KTH ACM Contest Template Library

KACTL version 2003-03-17

World Finals Edition

Contents

Chapter 1

Contest

1.1 Practice session

1.1.1 Checklist

Listing – checklist.cpp, p. 8

1.1.2 us_key.modmap

Listing – uskey.cpp, p. 8

1.2 Mandatory contest material

1.2.1 Problem assessment sheet

Usage

1.2.2 Template

Listing – Template.cpp, p. 9

Usage Standard problem template. Problems are classified as either of:

> simple-solve – Number of test cases is 1.
>
> for-solve – Number of test cases is given in the input.
>
> while-solve – End of test cases is indicated by special values or end of input. Stop when `solve` returns false.

1.2.3 Script

Listing – script.cpp, p. 9

Usage `sh script`

> c – Compile
>
> i – Enter program input
>
> o – Enter correct program answer
>
> t – Test using entered program input
>
> td – Test with direct typed input
>
> d – Diff the program output with the entered correct answer
>
> p – Print source code
>
> submit – Submit a solution!!
>
> n – New problem, copy Template
>
> f – Finished problem, move to done

1.2.4 adler

Listing – adler.cpp, p. 9

Usage `adler < code.cpp`

> Adler gives a checksum for all non-whitespace character it reads. All source code listings in this document have their checksum attached, calculated after comments and preprocessor directives have been stripped off (except for preprocessor directives in the code library utilities themselves).

1.3 Optional contest material

1.3.1 linecode

Listing – linecode.cpp, p. 10

Usage `linecode < code.cpp`

Linecode gives 16 values encoding line number xor:s of each line's adler checksum. All source code listings in this document have their line code attached. If a checksum doesn't match for a typed-in file, xor the line code from the document and the typed-in file to obtain candidate line numbers where the error might be.

1.3.2 xor

Listing – xor.cpp, p. 10 – A simple xor calculator.

1.3.3 Emacs Key Bindings

Listing – contest-keys.el.cpp, p. 10

Usage `M-x load-file RET contest-keys.el` contest-keys.el provides useful key bindings for Emacs.

C-x C-f – New file (overloads the usual find-file, uses Template.cpp)

C-c C-c – Compile (uses compile command as specified in c-lite.el)

C-c C-t – Test solution (using "`FILE < FILE.in`". The commented line is for testing using "`FILE`", i.e. when files are used instead of stdio)

C-c C-s – Submit solution (using "`submit FILE.cc`")

Practice Session

Listing 1.1: checklist.cpp — 1

0 lines
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0

```
Before contest checklist

Compiler
   What compiler?, what version?, what options?
Judge
   What types of error are there?

Practice session checklist

test keyboard
   Layout?
test contest utilities
   Scripts and checksum
test compiler
   Is long long available?
test STL (maybe not if compiler == g++ v2.95.3)
   Run testcode that tests all known (and used) features.
test editor
   Macros?
test judge
   Is stderr checked?
   Write code to cause every type of error? (except restricted function)
   What information is in the errormessages?
test printouts
   How to print?
   How long time to print?
test checking of scoreboard
   Is it possible to print the scoreboard?
```

Listing 1.2: uskey.cpp — 1

0 lines
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0

```
keycode 38 = 2 at
keycode 42 = 6 asciicircum
keycode 43 = 7 ampersand
keycode 44 = 8 asterisk
keycode 45 = 9 parenleft
keycode 46 = 0 parenright
keycode 52 = minus underscore
keycode 53 = equal plus
keycode 54 = bracketleft braceleft
keycode 55 = bracketright braceright
keycode 57 = backslash bar
keycode 58 = semicolon colon
keycode 59 = apostrophe quotedbl
keycode 61 = comma less
keycode 62 = period greater
keycode 63 = slash question
keycode 107 = grave asciitilde
```

Mandatory Contest Material

Listing 1.3: Template.cpp — 1

0 lines
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0

Listing 1.4: adler.cpp — ea0f4681

8 lines
c, 5, e, d, a, c, c, 3,
0, 9, 9, 8, d, d, a, 0

```
using namespace std;
int main() { // "Adler-32 by Mark Adler" -> 411f06c9
  unsigned long crcbase = 65521, s1 = 1, s2 = 0; unsigned char c;
  while (cin >> c) s1 = (s1 + c) % crcbase, s2 = (s2 + s1) % \
crcbase;
  cout << hex << (s2 << 16 | s1) << endl; return 0;
}
```

Listing 1.5: script.cpp — 1

0 lines
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0

```
mkdir data # done done/data

echo 'bc++ -q -lq -e$1 $1.cpp' > c

echo 'cat > data/$1_$2.in' > i

echo 'cat > data/$1_$2.ans' > o

echo 'cp data/$1_$2.in $1.in; ./$1 | tee data/$1_$2.out' > t

echo 'cat > $1.in; ./$1 | tee data/$1_$2.out' > td

echo 'diff data/$1_$2.out data/$1_$2.ans' > d

echo 'a2ps $1.cpp' > p


chmod +x c i o t td d p # submit n f


#echo 'bc++ -q -lq -O2 -w-par- -w-pia- -w-ovf- -e$1 $1.cpp' > c

#echo 'g++ -Wall -o $1 $1.cpp' > c
#echo './$1 < data/$1_$2.in | tee data/$1_$2.out' > t
#echo './$1 | tee data/$1_$2.out' > td
#echo 'mail judge@... < $1.cpp' > submit
#echo 'cp ../Template.cpp $1.cpp' > n
#echo 'mv $1* done; mv data/$1* done/data' > f
```

Optional Contest Material

Listing 1.6: linecode.cpp — 605da2fc

24 lines
d,1d,18, 7, e, 6, f, 1,
0, 1, 6,1c, c,1c, b,1b

```
// Failure probabilities:
//(of not detecting a single correct erroneous line number)
// Error on 1 line: 1/2^16 < 16ppm
//         2 lines: (2/4)^16 < 16ppm
//         3 lines: (5/8)^16 < .06%
//         4 lines: (12/16)^16 < 1.01%
//         5 lines: (27/32)^16 < 6.6%
//         6 lines: (58/64)^16 < 21%

// copy util/adler.cpp to get most of the adler function!!!
using namespace std;
int adler(istream &in) { // "Adler-32 by Mark Adler" -> 411f06c9
  unsigned long crcbase = 65521, s1 = 1, s2 = 0; unsigned char c;
  while (in >> c) s1 = (s1 + c) % crcbase, s2 = (s2 + s1) % \
crcbase;
  return s2 << 16 | s1;
}

int par[32];
int main() {
  string s; int lines = 0; // for every line's checksum,
  while (getline(cin, s)) { lines++; // for every bit of the sum,
    istringstream in(s); int sum = adler(in) - 1; // construct a \
parity
    for (int i = 0; i < 32; i++) if (sum & 1 << i) par[i] ^= \
lines;
  }
  cout << lines << " lines" << hex;
  for (int i = 0; i < 32; i++)
    if (i % 16 < 8) cout << ',' << setw(2) << par[i];
  cout << endl;
  return 0; // when two numers differ, xor them to get a line \
number
}
```

Listing 1.7: xor.cpp — 53224fac

17 lines
2,12,14, 2,1e,12,15, b,
10,13,17, 3,1a,13,1b, 1

```
using namespace std;

int main() {
  for (int i = 0; i < 16; i++) { // xor table
    for (int j = 0; j < 16; j++) {
      if (j > 0) cout << ' ';
      cout << hex << (i ^ j);
    }
    cout << endl;
  }
  int a, b;
  while (cin >> hex >> a >> hex >> b) // xor calculator
    cout << hex << (a ^ b) << ' ' << dec << (a ^ b) << endl;
  return 0;
}
```

```
xor table:
 0 1 2 3 4 5 6 7 8 9 a b c d e f
 1 0 3 2 5 4 7 6 9 8 b a d c f e
 2 3 0 1 6 7 4 5 a b 8 9 e f c d
 3 2 1 0 7 6 5 4 b a 9 8 f e d c
 4 5 6 7 0 1 2 3 c d e f 8 9 a b
 5 4 7 6 1 0 3 2 d c f e 9 8 b a
 6 7 4 5 2 3 0 1 e f c d a b 8 9
 7 6 5 4 3 2 1 0 f e d c b a 9 8
 8 9 a b c d e f 0 1 2 3 4 5 6 7
 9 8 b a d c f e 1 0 3 2 5 4 7 6
 a b 8 9 e f c d 2 3 0 1 6 7 4 5
 b a 9 8 f e d c 3 2 1 0 7 6 5 4
 c d e f 8 9 a b 4 5 6 7 0 1 2 3
 d c f e 9 8 b a 5 4 7 6 1 0 3 2
 e f c d a b 8 9 6 7 4 5 2 3 0 1
 f e d c b a 9 8 7 6 5 4 3 2 1 0
```

Listing 1.8: contest-keys.el.cpp — 1

0 lines
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0

```
(setq kactl-ext "cc")
(defun c-lite-compile () (interactive)
  (shell-command (concat "g++ -ansi -lm -O2 -pedantic -Wall -o "
                    (file-name-sans-extension buffer-file-name) " "
                    buffer-file-name)))

(defun c-lite-new-file (N) (interactive "FCFF: ")
  (find-file N) (or (file-exists-p N)
                  (not (string-equal (file-name-extension N) kactl-ext))
                  (insert-file "Template.cpp")))

(defun c-lite-test () (interactive)
  (let ((N (file-name-sans-extension buffer-file-name)))
    (shell-command (concat N " < " N ".in &"))))
;;(shell-command (file-name-sans-extension buffer-file-name)))

(defun c-lite-send () (interactive)
  (and (string-equal (file-name-extension buffer-file-name) kactl-ext)
     (y-or-n-p "Send? ") (shell-command (concat "submit " buffer-file-name))))

(global-set-key "\C-x\C-f" 'c-lite-new-file)
(global-set-key "\C-cc" 'c-lite-compile)
(global-set-key "\C-ct" 'c-lite-test)
(global-set-key "\C-cs" 'c-lite-send)
```

Chapter 2

Data Structures

2.1 STL containers

2.1.1 STL container summary

	[]	List op.	Front op.	Back op.	Iterators
vector	$\mathcal{O}(1)$	Am. $\mathcal{O}(n)$		Am. $\mathcal{O}(1)$	Ran
list		$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	Bi
deque	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	Ran
stack				$\mathcal{O}(1)$	
queue			$\mathcal{O}(1)$	$\mathcal{O}(1)$	
priority_queue			$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	
map	$\mathcal{O}(\log n)$	Am. $\mathcal{O}(\log n)$			Bi
multimap		Am. $\mathcal{O}(\log n)$			Bi
set		Am. $\mathcal{O}(\log n)$			Bi
multiset		Am. $\mathcal{O}(\log n)$			Bi
string	$\mathcal{O}(1)$	Am. $\mathcal{O}(n)$	Am. $\mathcal{O}(n)$	Am. $\mathcal{O}(1)$	Ran
array	$\mathcal{O}(1)$				Ran
valarray	$\mathcal{O}(1)$				Ran
bitset	$\mathcal{O}(1)$				

2.1.2 pair

Include `utility` or `algorithm`

Usage `make_pair`, `p.first`, `p.second`, <, ==.

2.1.3 string

Include `string`

Usage `substr`, <, ==.

2.1.4 vector

Include `vector`

Usage `resize`, `push_back`, `[]`.

Books JOS s148

2.1.5 deque

Include `deque`

Usage `push_front`, `push_back`, `[]`.

Books JOS s160

2.1.6 set

Include `set`

Usage `insert`, `erase`, `count`.

Books JOS s175

2.1.7 map

Include `map`

Usage `[]`, `count`.

Books JOS s194

2.1.8 list

Include `list`

Usage `push_front`, `push_back`, `splice`, `merge`, `sort`.

Books JOS s166

2.1.9 queue

Include `queue`

Usage `push`, `empty`, `front`, `pop`.

2.1.10 priority queue

Note! The `front()` is the element with the *highest* key.

Include `queue`

Usage `push`, `empty`, `front`, `pop`.

2.1.11 stack

Include `stack`

Usage `push`, `empty`, `top`, `pop`.

2.1.12 heap

Include `algorithm`

Usage `make_heap`, `push_heap`, `pop_heap`, `sort_heap`.

2.2 Null vector

2.2.1 null vector

Listing – null vector.cpp, p. 18

> `null vector` acts like a vector, but simply keeps one value. The value is reset and its reference returned for any index referenced.

Usage `v[3612378] = 5; v[3612378] == 0;`

2.3 Disjoint sets

2.3.1 sets

Listing – sets.cpp, p. 18

The `Kruskal` minimum spanning tree algorithm uses a data structure called `sets` to efficiently determine whether two vertices belong to the same tree.

2.4 Modifiable priority queue

2.4.1 mpq

Listing – mpq.cpp, p. 18

`mpq` is a modifiable priority queue (implemented as a set). Its interface is identical to that of a `priority_queue`. When an element should be modified the `update` method should be called as: `update(elem, oldvalue, newvalue)` Where `oldvalue` should be a *reference* to the value of the `elem`.

A common use is to use indices as elements which is compared using external containers.

2.4.2 update heap

Listing – update heap.cpp, p. 18

An updatable heap has an interface identical to that of a `priority_queue`. The elements need to have a method `set_position` though. When an element is changed, the `key_increased` or `key_decreased` method should be called with its position as argument.

2.5 Named items

When items are named, for example graph nodes identified by strings or big non-contiguous integers, it is often practical to keep a map from the names to an index numbering starting from 0, and a vector to retrieve a name back from an index.

The `index mapper` does this. It has function semantics to retrieve an index for a name, and vector semantics to retrieve a name from an index:

2.5.1 Index mapper

Usage `int idx = mapper("x") => mapper("x") == idx, mapper[idx] == "x"`

Listing – index mapper.cpp, p. 18

2.6 Matrices kept in arrays

It is often convenient to keep a two-dimensional matrix in a one-dimensional array, but then one has to explicitly calculate absolute indices into the array from the row and column of the matrix.

The `matrix mapper` helps with this. It has function semantics for retrieving absolute indices from a (row,column) pair, and, even better: matrix double bracket semantics for accessing the array elements directly as if they were in a matrix.

2.6.1 Matrix mapper

Usage `vector_matrix_mapper m(v, 12);`

```
v[row*12 + col] <=> v[m(row, col)] <=> m[row][col]
```

Listing – matrix mapper.cpp, p. 18

2.7 Indexed arrays

2.7.1 indexed

Listing – indexed.cpp, p. 18

2.8 Numerical data structures

2.8.1 Complex

Usage `#include <complex>`

2.8.2 Sign

Listing – sign.cpp, p. 18

2.8.3 Rational

Listing – rational.cpp, p. 18

2.8.4 Bigint

Listing – bigint.cpp, p. 20

2.8.5 Bigint Simple

Listing – bigint simple.cpp, p. 21

Fully dynamic BigInt class which handles +,-,* for positive integers. Can output numbers in base-10 only. This is a stripped down version of `bigint full`.

2.8.6 Bigint Full

Listing – bigint full.cpp, p. 23

Fully dynamic BigInt class which handles +,-,*,/ for positive integers. Can output numbers in base-10 only. Division/modulus is simple and uses neither an iterative method, such as Newton-Raphson, nor FFT. Has an iterative sqrt-function.

2.8.7 Bigint Per

Listing – bigint per.cpp, p. 25

Fully dynamic BigInt class which handles +,-,*. Additionally it can do divison/modulus with `ints`, find the n:th root of a number and do exponentiation with ˆ . Input and output in base 10.

2.8.8 Bigint Summary

This table summarise the capabilities of the different Bigint implementations.

	Bigint	BI Simple	BI Full	BI Per
Lines	185	192	318	167
Add, Sub, Cmp	$\mathcal{O}(n)$ for all four			
Mul with limb	$\mathcal{O}(n)$	N/A	N/A	$\mathcal{O}(n)$
Multiplication	$\mathcal{O}(n^2)$ for all four			
DivMod with limb	$\mathcal{O}(n)$	N/A	N/A	$\mathcal{O}(n)$
DivMod	$\mathcal{O}(n^2)$	N/A	yes	N/A
Exponentiation N^e	N/A	N/A	N/A	$\mathcal{O}(e \cdot n^2)$
Square root $\sqrt{N}$	N/A	N/A	yes	N/A
eth root $\sqrt[e]{N}$	N/A	N/A	N/A	yes
I/O Base	Any, but fixed	10	10	10
Bitops with limb	$\mathcal{O}(1)$	N/A	N/A	N/A
gcd	N/A	N/A	yes	N/A

Here, N is an n-bit number.

Data Structures

Listing 2.1: null vector.cpp — 6bc72061

4 lines
1, 3, 2, 5, 7, 4, 3, 5,
6, 7, 5, 1, 4, 5, 2, 3

Listing 2.2: sets.cpp — ae2ab730

35 lines
32,25, 3, d, 0,31,10,12,
11,3e,3f, 3,2d,1b,29,22

Listing 2.3: mpq.cpp — cd046c27

16 lines
0, b, e, 5, 4, 8, a, b,
e, 7, 1, 4, f, e, 4, d

Listing 2.4: update heap.cpp — eff5bc9

61 lines
2d,35,37,19,36,2f, 5,28,
39,18,30, 0, 9,1e,13,1f

Listing 2.5: index mapper.cpp — 1

0 lines
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0

Listing 2.6: matrix mapper.cpp — b928e66a

21 lines
1a,14,1a, 1,14, 9, 2,1e,
10, b,1a,1a,15, d, 2,11

Listing 2.7: indexed.cpp — 1

0 lines
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0

Listing 2.8: sign.cpp — 99285fac

36 lines
2b, 0, 6,17, 3, 4,25,17,
29,15, 5, 4,30,25,3d, 7

```
template <class T>
struct sign {
  static const T zero; // Requires declaration: const T sign<T>::zero = T();
  T x; bool neg;
  operator sign(T _x = zero, bool _neg = false) : x(_x), neg(_neg) { }
  bool operator <(const sign<T> &s) const {
    return neg==s.neg ? neg ? x>s.x : x<s.x : neg && !(x==zero&&s.x==zero);
  }
  bool operator ==(const sign<T> &s) const {
    return neg==s.neg ? x==s.x : x==zero&&s.x==zero;
  }
  sign<T> operator -() { return sign<T>(x, !neg); }
  sign<T> &addsub(bool add) {
    if (add) x+=s.x;
    else if (x<s.x) { T t=s.x; x = t-=x; neg=!neg; }
    else x-=s.x;
    return *this;
  }
  sign<T> &operator +=(const sign<T> &s) { return addsub(neg == s.neg); }
  sign<T> &operator -=(const sign<T> &s) { return addsub(neg != s.neg); }
  sign<T> &operator *=(const sign<T> &s) { x*=s.x, neg^=s.neg; return *this; }
  sign<T> &operator /=(const sign<T> &s) { x/=s.x, neg^=s.neg; return *this; }
};

template <class T>
sign<T> abs(const sign<T> &s) { return sign<T>(s.x, false); }

template <class T>
istream &operator >>(istream &in, sign<T> &s) {
  char c; in >> c; s.neg = c == '-'; if (!s.neg) in.unget(); in >> s.x;
}

template <class T>
ostream &operator <<(ostream &out, const sign<T> &s) {
  if (s.neg && s.x != s.zero) out << '-'; out << s.x;
}
```

Listing 2.9: rational.cpp — 169b04bf

81 lines
6e, d,4f, 7,37, 6,72,37,
1c,55,20,7c,2f,67,71,1e

```
#include "gcd.cpp"

template <class T>
struct rational {
  typedef rational<T> rT;
  typedef const rT & R;
  T n, d;
  rational(T _n=T(), T _d=T(1)) : n(_n), d(_d) { normalize(); }
  void normalize() {
    T f = gcd(n, d); n /= f; d /= f;
    if (d < T()) n *= -1, d *= -1;
  }
  bool operator < (R r) const { return n * r.d < d * r.n; }
  bool operator ==(R r) const { return n * r.d == d * r.n; }

  rT operator -() { return rT(-n, d); }

  rT operator +(R r) { return rT(n*r.d + r.n*d, d*r.d); }
  rT operator -(R r) { return rT(n*r.d - r.n*d, d*r.d); }

  rT operator *(R r) { return rT(n*r.n, d*r.d); }
  rT operator /(R r) { return rT( n*r.d,/**/d*r.n); }
  T/**//**/ div(R r) { return/**/(n*r.d) / (d*r.n); }
```

```
  rT operator %(R r) { return rT((n*r.d) % (d*r.n), d*r.d); }

  rT operator <<(int b) { return b<0 ? a>>-b : rT(n<<b, d); }
  rT operator >>(int b) { return b<0 ? a<<-b : rT(n, d<<b); }

  ostream &print_frac(ostream &out) {
    out << n; if (d != T(1)) out << '/' << d;
    return out;
  }

  istream &read_frac(istream &in) {
    in >> n;
    if (in.peek() == '/') { char c; in >> c >> d; } else d = T(1);
    normalize();
    return in;
  }
};

template <class T>
ostream &print_dec(ostream &out, const rational<T> &r,
                   int precision = 15, int radix = 10) {
  T n = r.n, d = r.d;
  if (n < T()) out << '-', n *= -1;
  out << n/d; n %= d;
  if (T() < n) {
    out << '.';
    for (int i = 0; n && i < precision; ++i) {
      n *= radix;
      out << n/d; n %= d;
    }
  }
  return out;
}

template <class T> ostream &operator <<(ostream &out, const rational<T> &r) {
  //return r.print_frac(out);
  return print_dec(out, r);
}

template <class T>
istream &read_dec(istream &in, rational<T> &r) {
  T i, f(0), z(1);
  in >> i;
  if (in.peek() == '.') {
    char c; in >> c;
    while (in.peek() == '0') { in >> c; z *= 10; }
    if (in.peek() >= '0' && in.peek() <= '9') in >> f;
  }
  r.d = T(1);
  while (r.d <= f) r.d *= 10;
  r.d *= z;
  r.n = i*r.d + f;
  r.normalize();
  return in;
}

template <class T> istream &operator >>(istream &in, rational<T> &r) {
  //return r.read_frac(in);
  return read_dec(in, r);
}
```

Bigint

Listing 2.10: bigint.cpp — fdf39bb8

185 lines
c7,21,a1, a,8d,75,20,9e,
6,97,1f,bc,28,bc,a6, 5

```
#include <vector>

template <class T, class M=T> // limb type, multiplication intermediate type
struct bigint {
 typedef bigint<T, M> S;
 typedef const S & R;

 static const T P; // maximum limb value
 static const unsigned N; // number of digits per limb
 vector<T> v; // limb vector

 bigint(T c = T()) { carry(c); }
 S &carry(T c) { while (c != T()) v.push_back(c % P), c /= P; return *this; }

 // limb access
 unsigned size() const { return v.size(); }
 T operator[](unsigned i) const { return v[i]; }

 // comparison
 bool operator <(R n) const {
  if (v.size() != n.size()) return v.size() < n.size();
  for (unsigned i = v.size(); i-- > 0; )
   if (v[i] != n[i]) return v[i] < n[i];
  return false;
 }
 bool operator ==(R n) const {
  if (v.size() != n.size()) return false;
  for (unsigned i = 0; i < v.size(); ++i)
   if (v[i] != n[i]) return false;
  return true;
 }

 // addition
 S &add(T c, unsigned i = 0) {
  while (c != T() && i < v.size())
   c += v[i], v[i] = c % P, c /= P, ++i;
  return carry(c);
 }
 S &operator ++() { return add(T(1)); }
 S operator ++(int) { S t = *this; add(T(1)); return t; }
 S &operator +=(T c) { return add(c); }
 S &operator +=(R n) {
  if (v.size() < n.size()) v.resize(n.size());
  T c = T();
  for (unsigned i = 0; i < n.size(); ++i)
   c += v[i] + n[i], v[i] = c % P, c /= P;
  add(c, n.size());
  return *this;
 }
 S operator +(T c) const { S t = *this; return t += c; }
 S operator +(R n) const { S t = *this; return t += n; }

 // subtraction
 S &sub(T c, unsigned i = 0) {
  for (; c != T() && i < v.size(); ++i)
   c += P-1 - v[i], v[i] = P-1 - c % P, c /= P;
  while (size() > 0 && v[size() - 1] == T()) v.pop_back();
  return *this;
 }
 S &operator --() { return sub(T(1)); }
 S operator --(int) { S t = *this; sub(T(1)); return t; }
 S &operator -=(T c) { return sub(c); }
 S &operator -=(R n) {
  if (v.size() < n.size()) v.resize(n.size()); // could be skipped
  T c = T();
  for (unsigned i = 0; i < n.size(); ++i)
   c += P-1 + n[i] - v[i], v[i] = P-1 - c % P, c /= P;
  sub(c, n.size());
  return *this;
 }
 S operator -(T c) const { S t = *this; return t -= c; }
 S operator -(R n) const { S t = *this; return t -= n; }

 // multiplication
 S &operator *=(T n) {
  M c = M();
  if (n == T())
   v.clear();
  else if (n != T(1))
   for (unsigned i = 0; i < v.size(); ++i)
    c += M(v[i]) * n, v[i] = T(c % P), c /= P;
  return carry(T(c));
 }
 S operator *(T c) const { S t = *this; return t *= c; }

 S operator *(R n) const {
  R m = *this;
  S r;
  if (m.size() > 0 && n.size() > 0) {
   r.v.resize(m.size() + n.size() - 1);
   for (unsigned i = 0; i < m.size(); ++i) {
    M c = M();
    for (unsigned j = 0; j < n.size(); ++j)
     c += r[i+j] + M(m[i]) * M(n[j]), r[i+j] = T(c % P), c /= P;
    r.add(T(c), i + n.size());
   }
  }
  return r;
 }
 S &operator *=(R n) { return *this = *this * n; }

 // division and modulo T
 S &divmod(T &d) {
  M c = M();
  for (unsigned i = size(); i-- > 0; )
   c = c * P + v[i], v[i] = T(c / d), c %= d;
  sub(T()); d = T(c); // sub to clear away zeros; return remainder in d
  return *this;
 }
 S &operator /=(T d) { return divmod(d); }
 S operator /(T d) const { S t = *this; t.divmod(d); return t; }
 S &operator %=(T d) { divmod(d); v.clear(); carry(d); return *this; }
 T operator %(T d) const { S t = *this; t.divmod(d); return d; }


 // long division
 S &divmod(R d, S &q) {
  S &r = *this; q = 0;
  S t = d, m = 1;
```

```
    while (t <= r) t *= 2, m *= 2;
    while (m > S(1)) {
      t /= 2, m /= 2;
      if (r >= t) r -= t, q += m;
    }
    return *this;
  }
  S &operator /=(R d) { S t = *this; t.divmod(d, *this); return *this; }
  S operator /(R d) const { S t = *this, q; t.divmod(d, q); return q; }
  S &operator %=(R d) { S q; return divmod(d, q); }
  S operator %(R d) const { S t = *this, q; t.divmod(d, q); return t; }


  // binary operations with T
  T operator &(T x) const { return v.empty() ? T(0) : v[0] & x; }
  S &operator <<=(int x) {
    while (x < 0) *this /= 2, ++x;
    while (x > 0) *this *= 2, --x;
    return *this;
  }
  S &operator >>=(int x) { return *this <<= -x; }

  S &operator &=(T x) { // allows *this &= ~3
    if (!v.empty()) v[0] &= ~T((1 << N) - 1) | x; // only the last N bits
    return *this;
  }
  S &operator |=(T x) {
    x &= T((1 << N) - 1);
    if (v.empty()) v.push_back(x); else v[0] |= x;
    return *this;
  }
  S &operator ^=(T x) {
    x &= T((1 << N) - 1);
    if (v.empty()) v.push_back(x); else v[0] ^= x;
    return *this;
  }
};

#include <iostream>
template <class T, class M>
ostream &operator <<(ostream &out, const bigint<T, M> &n) {
  if (n.size() > 0) {
    unsigned i = n.size() - 1;
    out << n[i];
    char fill = out.fill(); out.fill('0');
    while (i-- > 0)
      out.width(n.N), out << n[i]; // right-adjust is required (but default(?))
    out.fill(fill);
  }
  else
    out << '0';
  return out;
}

#include <string>
template <class T, class M>
istream& operator>>(istream& in, bigint<T, M> &n) {
  string s; in >> s;
  unsigned l = s.length();
  n.v.clear();
  while (l > 0) {
    T limb = T();
    for (unsigned k = l>n.N ? l-n.N : 0; k < l; ++k)
      limb = 10*limb + s[k]-'0';
    n.v.push_back(limb);
    l = l>n.N ? l-n.N : 0;
  }
  return in;
}

//typedef unsigned long ul;
//typedef unsigned long long ull;
//typedef bigint<ul, ull> big;
//const ul big::P = ul(1e9); // or 1e18, not using multiplication
//const unsigned big::N = 9; // or 18

//typedef unsigned short us;
//typedef unsigned long ul;
//typedef bigint<us, ul> big;
//const us big::P = us(1e4); // or 1e9, not using multiplication
//const unsigned big::N = 4; // or 9
```

Listing 2.11: bigint simple.cpp — f6ede821

192 lines
22,af,27,a4,99,63,c2,cd,
da,2d,cf,db,2a,11,b7,c3

```
class BigInt {
  // If no multiplication
  /* static const int NUMDIGITS = 9;
     static const int MAX = 1000000000; // 10^NUMDIGITS = 1e9
  */
  // If multiplication should be used
  static const int NUMDIGITS = 4;
  static const int MAX = 10000; // 10^NUMDIGITS = 1e4

  int *a;
  int l, res;

  void fix() { // Fix carry (when carry is -1, 0, +1)
    l = 1;
    for( int i=0; i<res; i++ ) {
      if( a[i] >= MAX ) {
        a[i] -= MAX;
        a[i+1]++;
      } else if( a[i] < 0 ) {
        a[i] += MAX;
        a[i+1]--;
      }
      if( a[i] != 0 )
        l = i+1;
    }
  }

  void set( const BigInt &x, int _res ) {
    int *newA = new int[res=_res];
    int i;

    for( i=0; i<x.l; i++ )
      newA[i] = x.a[i];
    for( ; i<res; i++ )
      newA[i] = 0;

    delete[] a;
    a = newA;
    l = x.l;
```

```
  }

public:
  static const BigInt zero;
  static const BigInt one;

  BigInt( unsigned int x=0 ) {
    a = new int[res=2];

    a[0] = x%MAX;
    a[1] = x/MAX;
    l = (x>=(unsigned)MAX ? 2:1);
  }
  BigInt( const BigInt &x ) {
    a = (int *)0;
    set( x, x.l );
  }
  ~BigInt() {
    delete[] a;
  }

  BigInt &operator=( const BigInt &x ) {
    set( x, x.l );
    return *this;
  }

  BigInt & operator+=( const BigInt &x ) {
    // Alloc larger array if there could be an overflow
    if( x.l > res || (l==res && x.l==res) )
      set( *this, x.l*2 );

    for( int i=0; i<x.l; i++ )
      a[i] += x.a[i];
    fix();

    return *this;
  }

  BigInt & operator-=( const BigInt &x ) {
    for( int i=0; i<x.l; i++ )
      a[i] -= x.a[i];
    fix();

    return *this;
  }

  BigInt & operator*=( const BigInt &x ) {
    BigInt prod;

    prod.set( 0, (l+x.l+1) );

    for( int i=0; i<x.l; i++ ) {
      for( int j=0; j<l; j++ ) {
        int s = prod.a[i+j]+x.a[i]*a[j];

        prod.a[i+j] = s%MAX;
        prod.a[i+j+1] += s/MAX;
      }
    }
    prod.fix();
    *this = prod;

    return *this;
  }

  int comp( const BigInt &x ) const {
    int d = l-x.l;

    if( d != 0 )
      return d;

    for( int i=l-1; i>=0; i-- ) {
      d = a[i]-x.a[i];

      if( d != 0 )
        return d;
    }
    return 0;
  }

  bool operator<( const BigInt &x ) const { return comp(x)<0; }
  bool operator<=( const BigInt &x ) const { return comp(x)<=0; }
  bool operator==( const BigInt &x ) const { return comp(x)==0; }
  bool operator!=( const BigInt &x ) const { return comp(x)!=0; }


  void print( ostream &out=cout ) const {
    bool flag = false;

    for( int i=l-1; i>=0; i-- ) {
      int b = a[i];

      if( flag ) {
        for( int j=MAX/10; j>b; j/=10 )
          out << '0';
        if( b>0 )
          out << b;
      } else if( i==0 || b>0 ) {
        out << b;
        flag = b>0;
      }
    }
  }

  void input( istream &in=cin ) {
    *this = 0;

    // Skip leading whitespace
    char c=' ';
    while(c==' ' || c=='\t' || c==10 || c==13) {
      in.get(c);
      if( !in.good() )
        return;
    }

    // Read word-wise
    int k=1, n=0;
    while(c>='0' && c<='9' && in.good() ) {
      n=n*10+(c-'0');
      k*=10;

      // Store word
      if( k>=MAX ) {
        *this *= MAX;
        a[0] = n;
        n = 0;
        k = 1;
      }
```

```
    in.get(c);
  }
  *this *= k;
  a[0] += n;

  if( in.good() )
    in.putback(c);
 }

 friend ostream &operator<<(ostream &lhs,const BigInt &rhs);
 friend istream &operator>>(istream &lhs,BigInt &rhs);
};

const BigInt BigInt::zero = BigInt(0);
const BigInt BigInt::one = BigInt(1);


ostream & operator<<(ostream &lhs,const BigInt &rhs) {
 rhs.print( lhs );
 return lhs;
}

istream & operator>>(istream &lhs,BigInt &rhs) {
 rhs.input( lhs );
 return lhs;
}
```

Listing 2.12: bigint full.cpp — 9c9b6cd

318 lines
59,1af,bb,e4,1b0,f2,184, 7,
1b9,19d,a9,9c, 6, 8,1e6,32

```
#include <algorithm> // for swap in _gcd

class BigInt {
 // If no multiplication
 /* static const int NUMDIGITS = 9;
    static const int MAX = 1000000000; // 10^NUMDIGITS = 1e9
 */
 // If multiplication should be used
 static const int NUMDIGITS = 4;
 static const int MAX = 10000; // 10^NUMDIGITS = 1e4

 int *a;
 int l, res;

 void fix() { // Fix carry (when carry is -1, 0, +1)
  l = 1;
  for( int i=0; i<res; i++ ) {
   if( a[i] >= MAX ) {
    a[i] -= MAX;
    a[i+1]++;
   } else if( a[i] < 0 ) {
    a[i] += MAX;
    a[i+1]--;
   }
   if( a[i] != 0 )
    l = i+1;
  }
 }

 void set( const BigInt &x, int _res ) {
  int *newA = new int[res=_res];
  int i;

  for( i=0; i<x.l; i++ )
   newA[i] = x.a[i];
  for( ; i<res; i++ )
   newA[i] = 0;

  delete[] a;
  a = newA;
  l = x.l;
 }

public:
 static const BigInt zero;
 static const BigInt one;

 BigInt( unsigned int x=0 ) {
  a = new int[res=2];

  a[0] = x%MAX;
  a[1] = x/MAX;
  l = (x>=(unsigned)MAX ? 2:1);
 }
 BigInt( const BigInt &x ) {
  a = (int *)0;
  set( x, x.l );
 }
 ~BigInt() {
  delete[] a;
 }

 BigInt &operator=( const BigInt &x ) {
  set( x, x.l );
  return *this;
 }

 BigInt & operator+=( const BigInt &x ) {
  // Alloc larger array if there could be an overflow
  if( x.l > res || (l==res && x.l==res) )
   set( *this, x.l*2 );

  for( int i=0; i<x.l; i++ )
   a[i] += x.a[i];
  fix();

  return *this;
 }

 BigInt & operator-=( const BigInt &x ) {
  for( int i=0; i<x.l; i++ )
   a[i] -= x.a[i];
  fix();

  return *this;
 }

 BigInt & operator*=( const BigInt &x ) {
  BigInt prod;

  prod.set( 0, (l+x.l+1) );

  for( int i=0; i<x.l; i++ ) {
```

```
    for( int j=0; j<l; j++ ) {
      int s = prod.a[i+j]+x.a[i]*a[j];

      prod.a[i+j] = s%MAX;
      prod.a[i+j+1] += s/MAX;
    }
  }
  prod.fix();
  *this = prod;

  return *this;
}


BigInt & operator/=( const BigInt &x ) {
  BigInt rem;

  div( x, *this, rem );
  return *this;
}

BigInt & operator%=( const BigInt &x ) {
  BigInt quot;

  div( x, quot, *this );
  return *this;
}

void div( const BigInt &d, BigInt &quot, BigInt &rem ) const {
  BigInt divisor = d;
  int scaling = 0;

  // Remainder = dividend
  rem.set( *this, l+1 );

  // Check for dividend < divisor (length-wise)
  if( l < d.l ) {
    quot.set( 0, 1 );
    return;
  }

  // Quotient = 0
  quot.set( 0, (l-d.l+1) );

  // Make sure a[l-1] is >=MAX/10 (for better guesses)
  if( divisor.l > 1 ) {
    int a = divisor.a[ divisor.l-1 ];

    while( (a*=10) < MAX ) {
      rem *= 10;
      divisor *= 10;
      scaling++;
    }
  }

  while( divisor <= rem ) {
    // Guess a quotient and subtract from remainder. We always
    // underestimate the quotient so we won't get any underflow.
    int dh = divisor.a[ divisor.l-1 ]+1;
    BigInt qadd;

    if( rem.l > 1 ) {
      int guess = (rem.a[ rem.l-1 ]*MAX + rem.a[ rem.l-2 ])/dh;

      // Scale guess to right position
      qadd.set( 0, rem.l-divisor.l+1 );
      qadd.a[ rem.l-divisor.l ] = guess/MAX;
      if( rem.l > divisor.l )
        qadd.a[ rem.l-divisor.l-1 ] = guess%MAX;
      qadd.l = rem.l-divisor.l+1;
      if( guess < MAX ) {
        if( qadd.l > 1 )
          qadd.l--;
        else        // (This implies that guess == 0)
          qadd.a[0]++; // Fix case where x/x = 0 due to round-down.
      }
    } else {
      int guess = rem.a[0]/dh;
      if( guess == 0 ) guess++;

      qadd.set( guess, 1 );
    }

    // Add guess to quotient
    quot += qadd;

    // Subtract div*guess from remainder
    BigInt remsub( qadd );
    remsub *= divisor;
    rem -= remsub;
  }

  while( scaling > 0 ) {
    rem /= 10;
    scaling--;
  }
}

void sqrt( BigInt &res ) const {
  // Newton-Raphson's method. Recursion: y' = y-(y^2-x)/(2y) = (y+x/y)/2
  if( *this == zero || *this == one ) {
    // special case for x=0,1 in sqrt(x) since x/2 is 0 in that case.
    res = *this;
  } else {
    res = *this;
    res /= 2;

    while( true ) {
      BigInt d = *this;
      d /= res;
      d += res;
      d /= 2;
      if( !(d<res) ) // acc. to Erik Nordenstam. Shouldn't d==res suffice?
        break;
      res = d;
    }
  }
}


int comp( const BigInt &x ) const {
  int d = l-x.l;

  if( d != 0 )
    return d;

  for( int i=l-1; i>=0; i-- ) {
    d = a[i]-x.a[i];
```

```
    if( d != 0 )
      return d;
  }
  return 0;
}

bool operator<( const BigInt &x ) const { return comp(x)<0; }
bool operator<=( const BigInt &x ) const { return comp(x)<=0; }
bool operator==( const BigInt &x ) const { return comp(x)==0; }
bool operator!=( const BigInt &x ) const { return comp(x)!=0; }


void print( ostream &out=cout ) const {
  bool flag = false;

  for( int i=l-1; i>=0; i-- ) {
    int b = a[i];

    if( flag ) {
      for( int j=MAX/10; j>b; j/=10 )
        out << '0';
      if( b>0 )
        out << b;
    } else if( i==0 || b>0 ) {
      out << b;
      flag = b>0;
    }
  }
}

void input( istream &in=cin ) {
  *this = 0;

  // Skip leading whitespace
  char c=' ';
  while(c==' ' || c=='\t' || c==10 || c==13) {
    in.get(c);
    if( !in.good() )
      return;
  }

  // Read word-wise
  int k=1, n=0;
  while(c>='0' && c<='9' && in.good() ) {
    n=n*10+(c-'0');
    k*=10;

    // Store word
    if( k>=MAX ) {
      *this *= MAX;
      a[0] = n;
      n = 0;
      k = 1;
    }

    in.get(c);
  }
  *this *= k;
  a[0] += n;

  if( in.good() )
    in.putback(c);
}
```

```
  friend ostream &operator<<(ostream &lhs,const BigInt &rhs);
  friend istream &operator>>(istream &lhs,BigInt &rhs);

  static void _gcd( BigInt &a, BigInt &b );
  static void gcd( const BigInt &a, const BigInt &b, BigInt &res );
};

const BigInt BigInt::zero = BigInt(0);
const BigInt BigInt::one = BigInt(1);


ostream & operator<<(ostream &lhs,const BigInt &rhs) {
  rhs.print( lhs );
  return lhs;
}

istream & operator>>(istream &lhs,BigInt &rhs) {
  rhs.input( lhs );
  return lhs;
}

void BigInt::_gcd( BigInt &a, BigInt &b ) {
  BigInt *pa=&a, *pb=&b;

  while( *pb != zero ) {
    *pa %= *pb;
    swap( pa, pb );
  }
  a = *pa;
}

void BigInt::gcd( const BigInt &a, const BigInt &b, BigInt &res ) {
  BigInt c=b;
  res = a;
  _gcd( res, c );
}
```

Listing 2.13: bigint per.cpp — 34746a10

167 lines
5,12,f9,36,7d,f1,5c,3c,
9d,80,71,44,67,b6,f9,35

```
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>

/* in order for subtraction to work properly, limb needs to be signed. */
typedef long long limb;
typedef vector<limb> bigint;
typedef bigint::const_iterator bcit;
typedef bigint::reverse_iterator brit;
typedef bigint::const_reverse_iterator bcrit;
typedef bigint::iterator bit;

/* size of limbs
 *          123456789 */
#define LSIZE 1000000000
#define LIMBDIGS 9

bigint BigInt(limb i) {
```

```
  bigint res;
  do {
    res.push_back(i % LSIZE);
  } while (i /= LSIZE);
  return res;
}

istream& operator>>(istream& i, bigint& n) {
  string s;
  i >> s;
  int l = s.length();
  n.clear();
  while (l > 0) {
    int j = 0;
    for (int k = l > LIMBDIGS ? l-LIMBDIGS: 0; k < l; ++k)
      j = 10*j + s[k]-'0';
    n.push_back(j);
    l -= LIMBDIGS;
  }
  return i;
}

/* Warning: the ostream must be configured to print things with right
 * justification.     */
ostream& operator<<(ostream& o, const bigint& n) {
  int began = 0;
  char ofill = o.fill();
  o.fill('0');
  for (bcrit i = n.rbegin(); i != n.rend(); ++i) {
    if (began) o << setw(LIMBDIGS);
    if (*i) began = 1;
    if (began) o << *i;
  }
  if (!began) o << "0";
  o.fill(ofill);
  return o;
}

/* The base comparison function. semantics like strcmp(...). */
int cmp(const bigint& n1, const bigint& n2) {
  int x = n2.size() - n1.size();
  bcit i = n1.end()-1;
  bcit j = n2.end()-1;
  if (x > 0) {
    while (x--)
      if (*j--) return -1;
  } else if (x < 0) {
    while (x++)
      if (*i--) return 1;
  }
  for (; i + 1 != n1.begin(); --i, --j)
    if (*i != *j)
      return *i-*j;
  return 0;
}

/* The other operators will be automatically defined by STL */
bool operator==(const bigint& n1, const bigint& n2) { return !cmp(n1,n2); }
bool operator<(const bigint& n1, const bigint& n2) { return cmp(n1,n2) < 0; }


bigint& operator+=(bigint& a, const bigint& b) {
  if (a.size() < b.size()) a.resize(b.size());
  limb cy = 0;
  bit i = a.begin();
  for (bcit j = b.begin(); i != a.end() && (cy || j < b.end()); ++j, ++i)
    cy += *i + (j < b.end() ? *j : 0), *i = cy % LSIZE, cy /= LSIZE;
  if (cy) a.push_back(cy);
  return a;
}

/* Returns true if sign changed */
bool sub(bigint& a, const bigint& b) {
  if (a.size() < b.size()) a.resize(b.size());
  limb cy = 0;
  bit i = a.begin();
  for (bcit j = b.begin(); i != a.end() && (cy || j < b.end()); ++j, ++i) {
    *i -= cy + (j < b.end() ? *j : 0);
    if ((cy = *i < 0)) *i += LSIZE;
  }
  /* If sign changed, flip all digits. These three lines can be
   * ignored if it is known that the sign will not change, e.g. when
   * using bigint in conjunction with sign.cpp or in many
   * combinatorial counting problems.      */
  if (cy)
    while (i > a.begin())
      *i = LSIZE - *--i;
  return cy;
}

bigint& operator-=(bigint& a, const bigint& b) {
  sub(a, b);
  return a;
}

bigint& operator*=(bigint& a, limb b) {
  limb cy = 0;
  for (bit i = a.begin(); i != a.end(); ++i)
    cy = cy / LSIZE + *i * b, *i = cy % LSIZE;
  while (cy /= LSIZE)
    a.push_back(cy % LSIZE);
  return a;
}

bigint& operator*=(bigint& a, const bigint& b) {
  bigint x = a, y, bb = b;
  a.clear();
  for (bcit i = bb.begin(); i != bb.end(); ++i, ++j)
    (y = x) *= *i, a += y, x.insert(x.begin(), 0);
  return a;
}


bigint& operator^=(bigint& a, limb b) {
  bigint aa = a;
  a.clear(); a.push_back(1);
  while (b) {
    if (b & 1) a *= aa;
    aa *= aa;
    b >>= 1;
  }
  return a;
}


bigint& divmod(bigint& a, limb b, limb* rest = NULL) {
  limb cy = 0;
  for (brit i = a.rbegin(); i != a.rend(); ++i)
```

```
    cy += *i, *i = cy / b, cy = (cy % b) * LSIZE;
  if (rest)
    *rest = cy / LSIZE;
  return a;
}

bigint& operator/=(bigint& a, limb b) { return divmod(a, b); }
limb operator%(const bigint& a, limb b) {
  limb res;
  bigint fubar = a;
  divmod(fubar, b, &res);
  return res;
}


/* Finds the e:th root of n in far worse time than necessary.
 * Returns 0 if the root doesn't exist.  */
bigint root(const bigint& n, limb e) {
  int f;
  bigint lo = BigInt(0), hi, m, n2;
  hi = BigInt(LSIZE);
  // let hi ~ LS^(1+log_LS(n))/e) = O(n^(1 + 1/e))
  hi ^= ((n.size()+1) / e) + 1;
  while (1) {
    (m = lo) += hi;
    divmod(m, 2);
    if (m == lo)
      break;
    (n2 = m) ^= e;
    f = cmp(n, n2);
    if (f < 0)
      hi = m;
    else if (f > 0)
      lo = m;
    else
      return m;
  }
  /* If just an approximation of the root is wanted, change return
   * statement to:
   * return lo;    (for floor(root))
   * return hi;    (for ceil(root))            */
  return BigInt(0);
}
```

Chapter 3

Number Theory

3.1 Divisibility

3.1.1 GCD

Listing – gcd.cpp, p. 37

Usage `d = gcd( a, b );`

Complexity $\mathcal{O}(\log(b))$

Listing – gcd fast.cpp, p. 37

Usage `d = gcd_fast(a, b);`

Complexity $\mathcal{O}(\log(a) + \log(b))$

Note! The `gcd_fast` routine is slightly more writing than the usual gcd routine, but can be useful where speed is *really essential*, as it only uses subtraction and shift operations instead of the evil modulo.

Valladolid 202

3.1.2 LCM

$\mathrm{lcm}(a,b) = \frac{ab}{\gcd(a,b)}$

3.1.3 Euclid

Listing – euclid.cpp, p. 37

Usage `d = euclid( a, b, &x, &y );`

x, y will satisfy $ax + by = d$ after the call.

Listing – poseuclid.cpp, p. 37

$x, y \geq 0$ will satisfy $ax - by = \pm d$ after the call.

Complexity $\mathcal{O}(\log(b))$

Note! x and y have the smallest absolute value

Example

```
euclid( 10, 6, x, y ) == 2, x == -1, y == 2
```

Valladolid 202

3.1.4 Chinese remainder theorem

Listing – chinese.cpp, p. 37

Given $x = a \mod m$, $x = b \mod n$, calculates $x \mod mn$.

Note! When poseuclid is used, mn may be returned.

3.1.5 ϕ-function

Listing – phi.cpp, p. 37

$\phi(n) = \#\{d < n | \gcd(n,d) = 1\}$

If $n = \prod p_i^{k_i}$, $\phi(n) = n \prod \left(1 - \frac{1}{p_i}\right) = \prod p_i^{k_i - 1}(p_i - 1)$

Complexity $\mathcal{O}(\sqrt{n})$

3.1.6 Perfect numbers

When n is even, it is a perfect number iff it is of the form $\frac{p(p+1)}{2}$, where p is a Mersenne prime. Mersenne primes are primes of the form $p = 2^k - 1$. The first 27 Mersenne primes (and thus the first 27 even perfect numbers) are obtained for $k = 2$, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941, 11213, 19937, 21701, 23209, 44497.

It is conjectured that there are no odd perfect numbers.

3.2 Primes

The first 100 primes are:

2	3	5	7	11	13	17	19	23
29	31	37	41	43	47	53	59	61
67	71	73	79	83	89	97	101	103
107	109	113	127	131	137	139	149	151
157	163	167	173	179	181	191	193	197
199	211	223	227	229	233	239	241	251
257	263	269	271	277	281	283	293	307
311	313	317	331	337	347	349	353	359
367	373	379	383	389	397	401	409	419
421	431	433	439	443	449	457	461	463
467	479	487	491	499	503	509	521	523
541								

The 1000th prime is 7919. The first every 10000th primes are:

104729	224737	350377	479909	611953	746773	882377	1020379	1159523
1299709								

Some primes closest below powers of two are (Mersenne primes are starred):

```
      3*       7*      13       31*      61      127*     251      509     1021
   2039    4093    8191*  16381    32749    65521   131071* 262139   524287*(19)
1048573              2147483647*(31)              2305843009213693951*(61)
    618970019642690137449562111*(89)  162259276829213363391578010288127*(107)
                              170141183460469231731687303715884105727*(127)
```

3.2.1 Primes

Listing – primes.cpp, p. 39

Usage `primes<int> p; p.generate(1000); n = factor(10); n==2;`

> `primes` calculates a vector of primes and has a factor method which returns the smallest factor in the given integer.

3.2.2 Prime Sieve

Listing – prime sieve.cpp, p. 40

Usage `prime_sieve p(1000); p.isprime(10)==false;`

> `prime_sieve` calculates a bool-vector containing whether an integer is prime. It is faster than `primes` when dealing with large numbers. Returns whether an integer is prime in constant time.

3.2.3 Primes Many

Listing – primes many simple.cpp, p. 39

Listing – primes many fast.cpp, p. 39

Usage `primes_many p(100000); p.primes[2] == 5;`

primes_many_simple and primes_many_fast calculates a vector of primes and is suitable when dealing with lots of primes (10000 or more). It calculates them by first sieving and then putting them into an array.

3.2.4 Miller-Rabin

Listing – miller-rabin.cpp, p. 40

Listing – miller-rabin-2.cpp, p. 41

Usage `isprime_rabin_miller( n, s );`

Complexity $\mathcal{O}(s \log(n))$

isprime_rabin_miller is a probabilistic primality test. It will never say that a prime is composite but may erroneous claim that a composite number is prime. The argument s is the number of iterations to be used. The probability of a false answer is not more than 2^{-s} so $s = 50$ is more than enough for most applications. The integer to be tested n may be as large as $2^{63} - 1$.

3.2.5 Pollard-ρ

Listing – pollard-rho.cpp, p. 41

Complexity $\mathcal{O}\left(\sqrt[4]{N}\right)$

Finds a non-trivial factor of N, provided that there is any.

3.2.6 Number of divisors

Listing – ndivisors.cpp, p. 41

Usage `primes_many_fast p(100000); ndivisors_prob(23424234234243,p.primes)==8;`

ndivisors and ndivisors_prob calculates the number of divisors. ndivisors needs a prime-table with all primes upto the square root of the number and ndivisors_prob only primes as large as the third root of the number. The second algorithm uses a probabilistic primality test (Miller-Rabin).

3.2.7 Factor

Listing – factor.cpp, p. 42

Usage `int n = 2*2*3*3*5; vector<pair<int, int> > &facs = Factor(n);`

Factor calculates the prime factorization of an integer. The returned vector is an increasing sequence of primes/exponents pairs. That is (letting p_i = Factor(n)[i].first, and e_i = Factor(n)[i].second): $n = \prod_i p_i^{e_i}$, where all p_i are prime and $i < j \Leftrightarrow p_i < p_j$.

As it basically works by dynamic programming, Factor requires roughly $\mathcal{O}(n \log\log n)$ memory, but the good news is that the time complexity for factoring all numbers from 1 to n is also roughly $\mathcal{O}(n \log\log n)$, and that once a number's been factored, Factor remembers the factorization and can give it in constant time.

3.2.8 Prime factorization

Listing – factor2.cpp, p. 42

Usage vector¡long long¿ factors; factor(N, factors);

`factor` calculates the prime factorization of an integer, but a lost faster than `Factor` above, since it uses Pollard-ρ.

3.3 Josephus

3.3.1 Josephus

Listing – josephus.cpp, p. 45

Complexity $\mathcal{O}\left(log_{\frac{k}{k-1}}(n)\right)$

Josephus is the problem to determine which person remains when repeatedly removing the k:th person from a total of n persons (cyclic).

3.4 Random

3.4.1 Pseudo random numbers

Listing – pseudo.cpp, p. 45

`pseudo` gives a pseudo-random integer in $[0, 2^{31} - 1]$.

`ullpseudo` gives a pseudo-random integer in $[0, 2^{62} - 1]$.

`fpseudo` gives a pseudo-random number in $[0, 1)$.

Complexity $\mathcal{O}(1)$

3.5 Linear Equations

3.5.1 Solving linear equations

Listing – solve linear.cpp, p. 43

Listing – solve linear TO.cpp, p. 43

3.5.2 Calculating determinant

Listing – determinant.cpp, p. 44

Listing – int determinant.cpp, p. 44

`determinant` and `int_determinant` both reduces the matrix to an upper diagonal form using elementary row operations. There could be an overflow in the integral variant and in that case the double variant can be used instead, rounding the answer at the end. The strength of `int_determinant` is that it can be used for `long long` or `BigInt`. Note that it uses `euclid` which could be rather slow in the `BigInt` case.

3.6 Big numerical operations

3.6.1 exp

Listing – exp.cpp, p. 45

Complexity $\mathcal{O}(\log e \text{ ops})$

`exp` calculates b^e by repeated squaring.

3.6.2 mulmod

Listing – mulmod.cpp, p. 45

Usage `mulmod(5678945893454353ULL, 2423948234343ULL, 3123123123123123ULL) == 1435590358119892;`

`mulmod` calculates $ab \bmod n$ in a way that allows numbers as large as MAXINT/2 to be used (which would otherwise give an overflow).

3.6.3 expmod

Listing – expmod.cpp, p. 45

Complexity $\mathcal{O}(\log e \text{ ops})$

`expmod` calculates $b^e \mod n$ by repeated squaring.

3.6.4 Bit manipulations

Listing – bitmanip.cpp, p. 45

Lowest bit, sign, power-of-two check, power-of-two round-up, next number with same number of bits, bit count, bit reversal, binary length.

3.7 Coordinates and directions

3.7.1 coords

Listing – coords.cpp, p. 46

Usage `sqrX X(cols); idx = X(3, 4); X(idx, &r, &c);`

Usage `cubeX X(rows, cols); idx = X(3, 4, 5); X(idx, &l, &r, &c);`

Usage `quadX X(levs, rows, cols); idx = X(3, 4, 5, 6); X(idx, &h, &l, &r, &c);`

Usage `triX X; idx = X(3, 4); X(idx, &r, &c);`

Coordinate to index and index to coordinate calculations for plane, space and hyperspace grid and plane triangle layouts.

Usage `dxy(dir, &dx, &dy);`

Usage `drc(dir, &dr, &dc);`

Usage `dknight(dir, &dr, &dc);`

Direction to x and y or row and column displacement. `dir` values 0, 1, 2, 3 mod 4 correspond to right, up, left, right or north, east, south, west, respectively. Or chess knight jump directions.

3.8 Optimization

3.8.1 Simplex method

Listing – simplex.cpp, p. 47

Solves a linear minimization problem. The first row of the input matrix is the objective function to be minimized. The first column is the maximum allowed value for each linear row.

3.9 Finding roots of polynomials

3.9.1 Newton's method

Listing – poly roots.cpp, p. 48

Finds roots of a polynomial without multiple roots. The roots are found from the left to the right and the input `xmin` variable needs to be lesser than all roots. A sufficiently small value (?) is $-\sum_{i=1}^{n}|a_i|/|a_0| - \epsilon$ where the polynomial is $a_0x^n + a_1x^{n-1} + \cdots + a_n$.

3.9.2 Newton's method

Listing – poly roots bisect.cpp, p. 48

Finds roots of a polynomial p without multiple roots. The roots are found by bisecting a given interval until it contains $\deg(p)$ sign-changing intervals. Then find the roots with either Newton's algorithm (with bisective fall-back) or the bisective method.

Divisibility

Listing 3.1: gcd.cpp — 4a431429

6 lines
2, 1, 7, 2, 0, 3, 2, 0,
6, 0, 3, 2, 6, 2, 4, 1

```
int gcd( int a, int b ) {
  if( b==0 )
    return a;
  else
    return gcd( b, a%b );
}
```

Listing 3.2: gcd fast.cpp — 552a5004

25 lines
9, 0, 4, a,16,1d,11,1a,
a,14, 0, a,1f,1a, a, d

```
template<class T>
T gcd_fast(T a, T b) {
  int twos = 0;
  if (a < 0) a = -a;
  if (b < 0) b = -b;
  if (!a)   a = 1;
  if (!b)   b = 1;
  while ((~a & 1) && (~b & 1)) {
    ++twos;
    a >>= 1;
    b >>= 1;
  }
  while (~a & 1) a >>= 1;
  while (~b & 1) b >>= 1;
  while (a != b) {
    if (a > b) {
      a -= b;
      while (~a & 1) a >>= 1;
    } else {
      b -= a;
      while (~b & 1) b >>= 1;
    }
  }
  return a <<= twos;
}
```

Listing 3.3: euclid.cpp — c1126eb

10 lines
8, f, 8, d, 9, 9, 3, 6,
f, 6, 4, 7, f, 8, f, 1

```
template <class T>
T euclid( T a, T b, T &x, T &y ) {
  if( b==T(0) )
    return x = T(1), y = T(0), a;
  else {
    T d = euclid( b, a%b, y, x );
    y -= a/b*x;
    return d;
  }
}
```

Listing 3.4: poseuclid.cpp — 2c92298d

10 lines
8, c, c, b, d, f, 1, 2,
9, 0, 0, 1, c, c, b, 1

```
template <class T>
T poseuclid( T a, T b, T &x, T &y ) {
  if( b==T(0) )
    return x = T(1), y = T(0), a;
  else {
    T d = poseuclid( b, a%b, y, x );
    y += a/b*x;
    return d;
  }
}
```

Listing 3.5: chinese.cpp — e3304c90

14 lines
3, d, 9, f, 3, 4, 9, 6,
4, 8, 3, d, 6, 3, b, 9

```
// x=a(mod m), x=b(mod n), find x (mod nm), assuming (n,m)=1
template <class T>
T chinese(T a, T m, T b, T n) {
  T x, y;
  poseuclid(m, n, x, y);
  big a1 = a * y * n;
  big b1 = b * x * m;
  big r = a1 > b1 ? a1 - b1 : b1 - a1;
  if (r % m != a) r = m * n - r % (m * n);
  return r;
}

big chinese(big a, big m, big b, big n, big c, big o) {
  return chinese(a, m, chinese(b, n, c, o), n * o);
}
```

Listing 3.6: phi.cpp — 26384705

25 lines
16,15,14, 1,10,14, c, 7,
e, 1,1c, 1,11,19, 0, c

```
template <class T>
void elim(T& n, T i) { while (!(n % i)) n /= i; }

template <class T>
T phi(T n) {
  T i, res = n;
  if (!(n % 2)) {
    elim(n, 2);
    res /= 2;
  }
  i = 3;
  while (i*i <= n) {
    if (!(n % i)) {
      elim(n, i);
```

```
      res /= i;
      res *= i-1;
    }
    i+= 2;
  }
  if (n > 1) {
    res /= n;
    res *= n-1;
  }
  return res;
}
```

Primes

Listing 3.7: primes.cpp — 150167ce

24 lines
7, 5,13,1d, d,14,10, 7,
5, c, 4, e,1b, 7, c, 0

```
#include <vector>

using namespace std;

template <class T>
struct primes {
  typedef vector<int> V;
  V v; // primes vector
  int k; // next number to check
  primes() : k(3) { v.push_back(2); }

  void generate(T n2) {
    while ((T) k * k <= n2) {
      if ((int) factor(k) == k)
        v.push_back(k);
      k += 2;
    }
  }

  T factor(T n) {
    generate(n);
    for (V::iterator i = v.begin(); i != v.end() && (*i) * (*i) <= n; i++)
      if (n % (*i) == 0) return (*i);
    return n;
  }
};
```

Listing 3.8: primes many simple.cpp — cf31cb0e

38 lines
c,1d,3c,1d,3f,39,23,3b,
3f,16,27,38, 1,1b,2a,1e

```
#include <vector>
#include <memory>

struct primes_many_simple {
  vector< unsigned int > primes;

  primes_many_simple( int n ) {
    unsigned char *v;
    unsigned int nPrimes, nPrimesEst;
    int bn, sn = (int)sqrt((double)n);
    n /= 2;
    bn = (n+7)/8;

    v = (unsigned char*)malloc(bn);
    memset( v, 0, bn );

    nPrimesEst = max(5000,(int)((n*2)*1.2/log((double)n*2)));
    primes.resize( nPrimesEst );

    nPrimes = 0;
    primes[nPrimes++] = 2;

    unsigned int word=0, bit=1;
    for( int i=0, p=3; i<n; i++, p+=2 ) {
      if( !(v[word]&bit) ) {
        if( p<=sn )
          for( int idx=i+p; idx<n; idx+=p )
            v[idx>>3] |= 1<<(idx&7);
        primes[nPrimes++] = p;
      }
      if( (bit <<= 1) >= 256 )
        word++, bit=1;
    }
    if( nPrimes > nPrimesEst )
      cout << "OOPS!" << endl;
    free( v );

    primes.resize( nPrimes );
  }
};
```

Listing 3.9: primes many fast.cpp — f9d2537e

77 lines
3,7c,79,6e,7e,26,48,3e,
b,16,1e,66,18,34,2f,57

```
#include <vector>
#include <memory>

struct primes_many_fast {
  vector< unsigned int > primes;

  primes_many_fast( int n ) {
    unsigned char *v;
    unsigned int nPrimes, nPrimesEst;
    int bn, sn = (int)sqrt((double)n);
    n /= 2;
    bn = (n+7)/8; // Round to nearest byte.

    v = (unsigned char*)malloc(bn);
    memset( v, 0, bn );

    nPrimesEst = max(5000,(int)((n*2)*1.2/log((double)n*2)));
    primes.resize( nPrimesEst );

    nPrimes = 0;
    primes[nPrimes++] = 2;

    unsigned int word=0, bit=1, f=8;
    for( int i=0, p=3; i<n; i++, p+=2 ) {
      if( !(v[word]&bit) ) {
        if( p<=sn ) {
          // p is a prime, remove all p-multiples
          if( f != 0 ) {
            f *= p;
            if( f+8 > (unsigned)n )
              f = 0;
          }
          if( f != 0 ) {    // Small prime optimization
            unsigned int idx; // (can be left out)

            // Generate small piece
```

```
            for( idx=i+p; idx<f+8; idx+=p )
              v[idx>>3] |= 1<<(idx&7);

            // Replicate
            int f2 = f/8;
            int w;
            for( w=f2+1; w<bn; ) {
              if( w+f2 <= bn ) {
                memcpy( &v[w], &v[1], f2 );
                w += f2;
              } else {
                memcpy( &v[w], &v[1], bn-w );
                w = bn;
              }
            }
          } else {
            int idx;
            int mult = (n-i-8*p-1)/(8*p)*8;
            int k=mult/8, mask = v[k];

            for( idx=i+(mult+8)*p; idx>i; mask=v[--k] ) {
              unsigned char t;
              for( t=128; t!=0; t>>=1, idx-=p ) {
                if( !(mask & t) )
                  v[idx>>3] |= 1<<(idx&7);
              }
            }

            for( idx=i+(mult+9)*p; idx<n; idx+=p )
              v[idx>>3] |= 1<<(idx&7);
          }
        }
        primes[nPrimes++] = p;
      }
      if( (bit <<= 1) >= 256 )
        word++, bit=1;
    }
    if( nPrimes > nPrimesEst )
      cout << "OOPS!" << endl;
    free( v );

    primes.resize( nPrimes );
  }
};
```

Listing 3.10: prime sieve.cpp — 54aa52e2

24 lines
10,13,10, 9,1c,17,1f, 2,
14,12, 5,11,19, 5,1f, 6

```
#include <vector>

struct prime_sieve {
  char *v;

  prime_sieve( int n ) {
    int sn = (int)sqrt((double)n);
    n /= 2;
    v = (char*)malloc(n);
    memset( v, 1, n );

    for( int i=0, p=3; i<sn; i++, p+=2 ) {
      if( v[i] ) {
        // p is a prime, remove all p-multiples
        for( int j=i+p; j<n; j+=p )
          v[j] = 0;
      }
    }
  }

  bool isprime( int m ) {
    if( !(m&1) )
      return (m==2);
    return v[(m-3)/2];
  }
};
```

Listing 3.11: miller-rabin.cpp — 91029f82

40 lines
11,2b,32,19, 1,19,27,37,
0,23,26,19,2a,1c, c,30

```
#include "mulmod.cpp"
#include "pseudo.cpp"

unsigned long long witness(unsigned long long a, unsigned long long n) {
  int i;
  unsigned long long t, d, pn, x, p;

  p = n-1;
  i=0; t=p; pn=1;
  while( t ) {
    t = t >> 1;
    pn = pn << 1;
    i++;
  }
  pn = pn >> 1;

  d = 1;
  while (i--) {
    x = d;

    d = n>0x80000000ULL ? mulmod(d,d,n) : (d*d%n);
    if ((d==1) && (x != 1) && (x != n-1)) return 1;
    if (p&pn)
      d = n>0x80000000ULL ? mulmod(d,a,n) : (d*a%n);
    p = p << 1;
  }
  if (d!=1) return 1;

  return 0;
}

int isprime_rabin_miller( unsigned long long n, int s ) {
  unsigned long long a;

  // Note: Make sure n > 1.
  for (int i=0;i<s;i++) {
    a = ulpseudo()%(n-1)+1; // ulpseudo may be changed to rand
    if (witness(a, n))
      return 0;
  }
  return 1;
```

```
}
```

Listing 3.12: miller-rabin-2.cpp — a57590b

20 lines
1a, d,11,1b,1d,15,14, 3,
8, 4, 5,14, e, 6, 6,15

```
#include "expmod.h"
#include "mulmod.h"

template <class T>
bool miller_rabin(T n, int tries = 15) {
  T n1 = n - 1, m = 1;
  int j, k = 0;
  if (n <= 3) return true;
  while (!(n1 & (m << k)))
    ++k;
  m = n1 >> k;
  for (int i = 0; i < tries; ++i) {
    T a = rand() % n1, b = expmod(++a, m, n);
    if (b == T(1))
      continue;
    for (j = 0; j < k && b != n1; ++j)
      b = mulmod(b, b, n);
    if (j == k)
      return false;
  }
  return true;
}
```

Listing 3.13: pollard-rho.cpp — fee67f25

37 lines
36,39,3e,11,19,1f,1a,2c,
c,24,3d,17,38, 2,38,22

```
#include "gcd.h"
#include "mulmod.h"

/* calculates x^2+1 (mod N) */
template <class T>
inline T pollard_step(T x, T N) {
  T r = mulmod(x, x, N);
  if (++r == N) r = 0;
  return r;
}

/*
 * Returns a non-trivial factor of N, if any. (Note that if N is
 * prime, pollard_rho never returns, so this should be checked first.)
 */
template <class T>
inline T pollard_rho(T N, int maxiter = 500) {
  /* replace rand by random number generator of choice. */
  T x = rand() % N, y = x;
  T d;
  int iter = 0;

  /* Check factor 2 */
  if (!(N & 1)) return 2;

  /* Check for a small factor. While this _shouldn't_ be necessary,
   * for some weird reason there is trouble factoring the number "25"
   * otherwise. Also, this gives a _considerable_ speed increase.
   */
  for (d = 3; d <= 9997; d += 2)
    if (!(N % d))
      return d;

  d = 1;
  while (d == 1) {
    /* Reseed if too many iterations passed. This shouldn't be
     * necessary either, but seemed to increase stability for
     * Valladolid 10290 ("sum{i++} = N")
     */
    if (iter++ == maxiter) {
      /* replace rand by random number generator of choice. */
      x = y = rand() % N;
      iter = 0;
    }
    x = pollard_step(x, N);
    y = pollard_step(pollard_step(y, N), N);
    d = gcd(y - x, N);
    if (d == N) d = 1;
  }
  return d;
}
```

Listing 3.14: ndivisors.cpp — 46a68d24

27 lines
19,1a, b,19, 0,12, b, 2,
1f, c,10, 6, 0,1d, c,18

```
#include <cmath>

template< class T >
unsigned long long ndivisors( unsigned long long x, T &primes ) {
  unsigned long long sn = (unsigned long long)sqrt((double)x);
  unsigned int nDivs = 1, nPrimes = primes.size();

  for( unsigned int i=0; i<nPrimes; i++ ) {
    unsigned int p = primes[i];

    if( p>sn )
      break;

    if( x % p == 0 ) {
      int nFactors = 0;

      while( x % p == 0 ) {
        x /= p;
        nFactors++;
      }
      sn = (unsigned long long)sqrt((double)x);
      nDivs *= (nFactors+1);
    }
  }

  if( x != 1 )
    nDivs *= 2;
  return nDivs;
}
```

Listing 3.15: ndivisors prob.cpp — 8ff4bd19

36 lines
13,14,2b,2e,19,2f,15,39,
2c,16,20,28,10,19,37,2d

```cpp
#include <cmath>

#include "rabin-miller.cpp"

template< class T >
unsigned long long ndivisors_prob( unsigned long long x, T &primes ) {
  unsigned long long sn = (unsigned long long)pow((double)x,1.0/3.0);
  unsigned int nDivs = 1, nPrimes = primes.size();

  for( unsigned int i=0; i<nPrimes; i++ ) {
    unsigned int p = primes[i];

    if( p>sn )
      break;

    if( x % p == 0 ) {
      int nFactors = 0;

      while( x % p == 0 ) {
        x /= p;
        nFactors++;
      }
      sn = (unsigned long long)pow((double)x,1.0/3.0);
      nDivs *= (nFactors+1);
    }
  }

  if( x != 1 ) {
    // x is either prime, a square or product of two primes.

    unsigned long long y = (unsigned long long)sqrt( (double)x );
    if( y*y == x )
      nDivs *= 3;
    else if( isprime_rabin_miller(x, 2) ) // Nr of iters, 10-50 may be proper.
      nDivs *= 2;
    else
      nDivs *= 4;
  }
  return nDivs;
}
```

Listing 3.16: factor.cpp — a2c940c

27 lines
14, 8, 6,10,16,1e, e,1a,
c, f, c, c, 5, a,1d, f

```cpp
#include <vector>

typedef pair<int, int> pii;
typedef vector<pii> vpii;

const vpii& Factor(int n) {
  static vector<vpii> factors;

  if (factors.size() <= (size_t)n)
    factors.resize(n+1);
  vpii& res = factors[n];
  if (res.empty() && n >= 2) {
    int fac = 2, count = 1;
    if (~n & 1) {
      while (~(n >>= 1) & 1) ++count;
    } else {
      fac = 3;
      while (n % fac) {
        fac += 2;
        if (fac*fac > n)
          fac = n;
      }
      while (!((n /= fac) % fac)) ++count;
    }
    res.push_back(pii(fac, count));
    res.insert(res.end(), Factor(n).begin(), Factor(n).end());
  }
  return res;
}
```

Listing 3.17: factor2.cpp — de807b85

21 lines
7,1b, b, d,14, 6, 3, b,
7,19,19,16,1f, 8, 7,17

```cpp
#include "miller-rabin.h"
#include "pollard-rho.h"


/* Factors N into prime factors. The factors are returned in the
 * "factors" vector.
 */
template <class T>
void factor(T N, vector<T>& factors) {
  vector<T> pending; // the pending vector is used as a stack
  if (N >= 2) {
    pending.push_back(N);
    while (!pending.empty()) {
      T x = pending.back();
      pending.pop_back();
      if (miller_rabin(x)) {
        factors.push_back(x);
      } else {
        T fac = pollard_rho(x);
        pending.push_back(fac);
        pending.push_back(x / fac);
      }
    }
  } else
    factors.push_back(N); // note that this adds 0 or 1 as single factor
  // if factors are wanted in order
  sort(factors.begin(), factors.end());
}
```

Linear Algebra

Listing 3.18: solve linear.cpp — 84b5aa44

47 lines
2a,2d,23,1c,36,34,10, a,
21,1f,3f,3b,21,3d,3a, 4

```cpp
template< class T, int N >
int solve_linear( T A[N][N], T x[N], T b[N], int nR, int nC ) {
  bool proc[N] = {false};

  for( int c=0; c<nC; c++ ) {
    for( int r=0; r<nR; r++ ) {
      if( proc[r] ) continue;

//    if( abs(A[r][c]) >= 1e-8 ) { // if T=double
      if( A[r][c] != 0 ) {
        // Eliminate column c using row r
        proc[r] = true;

        T f = A[r][c];
        for( int j=0; j<nC; j++ )
          A[r][j] /= f;
        b[r] /= f;

        for( int i=0; i<nR; i++ ) {
          if( i==r ) continue;
          f = A[i][c];
          for( int j=0; j<nC; j++ )
            A[i][j] -= A[r][j]*f;
          b[i] -= b[r]*f;
        }
        break;
      }
    }
  }

  int nFree = nC;

  for( int r=0; r<nR; r++ ) {
    if( !proc[r] ) {
      if( b[r] != 0 )
        return -1;
    } else {
      for( int c=0; c<nC; c++ ) {
//      if( abs(A[r][c]) >= 1e-8 ) { // if T=double
        if( A[r][c] != 0 ) {
          x[c] = b[r];
          break;
        }
      }
      nFree --;
    }
  }
  return nFree;
}
```

Listing 3.19: solve linear TO.cpp — bf7f688

47 lines
15, 2,36,31,2a,1d, 7, 8,
1b, f,1d,3b,23, 0,2e,20

```cpp
// solve(n,m) solves Ax = b, A nxm matrix
// Needs: number A[N][M], b[NN], x[MM], where NN,N>=n, MM,M>=m //
// I don't know your conventions for matrices:
//     number *, number ** or vector<number>
// A, b, x are destroyed on exit. solve() returns 0 if the system
// has a unique solution, -1 if no solution exists, otherwise
// the number of free variables (ideal for 10109)
// The comments prints the matrix after each elimination step.
template<class number> int solve(int n, int m) {
  int *r = new int[n], *c = new int[m];
  int t, ii, i, j, k, rankdef = 0;
  number pivot;
  /*
    for(i = 0; i<n; i++) {
      for(j = 0; j<m; j++) { A[i][j].print(); printf(" "); }
      b[i].print();
      printf("\n");
    }
    printf("\n");
  */
  for(i = 0; i<n; i++) r[i] = i;
  for(i = 0; i<m; i++) c[i] = i;
  for(ii = 0; ii<n; ii++) {
    i = ii - rankdef;
    j = i;
    while(j < m && A[r[i]][c[j]] == 0) j++;
    if(j < m) {
      t = c[i]; c[i] = c[j]; c[j] = t;
      for(j = i+1; j<n; j++) {
        pivot = A[r[j]][c[i]] / A[r[i]][c[i]];
        for(k = i; k<m; k++)
          A[r[j]][c[k]] -= pivot * A[r[i]][c[k]];
        b[r[j]] -= pivot * b[r[i]];
      }
    } else if(b[r[i]] != 0) {
      rankdef = -1;
      break;
    } else {
      rankdef++;
      t = r[i]; r[i] = r[n-rankdef]; r[n-rankdef] = t;
    }
    /*
    printf("pivot = "); pivot.print(); printf("\n");
    for(i = 0; i<n; i++) {
      for(j = 0; j<m; j++) { A[i][j].print(); printf(" "); }
      b[i].print();
      printf("\n");
    }
    printf("\n");
    */
  }
  /* printf("Rank deficiency %d (n=%d m=%d)\n", rankdef, n, m); */
  if(rankdef >=0 ) {
    if(m > n)
      rankdef += m-n;
    else if(rankdef == n-m) {
      rankdef = 0;
      for(i = m-1; i>=0; i--) {
        for(j = i+1; j<m; j++)
          b[r[i]] -= A[r[i]][c[j]] * b[r[j]];
        b[r[i]] /= A[r[i]][c[i]];
```

```
    }
    for(i = 0; i<m; i++)
      x[c[i]] = b[r[i]];
  }
 }
 delete[] c;
 delete[] r;
 return rankdef;
}
```

Listing 3.20: determinant.cpp — e6ec72d4

28 lines
17,17,1f,14, 4, 9, 7, d,
 e, 7, 4,13, d,1c,12, 5

```
template< int N >
double determinant( double m[N][N], int n ) {
  for( int c=0; c<n; c++ ) {
    for( int r=c; r<n; r++ ) {
      if( abs(m[r][c]) >= 1e-8 ) {
        // Eliminate column c with row r
        if( r!=c ) {
          for( int j=0; j<n; j++ ) {
            swap( m[c][j], m[r][j] );
            m[r][j] = -m[r][j];
          }
        }
        for( r++; r<n; r++ ) {
          double mul = m[r][c]/m[c][c];

          for( int j=0; j<n; j++ )
            m[r][j] -= m[c][j]*mul;
        }
      }
    }
  }
  // Matrix is now in upper-diagonal form
  double det = 1;

  for( int i=0; i<n; i++ )
    det *= m[i][i];
  return det;
}
```

Listing 3.21: int determinant.cpp — 14718b80

34 lines
3c,30, e,39,1f,27,2d,3f,
 c,37,32,10,15,1c, 9,28

```
#include "euclid.cpp"

template< class T, int N >
T int_determinant( T m[N][N], int n ) {
  for( int c=0; c<n; c++ ) {
    for( int r=c; r<n; r++ ) {
      if( m[r][c] !=0 ) {
        // Eliminate column c with row r
        if( r!=c ) {
          for( int j=0; j<n; j++ ) {
            swap( m[c][j], m[r][j] );
            m[r][j] = -m[r][j];
          }
        }
        for( r++; r<n; r++ ) {
          T x,y;
          T d = euclid( m[c][c], m[r][c], x,y );
          T x2 = -m[r][c]/d, y2 = m[c][c]/d;

          for( int j=c; j<n; j++ ) {
            T u = x*m[c][j]+y*m[r][j];
            T v = x2*m[c][j]+y2*m[r][j];
            m[c][j] = u;
            m[r][j] = v;
          }
        }
      }
    }
  }
  // Matrix is now in upper-diagonal form
  T det = 1;

  for( int i=0; i<n; i++ )
    det *= m[i][i];
  return det;
}
```

Numerical Misc

Listing 3.22: josephus.cpp — 6d071a21

6 lines
1, 4, 2, 2, 7, 0, 6, 6,
1, 1, 7, 4, 2, 2, 7, 5

```
int josephus(int n, int k) {
  int d = 1;
  while (d <= (k - 1) * n)
    d = (k * d + k - 2) / (k - 1);
  return k * n + 1 - d;
}
```

Listing 3.23: pseudo.cpp — 3ab67553

15 lines
4, 8, 7, c, 0, b, 0, 7,
a, f, f, 9, 0, 0, d, f

```
const int pseudo_mod = 1<<31;
const int pseudo_mul = 247590621; // Largest prime < psuedo_mod ( \
?)
int pseudo_seed = 0x12345678;

int pseudo() { // [0,1<<31)
  return pseudo_seed=(pseudo_seed*pseudo_mul+1)&(pseudo_mod-1);
}

double fpseudo() { // [0.0,1.0)
  return ((double)pseudo()/pseudo_mod+(double)pseudo())/pseudo_ \
mod;
}

unsigned long long ulpseudo() { // [0,1<<62)
  return ((unsigned long long)pseudo())*pseudo_mod+pseudo();
}
```

Listing 3.24: exp.cpp — 14e1f43

10 lines
8, 7, b, 9, 0, 4, b, 0,
f, 6, d, c, 6, 2, 7, d

```
template <class B, class E>
B exp(B b, E e) {
  B r = 1;
  if (e & 1) r = b;
  while (e > 1) {
    e >>= 1, b *= b;
    if (e & 1) r *= b;
  }
  return r;
}
```

Listing 3.25: mulmod.cpp — 332f2d0b

17 lines
1e,10, 9, 5, 6, 4,16, e,
14, 8, f, e,1e, d, 0, f

```
template< class T >
T mulmod( T a, T b, T mod ) {
  T c = 0;

  a %= mod;
  b %= mod;
  while( b > 0 ) {
    if( b & 1 ) {
      c += a;
      if( c>=mod ) c -= mod;
    }
    a *= 2;
    if( a>=mod ) a -= mod;
    b >>= 1;
  }
  return c;
}
```

Listing 3.26: expmod.cpp — 9b6527a0

10 lines
b, 5, 9, 9, 4, 2, b, 2,
8, 1, b, a, 5, 2, 6, 8

```
template <class B, class E>
B exp(B b, E e, B mod) {
  B r = 1; b %= mod;
  if (e & 1) r = b;
  while (e > 1) {
    e >>= 1, b *= b, b %= mod; // or mulmod!
    if (e & 1) r *= b, r %= mod; // or mulmod!
  }
  return r;
}
```

Listing 3.27: bitmanip.cpp — c9ef156a

47 lines
29,3c,31,3a,1d,2b,21,2b,
3c, 0,14, 1,3d, d,19, 2

```
// lowest bit
int lowest(int x) { return x & -x; }

// sign
int sign(int x) { return x >> 31; }

// is power of two
bool ispow2(int x) { return (x & x - 1) == 0; }

// power of two round up
int nlpow2(int x) {
  for (int i = 0; i < 5; ++i)
    x |= x >> (1 << i);
  return ++x;
}

// next higher number with the same number of bits set
unsigned nexthi_same_count_ones(unsigned a) { /* Gosper */
```

```
  /* works for any word length */
  unsigned c = (a & -a);
  unsigned r = a+c;
  return (((r ^ a) >> 2) / c) | r);
}


template <class T> // bit count, use with bitop
void bitcount(T &x, int s, T m) { x = (x >> s & m) + (x & m); }

template <class T> // bit reversal, use with bitop
void revbits(int &x, int s, int m) { x = x >> s & m | (x & m) << s; }

template <class F> int bitop(int x, F _fun) {
  _fun(x, 1, 0x55555555);
  _fun(x, 2, 0x33333333);
  _fun(x, 4, 0x0f0f0f0f);
  _fun(x, 8, 0x00ff00ff);
  _fun(x,16, 0x0000ffff);
  return x;
}

template <class F> long long bitop(long long x, F _fun) {
  _fun(x, 1, 0x5555555555555555ll);
  _fun(x, 2, 0x3333333333333333ll);
  _fun(x, 4, 0x0f0f0f0f0f0f0f0fll);
  _fun(x, 8, 0x00ff00ff00ff00ffll);
  _fun(x,16, 0x0000ffff0000ffffll);
  _fun(x,32, 0x00000000ffffffffll);
  return x;
}


// bit length, use with extended bitop
void bitlength(T &x, int s, T m, int c) { if (x & m << s) x >>= s, c |= s; }
```


Listing 3.28: coords.cpp — 3d6c6ff5

46 lines
30,35, b,11,36,22,33,13,
2,14,2a,10,22,3e,32,28

```
struct sqrX { // square [rows*]columns
  int c; sqrX(int cols) : c(cols) { }
  int operator()(int row, int col) { return row*c + col; }
  void operator()(int idx, int &row, int &col) { row = idx/c, col = idx%c; }
};

struct cubeX { // cube [levels*]square
  int r, c; cubeX(int rows, int cols) : r(rows), c(cols) { }
  int operator()(int lev, int row, int col) { return (lev*r + row)*c + col; }
  void operator()(int idx, int &lev, int &row, int &col) {
    col = idx%c, idx /= c, row = idx%r, lev = idx/r;
  }
};

struct quadX { // quad [hyper*]cube
  int l, r, c;
  quadX(int levs, int rows, int cols) : l(levs), r(rows), c(cols) { }
  int operator()(int hyp, int lev, int row, int col) {
    return ((hyp*l + lev)*r + row)*c + col;
  }
  void operator()(int idx, int &hyp, int &lev, int &row, int &col) {
    col = idx%c, idx /= c, row = idx%r, idx /= r, lev = idx%l, hyp = idx/l;
  }
};

struct triX { // triangle [row >= col]
  int operator()(int row, int col) { return row * (row + 1) / 2 + col; }
  void operator()(int idx, int &row, int &col) {
    for (row = 0, col = idx; col > row; col -= row) ++row;
  }
};

void dxy(int dir, int &dx, int &dy) { // direction is dir*90 degrees for x, y
  dx = dir & 1 ? 0 : 1 - (dir & 2);
  dy = dir & 1 ? 1 - (dir & 2) : 0;
}
void drc(int dir, int &dr, int &dc) { // direction is NESW for row, column
  dr = dir & 1 ? 0 : (dir & 2) - 1;
  dc = dir & 1 ? 1 - (dir & 2) : 0;
}

void dknight(int dir, int &dr, int &dc) { // chess knight jump
  int DR[] = { -2, -2, -1, 1, 2, 2, 1, -1 };
  int DC[] = { -1, 1, 2, 2, 1, -1, -2, -2 };
  dr = DR[dir], dc = DC[dir];
}
```

Optimization

Listing 3.29: simplex.cpp — c56b4925

55 lines
13,21,12,19,10,16, 2,16,
1,32, 6,29,29,1c,33,31

```
enum simplex_result { OK, UNBOUNDED, NO_SOLUTION };

template <class M, class I>
simplex_result simplex(M &a, I &var, int m, int n, int twophase = 0) {
  while (true) {
    // Choose a variable to enter the basis
    int idx = 0;
    for (int j = 1; j <= n; ++j)
      if (a[0][j] > 0 && (idx == 0 || a[0][j] > a[0][idx]))
        idx = j;
    // Done if all a[m][j]<=0
    if (idx == 0) return OK;
    // Find the variable to leave the basis
    int j = idx; idx = 0;
    for (int i = 1; i <= m; ++i)
      if (a[i][j] > 0 && (idx == 0 || a[i][0]/a[i][j] < a[idx][0]/a[idx][j]))
        idx = i;
    // Problem unbounded if all a[i][j]<=0
    if (idx == 0) return UNBOUNDED;
    // Pivot on a[i][j]
    int i = idx;
    for (int l = 0; l <= n; ++l)
      if (l != j) a[i][l] /= a[i][j];
    a[i][j] = 1;
    for (int k = 0; k <= m + twophase; ++k)
      if (k != i) {
        for (int l = 0; l <= n; ++l)
          if (l != j) a[k][l] -= a[k][j] * a[i][l];
        a[k][j] = 0;
      }
    // Keep track of the variable change
    var[i] = j;
  }
}

template <class M, class I>
simplex_result twophase_simplex(M &a, I &var, int m, int n, int artificial) {
  // Save primary objective, clear phase I objective
  for (int j = 0; j <= n + artificial; ++j)
    a[m + 1][j] = a[0][j], a[0][j] = 0;
  // Express phase I objective in terms of non-basic variables
  for (int i = 1; i <= m; ++i)
    for (int j = n + 1; j <= n + artificial; ++j)
      if (a[i][j] == 1)
        for (int l = 0; l <= n; ++l)
          if (l != j) a[0][l] += a[i][l];
  simplex(a, var, m, n + artificial, 1); // Simplex phase I
  // Check solution
  for (int j = n + 1; j <= n + artificial; ++j)
    if (a[0][j] >= 0) return NO_SOLUTION;
  // Restore primary objective
  for (int j = 0; j <= n; ++j)
    a[0][j] = a[m + 1][j];
  return simplex(a, var, m, n); // Simplex phase II
}
```

Polynomials

Listing 3.30: polynom.cpp — 4cfb80b5

42 lines
27,17,33, 4,30,2b,16,31,
2d,24,1b,10, c,34,1f,3b

```
#include <vector>

struct polynom {
  int n;
  vector<double> a;
  polynom( int _n ) : n(_n), a(n+1) {}

  // Calc value at x
  double operator()( double x ) const {
    double val = 0;

    for( int i=0; i<=n; i++ ) {
      val *= x;
      val += a[i];
    }
    return val;
  }

  // Calc derivative at x
  double deriv( double x ) const {
    double val = 0;

    for( int i=0; i<=n-1; i++ ) {
      val *= x;
      val += (n-i)*a[i];
    }
    return val;
  }

  // Divide this polynom with the factor (t-x) where x is a root...
  // The constant term is ignored (it should be zero but may be non-zero
  // due to rounding errors).
  void divroot( double x ) {
    double val = 0;

    for( int i=0; i<=n-1; i++ ) {
      val *= x;
      val += a[i];
      a[i] = val;
    }
    n--;
    a.resize(n+1);
  }
};
```

Listing 3.31: poly roots.cpp — f98c8f8a

31 lines
f,1a,1d,10,1b, a, 0, 6,
b, 8, 1, 3,17,1b,1f,14

```
#include <cmath>

#include "polynom.cpp"

template< class T >
double find_root_newton( double xmin, const T &calc, double eps=1e-5 )
{
  double x, newx;

  newx = xmin;
  do {
    x = newx;

    double xval = calc(x);
    newx = x - xval/calc.deriv(x);
  } while( fabs(x-newx) > eps );

  return newx;
}


void find_roots( const polynom &p, double xmin, vector<double> &roots )
{
  polynom p2 = p;
  double root;

  // Find roots repeatedly from the left.
  // No double-roots are allowed.
  while( p2.n > 0 ) {
    root = find_root_newton( xmin, p2 );
    roots.push_back( root );
    p2.divroot( root );
    xmin = root;
  }
}
```

Listing 3.32: poly roots bisect.cpp — 1bac8303

91 lines
12,13,49,49,36,2c,37, 6,
6b,32,5d,61,2b,50, 0,15

```
#include <cmath>

#include "polynom.cpp"

template< class T >
double find_root_newton_bisect( double x1, double x2, const T &calc,
                                double eps=1e-5 )
{
  bool p1 = (calc(x1)>0);
  bool p2 = (calc(x2)>0);
  double x, newx;

  // assertion: p1 != p2, i.e. sign-changing interval

  newx = (x1+x2)/2;
  do {
    x = newx;

    double xval = calc(x);
    bool pm = (xval>0);

    if( p1==pm )
      x1 = x;
    else if( p2==pm )
```

```
      x2 = x;

    newx = x - xval/calc.deriv(x);
    if( newx<x1 || newx>x2 )
      newx = (x1+x2)/2;
  } while( fabs(x-newx) > eps );

  return newx;
}

template< class T >
double find_root_bisect( double x1, double x2, const T &calc, double eps=1e-5 )
{
  bool p1 = (calc(x1)>0);
  bool p2 = (calc(x2)>0);

  // assertion: p1 != p2, i.e. sign-changing interval

  while( x2-x1 > eps ) {
    double xm = (x1+x2)/2;
    bool pm = (calc(xm)>0);

    if( p1==pm )
      x1 = xm;
    else if( p2==pm )
      x2 = xm;
    else
      return xm;
  }

  return (x1+x2)/2;
}

bool find_roots( const polynom &p, double xmin, double xmax,
                 vector<double> &roots )
{
  int nRoots;
  double step;

  roots.resize( p.n );

  // Find p.n sign-changing intervals
  for( int nInter = 8; ; nInter *= 10 ) {
    double lastVal = p(xmin);
    double lastX = xmin;

    step = (xmax-xmin)/nInter;
    nRoots = 0;

    for( int i=0; i<nInter; i++ ) {
      double x = lastX+step;
      double val = p(x);

      if( lastVal < 0 && val > 0 || lastVal > 0 && val < 0 )
        roots[nRoots++] = lastX;
      lastVal = val;
      lastX = x;
    }
    if( nRoots >= p.n )
      break;
  }
  if( nRoots != p.n )
    return false;

  for( int i=0; i<p.n; i++ )
  //    roots[i] = find_root_newton_bisect(roots[i], roots[i]+step, p);
    roots[i] = find_root_bisect(roots[i], roots[i]+step, p);

  return true;
}
```

Chapter 4

Combinatorial

4.1 Sorting

4.1.1 sort

Usage `sort( v.begin(), v.end(), less<int>() )`

4.1.2 isort

Usage `isort( array, n, idxarray, comp)`

Listing – isort.cpp, p. 56

4.1.3 indexed comparator

Listing – indexed comparator.cpp, p. 56

4.1.4 indexed less

Listing – indexed less.cpp, p. 56

4.2 Searching

4.2.1 median – nth element

Usage `nth_element( v.begin(), v.end(), 4 )`

Example

```
nth_element( v.begin(), v.end(), v.size() / 2 ) // mid value
```

4.2.2 binary search

Usage `bool binary_search( v.begin(), v.end(), 4 )`

Usage `iterator lower_bound( v.begin(), v.end(), 4 )`

Usage `iterator upper_bound( v.begin(), v.end(), 4 )`

4.2.3 binary search (numerical)

Listing – binary search num.cpp, p. 56

Usage `double binary_search_num( 0.0, 1.0, pred );`

4.2.4 golden search

Listing – golden search.cpp, p. 56

Usage `int golden_search( f, l, r, &min_value ) == min_index`

4.3 Permutations

4.3.1 next

Usage `next_permutation(begin, end [, comparator])`

4.3.2 previous

Usage `prev_permutation(begin, end [, comparator])`

4.3.3 random

Usage `random_shuffle(begin, end [, random_number_generator])`

4.3.4 permute

Listing – permute.cpp, p. 57

> `permute` recursively generates all permutations without comparisons. The permutations are given in lexiographically order using the original order (i.e. permute on "baa" gives baa, aba, aab). It can also be useful when the processing is done for each individual position (unnecessary duplicate work is avoided). Repetitions of non-adjacent elements will give strange results since there is no possible ordering (i.e "aba").

Usage `char a[n] = "caab"; permute(a+0,a+0,a+n);`

4.4 Counting

4.4.1 Binomial $\binom{n}{k}$

Listing – choose.cpp, p. 57

Complexity $\mathcal{O}\left(\min\{k, n-k\}\right)$

4.4.2 Multinomial $\binom{\Sigma k_i}{k_1\ k_2\ \ldots\ k_n}$

Listing – multinomial.cpp, p. 57

Complexity $\mathcal{O}\left((\Sigma k_i) - k_1\right)$

4.4.3 String permutations (multinomial)

Listing – nperms.cpp, p. 57

Usage `string s; int n = n_perms( s );`

> Algorithm for calculating the number of permutations of a string (multinomial numbers).

4.4.4 Stirling numbers of the first kind

Listing – stirling1.cpp, p. 58

Usage `s = stirling1(n,k);`

The Stirling numbers of the first kind s_{nk} count the number of ways to permute a list of n items into k cycles.

4.4.5 Stirling numbers of the second kind

Listing – stirling.cpp, p. 58

Usage `s = stirling(n,k);`

Calculates the stirling number s_{nk}, i.e. in how many ways can n different items be put in k boxes with at least one item in every box, or mathematically speaking – the number of partitions of n elements into k partitions.

4.4.6 Stirling numbers of the second kind modulo 2

Listing – stirling mod 2.cpp, p. 58

Usage `s = stirling_mod_2(n, k);`

Complexity $\mathcal{O}(\log k)$

4.4.7 Bell numbers

$B(n) = \sum_{k=1}^{n} \binom{n-1}{k-1} B(n-k) = \sum_{k=1}^{n} S(n,k)$, where S(n, k) is the Stirling numbers of the second kind.

The Bell numbers count the ways n elements can be partitioned.

4.4.8 Eulerian numbers

Listing – euler.cpp, p. 58

Usage `s = euler(n,k);`

The Eulerian number e_{nk} is the number of permutations $\pi_1\pi_2\cdots\pi_n$ of $\{1, 2, \cdots, n\}$ that have k places where $\pi_j < \pi_{j+1}$.

4.4.9 Second-order Eulerian numbers

Listing – euler2.cpp, p. 58

Usage `s = euler2(n,k);`

The second-order Eulerian number e_{nk} is the number of permutations $\pi_1\pi_2\cdots\pi_{2n}$ of the multiset $\{1, 1, 2, 2, \cdots, n, n\}$ with the property that all numbers between the two occurences of m are greater than m that have k places where $\pi_j < \pi_{j+1}$.

4.4.10 Catalan numbers

Among other things, the Catalan numbers describe the number of ways a polygon with n+2 sides can be cut into n triangles, the number of ways in which parentheses can be placed in a sequence of numbers to be multiplied, two at a time; the number of rooted, trivalent trees with n+1 nodes; and the number of paths of length 2n through an n-by-n grid that do not rise above the main diagonal.

$$C_n = \frac{\binom{2n}{n}}{n+1}$$

(from Math forum)

4.4.11 Derangements

A permutation that leaves no element in its original position.

$$D_{n+1} = n(D_n + D_{n-1}) \qquad D_n = n!\left(\frac{1}{2!} - \frac{1}{3!} + \ldots + (-1)^n \frac{1}{n!}\right), n \geq 2$$

Sorting & Searching

Listing 4.1: isort.cpp — 3dec7600

11 lines
7, 9, a, 1, 3, 5, f, 7,
3, 2, b, 8, c, 1, 0, 1

```
// V is a datastructure (vector) or RAI.
// I is RandomAccessIterator to objects/int.
template<class V, class I, class C>
void isort( const V &array, int n, I indexlist, C comp ) {
  for( int i=0; i<n; i++ )
    indexlist[i] = i;
  sort( indexlist+0, indexlist+n, indexed_comparator<V,C>(array, \
comp) );
}

template<class V, class I>
void isort( const V &array, int n, I indexlist ) {
  isort( array, n, indexlist, less<typename iterator_traits<V>:: \
value_type>());
}
```

Listing 4.2: indexed comparator.cpp — 6ad04101

6 lines
5, 6, 5, 5, 1, 7, 1, 6,
7, 3, 5, 6, 7, 5, 1, 2

```
template<class V, class C >
struct indexed_comparator {
  const V &v; const C &c; // array, comparator
  indexed_comparator( const V &_v, const C &_c ) : v(_v), c(_c) { \
 }
  bool operator()( int a, int b ) const { return c(v[a], v[b]); }
};
```

Listing 4.3: indexed less.cpp — 1a053088

6 lines
2, 7, 0, 2, 2, 6, 2, 7,
4, 0, 7, 1, 7, 1, 5, 7

```
template<class V >
struct indexed_less {
  const V &v; // array
  indexed_less( const V &v_ ) : v(_v) { }
  bool operator()( int a, int b ) const { return v[a] < v[b]; }
};
```

Listing 4.4: binary search num.cpp — bf213633

16 lines
1e, 1,11,1c,16,1b,1b, a,
e,12,15,1e,1f,1d,15, b

```
// This is only indented for double and may not work with ints.
// P is a predicate which should be equal to (a>m) for some m in \
[a,b],
// i.e. a step-function, and binary_search_num returns this m.
template <class T, class P>
T binary_search_num(T a, T b, P p, T eps = T(1e-13)) {
  T m;

  while (b-a > eps) {
    m = (a + b) / 2;
    if (p(m)) {
      if (b==m) break;
      b = m;
    } else {
      if (a==m) break;
      a = m;
    }
  }
  return m;
}
```

Listing 4.5: golden search.cpp — 7f1f9a80

27 lines
0,1e, 8, c, 8,1d, 9, 6,
a,17,10, 7, 2, b,17,17

```
int golden_ratio(int d) {
  const double golden_ratio = 1 - (sqrt(5) - 1) / 2;
  return (int) floor(golden_ratio * d);
}

template <class F, class T>
int golden_search(const F &f, int l, int r, T &v) {
  int d = golden_ratio(r-l);
  int m1 = l + d, m2 = r - d;
  T f1 = f(m1), f2 = f(m2);
  while (d > 0) {
    if (m1 == m2)
      if (m1 > l + 1)
        f1 = f(--m1);
      else
        f2 = f(++m2);
    if (f1 < f2)
      r = m2, m2 = m1, f2 = f1, d = golden_ratio(r-l), m1 = l + \
d, f1 = f(m1);
    else
      l = m1, m1 = m2, f1 = f2, d = golden_ratio(r-l), m2 = r - \
d, f2 = f(m2);
  }
  if (m2 - m1 > 1) {
    T f3 = f(m1 + 1);
    if (f3 < f1) f1 = f3, m1++;
  }
  return f1 < f2 ? (v = f1, m1) : (v = f2, m2);
}
```

Permutations & Counting

Listing 4.6: permute.cpp — ce084038

26 lines
1a, 0,17,17, 8, 8, 3,18,
15,14, 0, d, 4,18, 8,1c

```
template <class R>
void permute( R f, R c, R l ) {
  if( c==l ) {
    // Process whole perm
    while( f<l )
      cout << *f++;
    cout << endl;
    return;
  }

  for( R i=c; i!=l; ++i ) {
    if( i != c ) {
      if( *i == *c ) // for repetitions
        continue;
      swap( *i, *c );
    }
    // Process pos c
    // ...

    // Continue permutation seq
    permute( f, c+1, l );
  }
  while( l>c )
    swap( *--l, *c );
}
```

Listing 4.7: choose.cpp — 7e7320ee

11 lines
f, c, 4, 6, 4, 3, f, 6,
d, 5, 7, 3, a, 5, 3, 2

```
template <class T>
T choose(int n, int k) {
  k = max(k, n-k);

  T c = 1;
  for (int i = 1; i <= n-k; ++i)
    c *= k+i, c /= i;

  return c;
}
```

Listing 4.8: multinomial.cpp — da1a2a9d

10 lines
b, a, 6, 5, 0, 1, e, 7,
a, 0, 3, 2, c, 0, 0, 7

```
template <class T, class V>
T multinomial(int n, V &k) {
  T c = 1;
  int m = k[0];
  for (int i = 1; i < n; ++i)
    for (int j = 1; j <= k[i]; ++j)
      c *= ++m, c /= j;

  return c;
}
```

Listing 4.9: nperms.cpp — 541e9b85

38 lines
3e,34,2f, 8, 7,28,1b,2b,
27,3e,39,32,14,19,13,35

```
#define INT int
typedef unsigned int uint;

INT gcd(INT a, INT b) {
  INT r = b;
  do {
    b = r;
    r = a % b;
    a = b;
  } while (r != 0);
  return b;
}

/* Calculates the number of permutations of the string s.
 */
INT n_perms(string s) {
  int num[s.length()], den[s.length()];
  sort(s.begin(), s.end());
  for (uint i = 0; i < s.length(); ++i) {
    num[i] = i + 1;
    den[i] = 1;
    if (i > 0 && s[i] == s[i-1])
      den[i] = den[i-1]+1;
  }

  for (uint i = 0; i < s.length(); ++i) {
    for (uint j = 0; j < s.length(); ++j) {
      if (den[i] == 1)
        break;
      int x = gcd(num[j], den[i]);
      if (x > 1) {
        num[j] /= x;
        den[i] /= x;
      }
    }
  }
  INT res = 1;
  for (uint i = 0; i < s.length(); i++)
    res *= num[i];
  return res;
}
```

Listing 4.10: stirling1.cpp — 248828b6

7 lines
3, 1, 5, 1, 7, 7, 1, 2,
4, 3, 5, 3, 3, 4, 6, 0

```
template <class T>
T stirling1(T n, T k) {
  if (n < T(1) || n == k)
    return T(n==k ? 1:0);
  else
    return stirling1(n-1, k-1) + (n-1)*stirling1(n-1, k);
}
```

Listing 4.11: stirling.cpp — 27422771

7 lines
1, 3, 3, 1, 3, 1, 5, 2,
4, 7, 3, 1, 1, 0, 2, 6

```
template <class T>
T stirling(T n, T k) {
  if(n < T(1) || n == k)
    return T(n==k ? 1:0);
  else
    return stirling(n-1, k-1) + k*stirling(n-1, k);
}
```

Listing 4.12: stirling mod 2.cpp — 51df33f6

15 lines
a, c, 9, e, d, f, 4, e,
4, 0, 6, 0, 2, 2, e, c

```
/* Running time: O(log M) */

template <class T>
int stirling_mod_2(T n, T k) {
  T i = (k - 1) / 2;
  T p = 1;
  // let  p = 2^ceil(log2(i+1))
  while (p <= i) p <<= 1;
  T j = (n - k) % p;
  while (i | j) { // while (i != 0 || j != 0)
    if (i + j >= p) return 0;
    p >>= 1;
    if (i >= p) i -= p;
    if (j >= p) j -= p;
  }
  return 1;
}
```

Listing 4.13: euler.cpp — f55c2574

7 lines
3, 5, 3, 1, 7, 7, 5, 2,
6, 7, 3, 7, 3, 2, 6, 0

```
template <class T>
T euler(T n, T k) {
  if (n < T(1) || n == k)
    return T(k==0 ? 1:0);
  else
    return (n-k)*euler(n-1, k-1) + (k+1)*euler(n-1, k);
}
```

Listing 4.14: euler2.cpp — c13c269a

7 lines
3, 7, 7, 1, 3, 3, 1, 0,
6, 7, 7, 3, 1, 2, 4, 0

```
template <class T>
T euler2(T n, T k) {
  if (n < T(1) || n == k)
    return T(k==0 ? 1:0);
  else
    return (2n-1-k)*euler2(n-1, k-1) + (k+1)*euler2(n-1, k);
}
```

Chapter 5

Graph

5.1 Shortest Path and Connectivity

Parameter descriptions:

`f` is a function object that takes a node index and a funcion object, and that for each edge from that node calls the function object with the destination node index and distance of the edge. (`f` is most conveniently templatised on the type of the recieved function object.)

`adj` is an adjecency matrix. `adj[i][j]` is the distance from `i` to `j`.

`edges` is the graph given as an STL-container of `vectors`, `maps` or `multimaps` (or `set` in `dijkstra_1`) of the edges from each node. Edges can have negative distances, provided there is no negative cycle in the graph. Self and multiple edges are allowed.

`n` is the number of nodes is the graph.

`min` is filled in with the shortest distance to each node.

`path` **or** `from` is filled in with the node number from which each node was entered. (there is inconsistency in the name choice!)

5.1.1 Flood fill

Listing – flood fill.cpp, p. 68

Usage `flood_fill(edges, m, start, from, to);`

Complexity $\mathcal{O}(E)$

Flood fills around `start` changing the value `from` to `to` in `m`.

5.1.2 Connected components

Listing – connected components.cpp, p. 68

Usage `connected_components(edges, m, n);`

Complexity $\mathcal{O}(V+E)$

Fills in `m`, identifying each node with a connected component number, returning the number of connected components.

5.1.3 Transitive Closure

Listing – transitive closure.cpp, p. 68

Complexity $\mathcal{O}(V^3)$

Usage `transitive_closure(adj, path, n);`

The transitive closure, i.e. which nodes are connected, is found using a sligthly modified Floyd-Warshall. The algorithm also gives a path between two nodes, if they are connected.

The adjacency matrix is updated to contain whether nodes are connected.

5.1.4 Floyd Warshall

Listing – floyd warshall.cpp, p. 69

Complexity $\mathcal{O}\left(V^3\right)$

Usage `floyd_warshall(adj, path,n);`

> Calculates the distances between all pairs of nodes. The adjacency matrix is modified to contain the shortest distances.

5.1.5 Prijm

Listing – prijm.cpp, p. 69

Listing – prijm1.cpp, p. 69

Listing – for edge.cpp, p. 70

Complexity $\mathcal{O}\left((V+E)\log V\right)$

Usage `dijkstra(f, min, path, start);`

Usage `prim(f, min, path, start);`

5.1.6 Dijkstra 1

Listing – dijkstra 1.cpp, p. 70

Complexity $\mathcal{O}\left(V+E\right)$

Usage `dijkstra_1(edges, min, path, start, n);`

> Calculates the distance from a source to all other nodes, when all edges have weight 1.

5.1.7 Dijkstra Prim

Listing – dijkstra prim.cpp, p. 70

Listing – distfun.cpp, p. 72

Complexity $\mathcal{O}\left((V+E)\log V\right)$

Usage `dijkstra_prim(edges, min, path, start, n, distfun, mst);`

> Calculates the distance from a source to all other nodes. Takes two extra arguments: `distfun` is the edge distance function (see exaples). `mst = false` is used in `prim` to get a minimum spanning tree instead.

Note! Dijkstra Prim can handle negative edge-weights but the complexity is then worse, $\mathcal{O}\left(V(V+E)\log V\right)$ (?). Bellman-Ford is better suited for this case.

5.1.8 Dijkstra Prim Simple

Listing – dijkstra prim simple.cpp, p. 71

Complexity $\mathcal{O}((V+E)V)$

Usage `dijkstra_prim_simple(edges, min, path, start, n, mst);`

Calculates the distance from a source to all other nodes. Takes one extra arguments: `mst = false` is used in `prim` to get a minimum spanning tree instead.

Note! Dijkstra Prim Simple cannot handle negative edge-weights. It can be modified to work with negative edge-weights by inserting `proc[dest]=false;` when a shorter route is found, but with increased complexity.

5.1.9 Bellman-Ford

Listing – bellman ford.cpp, p. 71

Listing – distfun.cpp, p. 72

Complexity $\mathcal{O}(VE)$

Calculates the distance from a source to all other nodes. Returns whether there is a negative distance cycle in the graph.

5.1.10 Bellman-Ford-2

Listing – bellman ford 2.cpp, p. 72

Listing – distfun.cpp, p. 72

Complexity $\mathcal{O}(VE)$

Calculates the distance from a source to all other nodes. Returns whether there is a negative distance cycle in the graph. Instead of working with edge-lists as Bellman-Ford, the algorithm BF-2 uses a single edge-list with pairs ((from,to), dist).

5.1.11 Get shortest path

Listing – get shortest path.cpp, p. 72

Usage `get_shortest_path( from, path, start, end )`

Fills in `path` with the nodes from `start` to `end`, using `from` to see where the path got from to the end.

5.1.12 Shortest Tour

Shortest tour from A to B to A again not using any edge twice, in an undirected graph:

Convert the graph to a directed graph.

Take the shortest path from A to B.

Remove the paths used from A to B, but also negate the lengths of the reverse edges.

Take the shortest path again from A to B, using an algorithm which can handle negative-weight edges, such as Bellman-Ford. Note that there is no negative-weight *cycles*.

The shortest tour has the length of the two shortest paths combined.

5.2 Minimum Spanning Tree

5.2.1 Prim

Listing – prim.cpp, p. 74

Complexity $\mathcal{O}(V \log V + E)$

Usage `prim( graph, path, 0 )`

5.2.2 Kruskal

Usage `kruskal( graph, tree, n );`

Complexity $\mathcal{O}(E \log E)$

The resulting tree which is returned in `tree` may be the same variable as the graph.

Example

```
vector< vector<pair<int,double> > > edges;

edges.resize( 100 );
kruskal( edges, edges, 100 );
```

Listing – sets.cpp, p. 18

Listing – kruskal.cpp, p. 72

Valladolid 10147

5.3 Topological sorting

A topological sort of a dag (directed acyclic graph) is an ordering of its vertices such that for every edge (u, v), u appears before v in the ordering.

5.3.1 topo sort

Listing – topo sort.cpp, p. 74

Usage `topo_sort(edges, idxarray, n)`

5.4 Euler walk

5.4.1 Euler walk

Listing – euler walk.cpp, p. 74

Complexity $\mathcal{O}(E)$

Usage `euler_walk( V &edges, int start, list<int> &path, bool cyclic )`

Find an eulerian walk in a directed graph, i.e. a walk traversing all edges exactly once.

The algorithm *assumes* that there exists an eulerian walk. If it does not exists, it will return any maximal path, not neccessarily the longest.

If the graph is not cyclic, the start node must be a node with $\deg_{out} - \deg_{in} = 1$.

`euler_walk` can be used to test if a graph has an eulerian walk by first finding a start-node (or any node if it is cyclic) and then checking if `path.size() == nrOfEdges+1`. But obviously this is slower than checking that all out degrees are equal to the in degrees (or exactly one vertex has an extraneous entering edge and another vertex an extraneous leaving edge) and that the graph is connected.

Set `cyclic=true` if the path found must be cyclic, this is mostly of internal use.

`edges` is a vector/array with V edge-containers. The edge-containers should contain vertex-indices, and may contain repeated indices (i.e. multiple edges). **WARNING!** `edges` is modified and emptied by the algorithm.

`path` should be empty prior to the call and contains the euler-path given as *vertex numbers*. The first vertex is `start` which also is the last vertex if the path is cyclic.

Lexicographic Path If the edges are sorted in lexicographic order for each vertex, the resulting path will be lexicographically ordered. This is accomplished by the algorithm, adding extra loops from the end first.

5.4.2 Chinese postman

A generalised euler path/cycle problem, finding the shortest path/cycle that visits all edges even if some edges have to be traversed several times.

5.5 De Bruijn Sequences

Let Ω be an alphabet of size σ. A de Bruijn sequence is a sequence such that all words on L letters appear as a contiguous subrange of it. In a cyclic de Bruijn sequences a word may also wrap around the string. The shortest cyclic de Bruijn sequence is of length σ^L and the shortest non-cyclic de Bruijn sequence is of length $\sigma^L + L - 1$.

The shortest de Bruijn sequence of all words on 3 letters in the alphabet $\{0, 1\}$ which is lexicographically smallest is
00011101 (cyclic)
0001110100 (non-cyclic)

5.5.1 de Bruijn

Listing – deBruijn.cpp, p. 74

Listing – deBruijn fast.cpp, p. 75

Complexity $\mathcal{O}\left(N^L\right)$

Usage `deBruijn( int N, int L, char symbols[N] )`

`N` is the size of the alphabet and `symbols` the corresponding letters. `L` is the length of the words that should appear in the de Bruijn sequence.

The output is given as `cout`-statements.

5.6 Network Flow

Flow graphs are directed graphs with flow capacities on their edges.

To get quick access to the "back edge" of all egdes, a special flow edge struct is used in the network flow algorithms.

5.6.1 flow graph

Listing – flow graph.cpp, p. 76

Usage `flow_add_edge( edges, source, dest, cap [, back_cap] );`

> Flow graphs are constructed and updated by a couple of utility functions.
>
> A flow graph should be an STL-container of `vectors` with `flow_edges` (`maps` are not allowed).
>
> Edges should be added using `flow_add_edge`.
>
> Note that an edge *m*ust be added only once for each pair, simultaneously giving both forward and back capacity.

5.6.2 lift to front

Listing – lift to front.cpp, p. 76

Note! This is a much more effective algorithm than Ford Fulkerson, even on bi-partite graphs, and suitable for any flow graph.

Note! Ford Fulkerson *is* faster if $En_{aug\ paths} < V^3$.

Usage `flow = lift_to_front(edges, source, sink);`

Complexity $\mathcal{O}\left(V^3\right)$

5.6.3 ford fulkerson

Listing – ford fulkerson.cpp, p. 76

> This is a DFS or BFS Ford Fulkerson which maximize the flow in the augmenting paths. The BFS is more robust but may be slower.

Usage The maximum flow is calculated by repetitive calls to `flow_increase1`: `while( ap = flow_increase1(edges, source, sink) ) flow+=ap;`

Complexity $\mathcal{O}\left(E \cdot n_{aug\ paths}\right)$

5.6.4 ford fulkerson 1

Listing – ford fulkerson 1.cpp, p. 78

> This is a DFS Ford Fulkerson where all augmenting paths are 1 and thus specially suited for bipartite graphs.

Usage The maximum flow is calculated by repetitive calls to `flow_increase1`: `while( flow_increase1(edges, source, sink) ) flow++;`

Complexity $\mathcal{O}\left(E \cdot n_{aug\ paths}\right)$

5.6.5 Min cut

The minimum cut has the same capacity as the maximum flow, but perhaps we want an algorithm for finding it?

5.6.6 Flow constructions

Matching in a bipartite graph. A multisource, multisink flow with only one-capacities determines an optimal matching.

Edge and Vertex Connectivity of a graph, determines how connected it is. For vertex connectivity, each node is split in two.

Minimal cut of a graph, generalization of edge connectivity. A minimal cut is found by first finding a maximal flow. Then we consider the set A of all nodes that can be reached from the source using edges which has capacity left (i.e. edges in the residue network). The edges between A and the complement of A is a minimal cut.

Escaping problem on a grid, determines whether and how a set of points may be connected by grid-lines to the edge.

Minimal path cover of a graph, determines a minimum set of paths to cover it.

5.7 Matching

5.7.1 hopcroft karp

Listing – hopcroft karp.cpp, p. 79

Complexity $\mathcal{O}\left(\sqrt{V}E\right)$

5.7.2 max weight bipartite matching

Listing – mwbm.cpp, p. 80

Complexity $\mathcal{O}\left(V(E+V^2)\right)$

5.7.3 max weight bipartite matching of maximum cardinality

Listing – mwbm of max card.cpp, p. 81

Complexity $\mathcal{O}\left(V(E+V^2)\right)$

Connectivity

Listing 5.1: flood fill.cpp — 37977d03

18 lines
13, 1,1e,12,14,1e,1b, 1,
5,18,19,10,1a,1a,18, 3

```
#include <queue>

template <class E, class M, class T>
void flood_fill(E &edges, M &m, int start, T from, T to) {
  typedef typename E::value_type L;
  typedef typename L::const_iterator L_iter;
  queue<int> q;

  if (from == to) return;
  q.push(start);
  while (!q.empty()) {
    int node = q.front(); q.pop();
    if (m[node] == from) {
      m[node] = to;
      const L &l = edges[node];
      for (L_iter it = l.begin(); it != l.end(); it++)
        q.push(*it);
    }
  }
}
```

Listing 5.2: connected components.cpp — 56c44200

12 lines
8, 8, 2, 4, d, b, 2, 5,
6, e, 6, 6, 9, d, 2, 5

```
#include "flood_fill.cpp"

template <class E, class M>
int connected_components(E &edges, M &m, int n) {
  int count = 0;
  for (int i = 0; i < n; ++i)
    m[i] = 0;
  for (int i = 0; i < n; ++i)
    if (m[i] == 0) {
      ++count;
      flood_fill(edges, m, i, 0, count);
    }
  return count;
}
```

Listing 5.3: transitive closure.cpp — b0b75360

16 lines
f, 3, c, 2, 9, 9, 6, 4,
b, 9, 8, c, 4, a, 6, 9

```
template<class V, class T> // V is a bool n*n matrix
void transitive_closure( V &adj, T &paths, int n ) {
  for( int x=0; x<n; x++ )
    for( int y=0; y<n; y++ )
      paths[x][y] = -1;
  for( int m=0; m<n; m++ )
    for( int x=0; x<n; x++ )
      if( adj[x][m] )
        for( int y=0; y<n; y++ )
          if( adj[m][y] ) {
            adj[x][y] = true;
            paths[x][y] = m;
          }
}
```

Shortest Path / Minimum Spanning Tree

Listing 5.4: floyd warshall.cpp — cc5268e5

16 lines
1e, 0,1f,1a,18,1f,16, 5,
2,18,15,1a,17,17,11, 8

```
template<class V1, class V2>
void floyd_warshall( V1 &adj, V2 &path, int n ) {
  for( int x=0; x<n; x++ )
    for( int y=0; y<n; y++ )
      path[x][y] = -1;

  for( int m=0; m<n; m++ )
    for( int x=0; x<n; x++ )
      if( adj[x][m] >= 0 )
        for( int y=0; y<n; y++ )
          if( adj[m][y] >= 0 )
            if( adj[x][y] < 0 || adj[x][m] + adj[m][y] < adj[x][y] ) {
              adj[x][y] = adj[x][m] + adj[m][y];
              path[x][y] = m;
            }
}
```

Listing 5.5: prijm.cpp — 155ea7d

37 lines
2b,2b,1c,19,2e,3c,28,27,
28,3f, 0, 5,2f,23,17, 9

```
#include <set>

// min should be initialised before-hand to inf values [path to -1 values]
template <class M, class P, bool MST>
struct prijm {
  typedef typename M::value_type T;

  M &min; P &path; int node;

  set< pair<T, X> > q; // use as an mpq

  prijm(M &m, P &p, int start) : min(m), path(p) {
    min[start] = T();
    q.insert(make_pair(min[start], start));
    while (!q.empty()) {
      node = q.begin()->second;
      q.erase(q.begin());
      if (MST) min[node] = T(); // only difference between dijkstra and prim
      f(node);
    }
  }

  void relax(int dest, T dist) {
    if (min[node] + dist < min[dest]) {
      q.erase(make_pair(min[dest], dest)); //
      min[dest] = min[node] + dist; // update dest in the queue
      q.insert(make_pair(min[dest], dest)); //
      path[dest] = node;
    }
  }

  void f(int node) { // call relax on every edge that leaves node
  }
};

template <class M, class P>
void dijkstra(M &m, P &p, int start) { prijm<M, P, false>(m, p, start); }

template <class M, class P>
void prim(M &m, P &p, int start) { prijm<M, P, true>(m, p, start); }
```

Listing 5.6: prijm1.cpp — 5152c210

35 lines
4,1f,2e,21,2e,25,39,3c,
1c,2d, 3,15,13,26,1a, 1

```
#include <queue>

// min should be initialised before-hand to inf values [path to -1 values]
template <class M, class P, bool MST>
struct prijm1 {
  typedef typename M::value_type T;

  M &min; P &path; int node;

  queue< int > q;

  prijm1(M &m, P &p, int start) : min(m), path(p) {
    min[start] = T();
    q.push(start);
    while (!q.empty()) {
      node = q.front(); q.pop();
      if (MST) min[node] = T(); // only difference between dijkstra and prim
      f(node);
    }
  }

  void relax(int dest) {
    if (min[node] + 1 < min[dest]) {
      min[dest] = min[node] + dist;
      q.push(dest);
      path[dest] = node;
    }
  }

  void f(int node) { // call relax on every edge that leaves node
  }
};

template <class M, class P>
void dijkstra1(M &m, P &p, int start) { prijm1<M, P, false>(m, p, start); }

template <class M, class P>
void prim1(M &m, P &p, int start) { prijm1<M, P, true>(m, p, start); }
```

Listing 5.7: for edge.cpp — 6bc8854e

44 lines
d,10,31,13,2f,23, 3, 2,
f,20,32,2d,14, d,1e,3c

```
// example for_edge function objects:

// one fun
typedef vector<int> Vi; vector<Vi> g;
struct onefun {
  template <class F> void operator()(int node, F &f) {
    const Vi &l = g[node];
    for (Vi::const_iterator i = l.begin(); i != l.end(); ++i)
      f(*i, 1);
  }
};

// weighted edge fun
typedef vector<pair<int, int> > Vp; vector<Vp> h;
struct wefun {
  template <class F> void operator()(int node, F &f) {
    const Vp &l = h[node];
    for (Vp::const_iterator i = l.begin(); i != l.end(); ++i)
      f(i->first, i->second);
  }
};

// geometrical dist fun
typedef vector<point<double> > VP; int n; VP pts(n);
struct distfun {
  template <class F> void operator()(int node, F &f) {
    for (int i = 0; i < pts.size(); ++i)
      f(i, dist(pts[node] - pts[i]));
  }
};

// the following funs fetches min[node] from inside f, which is \
a bit ugly:

// for time-table searches without mod
typedef vector<pair<int, pair<int, int> > > Vpp; vector<Vpp> g;
struct stepfun {
  template <class F> void operator()(int node, F &f) {
    const Vi &l = g[node]; int t = f.min[node];
    for (Vpp::const_iterator i = l.begin(); i != l.end(); ++i)
      if (i->second.first >= t)
        f(i->first, i->second.first - t + i->second.second);
  }
};

// for time-table searches
typedef vector<pair<int, pair<int, int> > > Vpp; vector<Vpp> g;
template <int MOD> struct modfun {
  template <class F> void operator()(int node, F &f) {
    const Vi &l = g[node]; int t = f.min[node];
    for (Vpp::const_iterator i = l.begin(); i != l.end(); ++i)
      f(i->first, (i->second.first - t % MOD + MOD) % MOD + i-> \
second.second);
  }
};
```

Listing 5.8: dijkstra 1.cpp — 7cbeba0c

40 lines
11,33,31, a,15, 0,1e, d,
8, 5, 5,38, 0,2b,11,20

```
#include <queue>


template<class V, class S, class T>
void dijkstra_1( const V &edges, S &min, T &from, int start, int n ) {
  typedef typename V::value_type::const_iterator E_iter;
  const int inf = 1 << 29;
  queue<int> q;

  // Initialize min
  for( int i=0; i<n; i++ ) {
    min[i] = inf;
    from[i] = -1; // ****
  }
  min[start] = 0;

  q.push( start );

  while( !q.empty() ) {
    int node = q.front();
    int length = min[node]+1;

    q.pop();

    // Process node
    for( E_iter e=edges[node].begin(); e!=edges[node].end(); e++ ) {
      int destNode = *e;

      if( length < min[destNode] ) {
        // Process this node the next run
        min[destNode] = length;
        q.push( destNode );

        from[destNode] = node; // ****
      }
    }
  }

  for( int i=0; i<n; i++ ) {
    if( min[i] == inf )
      min[i] = -1;
  }
}
```

Listing 5.9: dijkstra prim.cpp — 6bdefbff

35 lines
1c, c, 7, f,1a,19, 1, 1,
d,1c, d,1e, 8, f, 5, 8

```
#include <set>

template <class E, class M, class P, class D>
void dijkstra_prim(E &edges, M &min, P &path, int start, int n, D distfun,
                   bool mst = false) {
  typedef typename M::value_type T;
  T inf(1<<29);

  set< pair<T, int> > q; // use a set as a modifiable priority queue
```

```cpp
  for (int i = 0; i < n; i++) {
    min[i] = inf;
    path[i] = -1;
  }
  min[start] = T();

  q.insert(make_pair(min[start], start));
  while (!q.empty()) {
    int node = q.begin()->second;
    q.erase(q.begin());
    if (mst) min[node] = T(); // only difference between dijkstra and prim
    typedef typename E::value_type L;
    typedef typename L::const_iterator L_iter;

    const L &l = edges[node];
    for (L_iter it = l.begin(); it != l.end(); ++it) {
      pair<int, T> p = distfun(*it, min[node]);
      int dest = p.first; T dist = min[node] + p.second;
      if (dist < min[dest]) {
        q.erase(make_pair(min[dest], dest)); //
        min[dest] = dist; // update dest in the queue
        q.insert(make_pair(min[dest], dest)); //
        path[dest] = node;
      }
    }
  }
}
```

Listing 5.10: dijkstra prim simple.cpp — 7cce8b2

44 lines
3c, 2,30,33,1c, 5,19,3e,
10, 9,18,1e,28,28,33,17

```cpp
template<class E, class M, class P>
void dijkstra_prim_simple( const E &edges, M &min, P &path, int start, int n,
                bool mst = false ) {
  typedef typename M::value_type T;
  T inf(1<<29);

  // Initialize min & path
  for( int i=0; i<n; i++ ) {
    min[i] = inf;
    path[i] = -1;
  }
  min[start] = T();

  // Initalize proc
  vector<bool> proc( n, false );

  // Find shortest path
  while( true ) {
    int node = -1;
    T   least = inf;

    for( int i=0; i<n; i++ )
      if( !proc[i] && min[i] < least )
        node = i, least = min[i];
    if( node < 0 ) break;

    if( mst ) min[node] = T();
    typedef typename E::value_type L;
    typedef typename L::const_iterator L_iter;

    const L &l = edges[node];
    for( L_iter it=l.begin(); it!=l.end(); ++it ) {
      int dest = (*it).first;
      T dist = min[node]+(*it).second;

      if( !proc[dest] && dist < min[dest] ) {
        min[dest] = dist;
        path[dest] = node;
      }
    }

    proc[node] = true;
  }
}
```

Listing 5.11: bellman ford.cpp — bac8db49

33 lines
e,32,39, d, 7,38,2a,24,
2e,23,2c,3a,23,27,1d,2f

```cpp
template <class E, class M, class P, class D>
bool bellman_ford(E &edges, M &min, P &path, int start, int n, D distfun) {
  typedef typename M::value_type T;
  typedef typename E::value_type L;
  typedef typename L::const_iterator L_iter;
  T inf(1<<29);

  for (int i = 0; i < n; i++) {
    min[i] = inf;
    path[i] = -1;
  }
  min[start] = T();

  bool changed = true;
  for (int i = 1; changed; ++i) { // max V-1 times
    changed = false;
    for (int node = 0; node < n; ++node) {
      const L &l = edges[node];
      for (iterator it = l.begin(); it != l.end(); ++it) {
        pair<int, T> p = distfun(*it, min[node]);
        int dest = p.first; T dist = min[node] + p.second;
        if (dist < min[dest]) {
          if( i>=N )
            return false; // negative cycle!
          min[dest] = dist;
          path[dest] = node;
          changed = true;
        }
      }
    }
  }
  return true; // graph is negative-cycle-free
}
```

Listing 5.12: bellman ford 2.cpp — f664aaa0

30 lines
a,16, d, c,1e,15,11, b,
1b,12,13, a,12, b, e,1e

```
template <class E, class M, class P, class D>
bool bellman_ford_2(E &edges, M &min, P &path, int start, int n, int m) {
  typedef typename M::value_type T;
  T inf(1<<29);

  for (int i = 0; i < n; i++) {
    min[i] = inf;
    path[i] = -1;
  }
  min[start] = T();

  bool changed = true;
  for (int i = 1; changed; ++i) { // V-1 times
    changed = false;
    for (int j = 0; j < m; ++j) {
      int node = edges[j].first.first;
      int dest = edges[j].first.second;
      T dist = min[node] + edges[j].second;

      if (dist < min[dest]) {
        if( i>=N )
          return false; // negative cycle!
        min[dest] = dist;
        path[dest] = node;
        changed = true;
      }
    }
  }
  return true; // graph is negative-cycle-free
}
```

Listing 5.13: distfun.cpp — e98ba1d7

15 lines
0, 5, 9, 4, 9, 8, b, d,
b, d, b, f, 4, 8, b, 0

```
// one distfun
pair<int, int> one_dist(int node, int) { return make_pair(node, \
1); }

// weighted edge distfun
template <class P, class T> P id_dist(P edge, T) { return edge; }

// for time-table searches without mod:
template <class T, T inf>
pair<int, T> step_dist(pair<int, pair<T, T> > &edge, T t) {
  return make_pair(edge.first, edge.second.first < t ? inf :
                edge.second.first - t + edge.second.second);
}

// for time-table searches:
template <int MOD>
pair<int, int > mod_dist(const pair<int, pair<int, int> > &edge, \
int t) {
  return make_pair(edge.first, (edge.second.first - t % MOD + \
MOD) % MOD +
    edge.second.second);
}
```


Listing 5.14: get shortest path.cpp — fc3151c0

15 lines
8, 8, 3, 4, b, 3, 8, 4,
e, 5, 0, 9, 1, 1, e, 6

```
// T is a random access iterator into a container containing
// node-indices of size V.

template<class T>
bool get_shortest_path( const T &from, T &path, int start, int \
end ) {
  int   n=0;

  if( from[end]<0 )
    return false;

  for( int node=end; node!=start; node = from[node] )
    path[n++] = node;
  path[n++] = start;

  reverse( path, path+n );

  return true;
}
```

Listing 5.15: kruskal.cpp — 826d118b

37 lines
6, 8,1e, c, 2,16, e,1a,
3b,3e,28,37,2d,1c,30, 7

```
#include <algorithm>
#include <vector>

#include "../../datastructures/sets.cpp"

template<class V>
void kruskal( const V &graph, V &tree, int n ) {
  typedef typename V::value_type      E;
  typedef typename E::const_iterator  E_iter;
  typedef typename E::value_type::second_type D;

  sets sets(n);
  vector< pair< D,pair<int,int> > > edges;

  // Convert all edges into a single edge-list
  for( int i=0; i<n; i++ ) {
    for( E_iter iter=graph[i].begin(); iter!=graph[i].end(); iter++ ) {
      if( i < (*iter).first ) // Undirected: only use half of the edges
        edges.push_back( make_pair((*iter).second,
                             make_pair(i,(*iter).first)) );
    }
  }

  // Clear tree
  for( int i=0; i<n; i++ )
    tree[i].clear();

  sort( edges.begin(), edges.end() );

  // Add edges in order of non-decreasing weight
  int numEdges = edges.size();
  for( int i=0; i<numEdges; i++ ) {
    pair<int,int> &edge = edges[i].second;
```

```
    // Add edge if the edge-endpoints aren't in the same set
    if( !sets.equal(edge.first, edge.second) ) {
      sets.link( edge.first, edge.second );
      tree[edge.first].push_back( make_pair(edge.second, edges[i].first) );
      tree[edge.second].push_back( make_pair(edge.first, edges[i].first) );
    }
  }
}
```

Graph Misc

Listing 5.16: prim.cpp — 55504a52

6 lines
5, 5, 4, 6, 6, 3, 7, 3,
6, 5, 1, 0, 7, 4, 1, 2

```
template <class E, class P>
void prim(int root, E &edges, P &path, int n) {
  typedef typename E::value_type::value_type::second_type T;
  vector<T> min(n);
  dijkstra_prim(edges, min, path, root, n, id_dist<pair<int,T>,T> \
, true);
}
```

Listing 5.17: topo sort.cpp — 8ee19e49

25 lines
9,13,12, b,17, f, 7,10,
15,12, 3, 6,18, d, d,1e

```
#include <vector>
#include <queue>

template <class V, class I>
bool topo_sort(const V &edges, I &idx, int n) {
  typedef typename V::value_type::const_iterator E_iter;
  vector<int> indeg;
  indeg.resize(n, 0);
  for (int i = 0; i < n; i++)
    for (E_iter e = edges[i].begin(); e != edges[i].end(); e++)
      indeg[*e]++;
  //queue<int> q;
  priority_queue<int> q; // **
  for (int i = 0; i < n; i++)
    if (indeg[i] == 0)
      q.push(-i);
  int nr = 0;
  while (q.size() > 0) {
    //int i = -q.front();
    int i = -q.top(); // **
    idx[i] = nr++;
    q.pop();
    for (E_iter e = edges[i].begin(); e != edges[i].end(); e++)
      if (--indeg[*e] == 0)
        q.push(-*e);
  }
  return nr == n;
}
```

Listing 5.18: euler walk.cpp — c518e621

43 lines
35,12,19,2f,34, 5,29,32,
9,15,28, 9,35,2b,17,22

```
#include <list>

template<class V>
void euler_walk( V &edges, int start, list< int > &path, bool cyclic=false ) {
  int node = start, next_node;

  // Find a maximal path
  while( true ) {
    typename V::value_type &s = edges[node];

    path.push_back( node );

    if( s.empty() )
      break;

    // Follow the first edge and remove it
    next_node = *s.begin();
    s.erase( s.begin() );

    node = next_node;
  }

  // If no cyclic path was found, return an "empty" path, i.e. only the start node
  if( cyclic && node != start ) {
    path.clear();
    path.push_back( node );
    return;
  }

  // Extend path with cycles
  //for( list<int>::iterator iter = path.begin(); iter != path.end(); iter++ )

  for( list<int>::iterator iter = --path.end(); iter != path.begin(); ) {
    list<int>::iterator iter2 = iter; iter2--;
    node = *iter;

    typename V::value_type &s = edges[node];
    while( !s.empty() ) {
      list<int> extra_list;
      euler_walk( edges, node, extra_list, true /*must be cyclic*/ );
      path.splice( iter, extra_list, extra_list.begin(), --extra_list.end() );
    }
    iter = iter2;
  }
}
```

Listing 5.19: deBruijn.cpp — 96e90632

49 lines
32, 3,23,19,14, 6,2c, 3,
3b,32,2f, 2,1c, 5,27,36

```
using namespace std;

void deBruijn( int numSymbols, int L, char symbols[]) {
  int             numNodes;
  vector< vector<int> > edges;
  list<int>       path;

  // Number of nodes is numSymbols^(L-1)
  numNodes = 1;
  for( int i=0; i<L-1; i++ )
    numNodes *= numSymbols;

  // Create edges
  edges.resize( numNodes );
```

```
  for( int i=0; i<numNodes; i++ ) {
    edges[i].resize( numSymbols );

    for( int j=0; j<numSymbols; j++ )
      edges[i][j] = (i*numSymbols)%numNodes + j;
  }

  // Find euler walk
  path.clear();
  euler_walk( edges, 0, path );

  // Non-cyclic deBruijn sequences
  cout << "Non-cyclic:" << endl;

  string answer;
  for( list<int>::iterator iter = path.begin(); iter != path.end( \
); iter++ ) {
    int node = *iter;

    if( iter == path.begin() ) {
      int d = numNodes;

      for( int j=0; j<L-1; j++ ) {
        d/= numSymbols;
        answer += symbols[ node % numSymbols ];
      }
    } else
      answer += symbols[ node % numSymbols ];
  }
  cout << answer << endl;

  // Cyclic deBruijn sequences
  cout << "Cyclic:" << endl;
  cout << answer.substr(0, answer.length()-(L-1)) << endl << \
endl;
}
```

Listing 5.20: deBruijn fast.cpp — 382b9aaf

80 lines
4,31,58,61,29,5c,23,64,
53,76,38,7a, e,53, 8,47

```
template<class V>
void euler_walk_dB( V &edges, int start, list< int > &path, int nSymb,
                    int nNodes )
{
  int node = start;

  while( true ) {
    int &s = edges[node];

    path.push_back( node );

    if( s == 0 )
      break;

    for( int i=0; i<nSymb; i++ ) {
      if( s & (1<<i) ) {
        node = (node*nSymb)%nNodes + i;
        s ^= (1<<i);
        break;
      }
    }
  }

  //for( list<int>::iterator iter = path.begin(); iter != path.end(); iter++ )

  for( list<int>::iterator iter = --path.end(); iter != path.begin(); ) {
    list<int>::iterator iter2 = iter; iter2--;
    node = *iter;

    int &s = edges[node];
    while( s != 0 ) {
      list<int> extra_list;
      euler_walk_dB( edges, node, extra_list, nSymb, nNodes );
      path.splice( iter, extra_list, extra_list.begin(), --extra_list.end() );
    }
    iter = iter2;
  }
}


void deBruijn_fast( int nSymb, int L, char symbols[]) {
  int             nNodes;
  vector< int >   edges;
  list<int>       path;

  nNodes = 1;
  for( int i=0; i<L-1; i++ )
    nNodes *= nSymb;

  edges.reserve( nNodes );
  for( int i=0; i<nNodes; i++ )
    edges.push_back( (1<<nSymb)-1 );

  euler_walk_dB( edges, 0, path, nSymb, nNodes );


  // Non-cyclic deBruijn sequences
  cout << "Non-cyclic:" << endl;

  string answer;
  for( list<int>::iterator iter = path.begin(); iter != path.end(); iter++ ) {
    int node = *iter;

    if( iter==path.begin() ) {
      int d = nNodes;

      for( int j=0; j<L-1; j++ ) {
        d/= nSymb;
        answer += symbols[ node % nSymb ];
      }
    } else
      answer += symbols[ node % nSymb ];
  }
  cout << answer << endl;

  // Cyclic deBruijn sequences
  cout << "Cyclic:" << endl;
  cout << answer.substr(0, answer.length()-(L-1)) << endl << endl;
}
```

Network Flow

Listing 5.21: flow graph.cpp — b2909a74

17 lines
e, 0, c, f, 8,1e,19,12,
8, 3, 7,13, f,19, b, 1

```cpp
#include <vector>


template <class T>
struct flow_edge {
  typedef T flow_type;
  int dest, back; // back is the index of the back-edge in graph[dest]
  T c, f; // capacity and flow
  flow_edge() {}
  flow_edge(int _dest, int _back, T _c, T _f = T())
    : dest(_dest), back(_back), c(_c), f(_f) { }
};

template <class E, class T>
void flow_add_edge(E &flow, int node, int dest, T c, T back_c = T()) {
  flow[node].push_back(flow_edge<T>(dest, flow[dest].size(), c));
  flow[dest].push_back(flow_edge<T>(node, flow[node].size() - 1, back_c));
}

typedef vector< vector< flow_edge<int> > > flow_graph;
```

Listing 5.22: lift to front.cpp — b132b555

64 lines
69,20,62,76,46,53,7e,28,
1f,77,43,67,67,4e,7d,37

```cpp
#include <vector>
#include "flow_graph.cpp"


template <class E, class T, class V>
void add_flow(E &flow, flow_edge<T> &edge, T f, V &excess) {
  flow_edge<T> &back = flow[edge.dest][edge.back];
  edge.f += f; edge.c -= f;
  back.f -= f; back.c += f;
  excess[edge.dest] += f;
  excess[back.dest] -= f;
}

template <class E>
typename E::value_type::value_type::flow_type lift_to_front(E &flow,
                                                 int source, int sink) {
  typedef typename E::value_type::value_type::flow_type T;
  typedef typename E::value_type L;
  typedef typename L::iterator L_iter;
  int v = flow.size();

  // init preflow
  vector<int> height(v, 0);
  vector<T> excess(v, T());
  height[source] = v - 2;

  for (L_iter it = flow[source].begin(); it != flow[source].end(); it++)
    add_flow(flow, *it, it->c, excess);

  // init lift-to-front
  vector<int> l(v, sink); // lift-to-front list
  vector<L_iter> cur; // current edge, per node
  int p = sink;
  for (int i = 0; i < v; i++)
    if (i != source && i != sink)
      l[i] = p, p = i; // turn l into a linked list from p to sink
  for (int i = 0; i < v; i++)
    cur.push_back(flow[i].begin());

  // lift-to-front loop
  int r = source, u = p;
  while (u != sink) {
    int oldheight = height[u];

    // discharge u
    while (excess[u] > 0)
      if (cur[u] == flow[u].end()) {
        // lift u
        height[u] = 2 * v - 1;
        for (L_iter it = flow[u].begin(); it != flow[u].end(); it++)
          if (it->c > 0 && height[it->dest]+1 < height[u]) {
            height[u] = height[it->dest]+1;
            cur[u] = it; // start from an admissable edge!
          }
      }
      else if (cur[u]->c > 0 && height[u] == height[cur[u]->dest] + 1)
        // push on edge cur[u]
        add_flow(flow, *cur[u], min(excess[u], (*cur[u]).c), excess);
      else
        ++cur[u];

    // the lift-to-front bit:
    if (height[u] > oldheight)
      l[r] = l[u], l[u] = p, p = u; // move u to front of list
    r = u, u = l[r]; // move to next in list
  }
  return excess[sink];
}
```

Listing 5.23: ford fulkerson.cpp — 49c8547b

8 lines
0, 7, 1, 1, 3, 0, 7, 0,
7, 3, 0, 3, 4, 1, 3, 2

```cpp
#include <queue>
#include "flow_graph.cpp"

// Function prototypes
int flow_increase( flow_graph &g, int source, int sink );

// Internal auxillary functions
bool flow_findaugpath_dfs( const flow_graph &g, vector<int> &backEdges,
                      int source, int sink );
void flow_findaugpath_bfs( const flow_graph &g, vector<int> &backEdges,
                      int source, int sink );


/*********************************************************
 * Max-flow in a general flow-graph
```

```
 ************************************************************/

int flow_increase( flow_graph &g, int source, int sink ) {
  const int inf = 0x20000000;
  vector<int> backEdges;

  backEdges.resize( g.size(), -1 );
  backEdges[source] = 0; // backEdges>=0 is also used to mark
  // if the node has been traversed

  // Find augmenting path (choose one of these)
  flow_findaugpath_dfs( g, backEdges, source, sink );
  //flow_findaugpath_bfs( g, backEdges, source, sink );

  if( backEdges[sink] < 0 )
    return 0;

  // Find min-slack
  int minSlack = inf;

  for( int node=sink; node!=source; ) {
    flow_edge<int> &be = g[node][backEdges[node]];
    flow_edge<int> &fe = g[be.dest][be.back];

    minSlack = min( minSlack, fe.c );
    node = be.dest;
  }

  // Increase flow
  for( int node=sink; node!=source; ) {
    flow_edge<int> &be = g[node][backEdges[node]];
    flow_edge<int> &fe = g[be.dest][be.back];

    fe.f += minSlack; fe.c -= minSlack;
    be.f -= minSlack; be.c += minSlack;
    node = be.dest;
  }

  return minSlack;
}


// The backEdges is an array containing the index of the backEdge
// which should be followed to get back to the source

void flow_findaugpath_bfs( const flow_graph &g, vector<int> &backEdges,
                      int source, int sink )
{
  const int inf = 0x20000000;
  queue<int> q;
  vector<int> min;

  min.resize( g.size() );

  // Initialize min/backEdges
  int n = g.size();
  for( int i=0; i<n; i++ ) {
    min[i] = inf;
    backEdges[i] = -1;
  }
  min[source] = 0;

  // BFS-search
  q.push( source );

  while( !q.empty() ) {
    int node = q.front();
    int length = min[node]+1;

    q.pop();

    // Process node
    const vector<flow_edge<int> > &edges = g[node];
    int numEdges = edges.size();
    for( int i=0; i<numEdges; i++ ) {
      const flow_edge<int> &fe = edges[i];

      if( fe.c <= 0 )
        continue;

      int dest = fe.dest;

      if( length < min[dest] ) {
        // Process this node the next run
        min[dest] = length;
        backEdges[dest] = fe.back;
        q.push( dest );

        if( dest == sink )
          return;
      }
    }
  }
}

// The backEdges is an array containing the index of the backEdge
// which should be followed to get back to the source.
// Make sure backEdges is initialized to -1 prior to this function
// except for the source which should have a value >=0!

bool flow_findaugpath_dfs( const flow_graph &g, vector<int> &backEdges,
                      int node, int sink )
{
  typedef const vector<flow_edge<int> > E;
  typedef E::const_iterator E_iter;

  E & el = g[node];
  for( E_iter e=el.begin(); e!=el.end(); ++e ) {
    if( e->c <= 0 )
      continue;

    int dest = e->dest;
    if( backEdges[dest] < 0 ) {
      // Process this node
      backEdges[dest] = e->back;

      if( dest == sink || flow_findaugpath_dfs(g,backEdges,dest,sink) )
        return true;     // Found augmenting path
    }
  }

  return false;
}
```

Listing 5.24: ford fulkerson 1.cpp — 9511f2d0

41 lines
11, 2, b,23,31,24,21,26,
2,18, 1,22, e,2b,1f, 7

```
// Function prototypes
bool flow_increase1( flow_graph &g, int source, int sink );
// Internal auxillary function
bool flow_dfs1( flow_graph &g, vector<int> &proc, int source, \
int sink );


bool flow_increase1( flow_graph &g, int source, int sink ) {
  vector<int> proc; // in reality bool but vector<bool> is \
slower

  proc.resize( g.size(), false );

  return flow_dfs1( g, proc, source, sink );
}

bool flow_dfs1( flow_graph &g, vector<int> &proc, int node, int \
sink ) {
  typedef vector<flow_edge<int> >::iterator E_iter;
  typedef vector<flow_edge<int> >  E;
  bool found = false;

  proc[node] = true;

  E & el = g[node];
  for( E_iter e=el.begin(); e!=el.end(); e++ ) {
    if( e->c <= 0 )
      continue;

    int dest = e->dest;

    if( !proc[dest] ) {
      // Process this node
      if( dest == sink || flow_dfs1(g,proc,dest,sink) ) {
        // Found augmenting path - add flow 1
        e->f++; e->c--;
        g[dest][e->back].f--; g[dest][e->back].c++;

        found = true;
        break;
      }
    }
  }

  return found;
}
```

Matching

Listing 5.25: hopcroft karp.cpp — 675b290f

104 lines
31, 6,48,3a,4c,65,21,66,
79,3b,41,45, 8,5a,7d,3f

```
#include <queue>
#include <vector>
#include <utility>

template< class M >
bool hk_recurse( int b, int *lPred, vector<int> *rPreds, M match_b ) {
  vector< int > L;

  L.swap( rPreds[b] );

  for( unsigned int i=0; i<L.size(); ++i ) {
    int a = L[i];
    int b2 = lPred[a];

    lPred[a] = -2;
    if( b2 == -2 )
      continue;
    if( b2 == -1 || hk_recurse(b2, lPred, rPreds, match_b) ) {
      match_b[b] = a;
      return true;
    }
  }
  return false;
}


template< class G, class M, class T >
int hopcroft_karp( G g, int n, int m, M match_b, T mis_a, T mis_b ) {
  typedef typename G::value_type::const_iterator E_iter;

  int lPred[n];
  vector< int > rPreds[m];
  queue< int > leftQ, rightQ, unmatchedQ;
  bool rProc[m], rNextProc[m];

  for( int i=0; i<m; i++ )
    match_b[i] = -1;

  // Greedy matching (start)
  for( int i=0; i<n; i++ ) {
    for( E_iter e=g[i].begin(); e!=g[i].end(); ++e ) {
      if( match_b[*e]<0 ) {
        match_b[*e] = i;
        break;
      }
    }
  }

  while( true ) {
    for( int i=0; i<n; i++ )
      lPred[ i ] = -1; // i is in the first layer
    for( int j=0; j<m; j++ )
      if( match_b[j]>=0 )
        lPred[match_b[j]] = -2; // remove from layer alltogether
    for( int j=0; j<m; j++ ) {
      rPreds[j].clear();
      rProc[j] = rNextProc[j] = false;
    }

    for( int i=0; i<n; i++ )
      if( lPred[i]==-1 )
        leftQ.push( i );

    while( !leftQ.empty() && unmatchedQ.empty() ) {
      while( !leftQ.empty() ) {
        int a = leftQ.front(); leftQ.pop();
        for( E_iter e=g[a].begin(); e!=g[a].end(); ++e )
          if( !rProc[*e] ) {
            rPreds[*e].push_back( a );
            if( !rNextProc[*e] ) {
              rightQ.push( *e );
              rNextProc[*e] = true;
            }
          }
      }
      while( !rightQ.empty() ) {
        int b = rightQ.front(); rightQ.pop();

        rProc[b] = true;
        if( match_b[b] >= 0 ) {
          leftQ.push( match_b[b] );
          lPred[ match_b[b] ] = b;
        } else
          unmatchedQ.push( b );
      }
    }

    while( !leftQ.empty() )
      leftQ.pop();

    if( unmatchedQ.empty() ) { // No more alternating paths
      int nMatch = 0;
      for( int i=0; i<n; i++ )
        mis_a[i] = lPred[i]>=-1;
      for( int j=0; j<m; j++ ) {
        mis_b[j] = !rProc[j];
        nMatch += match_b[j]>=0;
      }
      return nMatch;
    }

    while( !unmatchedQ.empty() ) {
      int b = unmatchedQ.front(); unmatchedQ.pop();
      hk_recurse( b, lPred, rPreds, match_b );
    }
  }
}
```

Maximum Weight Bipartite Matching

Listing 5.26: mwbm.cpp — 99ea72c

143 lines
ee,8f, 5, 0,b7, 5,8a, 1,
5f,1c,1f,39,bc,a5,c5,3f

```
#include <vector>

template< class E, class M, class W >
inline bool augment( E &edges, int a, int n, int m,
                     vector<W> &pot, vector<bool> &free,
                     vector<int> &pred, vector<W> &dist, M &match_b,
                     bool perfect )
{
  typedef typename E::value_type L;
  typedef typename L::const_iterator L_iter;

  vector<bool> proc(m, false);
  dist[a] = 0;
  pred[a] = a; // Start of alternating path
  int best_a = a, a1 = a, v;
  W minA = pot[a], delta;

  while( true ) {
    // Relax all edges out of a1
    for( L_iter e = edges[a1].begin(); e != edges[a1].end(); ++e ) {
      int b = n+e->first;
      if( match_b[b-n] == a1 )
        continue;

      W db = dist[a1] + (pot[a1]+pot[b]-e->second);

      if( pred[b] < 0 || db < dist[b] ) {
        dist[b] = db; pred[b] = a1;
      }
    }

    // Select a node b with minimal distance db
    int b1 = -1;
    W db=0; // unused but makes compiler happy
    for( int b=n; b<n+m; b++ ) {
      if( !proc[b-n] && pred[b]>=0 && (b1<0 || dist[b]<db) ) {
        b1 = b;
        db = dist[b];
      }
    }

    if( b1>=0 )
      proc[b1-n] = true;


    // End conditions
    if( !perfect && (b1<0 || db >= minA) ) {
      // Augment by path to best node in A
      delta = minA;
      free[a] = false; free[best_a] = true; // NB! Order is important
      v = best_a;
      break;
    } else if( b1<0 ) {
      return false;
    } else if( free[b1] ) {
      // Augment by path to b
      delta = db;
      free[a] = free[b1] = false;
      v = b1;
      break;
    }

    // Continue shortest-path computation
    a1 = match_b[ b1-n ];
    pred[a1] = b1;
    dist[a1] = db;
    if( db+pot[a1] < minA ) {
      best_a = a1;
      minA = db+pot[a1];
    }
  }

  // Augment path
  while( true ) {
    int vn = pred[v];

    if( v==vn )
      break;

    if( v>=n ) match_b[v-n] = vn;
    v = vn;
  }

  for( int a=0; a<n; a++ ) {
    if( pred[a]>=0 ) {
      W dpot = delta - dist[a];
      pred[a] = -1;
      if( dpot > 0 ) pot[a] -= dpot;
    }
  }
  for( int b=n; b<n+m; b++ ) {
    if( pred[b]>=0 ) {
      W dpot = delta - dist[b];
      pred[b] = -1;
      if( dpot > 0 ) pot[b] += dpot;
    }
  }
  return true;
}

template< class E, class M, class W >
bool max_weight_bipartite_matching( E &edges, int n, int m, M &match_b,
                                    W &max_weight, bool perfect )
{
  typedef typename E::value_type L;
  typedef typename L::const_iterator L_iter;

  vector<W> pot( n+m, 0 );
  vector<bool> free(n+m, true );
  vector<int> pred( n+m, -1 );
  vector<W> dist( n+m, 0 );

  for( int b=0; b<m; b++ )
    match_b[b] = -1;

  // Initialize pot and matching with simple heuristics
  for( int a=0; a<n; a++ ) {
    int b = -1;
    W Cmax = 0;
```

```
    for( L_iter e = edges[a].begin(); e != edges[a].end(); ++e ) {
      if( b<0 || e->second > Cmax || e->second==Cmax && free[n+e->first] ) {
        b = n+e->first;
        Cmax = e->second;
      }
    }
    pot[a] = Cmax;
    if( b>=0 && free[b] ) {
      match_b[b-n] = a;
      free[a] = free[b] = false;
    }
  }

  // Augment matching
  for( int a=0; a<n; a++ )
    if( free[a] )
      if( !augment(edges, a, n, m, pot, free, pred, dist, match_b, perfect) )
        return false;

  max_weight = 0;
  for( int i=0; i<n+m; i++ )
    max_weight += pot[i];

  return true;
}
```


Listing 5.27: mwbm of max card.cpp — efd6d2bd

27 lines
12,13,10, 7,1e, 9,16,1b,
19,1c,12,11, 3,19, c,16

```
#include "max_weight_bipartite_matching.cpp"

template< class E, class M, class W >
void max_weight_b_m_of_max_card( E &edges, int n, int m, M &match_b,
                                 W &max_weight )
{
  typedef typename E::value_type L;
  typedef typename L::iterator L_iter;

  W Cmax = 0;
  for( int a=0; a<n; a++ )
    for( L_iter e = edges[a].begin(); e != edges[a].end(); ++e )
      Cmax = max( Cmax, max(e->second, -e->second) );
  Cmax = 1 + 2*max(n,m)*Cmax;

  for( int a=0; a<n; a++ )
    for( L_iter e = edges[a].begin(); e != edges[a].end(); ++e )
      e->second += Cmax;

  max_weight_bipartite_matching( edges, n, m, match_b, max_weight, false );

  for( int b=0; b<m; b++ )
    if( match_b[b] >= 0 )
      max_weight -= Cmax;

  for( int a=0; a<n; a++ )
    for( L_iter e = edges[a].begin(); e != edges[a].end(); ++e )
      e->second -= Cmax;
}
```

Chapter 6

Geometry

6.1 Geometric primitives

6.1.1 π, sqr, point and line

Listing – geometry.cpp, p. 92

The recomended way of defining pi in contests.
The useful `sqr` function.
A very simple point struct. Normally one should use the one in `point.cpp`.
A line struct.

6.1.2 Point

Listing – point.cpp, p. 92

A point struct with comparison, difference, inverse scaling, dot and scalar cross product.

6.1.3 Point operations

Listing – point ops.cpp, p. 92

The operations `dist2, dx, dy, dz, dist, angle, theta, unit, perm, normal` on points. Theta is a rational meta-angle function in the rangle $(-2, 2]$.

6.1.4 Point 3D

Listing – point3.cpp, p. 92

A 3D point struct with comparison, difference, inverse scaling, dot and vector cross product.

6.1.5 Point-line relations

Listing – point line.cpp, p. 92

`sideof` Determine on which side of a line a point is. +1/-1 is left/right of vector $p_1 - p_0$ and 0 is on the line.

`onsegment` Determine if a point is on a line segment (incl the end points).

`linedist` Get a measure of the distance of a point from a line (0 on the line and positive/negative on the different sides).

6.1.6 Line intersection

Listing – line isect.cpp, p. 93

Intersection point between two lines.

6.1.7 Interval intersection

Listing – interval.cpp, p. 93

Intersection between two intervals or rectangles.

6.1.8 Interval intersection

Listing – interval.cpp, p. 93

Intersection between two intervals or rectangles.

6.1.9 Interval union

Listing – ival union.cpp, p. 93

The union of several intervals given as pair¡first,last¿ in a container. The result is a disjoint list of intervals in ascending order.

6.1.10 Circle tangents

Listing – circle tangents.cpp, p. 93

The tangent points from a point to a circle. The algorithm returns if the point lies on the circles perimeter (in which case the two tangent points are equal).

6.1.11 Counter-Clock-Wise

Listing – ccw.cpp, p. 93

Sedgewick ccw function.

6.1.12 CCW Line segment intersection test

Listing – isect test.cpp, p. 94

Based on Sedgewick's ccw function.

6.2 Triangles

6.2.1 Heron triangle area

Listing – heron.cpp, p. 95

Triangle area using Heron's formula $K = \sqrt{p(p-a)(p-b)(p-c)}$, where $p = \frac{a+b+c}{2}$.

6.2.2 Enclosing circle

Listing – incircle.cpp, p. 95

`incircle` returns a determinant, whose sign determines whether a point lies inside the circle enclosing three other points.

Usage `bool enclosing_centre(a, b, c, &p[, eps]);`

Fills in the enclosing circle centre of points a, b, c in point p. Returns false if the points are colinear within the eps limit.

Usage `bool enclosing_radius(a, b, c, &r[, eps]);`

Fills in the enclosing circle radius in r, using $r = \frac{abc}{4K}$, where K is the triangle area (as in Heron). Returns false if the points are colinear within the eps limit.

6.3 Polygons

6.3.1 Inside polygon

Listing – inside.cpp, p. 96

Complexity $\mathcal{O}(n)$

Usage `inside(polygon,nPts,point) == true;`

Determine whether a point is inside a polygon. If it is on an edge, standard computer graphics rules determine the returned value (inside above and to the left of the polygon, but not below or to the right). This is usually *not* the desired behaviour in contest geometry problems. Use `on_edge` in `pointline.cpp` to check if a point is on the edge.

6.3.2 Winding number

Listing – winding number.cpp, p. 96

Complexity $\mathcal{O}(n)$

Theta meta-angle winding number of a point with a polygon. The polygon should be given in ccw-order. Also function `inside_wn` which does the same as `inside` but uses the winding number. The value of `inside_wn` is +1/-1 for inside/outside and 0 for on the edge.

6.3.3 Polygon area

Listing – poly area.cpp, p. 96

Listing – poly area too.cpp, p. 96

The signed area of a polygon calculated by adding cross product or trapezium areas.

6.3.4 Polyhedron volume

Listing – poly volume.cpp, p. 96

The signed volume of a polyhedron calculated by adding vector tripple product volumes.

6.3.5 Polygon cut

Listing – poly cut.cpp, p. 96

Usage `iterator r_end = poly_cut(v.begin(), v.end(), p0, p1, r.begin())`

Cuts a polygon with (a half plane specified by) a line. `r` is filled in with the cut polygon, and the end of the filled in interval is returned. The polygon is kept connected by (overlapping) line segments along the cutting line if the cut splits the polygon in parts.

6.3.6 Center of mass

Listing – center of mass.cpp, p. 97

Polygon and triangular center of mass.

6.4 Convex Hull

NOTE None of the Graham scans handle multiple coinciding points, so make sure the points are unique before calling!

6.4.1 Graham scan

Listing – convex hull.cpp, p. 98

Complexity $\mathcal{O}(n \log(n))$

Usage `iterator hull_end = convex_hull(p.begin(), p.end())`

Swaps the points in `p` so the hull points are in order at the beginning.

Note! Handles colinear points on the hull

6.4.2 Graham scan, indexed

Listing – convex hull idx.cpp, p. 98

Complexity $\mathcal{O}(n \log(n))$

Usage `convex_hull_idx(points p, idx, int n)`

Fills the index vector `idx` with indeces to the point vector `p`, returning the number of points in the hull.

Note! Does not handle colinear points on the hull consistently

6.4.3 Graham scan, colinearly robust, indexed

Listing – convex hull robust idx.cpp, p. 98

Complexity $\mathcal{O}(n \log(n))$

Note! Does handle colinear points on the hull consistently

6.4.4 Three dimensional hull

Complexity $\mathcal{O}(n^2)$

Listing – convex hull space.cpp, p. 99

Usage `convex_hull_space(points p, int n, list<ABC> &trilist)`

`trilist` is a list of ABC-tripples of indices of vertices in the 3D point vector `p`.

Note! Requires the hull to have positive volume. Arbitrarily triangulates the surface of the hull.

6.4.5 Point inside hull

Listing – inside hull.cpp, p. 99

Complexity $\mathcal{O}(\log(n))$

Usage `inside_hull(hull p, int n, point t)`

Determine whether a point `t` lies inside the hull given by the point vector `p`. The hull should not contain colinear points. A hull with 2 points are ok. The result is given as: 1 inside, 0 onedge, -1 outside.

6.4.6 Point inside hull simple

Listing – inside hull simple.cpp, p. 100

Complexity $\mathcal{O}(n)$

Usage `inside_hull_simple(It begin, It end, point t)`

Determine whether a point `t` lies inside the hull given by begin and end. Colinear points are ok. If duplicate points exists, it will return *onedge* when it is inside. The hull must have at least one point. The result is given as: 1 inside, 0 onedge, -1 outside.

6.4.7 Hull diameter

Listing – hull diameter.cpp, p. 100

Complexity $\mathcal{O}(n)$

Usage `hull_diameter2(hull p, int n, &i1, &i2)`

Determine the points that are farthest apart in a hull. `i1, i2` will be the indices to those points after the call. The squared distance is returned.

6.4.8 Minimum enclosing circle

Listing – mec.cpp, p. 100

Complexity $\mathcal{O}(n)$

Usage `bool mec(p, n, c, &i1, &i2, &i3[, eps]);`

Usage `double mec(p, n, c[, eps]);`

Fills in c with the centre point of the minimum circle, enclosing the n point vector p. The first version fills in indices to the points determining the circle, and returns whether the third index is used. The second version returns the enclosing circle radius as a double. Colinearity of a third point is determined by the eps limit.

6.4.9 Line-hull intersect

Listing – line hull intersect.cpp, p. 100

Complexity $\mathcal{O}(\log(n))$

Usage `line_hull_intersect(hull p, int n, point p1, point p2, &s1, &s2)`

Determine the intersection points of a hull with a line. `p1, p2, s1, s2` will be the intersection points and indices to the hull line segments that intersect after the call. Returns whether there is an intersection.

6.5 Minimum enclosing circle

See Convex hull, Minimum enclosing circle.

6.6 Voronoi diagrams

6.6.1 Simple Delaunay triangulation

Listing – delaunay simple.cpp, p. 102

Complexity $\mathcal{O}\left(n^4\right)$

Usage `delaunay(points p, int n, trifun)`

Uses a `trifun(int, int, int)` to return all possible delaunay triangles as tripple indices to the point vector.

Note! Triangles may overlap if points are cocircular.

6.6.2 Convex hull Delaunay triangulation

Listing – delaunay hull.cpp, p. 102

Complexity $\mathcal{O}$(3d convex hull)

Usage `delaunay(points p, int n, trifun)`

Returns an arbitrary triangulation if points are cocircular.

Note! Depending on convex hull implementation it may fail if *all* points are cocircular, as is currently the case.

6.7 Nearest Neighbour

6.7.1 Divide and conquer

Listing – closest pair.cpp, p. 103

Complexity $\mathcal{O}(n \log n)$

Usage `closestpair( points p, int n, &i1, &i2 )`

`i1, i2` are the indices to the closest pair of points in the point vector `p` after the call. The distance is returned.

6.7.2 Simpler method

Listing – closest pair simple.cpp, p. 104

Complexity $\mathcal{O}\left(n^2 \text{ (average } n)\right)$

Usage `closestpair( points p, int n, &i1, &i2 )`

Primitives

Listing 6.1: geometry.cpp — 4f264008

10 lines
8, 8, 2, a, f, e, 4, e,
a, 2, 8, 8, 1, 1, 5, e

```
#include <cmath>

const double pi = acos(0.0) * 2;


template <class T> T sqr(T x) { return x * x; }


template <class P> struct line {
  typedef typename P::coord_type coord_type;
  P p1, p2; line(P _p1 = P(), P _p2 = P()) : p1(_p1), p2(_p2) { }
};
```

Listing 6.2: point.cpp — 3c1eb2e1

20 lines
9,11, e, 7,14,10,12, 7,
1, 8,10,19, 2, 2, 3,18

```
template <class T>
struct point {
  typedef T coord_type;
  typedef point S;
  typedef const S &R;
  T x, y;
  point(T _x=T(), T _y=T()) : x(_x), y(_y) { }
  bool operator< (R p) const {
    return x < p.x || x <= p.x && y < p.y;
  }
  S operator-(R p) const { return S(x - p.x, y - p.y); }
  S operator+(R p) const { return S(x + p.x, y + p.y); }
  S operator/(T d) const { return S(x / d, y / d); }
  T dot(R p) const { return x*p.x + y*p.y; }
  T cross(R p) const { return x*p.y - y*p.x; }
};

template <class T> T dot(point<T> p, point<T> q) { return p.dot(q); }

template <class T> T cross(point<T> p, point<T> q) { return p.cross(q); }
```

Listing 6.3: point ops.cpp — 5dea0226

19 lines
12,17,1d, b,19,1e,11,14,
8,10,12,19, 9,11, 1, f

```
template <class P> typename P::coord_type dist2(P p) { return dot(p, p); }
template <class P> typename P::coord_type dx(P p, P q) { return q.x - p.x; }
template <class P> typename P::coord_type dy(P p, P q) { return q.y - p.y; }
// for point3:
template <class P> typename P::coord_type dz(P p, P q) { return q.z - p.z; }
#include <cmath>
template <class P> double dist(P p) { return sqrt((double)dist2(p)); }
template <class P> double angle(P p) { return atan2((double)p.y, (double)p.x);}
template <class P> double theta(P p) {
  double x = p.x, y = p.y; if (x==0 && y==0) return 0;
  double t = y / (x<0^y<0 ? x-y : x+y);
  return x<0 ? y<0 ? t-2 : t+2 : t;
}


template <class P> P unit(P p) { return p / dist(p); }
template <class P> P perp(P p) { return P(-p.y, p.x); }
template <class P> P normal(P p) { return unit(perp(p)); }
// for point3: (unit normal to a plane from two vectors)
template <class P> P normal(P p, P q) { return unit(cross(p, q)); }
```

Listing 6.4: point3.cpp — a4eec7d8

22 lines
1f, 3,1f,17, a,1e,1b,13,
f, 4, d,1b, b,13, f, c

```
template <class T>
struct point3 {
  typedef T coord_type;
  typedef point3 S;
  typedef const S &R;
  T x, y, z;
  point3(T _x=T(), T _y=T(), T _z=T()) : x(_x), y(_y), z(_z) { }
  bool operator< (R p) const {
    return x < p.x || x <= p.x && (y < p.y || y <= p.y && z < p.z);
  }
  S operator-(R p) const { return S(x - p.x, y - p.y, z - p.z); }
  S operator+(R p) const { return S(x + p.x, y + p.y, z + p.z); }
  S operator/(T d) const { return S(x / d, y / d, z / d); }
  T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
  S cross(R p) const { return S(y*p.z - z*p.y,
                               z*p.x - x*p.z,
                               x*p.y - y*p.x); }
};

template <class T> T dot(point3<T> p, point3<T> q) { return p.dot(q); }

template <class P> P cross(P p, P q) { return p.cross(q); }
```

Listing 6.5: point line.cpp — ebbc6ca0

20 lines
e, 6, e, 3, c, c,19, f,
1f,1d,18,18, 2,16,1e, 5

```
#include "point_ops.cpp"

template <class P>
int sideof(P p0, P p1, P q) {
  typename P::coord_type d = cross(p1-p0, q-p0);
  return d > 0 ? 1 : d < 0 ? -1 : 0;
}

template<class P>
bool onsegment(P p0, P p1, P q) {
  // Check if point is outside line segment rectangle.
```

```
  if( dx(p0,q)*dx(p1,q) > 0 || dy(p0,q)*dy(p1,q) > 0 )
    return false;

  // Check if point is on line (not line segment)
  return cross(p1-p0, q-p0) == 0;
}

template <class P>
double linedist(P p0, P p1, P q) {
  return (double) cross(p1-p0, q-p0) / dist(p1-p0);
}
```

Listing 6.6: line isect.cpp — ba855170

14 lines
f, 6, 6, 1, 9, 8, 4, b,
f, 2, 1, f, 1, c, 1, 4

```
template <class P, class R>
bool line_isect(P p0, P p1, P q0, P q1, R &x) {
  typedef typename R::coord_type T;

  P p = p1-p0, q = q1-q0;
  T det = cross(p, q);
  if (det == 0)
    return false;

  T a = dot(perp(p), p0), b = dot(perp(q), q0);
  x.x = (a*q.x - b*p.x) / det;
  x.y = (a*q.y - b*p.y) / det;
  return true;
}
```

Listing 6.7: interval.cpp — 5dec770f

14 lines
c, 2, d, 9, 1, 5, 2, 3,
5, a, 3, a, c, 6, a, 3

```
#include "geometry.h"

template <class T>
bool ivalisect(T p0, T p1, T q0, T q1, T &r0, T &r1) {
  T pmin = min(p0, p1), pmax = max(p0, p1);
  T qmin = min(q0, q1), qmax = max(q0, q1);
  r0 = max(pmin, qmin), r1 = min(pmax, qmax);
  return r0 <= r1;
}

template <class P>
bool rectisect(P p0, P p1, P q0, P q1, P &r0, P &r1) {
  bool xflag = ivalisect(p0.x, p1.x, q0.x, q1.x, r0.x, r1.x);
  bool yflag = ivalisect(p0.y, p1.y, q0.y, q1.y, r0.y, r1.y);
  return xflag && yflag;
}
```

Listing 6.8: ival union.cpp — 7d2b50e5

14 lines
8, 0, 0, 6, 4, c, 6, 3,
9, 6, 5, 1, e, 2, f, e

```
template <class F, class O>
O ival_union( F begin, F end, O dest ) {
  sort( begin, end );
  while( begin != end ) {
    *dest = *begin++;
    while( begin != end && begin->first <= dest->second ) {
      dest->second = max(dest->second,begin->second);
      ++begin;
    }
    ++dest;
  }
  return dest;
}
```

Listing 6.9: circle tangents.cpp — 525e3e94

10 lines
8, 6, f, b, 3, 3, 9, d,
7, b, 6, a, e, 7, d, e

```
template <class P, class T>
bool circle_tangents(const P &p, const P &c, T r, P &t1, P &t2) {
  P a = (c-p), ap = perp(a);
  double a2 = dist2(a), r2 = r*r;
  P x = p+a*(1-r2/a2), y = ap*(sqrt(a2-r2)*r/a2);

  t1 = x + y;
  t2 = x - y;
  return a2==r2;
}
```

Listing 6.10: ccw.cpp — 37f342be

21 lines
d,14, d,1a,11, d,1d, 4,
13,1e,1e, 1,14, e,14,17

```
template <class P>
int ccw(P p0, P p1, P p2) {
  typedef typename P::coord_type T;
  P d1 = p1-p0, d2 = p2-p0;

  T d = cross(d1, d2);
  if( d != 0 ) return d>0 ? 1:-1;

  // Points are on a line

  // If points are on different sides of p1, the angle is
  // 180 degrees (a degenerated triangle). This is a ambigous
  // case which never occurs for the convex_hull algorithm.
  if (d1.x * d2.x < 0 || d1.y * d2.y < 0) return -1;

  // The correct ordering of 3 points on a row is p0-p1-p2.
  if (dist2(d1) < dist2(d2)) return +1;

  // If all three points coincide return 0.
  return 0;
}
```

Listing 6.11: isect test.cpp — 2e8e3cdf

14 lines
a, c, 7, d, b, 4, c, 9,
1, e, 9, 5, 0, a, 5, 0

```
template <class P>
bool isect_test( P p1, P p2, P q1, P q2) {
  int c11, c12, c21, c22;

  c11 = ccw( p1, p2, q1 );
  c12 = ccw( p1, p2, q2 );
  c21 = ccw( q1, q2, p1 );
  c22 = ccw( q1, q2, p2 );

  if( c11*c12<=0 && c21*c22<=0 )
    return true;

  return c11*c12*c21*c22==0;
}
```

Triangles

Listing 6.12: heron.cpp — 30de3a30

8 lines
a, 6, 8, a, f, 9, 9, 0,
0, f, e, 9, b, e, c, 0

```
#include <cmath>

double heron(double a, double b, double c) {
  double s=(a+b+c)/2;
  return sqrt(s*(s-a)*(s-b)*(s-c));
}

template <class P> double heron(P A, P B, P C) {
  return heron(dist(B-C), dist(C-A), dist(A-B));
}
```

Listing 6.13: incircle.cpp — c98ccf9e

31 lines
18, e, 0, 4, 4,1e, d, 2,
17, 0,1e,1b, 3, e, 8, c

```
template <class P>
double incircle(P A, P B, P C, P D) {
  typedef typename P::coord_type T;
  P a = A - D; T a2 = dist2(a);
  P b = B - D; T b2 = dist2(b);
  P c = C - D; T c2 = dist2(c);
  return (a2 * cross(b, c) +
          b2 * cross(c, a) +
          c2 * cross(a, b));
}

template <class P, class R>
bool enclosing_centre(P A, P B, P C, R &p, double eps = 1e-13) {
  typedef typename R::coord_type T;
  P a = A - C, b = B - C;
  T det2 = cross(a, b) * 2;
  if (-eps < det2 && det2 < eps) return false;
  T a2 = dist2(a), b2 = dist2(b);
  p.x = (b.y * a2 - a.y * b2) / det2 + C.x;
  p.y = (a.x * b2 - b.x * a2) / det2 + C.y;
  return true;
}

#include "heron.cpp"
template <class P, class T>
bool enclosing_radius(P A, P B, P C, T &r, T eps = 1e-13) {
  T a = dist(B-C), b = dist(C-A), c = dist(A-B);
  T K4 = heron(a, b, c) * 4;
  if (K4 < eps) return false;
  r = a * b * c / K4;
  return true;
}
```

Polygons

Listing 6.14: inside.cpp — 86d649df

11 lines
d, 2, 2, 4, 6, e, d, 2,
d, f, 1, 3, 6, 8, 7, 2

```
template<class P, class V> // V is vector/array of point<T>s
bool inside(const V &p, int n, P t) {
  bool c = false;

  for (int i=0, j=n-1; i<n; j=i++) {
    if ( ((p[i].y<=t.y && t.y<p[j].y) || (p[j].y<=t.y && t.y<p[i].y)) &&
         (dx(p[i],t) < dx(p[i],p[j])*dy(p[i],t)/dy(p[i],p[j])) )
      c = !c;
  }
  return c;
}
```

Listing 6.15: winding number.cpp — 9269969

31 lines
14, 6, 9, 7,16, e,13,15,
16,1c,1b, 3, f,14,14, 2

```
#include "pointline.cpp"

template<class V, class P>
double winding_nr(const V &p, int n, const P &t, bool &onEdge) {
  double wind = 0;

  onEdge = false;

  for (int i=0, j=n-1; i<n; j=i++) {
    if( onsegment(p[i], p[j], t) ) {
      onEdge = true;
      continue;
    }
    double t1 = theta(t-p[i]), t2 = theta(t-p[j]);
    double dt = t1-t2;
    if( dt > 2 ) dt -= 4;
    if( dt < -2 ) dt += 4;
    wind += dt;
  }

  return wind;
}

template<class P, class V> // V is vector/array of point<T>s
int inside_wn(const V &p, int n, P t) {
  bool edge;
  double wind = winding_nr(p,n,t, edge);

  if( edge ) return wind>4 ? 1:0;

  // Not on edge, i.e. wind is (approx) 4*nr of turns.
  return wind > 2 ? 1:-1;
}
```

Listing 6.16: poly area.cpp — 4e8c340c

7 lines
5, 2, 4, 6, 7, 0, 1, 7,
7, 7, 6, 7, 3, 5, 2, 0

```
#include "point.cpp"

template <class V>
double poly_area(V p, int n) {
  typename V::value_type::coord_type a = 0;
  for (int i = 0, j = n - 1; i < n; j = i++)
    a += cross(p[j], p[i]);
  return (double) a / 2;
}
```

Listing 6.17: poly area too.cpp — a1404216

11 lines
f, 0, 5, a, 0, b, f, 5,
7, c, c, 6, e, a, 0, 2

```
#include <iterator>
#include "geometry.h"


template <class V>
double poly_area(V p, int n) {
  int j = n - 1;
  typename iterator_traits<V>::value_type::coord_type a = 0;
  for (int i = 0; i < n; i++) {
    a += (p[j].x - p[i].x) * (p[j].y + p[i].y);
    j = i;
  }
  return (double) a / 2;
}
```

Listing 6.18: poly volume.cpp — 991a53ca

7 lines
6, 0, 4, 5, 3, 6, 0, 0,
2, 0, 5, 1, 6, 1, 6, 4

```
template <class V, class L>
double poly_volume(const V &p, const L &trilist) {
  typename L::value_type::coord_type v = 0;
  for (typename L::const_iterator i = trilist.begin(); i != \
trilist.end(); ++i)
    v += dot(cross(p[i->a], p[i->b]), p[i->c]);
  return (double) v / 6;
}
```

Listing 6.19: poly cut.cpp — 83e56b0f

17 lines
1e, 1,16,12,17,18,1f, d,
8,17,1f,11,11,1f,1f, 1

```
#include "line_isect.cpp"

template <class CI, class OI, class P>
OI poly_cut(CI first, CI last, P p0, P p1, OI result) {
  if (first == last) return result;
  P p = p1-p0;
```

```
  CI j = last; --j;
  bool pside = cross(p, *j-p0) > 0;
  for (CI i = first; i != last; ++i) {
    bool side = cross(p, *i-p0) > 0;
    if (pside ^ side)
      line_isect(p0, p1, *i, *j, *result++);
    if (side)
      *result++ = *i;
    j = i; pside = side;
  }
  return result;
}
```

Listing 6.20: center of mass.cpp — 94bbda10

41 lines
2c,2c,3c,32, b,3d,38,17,
39,13,3d,22,36, 4,21,2b

```
template <class V>
inline double tri_area(V p) { // cross-product / 2
  return ((double)dx(p[0],p[1])*dy(p[0],p[2])-
        (double)dy(p[0],p[1])*dx(p[0],p[2]))/2;
}

template <class V>
void centerofmass( V p, int n, point<double> &com ) {
  com.x = com.y = 0.0;

  if( n<=3 ) {
    // Simple case
    for( int i=0; i<n; i++ ) {
      com.x += p[i].x;
      com.y += p[i].y;
    }
    com.x /= n;
    com.y /= n;
  } else {
    // More difficult case (NB! poly must be in ccw order!)
    typedef typename iterator_traits<V>::value_type::coord_type \
T;
    point<T> tri[3];

    tri[0] = p[0];

    double totarea=0.0, area;
    point<double> tri_com;
    for( int i=2; i<n; i++ ) {
      tri[1] = p[i-1];
      tri[2] = p[i];
      area = tri_area( tri ); // (with orientation)

      centerofmass( tri, 3, tri_com );
      com.x += area*tri_com.x;
      com.y += area*tri_com.y;
      totarea += area<0 ? -area:area;
    }
    com.x /= totarea;
    com.y /= totarea;
  }
}
```

Hull

Listing 6.21: convex hull.cpp — 9530d9a8

35 lines
3a,30,13,1b, b,3e,10,37,
 0, f,34,37,20,3e,35,26

```
template <class P>
struct cross_dist_comparator {
  P o; cross_dist_comparator(P _o) : o(_o) { }
  bool operator ()(const P &p, const P &q) const {
    typename P::coord_type c = cross(p-o, q-o);
    return c != 0 ? c > 0 : dist2(p-o) > dist2(q-o);
  }
};

template <class It>
It convex_hull(It begin, It end) {
  typedef typename iterator_traits<It>::value_type P;
  // zero, one or two points always form a hull
  if (end - begin < 3) return end;
  // find a guaranteed hull point, sort in scan order around it
  swap(*begin, *min_element(begin, end));
  cross_dist_comparator<P> comp(*begin);
  sort(begin + 1, end, comp);
  // colinear points on the first line of the hull must be reversed
  It i = begin + 1;
  for (It j = i++; i != end; j = i++)
    if (cross(*i-*begin, *j-*begin) != 0)
      break;
  reverse(begin + 1, i);
  // place hull points first by doing a Graham scan
  It r = begin + 2;
  for (It i = begin + 3; i != end; ++i) {
    // change < 0 to <= 0 if colinear points on the hull are not desired
    while (cross(*r-*(r-1), *i-*(r-1)) < 0)
      --r;
    swap(*++r, *i);
  }
  // return the iterator past the last hull point
  return ++r;
}
```

Listing 6.22: convex hull idx.cpp — 18989d8d

32 lines
29, e,28,34,2e,3b,37, f,
17,25,3d,32,34,2b,3d,1d

```
// Define an ordering on the points given by their angle
template<class P>
struct ch_sweep {
  P &p;
  ch_sweep(P &_p) : p(_p) {}
  ch_sweep(const ch_sweep<P> &x) : p(x.p) {}

  bool operator()(const P &p1, const P &p2) const
  { return 0 < ccw(p, p1, p2); }
};

// V should be RandomAccessIterator to point<T>s.
// R should be RandomAccessIterator to ints.
template <class V, class R>
int convex_hull(V p, R idx, int n) {
  typedef typename iterator_traits<V>::value_type P;

  // Find bottom-left point
  int i, m = 0;
  for (i = 1; i < n; i++)
    if (p[i] < p[m])
      m = i;

  isort(p, n, idx, ch_sweep<P>(p[m]));

  int r = 3;
  indexed<V, R> q(p, idx);

  for (i = 3; i < n; i++) {
    while (ccw(q[r - 2], q[r - 1], q[i]) < 0)
      r--;
    idx[r++] = idx[i];
  }
  return r;
}
```

Listing 6.23: convex hull robust idx.cpp — 9e4b3ac9

54 lines
23,2d, f, e,33,10,1e,31,
36,3b,1a,3f,27, e,20, 2

```
template <class T>
int ccw_simple(point<T> p0, point<T> p1, point<T> p2) {
  T d = dx(p0,p1)*dy(p0,p2)-dy(p0,p1)*dx(p0,p2);

  return d!=0 ? d>0 ? 1:-1:0;
}

// V should be RandomAccessIterator to point<T>s.
// R should be RandomAccessIterator to ints.
template <class V, class R, class T>
void sort_ccw(V p, R idx, int n, const point<T> &center ) {
}


// V should be RandomAccessIterator to point<T>s.
// R should be RandomAccessIterator to ints.
template <class V, class R>
int convex_hull_robust(V p, R idx, int n) {
  typedef typename iterator_traits<V>::value_type P;
  P center = P();

  // Take an arbitrary point that is *strictly* inside the hull.
  int m = 0;
  for( int i=0; i<n; i++ ) {
    center.x += p[i].x, center.y += p[i].y;
    if( p[i] < p[m] )
      m = i;
  }
  center.x /= n, center.y /= n;


  /* Sofisticated total ordering of points:
   *
```

```
   * Sort first on angle to startpoint (m), then on angle to \
center and
   * finally if the points are on the line m-center, sort on \
distance.
   */
  vector< pair<double,double> > angles;
  angles.reserve( n );

  for( int i=0; i<n; i++ ) {
    if( p[m].x == p[i].x && p[m].y == p[i].y )
      angles.push_back( make_pair(-100,0) );
    else if( ccw_simple(p[m],center,p[i]) == 0 )
      angles.push_back( make_pair(angle(p[i]-p[m]), dist2(p[i]-p[ \
m])) );
    else
      angles.push_back( make_pair(angle(p[i]-p[m]), angle(p[i]- \
center)) );
  }

  isort( angles.begin(), n, idx );


  int r = 3;
  indexed<V, R> q(p, idx);

  // Change <0 to <=0 if colinear points on the hull are not \
desired.
  for (int i = 3; i < n; i++) {
    while (ccw_simple(q[r - 2], q[r - 1], q[i]) < 0)
      r--;
    idx[r++] = idx[i];
  }
  return r;
}
```

Listing 6.24: convex hull space.cpp — 888aa0cb

50 lines
3,15, 1,10, e,1a, 0,32,
29,1c,3b,21,16,2c, 9,18

```
#include <set>

struct ABC {
  int a, b, c; ABC(int _a, int _b, int _c) : a(_a), b(_b), c(_c) { }
  bool operator<(const ABC &o) const {
    return a!=o.a ? a<o.a : b!=o.b ? b<o.b : c<o.c;
  }
};

template <class V, class L>
bool convex_hull_space(V p, int n, L &trilist) {
  typedef typename V::value_type P3;
  typedef typename P3::coord_type T;
  typedef typename L::value_type I3;
  int a, b, c; // Find a proper tetrahedron
  for (a = 1; a < n; ++a) if (dist2(p[a]-p[0]) != T()) break;
  for (b = a + 1; b < n; ++b) if (dist2(cross(p[a]-p[0],p[b]-p[0]))) break;
  for (c = b + 1; c < n; ++c) if (dot(cross(p[a]-p[0],p[b]-p[0]), p[c]-p[0])
                                  != T()) break;
  if (c >= n) return false;
  if (dot(cross(p[a]-p[0],p[b]-p[0]), p[c]-p[0]) > T()) swap(a, b);
  trilist.push_back(I3(0, a, b)); // Use it as initial hull
  trilist.push_back(I3(0, b, c));
  trilist.push_back(I3(0, c, a));
  trilist.push_back(I3(a, c, b));
  for (int i = 1; i < n; ++i) {
    typedef pair<int, int> I2;
    set< pair<int, int> > edges;
    P3 &P = p[i];
    {
      typename L::iterator it = trilist.begin();
      while (it != trilist.end()) {
        int a = it->a, b = it->b, c = it->c;
        P3 &A = p[a], &B = p[b], &C = p[c];
        P3 normal = cross(B-A, C-A);
        T d = dot(normal, P-A);
        if (d > T()) {
          edges.insert(make_pair(a, b));
          edges.insert(make_pair(b, c));
          edges.insert(make_pair(c, a));
          trilist.erase(it++); // ugly!!
        }
        else
          ++it;
      }
    }
    for (set<I2>::iterator it = edges.begin(); it != edges.end(); ++it)
      if (edges.count(make_pair(it->second, it->first)) == 0)
        trilist.push_back(I3(i, it->first, it->second));
  }
  return true;
}
```

Listing 6.25: inside hull.cpp — 2b31bd9c

27 lines
1f,15,1d,13,1c, d,14,1c,
13,14,15,1e,1f, 9,1b,1e

```
#include "../geometry.h.cpp"
#include "../pointline.cpp"

template <class V, class T>
int inside_hull_sub(const V &p, int n, const point<T> &t, int i1, int i2) {
  if (i2 - i1 <= 2) {
    int s0 = sideof(p[0], p[i1], t);
    int s1 = sideof(p[i1], p[i2], t);
    int s2 = sideof(p[i2], p[0], t);
    if (s0 < 0 || s1 < 0 || s2 < 0)
      return -1;
    if (i1 == 1 && s0 == 0 || s1 == 0 || i2 == n - 1 && s2 == 0)
      return 0;
    return 1;
  }
  int i = (i1 + i2) / 2;
  int side = sideof(p[0], p[i], t);
  if (side > 0)
    return inside_hull_sub(p, n, t, i, i2);
  else
    return inside_hull_sub(p, n, t, i1, i);
}

template <class V, class T>
int inside_hull(const V &p, int n, const point<T> &t) {
  if (n < 3)
```

```
    return onsegment(p[0], p[n - 1], t) ? 0 : -1;
  else
    return inside_hull_sub(p, n, t, 1, n - 1);
}
```

Listing 6.26: inside hull simple.cpp — 66d97f4f

27 lines
12, 3, 0, 0, 0,1e,10, 8,
c,1b,10,19,1d, b,16, f

```
// If the hull only consist of non-colinear points the \
degenerated-hull-check
// can be replaced with a onsegment-call if end-begin==2.

template <class It, class T>
int inside_hull_simple(It begin, It end, const point<T> &t) {
  bool on_edge = false;

  point<T> p, q; // degenerated hulls
  p = q = *begin; //

  for( It i=begin, j=end-1; i!=end; j=i++ ) {
    T d = cross(*i-*j,t-*j);
    if( d<0 )
      return -1;
    if( d==0 ) on_edge = true;

    p.x = min(p.x,i->x); // degenerated hulls
    p.y = min(p.y,i->y); //
    q.x = max(q.x,i->x); //
    q.y = max(q.y,i->y); //
  }

  // Extra check for degenerated hulls
  if( on_edge ) {
    if( t.x<p.x || t.x>q.x || t.y<p.y || t.y>q.y )
      return -1;
  }

  return on_edge ? 0:1;
}
```

Listing 6.27: hull diameter.cpp — dd7b6621

21 lines
8,10,15,13,12,18,14, d,
12, 0, 9,19, a, 8, 8,1c

```
#include "../point_ops.cpp"

template <class V>
double hull_diameter2(const V &p, int n, int &i1, int &i2) {
  typedef typename V::value_type::coord_type T;
  if (n < 2) { i1 = i2 = 0; return 0; }
  T m = 0;
  int i, j = 1, k = 0;
  // wander around
  for (i = 0; i <= k; i++) {
    // find opposite
    T d2 = dist2(p[j]-p[i]);
    while (j + 1 < n) {
      T t = dist2(p[j+1]-p[i]);
      if (t > d2) d2 = t; else break;
      j++;
    }
    if (i == 0) k = j; // remember first opposite index
    if (d2 > m) m = d2, i1 = i, i2 = j;
  }
  // cout << "first opposite: " << k << endl;
  return m;
}
```

Listing 6.28: mec.cpp — 882a92be

22 lines
1f,11, 3,1d, c,1f,11, 7,
c, f, 0,1c, 2,1b,1c, b

```
#include "hull_diameter.cpp"
#include "../incircle.cpp"

template <class V, class P>
bool mec(V p, int n, P &c, int &i1, int &i2, int &i3, double eps = 1e-13) {
  typedef typename P::coord_type T;
  hull_diameter2(p, n, i1, i2);
  c = (p[i1] + p[i2]) / 2;
  T r2 = dist2(c, p[i1]);
  bool f = false;
  for (int i = 0; i < n; ++i)
    if (dist2(c, p[i]) > r2) {
      i3 = i, f = true;
      enclosing_centre(p[i1], p[i2], p[i3], c, eps);
      r2 = dist2(c, p[i]);
    }
  return f;
}

template <class V, class P>
double mec(V p, int n, P &c, double eps = 1e-13) {
  int i1, i2, i3;
  mec(p, n, c, i1, i2, i3, eps);
  return dist(c, p[i1]);
}
```

Listing 6.29: line hull intersect.cpp — aa10575f

52 lines
17,1d,32, b,3a,3d,1c,11,
10,16,3c, d,22,27,37,2e

```
#include "../point.cpp"
#include "../geometry.h.cpp"
#include "../pointline.cpp"

template <class V, class T>
struct line_hull_isct {
  const V &p;
  int n;
  const point<T> &p1, &p2;
  int &s1, &s2;
  line_hull_isct(const V &_p, int _n, const point<T> &_p1, const point<T> &_p2,
                 int &_s1, int &_s2)
    : p(_p), n(_n), p1(_p1), p2(_p2), s1(_s1), s2(_s2) {
```

```
  }

// assumes 0 <= md <= i1d, i2d
  bool isct(int i1, int m, int i2, double md) {
    if (md <= 0) {
      s1 = findisct(i1, m) % n;
      s2 = findisct(i2, m) % n;
      return true;
    }
    if( i2-i1 <= 2 )
      return false;
    int l = (i1 + m) / 2;
    int r = (m + i2) / 2;
    double ld = linedist(p1, p2, p[l % n]);
    double rd = linedist(p1, p2, p[r % n]);
    if (ld <= md && ld <= rd)
      return isct(i1, l, m, ld);
    if (rd <= md && rd <= ld)
      return isct(m, r, i2, rd);
    else
      return isct(l, m, r, md);
  }
  int findisct(int pos, int neg) {
    int m = (pos + neg) / 2;
    if (m == pos) return pos;
    if (m == neg) return neg;
    double d = linedist(p1, p2, p[m % n]);
    if (d <= 0)
      return findisct(pos, m);
    else
      return findisct(m, neg);
  }
};

template <class V, class T>
bool line_hull_intersect(const V &p, int n,
                         const point<T> &p1, const point<T> &p2,
                         int &s1, int &s2) {
  double d = linedist(p1, p2, p[0]);
  if (d >= 0)
    return line_hull_isct<V, T>(p, n, p1, p2, s1, s2).isct(0, n, 2 * n, d);
  else
    return line_hull_isct<V, T>(p, n, p2, p1, s1, s2).isct(0, n, 2 * n, -d);
}
```

Voronoi

Listing 6.30: delaunay simple.cpp — 4e999436

26 lines
15,1e, 2, c, 5,1c, 2,1e,
13, 0, c, b, a, b, f,17

```
#include "../point.cpp"

template <class V, class F>
void delaunay(V p, int n, F trifun) {
  typedef typename V::value_type P;
  typedef typename P::coord_type T;
  for (int i = 0; i < n; ++i) {
    for (int j = i + 1; j < n; ++j) {
      P J = p[j] - p[i]; T jd = dist2(J);
      for (int k = i + 1; (j != k || ++k) && k < n; ++k) {
        P K = p[k] - p[i]; T kd = dist2(K);
        T qd = cross(J,K);
        if (qd > T()) {
          P q = P(J.y*kd - K.y*jd, jd*K.x - kd*J.x);
          bool flag = true;
          for (int l = 0; l < n; ++l) {
            P L = p[l] - p[i]; T dl = dist2(L);
            if (dot(L, q) + dl * qd < T()) {
              flag = false;
              break;
            }
          }
          if (flag) trifun(i, j, k);
        }
      }
    }
  }
}
```

Listing 6.31: delaunay hull.cpp — 28a87fe3

14 lines
9, a, 0, b, 5, d, b, a,
e, e, 4, c, 2, c, 7, 7

```
#include <vector>
#include <list>

#include "../point3.cpp"
#include "../hull/convex_hull_space.cpp"

template <class V, class F>
void delaunay(V &p, int n, F trifun) {
  typedef point3<typename V::value_type::coord_type> P3;
  typedef vector<P3> V3;
  typedef list<ABC> L;
  V3 p3(n);
  for (int i = 0; i < n; ++i)
    p3[i] = P3(p[i].x, p[i].y, dist2(p[i]));
  L l;
  convex_hull_space(p3, n, l);
  for (L::iterator it = l.begin(); it != l.end(); ++it)
    if (dot(cross(p3[it->b]-p3[it->a], p3[it->c]-p3[it->a]), P3(0, 0, 1)) < 0)
      trifun(it->a, it->c, it->b); // triangles are turned!
```

```
}
```

Closest pair

Listing 6.32: closest pair.cpp — 18d9f684

99 lines
67,29,3d, f,57,7b,51,51,
79,1d,7a,59,20,6b,53,64

```cpp
#include <iterator>
#include <vector>

struct x_sort {
  template<class P>
  bool operator()(const P &p1, const P &p2) const
  { return p1.x < p2.x; }
};
struct y_sort {
  template<class P>
  bool operator()(const P &p1, const P &p2) const
  { return p1.x < p2.x; }
};

// Gives square distance of closest pair.
template<class V, class R>
double closestpair_sub(const V &p, int n, R xa, R ya, int &i1, int &i2) {
  typedef typename iterator_traits<V>::value_type P;
  vector< int > lefty, righty;

  // 2 or 3 points
  if( n <= 3 ) {
    // Largest dist is either between the two farthest in x or y.
    double a = dist2( p[xa[1]]-p[xa[0]] );
    if( n == 3 ) {
      double b = dist2( p[xa[2]]-p[xa[0]] );
      double c = dist2( p[xa[2]]-p[xa[1]] );

      return min(a,min(b,c));
    } else
      return a;
  }

  // Divide
  int split = n/2;
  double splitx = p[xa[split]].x;

  for( int i=0; i<n; i++ ) {
    if( p[ya[i]].x < splitx )
      lefty.push_back( ya[i] );
    else
      righty.push_back( ya[i] );
  }

  // Conquer
  int j1,j2;
  double a = closestpair_sub( p, split, xa, lefty.begin(), i1, i2 );
  double b = closestpair_sub( p, n-split, xa+split, righty.begin(), j1, j2 );

  if( b<a ) a = b, i1=j1, i2=j2;

  // Combine: Create strip (with sorted y)
  vector<int> stripy;

  for( int i=0; i<n; i++ ) {
    double x = p[ya[i]].x;

    if( x >= splitx-a && x <= splitx+a )
      stripy.push_back( ya[i] );
  }

  int nStrip = stripy.size();
  double a2 = a*a;

  // cout << "Combining " << nStrip << " points...";
  // cout.flush();

  for( int i=0; i<nStrip; i++ ) {
    P &p1 = p[stripy[i]];

    for( int j=i+1; j<nStrip; j++ ) { // This loop will be run <8 times/"i"
      P &p2 = p[stripy[j]];

      if( dy(p1,p2) > a )
        break;

      double d2 = dist2(p2-p1);
      if( d2<a2 ) {
        i1 = stripy[i];
        i2 = stripy[j];
        a2 = d2;
      }
    }
  }

  // cout << " done" << endl;
  return sqrt(a2);
}

template<class V> // R is random access iterators of point<T>s
double closestpair( const V &p, int n, int &i1, int &i2 ) {
  vector< int > xa, ya;

  if( n < 2 )
    throw "closestpair called with less than 2 points";

  xa.resize( n );
  ya.resize( n );
  isort( p, n, xa.begin(), x_sort() );
  isort( p, n, ya.begin(), y_sort() );

  return closestpair_sub( p, n, xa.begin(), ya.begin(), i1, i2 );
}
```

Listing 6.33: closest pair simple.cpp — c806b04b

32 lines
38, 7,26,31,2c,20,2b, e,
10,2f,24,31,28,30,26,12

```
template<class R> // R is random access iterators of point<T> \
s
double closestpair_simple( R p, int n, int &i1, int &i2 ) {
  typedef typename iterator_traits<R>::value_type P;
  vector< int > idx;

  if( n < 2 )
    throw "closestpair called with less than 2 points";

  // Sort points "naturally" (i.e. first after x then after y)
  idx.resize( n );
  isort( p, n, idx.begin() );

  indexed<R, vector<int>::iterator > q(p, idx.begin() );

  double minDist = dist2(q[1]-q[0]);
  i1 = 0; i2 = 1;
  for( int i=0; i<N; i++ ) {
    double stopX = q[i].x+sqrt(minDist);
    for( int j=i+1; j<N; j++ ) {
      if( q[j].x >= stopX )
        break;
      double d = dist2(q[j]-q[i]);
      if( d<minDist ) {
        i1 = i;
        i2 = j;
        minDist = d;
      }
    }
  }

  return sqrt(minDist);
}
```

Chapter 7

Pattern

7.1 String Matching

7.1.1 Knuth-Morrison-Pratt

Listing – kmp.cpp, p. 107

7.1.2 Regular expressions

Listing – regexp.cpp, p. 107

Usage
```
NFA n = parseRegExp("(abc|def)*");
char* s = ...;
size_t match = n.match(s);
```

`RegExp::match(string::iterator s)` returns the length of the longest string beginning in `s` which matches the regexp, or `string::npos` if there is no string beginning in `s` which matches the regexp.

The supported regexp constructions are

- `a`, where `a` is any non-special (like `*` for instance) character, matches `a` exactly.
- `R*`, where `R` is a regexp, is the Kleene closure.
- `R1|R2`, where `R1` and `R2` are regexps, is concatenation.
- `(R)`, where `R` is a regexp, is the grouping operation.
- `[aX]`, where `a` is any character ≠ `]` and `X` is a sequence of characters ≠ `]`, is equivalent to `a|[X]`.
- `"X"`, where `X` is a sequence of characters ≠ `"`, matches the literal string `X` exactly.

Something you should know is that operator priority is not implemented. Therefore, the regexp `abc` is interpreted as `(abc)*`, `abc|def*` as `abc|((def))`, and `(mupp)*(abc|def)*` as `((mupp)*(abc|def))*`.

Any regexp `R` of the form `X(Y`, where `Y` is a correctly parenthesized regexp and `X` is an arbitrary sequency of characters, will be interpreted as the regexp `X(Y)`. That is: missing right parenthesis will be "added" to the end of the string.

Any regexp `R` of the form `X)Y`, where `X` is a correctly parenthesized regexp and `Y` is an arbitrary sequence of characters, will be interpreted as the regexp `X`. That is: the first inbalanced parenthesis will be considered the end of string.

7.2 Automata

7.2.1 NFA

Listing – nfa.cpp, p. 108

Note! The primary use of the NFA in its current state is for RegExp matching.

7.2.2 DFA

Listing – dfa.cpp, p. 108

A DFA class implementation. Biggest feature is the possibility to create a DFA from an NFA, which provides for faster matching for automata that will run several times.

No DFA compression is made. This means, for example, that the regexp `(a|b|c|d|e|f)*` will result in a DFA with seven states, occupying roughly $7 \cdot 256$ bytes of memory, (whereas the optimal number of states is 1, occupying roughly 256 bytes of memory). The equivalent regexp `[abcdef]*` will result in a DFA of 2 states.

7.3 Sequences

7.3.1 Longest Increasing Subsequence

Listing – lis.cpp, p. 110

Usage `iterator last = long_inc_seq(begin, end, cmp);`

Complexity $\mathcal{O}\left(n^2\right)$

Note! The length of the subsequence is `last - begin`, and the actual values are stored in [`begin`, `last`). Alternatively, `cmp` may be omitted, in which case the standard less comparator will be used.

Valladolid 10131 (tested), 231, 497

String Matching

Listing 7.1: kmp.cpp — ddea6fcf

31 lines
6,1a, 3,1d,18, 5, 2,10,
1b,1c, 4, c, e,15, e, 9

```
template<class S, class T>
int kmp( S str, T p ) {
  vector<int> prefix;
  int       m;

  for( m = 0; p[m]; m++ )
    ;

  // Compute prefix-function
  prefix.resize(m);
  prefix[0] = -1;
  for( int i=1, k=-1; i<m; i++ ) {
    while( k>=0 && p[k+1] != p[i] )
      k = prefix[k];
    if( p[k+1] == p[i] )
      k++;
    prefix[i] = k;
  }

  // Match string
  for( int i=0, k=-1; str[i]; i++ ) {
    while( k>=0 && p[k+1] != str[i] )
      k = prefix[k];
    if( p[k+1] == str[i] )
      k++;
    if( k==m-1 )
      return i-(m-1);
  }

  return -1;
}
```

Listing 7.2: regexp.cpp — 4f68019d

47 lines
3,22,32,3a,2e,26,26, 3,
33,38,30,2b,3c, 3,18, f

```
#include "nfa.cpp"

int parseRegExp(NFA& nfa, char* s, int entry);

NFA parseRegexp(const char* s) {
  NFA res;
  res.newState();
  res.accepting[parseRegExp(res, s, 0)] = true;
  return res;
}


int parseRegExp(NFA& nfa, const char* s, int entry) {
  int res = entry;
  char literal = 0;
  while (*i && *i != ')') {
    int oldRes = res;
    switch (*i | (*i == '"' ? 0 : literal)) {
    case '"':
      literal = ~literal;
      break;
    case '(':
      res = parseRegExp(++i, res);
      break;
    case '|': {
      int altEnd = parseRegExp(++i, entry);
      res = nfa.newState();
      nfa.newTransition(oldRes, res, 0);
      nfa.newTransition(altEnd, res, 0);
      --i;
      break;
    }
    case '*':
      nfa.newTransition(res, entry, 0);
      res = entry;
      break;
    case '[':
      res = nfa.newState();
      while (*++i != ']' && *i)
        nfa.newTransition(oldRes, res, *i);
      break;
    default:
      res = nfa.newState();
      nfa.newTransition(oldRes, res, (unsigned char)*i);
    }
    if (*i) ++i;
  }
  return res;
}
```

Automata

Listing 7.3: dfa.cpp — 830c4b59

60 lines
1f, 8,3e,1d,34,16, e,21,
8,25,23,2f,13, f,3b,3d

```
#include "nfa.cpp" // nfa.cpp includes vector, and other necessities.
#include <map>

struct DFA {
  vector<vi> states;
  vector<bool> accepting;

  DFA() {}
  DFA(const NFA& nfa) {
    map<StateSet, int> stateIdx;
    vector<StateSet> stateSets;
    unsigned int state = 0;

    /* Create initial state */
    StateSet start;
    start.insert(0);
    nfa.epsilonClosure(start);
    stateIdx[start] = 1;
    stateSets.push_back(start);
    states.push_back(vi(256, -1));
    accepting.push_back(false);

    /* Process states as they come */
    while (state < states.size()) {
      StateSet s = stateSets[state];
      /* Is this an accepting state? */
      if (s.count(nfa.accept))
        accepting[state] = true;
      /* Which chars trigger state transitions from this state? */
      for (int i = 1; i < 256; ++i) {
        StateSet nS = nfa.nextStates(s, i);
        if (!nS.empty()) {
          nfa.epsilonClosure(nS);
          int& idx = stateIdx[nS];
          /* Is this a new state? */
          if (idx == 0) {
            states.push_back(vi(256, -1));
            accepting.push_back(false);
            stateSets.push_back(nS);
            idx = states.size();
          }
          /* Add state transition */
          states[state][i] = idx-1;
        }
      }
      ++state;
    }
  }

  size_t match(string::const_iterator s) {
    int state = 0;
    string::const_iterator begin = s;
    size_t lastAccept = string::npos;
    while (state != -1) {
      if (accepting[state])
        lastAccept = distance(begin, s);
      if (!*s)
        break;
      state = states[state][*s++];
    }
    return lastAccept;
  }
};
```

Listing 7.4: nfa.cpp — 73898484

60 lines
34,3e,3b,34,2b,13,34, 2,
29,20,3b,34,23,1c,11,32

```
#include <queue> /* Used in the epsilon closure */
#include <vector>
#include <set>

typedef vector<int> vi;
typedef set<int> StateSet;

struct NFA {
  typedef vector<vi> NFAState;
  vector<NFAState> states;
  vector<bool> accepting;

  NFA() { }

  size_t match(char* s) {
    size_t *lastAccept = string::npos;
    char *i = s;
    StateSet states;
    states.insert(0);
    while (!states.empty()) {
      epsClosure(S);
      for (StateSet::iterator si = states.begin(); si != states.end(); ++si)
        if (accepting[*si])
          lastAccept = i - s;
      if (!*i)
        break;
      states = nextStates(states, *i++);
    }
    return lastAccept;
  }

  int newState() {
    states.push_back(NFAState(256, vi()));
    accepting.push_back(false);
    return states.size() - 1;
  }

  void newTransition(int from, int to, char c) {
    states[from][c].push_back(to);
  }

  void epsClosure(StateSet& S) {
    queue<int> q;
    for (StateSet::iterator i = S.begin(); i != S.end(); ++i)
      q.push(*i);
    while (!q.empty()) {
      vi &epsStates = states[q.front()][0];
      q.pop();
```

```
      for (vi::iterator i = epsStates.begin(); i != epsStates.end(); ++i)
        if (S.insert(*i).second)
          q.push(*i);
    }
  }

  StateSet nextStates(StateSet& S, char c) {
    StateSet res;
    for (StateSet::iterator i = S.begin(); i != S.end(); ++i) {
      vi &nStates = states[*i][(unsigned char)c];
      for (vi::iterator j = nStates.begin(); j != nStates.end(); ++j)
        res.insert(*j);
    }
    return res;
  }
};
```

Longest Increasing Subsequence

Listing 7.5: lis.cpp — 5aea809

25 lines
a,1e, 9, f,17, 9, c, 8,
1c,14,1b, c,1f,19,1d,14

```cpp
#include <functional>

/* It needs to be Random Access Iterator */
template <class It>
It long_inc_seq(It begin, It end) {
  return long_inc_seq(begin, end, less<iterator_traits<It>::value_type>());
}

/* It needs to be Random Access Iterator */
template <class It, class Cmp>
It long_inc_seq(It begin, It end, Cmp cmp) {
  int n = end - begin, max[n], forw[n], best = n-1;
  for (int i = n-1; i >= 0; --i) {
    max[i] = 1;
    forw[i] = -1;
    for (int j = i+1; j < n; ++j)
      // Swap i and j in the call to cmp and negate it, if nondecreasing
      // sequences are wanted rather than just increasing.
      if (max[j] + 1 > max[i] && cmp(*(begin + i), *(begin + j)))
        forw[i] = j, max[i] = max[j] + 1;
    if (max[i] > max[best]) best = i;
  }
  It pos = begin;
  while (best != -1) {
    swap(*pos++, *(begin + best));
    best = forw[best];
  }
  return pos;
}
```

Chapter 8

Games

8.1 Repetitive Asymmetric Games

Repetitive asymmetric games are best solved by recursing backwards from known winning & losing positions.

8.2 Card Games

8.2.1 Poker Hands

Listing – poker hands.cpp, p. 112

Usage `int i = hand_value(int hand[5]); string s = hand_names[i];`

Valladolid 131

Cards are assumed to be integers from 0..51 where 0 is the ace of the first color, 12 the king of the first color, 13 the ace of the second color, and so on. (Note: no distinction is made between two hands of equal value (t.b.a.).)

Listing 8.1: rep asymm.cpp — 26331d14

104 lines
52,5b,4f,63, 8,14, b,62,
6b,7e,1c,5f,7b,57,60,63

```
struct position{
  char win;   // vem som vinner, den som är vid draget (isf 1) eller inte (isf -1)
  mer data som behövs för att beskriva brädet...
  int nMoves;  // hur många valmöjligheter man har kvar
}positions[antal_positioner];


#define QUEUELEN 0x10000

struct{
  ställning
}queue[QUEUELEN];

int pushPtr;
int popPtr;

int pop(*ställning){
  if(popPtr==pushPtr)
    return 0;
  *ställningen=pop...
    popPtr=(popPtr+1)%QUEUELEN;
  return 1;
}

void push(ställning){
  pusha ställningen...
    pushPtr=(pushPtr+1)%QUEUELEN;
}

int generateMoves(ställning, int *moves, int backwards){
  // en funktion som returnerar en lista med drag som går att göra
  // från en given position (pos)

  if(!backwards){
    for(varje drag man kan göra framlänges){
      if(moves)
        moves[nMoves]=den ställningen som uppkommer om man gör det draget;
      nMoves++;
    }
    return nMoves;
  }else{
    for(varje drag man kan göra baklänges){
      if(moves)
        moves[nMoves]=den ställningen som uppkommer om man
                      gör det draget baklänges;
      nMoves++;
    }
    return nMoves;
  }
}

void set((nån typ) ställning, char winning) {
  // en funktion som sätter att en viss ställning är evaluerad till något

  positions[ställning].win=winning;
  push(ställning);
}

void test(){
  // nollställ kön
  pushPtr=0;
  popPtr=0;

  // initiera listan med varje ställning
  for(alla ställningar som finns){
    positions[ställning].willWin=UNKNOWN;

    // antalet drag man kan göra från den ställningen
    positions[ställning].nMoves=generateMoves(ställning, 0, 0);
  }

  // sätt alla positioner där man vinner till vunna
  for(alla ställningar där man vinner){
    set(ställning, 1);
    set(ställning, 1);
  }
  for(alla ställningar där man förlorar){
    set(ställning, -1);
    set(ställning, -1);
  }

  // gå igenom spelträdet
  ställning moves[massa];
  while(pop(&ställning)){
    // processa allt som ligger i kön

    // generera alla platser baklänges
    nMoves=generateMoves(ställning, moves, 1);

    if(positions[ställning].willWin==-1){
      // isåfall kommer ju alla dragen som ledde hit vara vinnande

      for(a=0;a<nMoves;a++)
        if(positions[moves[a]].willWin==UNKNOWN)
          set(moves[a],1);
    }else{
      // man vinner, alltså minskar valmöjligheterna för draget innan.
      // Om den inte har några valmöjligheter kvar förlorar den.

      for(a=0;a<nMoves;a++)
        if( positions[moves[a]].willWin==UNKNOWN &&
            --positions[moves[a]].nMoves==0)
          set(moves[a],-1);
    }
  }
}
```

Listing 8.2: poker hands.cpp — 915c599f

60 lines
19,35,38,17,3c,30,18, f,
39,1e, c, 4,19, f,2f,3a

```
#include <string>

string hand_names[] = {
  "highest-card",
  "one-pair",
  "two-pairs",
  "three-of-a-kind",
  "straight",
  "flush",
  "full-house",
  "four-of-a-kind",
```

```
  "straight-flush"};

inline int color(int card) { return card / 13; }
inline int value(int card) { return card % 13; }

int hand_value(int hand[5]) {
  int pairs = 0, triples = 0, quads = 0;
  int vcnt[13]; // count of each value
  int ccnt[4]; // count of each color
  bool flush = false, straight = true;
  for (int i = 0; i < 13; ++i) vcnt[i] = 0;
  for (int i = 0; i < 4; ++i) ccnt[i] = 0;
  for (int i = 0; i < 5; ++i) {
    int v = ++vcnt[value(hand[i])];
    if (v == 2)    ++pairs;
    else if (v == 3) --pairs, ++triples;
    else if (v == 4) --triples, ++quads;
    if (++ccnt[color(hand[i])] == 5)
      flush = true;
  }
  int began = -1;
  bool rstraight = true; // royal straight is special (value(ace) = 0)
  for (int i = 0; i < 13; ++i)
    if (vcnt[i]) {
      if (began == -1)
        began = i;
      straight &= i < began + 5;
      rstraight &= !began && (!i || i >= 9);
    }
  straight |= rstraight;
  straight &= !pairs && !triples && !quads;
  if (straight && flush) {   // straight flush
    return 8;
  } else if (quads) {         // four of a kind
    return 7;
  } else if (triples && pairs) { // full house
    return 6;
  } else if (flush) {         // flush
    return 5;
  } else if (straight) {      // straight
    return 4;
  } else if (triples) {       // three of a kind
    return 3;
  } else if (pairs > 1) {     // two pairs
    return 2;
  } else if (pairs) {         // one pair
    return 1;
  }
  // no hand
  return 0;
}
```

Chapter 9

Hard problems

9.1 Knapsack

9.1.1 Knapsack

Listing – knapsack.cpp, p. 116

Usage `R res = knapsack<R>(n, C, costs, values [, bound = 500000]);`

Complexity $\mathcal{O}(\min(bound, nC))$

Knapsack heuristic. Returns the maximum value achievable. `n` is the number of objects, `C` is the capacity of the knapsack, `costs` is a random access container with the `n` costs, and `values` is a random access container with the `n` values. `bound` is an approximation factor; lower values of `bound` means shorter running time, but also a greater risk that the algorithm will produce an incorrect answer (if $nC \leq$ `bound` and the costs are integers, the answer is "guaranteed" to be correct). Note that actually, all scaled values (i.e. multiplications with `scale`) should be rounded upwards. However, empirical tests (using the test data from NADA Open 2002) has shown this approach to be less accurate (i.e. requiring a higher bound to produce a correct answer) in practice. For the NADA Open 2002 test cases, a bound of $\approx$ 90000 was sufficient.

Listing 9.1: knapsack.cpp — 7deca420

29 lines
7,16, 4,10,1f,1b,17,1f,
1b, 4, e, d,1c, 1,1c, 7

```cpp
#include <vector>

/* Templates:
 * R is the value type (needs to be constructable from "-1").
 * T is the cost type (needs to be multipliable with doubles).
 * W is a random access container of costs.
 * V is a random access container of values.
 */
template <class R, class T, class W, class V>
R knapsack(int n, const T& C, const W& costs, const V& values,
           int bound=500000) {
  double scale = bound / ((double) n * C);

  // (Usually) no point in scaling upwards... This line should be
  // removed if the costs are all small-valued doubles, in which case
  // it will be very healthy to stretch 'em out a bit.
  if (scale > 1) scale = 1;

  int C_max = (int) (scale * C) + 1;
  R max = R();
  vector<R> val_max(C_max, R(-1));
  val_max[0] = R();

  for (int i = 0; i < n; ++i) {
    int scaled_cost = (int) (scale * costs[i]);
    for (int j = C_max - 1; j >= scaled_cost; --j) {
      R v = val_max[j - scaled_cost];
      if (v != -1 && v + values[i] > val_max[j]) {
        val_max[j] = v + values[i];
        if (val_max[j] > max)
          max = val_max[j];
      }
    }
  }

  return max;
}
```

Chapter 10

Input/Output

10.1 Stream manipulators

10.1.1 Eat whitespace

Usage `cin >> ws;`

10.1.2 Eat line

Listing – manipulators.cpp, p. 118

Usage `cin >> eatline;`

10.1.3 Eat to blank line

Listing – manipulators.cpp, p. 118

Usage `cin >> blankline;`

10.1.4 Print number with words

Listing – manipulators.cpp, p. 118

Usage `cout << expand<int>(-4711);`

10.2 String stream

10.2.1 istringstream

Listing – istringstream.cpp, p. 118

Fixes an `istringstream` if `sstream` is not present.

Listing 10.1: manipulators.cpp — 66f5614a

46 lines
3a,1b, 3,20,31, 0,32,34,
2,2d,11,3c,1f,39,18,26

```
#include <iostream>

istream &eatline(istream &in) {
  while (in && in.get() != '\n');
  return in;
}

istream &blankline(istream &in) {
  char c, lastc=0;
  while (in && ((c = in.get()) != '\n' || lastc != '\n')) lastc = c;
  return in;
}

template <class T, bool Z = true>
struct expand {
  T n;

  expand(T _n) : n(_n) {}
  ostream &operator()(ostream &out) const {
    static char const* units[] = {
      "zero", "one", "two", "three", "four",
      "five", "six", "seven", "eight", "nine",
      "ten", "eleven", "twelve", "thirteen", "fourteen",
      "fifteen", "sixteen", "seventeen", "eighteen", "nineteen"
    };

    static char const* tens[] = {
      "zero", "ten", "twenty", "thirty", "fourty",
      "fifty", "sixty", "seventy", "eighty", "ninety"
    };

    typedef expand<T, false> E;
    if (n < 0) return out << "minus " << E(-n);
    if (n == 0) return Z ? out << units[n] : out;
    if (n < 20) return out << units[n];
    if (n < 100) return out << tens[n/10] << E(n%10);
    if (n < 1000) return out << E(n/100) << "hundred" << E(n%100);
    if (n < T(1e6)) return out << E(n/1000) << "thousand" << E(n%1000);
    if (n < T(1e9)) return out << E(n/T(1e6)) << "million" << E(n%T(1e6));
    return out << E(n/T(1e9)) << "billion" << E(n%T(1e9));
  }
};

template <class T, bool Z>
ostream &operator<<(ostream &out, const expand<T,Z> &x) {
  return x(out);
}
```

Listing 10.2: istringstream.cpp — 7a64259f

3 lines
0, 2, 1, 3, 0, 0, 0, 1,
1, 0, 1, 3, 0, 1, 3, 1

```
#include <strstream>
#include <string>
struct istringstream : istrstream {
  istringstream( const string &s ) : istrstream( s.c_str(), s.length() ) {}
};
```

Listings

Index