Nanyang Technological Univerisity

Orange Juice - ACM-ICPC Cheat Sheet

Basic

C++ Solution Template

```
#include <bits/stdc++.h>

#define GET_BIT(n, i) (((n) & (1 << ((i)-1))) >> ((i)-1)) // i start from 1
#define SET_BIT(n, i) ((n) | (1 << ((i)-1)))
#define CLR_BIT(n, i) ((n) & ~(1 << ((i)-1)))
#define SHOW_A(x) {cout << #x << " = " << x << endl;}
#define SHOW_B(x, y) {cout << #x << " = " << x << ", " << #y << " = " << y << endl;}
#define SHOW_C(x, y, z) {cout << #x << " = " << x << ", " << #y << " = " << y << ", " << #z << " = " << z << endl;}
#define REACH_HERE {cout << "REACH_HERE! line: " << __LINE__ << endl;}
#define ASSERT(x) { _assert(x); cout << #x << endl; }

const double E = 1e-8;
const double PI = acos(-1);

using namespace std;

int test = 0, pass = 0;

void _assert(bool result) {
    test++; pass+=result;
    cout << (result ? "PASS: " : "FAIL: ");
}

int main() {
    ios::sync_with_stdio(false);

    // int x = 0;
    // ASSERT(x==0)
    // ASSERT((x = SET_BIT(x, 3))==4)
    // ASSERT((x = CLR_BIT(x, 3))==0)

    // cout << "Total test: " << test << endl;
    // cout << "Passed: " << pass << " (" << 100.0*pass/test << "%%)" << endl;

    return 0;
}
```

Optional include list

```
#include <iostream>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <climits>
#include <stack>
#include <queue>
#include <vector>
#include <set>
#include <map>
#include <list>
```

C++ String

Input string

get one string with no space and new-line.

```
string a;
cin >> a;
```

read one line

getline()

```
string a;
getline(cin, a);
```

Convert to char array

```
string cppstr = "this is a string";
char target[1024];
strcpy(target, cppstr.c_str());
```

or

```
string hi = "hi";
const char[] = hi.c_str();
```

C String (Character Array)

Input C String

gets()

Reads characters from the standard input (stdin) and stores them as a C string into str until a newline character or the end-of-file is reached.

```
char b[10];
gets(b);
cout << "[C++] You have input \"" << b << "\", "
     << ", whose length is " << strlen(b) << endl;
```

Note: There is a space in front of "world", which will be part of the string. However, using gets() is "unsafe", but we are not to discuss the details here.

Convert to C++ string

```
char arrstr[] = "this is a string";
string target = string(arr);
```

Algorithm

```
#include <algorithm>
```

Permutation

Usage

```
bool next_permutation (BidirectionalIterator first,
BidirectionalIterator last);
bool next_permutation (BidirectionalIterator first,
BidirectionalIterator last, Compare comp);
```

Example

```
#include <iostream>     // std::cout
#include <algorithm>    // std::next_permutation, std::sort

int main () {
  int myints[] = {1,2,3};

  std::sort (myints,myints+3);

  std::cout << "The 3! possible permutations with 3
elements:\n";
  do {
    std::cout << myints[0] << ' ' << myints[1] << ' ' <<
myints[2] << '\n';
  } while ( std::next_permutation(myints,myints+3) );

  std::cout << "After loop: " << myints[0] << ' ' << myints[1]
<< ' ' << myints[2] << '\n';

  return 0;
}
```

Output

```
The 3! possible permutations with 3 elements:
1 2 3
1 3 2
```

```
2 1 3
2 3 1
3 1 2
3 2 1
After loop: 1 2 3
```

Binary Search

Usage

```
bool binary_search (ForwardIterator first, ForwardIterator
last, const T& val, Compare comp);
// return true if found, false if not
```

Lower Bound

Returns an iterator pointing to the first element in the range [first,last) which does not compare less than val.

Usage

```
ForwardIterator lower_bound (ForwardIterator first,
ForwardIterator last,
                             const T& val);
ForwardIterator lower_bound (ForwardIterator first,
ForwardIterator last,
                             const T& val, Compare comp);
```

`binary_search` is defined by:

```
template <class ForwardIterator, class T>
  bool binary_search (ForwardIterator first, ForwardIterator
last, const T& val) {
  first = std::lower_bound(first,last,val);
  return (first!=last && !(val<*first));
}
```

Swap

Usage

```
void swap (T& a, T& b);
void iter_swap (ForwardIterator1 a, ForwardIterator2 b);
```

`iter_swap` example

```
int myints[]={10,20,30,40,50 };              //   myints:  10
20  30  40  50
std::vector<int> myvector (4,99);            // myvector:  99
99  99  99

std::iter_swap(myints,myvector.begin());     //   myints: [99]
20  30  40  50
                                             // myvector: [10]
99  99  99

std::iter_swap(myints+3,myvector.begin()+2); //   myints:  99
20  30 [99] 50
                                             // myvector:  10
99 [40] 99
```

Heap

// not completed

// TODO add method and example to maintain the size of vector

- make_heap: Rearranges the elements in the range [first,last) in such a way that they form a heap. The element with the highest value is always pointed by first.
- pop_heap: Rearranges the elements in the heap range [first,last) in such a way that the part considered a heap is shortened by one: The element with the highest value is moved to (last-1).
- push_heap: Given a heap in the range [first,last-1), this function extends the range considered a heap to [first,last) by

placing the value in (last-1) into its corresponding location within it.

- sort_heap: Sorts the elements in the heap range [first,last) into ascending order.

Usage

```
void make_heap (RandomAccessIterator first,
RandomAccessIterator last, Compare comp);
void pop_heap (RandomAccessIterator first, RandomAccessIterator
last, Compare comp);
void push_heap (RandomAccessIterator first,
RandomAccessIterator last, Compare comp);
void sort_heap (RandomAccessIterator first,
RandomAccessIterator last); Compare comp);
```

Note: Priority queue is more recoomeneded.

Sort

Sorts the elements in the range [first,last) into ascending order. `stable_sort` preserves the relative order of the elements with equivalent values.

Usage

```
void sort (RandomAccessIterator first, RandomAccessIterator
last);
void sort (RandomAccessIterator first, RandomAccessIterator
last, Compare comp);
void stable_sort ( RandomAccessIterator first,
RandomAccessIterator last );
void stable_sort ( RandomAccessIterator first,
RandomAccessIterator last,
                   Compare comp );
```

Compare

bool operator< (ele); true at bottom (for sort, true at front)

Compare function

Binary function that accepts two elements in the range as arguments, and returns a value convertible to bool. It should returns true if the first element is considered to be "smaller" than the second one.

Using by `sort`, `make_heap` and etc.

```
bool myfunction (int i,int j) { return (i<j); }
```

Define operator <()

Member function

recommended // can use for priority_queue, sort,

```
struct Edge {
   int from, to, weight;
  bool operator<(Edge that) const {
        return weight > that.weight;
    }
};
```

verbal version

```
struct Edge {
   int from, to, weight;
  bool operator<(const Edge& that) const {
        return this->weight > that.weight;
    }
};
```

Non-member function

```
struct Edge {
    int from, to, weight;
```

```
    friend bool operator<(Edge a, Edge b) {
        return a.weight > b.weight;
    }
};
```

Define operator()()

You can use comparison function for STL containers by passing them as the first argument of the constructor, and specifying the function type as the additional template argument. For example:

```
set<int, bool (*)(int, int)> s(cmp);
```

A functor, or a function object, is an object that can behave like a function. This is done by defining operator()() of the class. In this case, implement operator()() as a comparison function:

```
vector<int> occurrences;
struct cmp {
    bool operator()(int a, int b) {
        return occurrences[a] < occurrences[b];
    }
};
set<int, cmp> s;
priority_queue<int, vector<int>, cmp> pq;
```

Used by `priority_queue`.

Map

```
#include <map>
```

Define a Map

```
template < class Key,                                     //
map::key_type
           class T,                                       //
map::mapped_type
           class Compare = less<Key>,                     //
map::key_compare
           class Alloc = allocator<pair<const Key,T> >    //
map::allocator_type
           > class map;
```

Commonly used method

```
begin()
end()

empty()
size()

operator[] // if not found, insert one

insert(pair<first type, second type)
erase()
clear()

find() // if not found, return end()
count() // return 1 or 0
```

Red-black Tree

deleted...

Hash Map

Unordered Map

// TODO add interface

Unordered map is implemented as a hash table.

```
template < class Key,                                     //
unordered_map::key_type
           class T,                                       //
unordered_map::mapped_type
```

```
          class Hash = hash<Key>,                    // unordered_map::hasher
          class Pred = equal_to<Key>,                // unordered_map::key_equal
          class Alloc = allocator< pair<const Key,T> >  // unordered_map::allocator_type
          > class unordered_map;
```

Deprecated Hash Map

```
#include <ext/hash_map>
__gnu_cxx::hash_map<string, int> months;
months["january"] = 31;
months["february"] = 28;
```

Pair

Vector

Constructor

```
std::vector<int> second (4,100);  // four ints with value 100
```

Methods

- begin(), end()
- front(), back()
- clear()
- size()
- push_back(const value_type& val)
- pop_back()

List

List containers are implemented as doubly-linked lists.

Methods

- begin(), end()
- front(), back()
- clear()
- push_front(const value_type& val)
- push_back(const value_type& val)
- pop_front(): remove the first element.
- pop_back(): remove the last element.
- remove(const value_type& val): remove all elements of value val.
- insert(iterator position, const value_type& val)
- size()
- reverse()
- sort(), sort (Compare comp)

Queue

```
include <queue>
```

Constructor

```
queue<int> my_queue;
queue<int, list<int> > my_queue (my_list);
// use list<int> as container, copy my_list into my_queue
```

Methods

```
void queue::push(const value_type& val);
void queue::pop();
bool queue::empty() const;
size_type queue::size() const;
const_reference& queue::front() const;
```

Double-ended Queue

```
include <dequeue>
```

Stack

Constructor

```
stack<int, vector<int> > my_stack (my_data);
// use vector<int> as container, copy my_data into my_stack

bool stack::empty() const;
size_type stack::size() const;
const_reference& stack::top() const;
void stack::push (const value_type& val);
void stack::pop();
```

Iterator

Priority Queue

```
// constructor
priority_queue<int> my_priority_queue;
priority_queue<int, vector<int>, greater<int> >
two_priority_queue;
// if use greater<int>, must have vector<int>
priority_queue<My_type, vector<My_type>, Comparator_class>
            my_priority_queue (my_data.begin(), my_data.end());
// use Comparator_class as comparator, use vector<My_type> as
container, copy my_data into my_priority_queue

bool empty() const;
// return true if empty, false if not
size_type size() const;
// return size of queue
const_reference :top() const;
// returns a constant reference to the top element
void push(const value_type& val);
// inserts a new element, initialize to val
void pop();
```

```
// removes the element on top

struct My_type {
    int weight;
    int other;
};

struct My_class_for_compare {
    public:
        bool operator() (My_type a, My_type b) {
            return a.weight < b.weight;
        }
};

vector<My_type> my_vector = {(My_type){2, 789}, (My_type){1,
127}, (My_type){3, 456}};

priority_queue<My_type, vector<My_type>, My_class_for_compare>
            one_priority_queue (my_vector.begin(),
my_vector.end());
one_priority_queue.push((My_type){4, 483});
while (one_priority_queue.size() != 0) {
    My_type temp = one_priority_queue.top();
    one_priority_queue.pop();
    SHOW_B(temp.weight, temp.other);
}

vector<int> my_int = {2, 3, 1};

priority_queue<int, vector<int>, greater<int> >
two_priority_queue (my_int.begin(), my_int.end());
while (two_priority_queue.size() != 0) {
    SHOW_A(two_priority_queue.top());
    two_priority_queue.pop();
}

// output
// temp.weight = 4, temp.other = 483
// temp.weight = 3, temp.other = 456
// temp.weight = 2, temp.other = 789
// temp.weight = 1, temp.other = 127
// two_priority_queue.top() = 1
```

```
// two_priority_queue.top() = 2
// two_priority_queue.top() = 3
```

BigInteger & BigDecimal

C++ Big Integer

```
const int BASE_LENGTH = 2;
const int BASE = (int) pow(10, BASE_LENGTH);
const int MAX_LENGTH = 500;

string int_to_string(int i, int width, bool zero) {
    string res = "";
    while (width--) {
        if (!zero && i == 0) return res;
        res = (char)(i%10 + '0') + res;
        i /= 10;
    }
    return res;
}

struct bigint {
    int len, s[MAX_LENGTH];

    bigint() {
        memset(s, 0, sizeof(s));
        len = 1;
    }

    bigint(unsigned long long num) {
        len = 0;
        while (num >= BASE) {
            s[len] = num % BASE;
            num /= BASE;
            len ++;
        }
        s[len++] = num;
    }

    bigint(const char* num) {
        int l = strlen(num);
        len = l/BASE_LENGTH;
        if (l % BASE_LENGTH) len++;
        int index = 0;
        for (int i = l - 1; i >= 0; i -= BASE_LENGTH) {
            int tmp = 0;
            int k = i - BASE_LENGTH + 1;
            if (k < 0) k = 0;
            for (int j = k; j <= i; j++) {
                tmp = tmp*10 + num[j] - '0';
            }
            s[index++] = tmp;
        }
    }

    void clean() {
        while(len > 1 && !s[len-1]) len--;
    }

    string str() const {
        string ret = "";
        if (len == 1 && !s[0]) return "0";
        for(int i = 0; i < len; i++) {
            if (i == 0) {
                ret += int_to_string(s[len - i - 1],
BASE_LENGTH, false);
            } else {
                ret += int_to_string(s[len - i - 1],
BASE_LENGTH, true);
            }
        }
        return ret;
    }

    unsigned long long ll() const {
        unsigned long long ret = 0;
        for(int i = len-1; i >= 0; i--) {
            ret *= BASE;
            ret += s[i];
        }
        return ret;
    }

    bigint operator + (const bigint& b) const {
        bigint c = b;
        while (c.len < len) c.s[c.len++] = 0;
        c.s[c.len++] = 0;
```

```
        bool r = 0;
        for (int i = 0; i < len || r; i++) {
            c.s[i] += (i<len)*s[i] + r;
            r = c.s[i] >= BASE;
            if (r) c.s[i] -= BASE;
        }
        c.clean();
        return c;
    }

    bigint operator - (const bigint& b) const {
        if (operator < (b)) throw "cannot do subtract";
        bigint c = *this;
        bool r = 0;
        for (int i = 0; i < b.len || r; i++) {
            c.s[i] -= b.s[i];
            r = c.s[i] < 0;
            if (r) c.s[i] += BASE;
        }
        c.clean();
        return c;
    }

    bigint operator * (const bigint& b) const {
        bigint c;
        c.len = len + b.len;
        for(int i = 0; i < len; i++)
            for(int j = 0; j < b.len; j++)
                c.s[i+j] += s[i] * b.s[j];
        for(int i = 0; i < c.len-1; i++){
            c.s[i+1] += c.s[i] / BASE;
            c.s[i] %= BASE;
        }
        c.clean();
        return c;
    }

    bigint operator / (const int b) const {
        bigint ret;
        int down = 0;
        for (int i = len - 1; i >= 0; i--) {
            ret.s[i] = (s[i] + down * BASE) / b;
            down = s[i] + down * BASE - ret.s[i] * b;
        }
        ret.len = len;
        ret.clean();
        return ret;
    }

    bool operator < (const bigint& b) const {
        if (len < b.len) return true;
        else if (len > b.len) return false;
        for (int i = 0; i < len; i++)
            if (s[i] < b.s[i]) return true;
            else if (s[i] > b.s[i]) return false;
        return false;
    }

    bool operator == (const bigint& b) const {
        return !(*this<b) && !(b<(*this));
    }

    bool operator > (const bigint& b) const {
        return b < *this;
    }
};
```

The Java Approach

BigInteger

```
import java. math. BigInteger;
```

Constructor and Description

```
BigInteger(String val)
// translates the decimal String representation of a BigInteger
into a BigInteger.
BigInteger(String val, int radix)
// translates the String representation of a BigInteger in the
specified radix into a BigInteger.

//Modifier and Type   Method and Description
BigInteger  abs()
```

```
// returns a BigInteger whose value is the absolute value of
this BigInteger.

BigInteger  add(BigInteger val)
// returns a BigInteger whose value is (this + val).
BigInteger  subtract(BigInteger val)
BigInteger  multiply(BigInteger val)
BigInteger  divide(BigInteger val)
BigInteger  pow(int exponent)
// returns a BigInteger whose value is (thisexponent).

BigInteger  and(BigInteger val)
// returns a BigInteger whose value is (this & val).
BigInteger  or(BigInteger val)
BigInteger  not()

BigInteger  shiftLeft(int n)
// returns a BigInteger whose value is (this << n).
BigInteger  shiftRight(int n)

BigInteger  mod(BigInteger m)
// returns a BigInteger whose value is (this mod m).
BigInteger  remainder(BigInteger val)
// returns a BigInteger whose value is (this % val).
BigInteger  gcd(BigInteger val)
// returns a BigInteger whose value is the greatest common
divisor of abs(this) and abs(val).

BigInteger  max(BigInteger val)
// returns the maximum of this BigInteger and val.
BigInteger  min(BigInteger val)

int compareTo(BigInteger val)
// compares this BigInteger with the specified BigInteger.
String  toString(int radix)
// returns the String representation of this BigInteger in the
given radix.
static BigInteger   valueOf(long val)
// returns a BigInteger whose value is equal to that of the
specified long.
long    longValue()
// converts this BigInteger to a long.
```

BigDecimal

```
// constructor
BigDecimal(BigInteger val)
// Translates a BigInteger into a BigDecimal.
BigDecimal(BigInteger unscaledVal, int scale)
// Translates a BigInteger unscaled value and an int scale into
a BigDecimal.
BigDecimal(BigInteger unscaledVal, int scale, MathContext mc)
// Translates a BigInteger unscaled value and an int scale into
a BigDecimal, with rounding according to the context settings.
BigDecimal(BigInteger val, MathContext mc)
// Translates a BigInteger into a BigDecimal rounding according
to the context settings.
BigDecimal(char[] in)
// Translates a character array representation of a BigDecimal
into a BigDecimal, accepting the same sequence of characters as
the BigDecimal(String) constructor.
BigDecimal(char[] in, int offset, int len)
// Translates a character array representation of a BigDecimal
into a BigDecimal, accepting the same sequence of characters as
the BigDecimal(String) constructor, while allowing a sub-array
to be specified.
BigDecimal(char[] in, int offset, int len, MathContext mc)
// Translates a character array representation of a BigDecimal
into a BigDecimal, accepting the same sequence of characters as
the BigDecimal(String) constructor, while allowing a sub-array
to be specified and with rounding according to the context
settings.
BigDecimal(char[] in, MathContext mc)
// Translates a character array representation of a BigDecimal
into a BigDecimal, accepting the same sequence of characters as
the BigDecimal(String) constructor and with rounding according
to the context settings.
BigDecimal(double val)
// Translates a double into a BigDecimal which is the exact
decimal representation of the double's binary floating-point
value.
BigDecimal(double val, MathContext mc)
// Translates a double into a BigDecimal, with rounding
according to the context settings.
BigDecimal(int val)
```

```
// Translates an int into a BigDecimal.
BigDecimal(int val, MathContext mc)
// Translates an int into a BigDecimal, with rounding according
to the context settings.
BigDecimal(long val)
// Translates a long into a BigDecimal.
BigDecimal(long val, MathContext mc)
// Translates a long into a BigDecimal, with rounding according
to the context settings.
BigDecimal(String val)
// Translates the string representation of a BigDecimal into a
BigDecimal.
BigDecimal(String val, MathContext mc)
// Translates the string representation of a BigDecimal into a
BigDecimal, accepting the same strings as the
BigDecimal(String) constructor, with rounding according to the
context settings.

BigDecimal  abs()
// Returns a BigDecimal whose value is the absolute value of
this BigDecimal, and whose scale is this.scale().
BigDecimal  abs(MathContext mc)
// Returns a BigDecimal whose value is the absolute value of
this BigDecimal, with rounding according to the context
settings.
BigDecimal  add(BigDecimal augend)
// Returns a BigDecimal whose value is (this + augend), and
whose scale is max(this.scale(), augend.scale()).
BigDecimal  add(BigDecimal augend, MathContext mc)
// Returns a BigDecimal whose value is (this + augend), with
rounding according to the context settings.
byte    byteValueExact()
// Converts this BigDecimal to a byte, checking for lost
information.
int compareTo(BigDecimal val)
// Compares this BigDecimal with the specified BigDecimal.
BigDecimal  divide(BigDecimal divisor)
// Returns a BigDecimal whose value is (this / divisor), and
whose preferred scale is (this.scale() - divisor.scale()); if
the exact quotient cannot be represented (because it has a non-
terminating decimal expansion) an ArithmeticException is
thrown.
```

```
BigDecimal  divide(BigDecimal divisor, int roundingMode)
// Returns a BigDecimal whose value is (this / divisor), and
whose scale is this.scale().
BigDecimal  divide(BigDecimal divisor, int scale, int
roundingMode)
// Returns a BigDecimal whose value is (this / divisor), and
whose scale is as specified.
BigDecimal  divide(BigDecimal divisor, int scale, RoundingMode
roundingMode)
// Returns a BigDecimal whose value is (this / divisor), and
whose scale is as specified.
BigDecimal  divide(BigDecimal divisor, MathContext mc)
// Returns a BigDecimal whose value is (this / divisor), with
rounding according to the context settings.
BigDecimal  divide(BigDecimal divisor, RoundingMode
roundingMode)
// Returns a BigDecimal whose value is (this / divisor), and
whose scale is this.scale().
BigDecimal[]    divideAndRemainder(BigDecimal divisor)
// Returns a two-element BigDecimal array containing the result
of divideToIntegralValue followed by the result of remainder on
the two operands.
BigDecimal[]    divideAndRemainder(BigDecimal divisor,
MathContext mc)
//Returns a two-element BigDecimal array containing the result
of divideToIntegralValue followed by the result of remainder on
the two operands calculated with rounding according to the
context settings.
BigDecimal  divideToIntegralValue(BigDecimal divisor)
// Returns a BigDecimal whose value is the integer part of the
quotient (this / divisor) rounded down.
BigDecimal  divideToIntegralValue(BigDecimal divisor,
MathContext mc)
// Returns a BigDecimal whose value is the integer part of
(this / divisor).
double  doubleValue()
// Converts this BigDecimal to a double.
boolean equals(Object x)
// Compares this BigDecimal with the specified Object for
equality.
float   floatValue()
// Converts this BigDecimal to a float.
```

```
int hashCode()
// Returns the hash code for this BigDecimal.
int intValue()
// Converts this BigDecimal to an int.
int intValueExact()
// Converts this BigDecimal to an int, checking for lost
information.
long    longValue()
// Converts this BigDecimal to a long.
long    longValueExact()
// Converts this BigDecimal to a long, checking for lost
information.
BigDecimal  max(BigDecimal val)
// Returns the maximum of this BigDecimal and val.
BigDecimal  min(BigDecimal val)
// Returns the minimum of this BigDecimal and val.
BigDecimal  movePointLeft(int n)
// Returns a BigDecimal which is equivalent to this one with
the decimal point moved n places to the left.
BigDecimal  movePointRight(int n)
// Returns a BigDecimal which is equivalent to this one with
the decimal point moved n places to the right.
BigDecimal  multiply(BigDecimal multiplicand)
// Returns a BigDecimal whose value is (this × multiplicand),
and whose scale is (this.scale() + multiplicand.scale()).
BigDecimal  multiply(BigDecimal multiplicand, MathContext mc)
// Returns a BigDecimal whose value is (this × multiplicand),
with rounding according to the context settings.
BigDecimal  negate()
// Returns a BigDecimal whose value is (-this), and whose scale
is this.scale().
BigDecimal  negate(MathContext mc)
// Returns a BigDecimal whose value is (-this), with rounding
according to the context settings.
BigDecimal  plus()
// Returns a BigDecimal whose value is (+this), and whose scale
is this.scale().
BigDecimal  plus(MathContext mc)
// Returns a BigDecimal whose value is (+this), with rounding
according to the context settings.
BigDecimal  pow(int n)
// Returns a BigDecimal whose value is (thisn), The power is
computed exactly, to unlimited precision.
BigDecimal  pow(int n, MathContext mc)
// Returns a BigDecimal whose value is (thisn).
int precision()
// Returns the precision of this BigDecimal.
BigDecimal  remainder(BigDecimal divisor)
// Returns a BigDecimal whose value is (this % divisor).
BigDecimal  remainder(BigDecimal divisor, MathContext mc)
// Returns a BigDecimal whose value is (this % divisor), with
rounding according to the context settings.
BigDecimal  round(MathContext mc)
// Returns a BigDecimal rounded according to the MathContext
settings.
int scale()
// Returns the scale of this BigDecimal.
BigDecimal  scaleByPowerOfTen(int n)
// Returns a BigDecimal whose numerical value is equal to (this
* 10n).
BigDecimal  setScale(int newScale)
// Returns a BigDecimal whose scale is the specified value, and
whose value is numerically equal to this BigDecimal's.
BigDecimal  setScale(int newScale, int roundingMode)
// Returns a BigDecimal whose scale is the specified value, and
whose unscaled value is determined by multiplying or dividing
this BigDecimal's unscaled value by the appropriate power of
ten to maintain its overall value.
BigDecimal  setScale(int newScale, RoundingMode roundingMode)
// Returns a BigDecimal whose scale is the specified value, and
whose unscaled value is determined by multiplying or dividing
this BigDecimal's unscaled value by the appropriate power of
ten to maintain its overall value.
short   shortValueExact()
// Converts this BigDecimal to a short, checking for lost
information.
int signum()
// Returns the signum function of this BigDecimal.
BigDecimal  stripTrailingZeros()
// Returns a BigDecimal which is numerically equal to this one
but with any trailing zeros removed from the representation.
BigDecimal  subtract(BigDecimal subtrahend)
```

```
// Returns a BigDecimal whose value is (this - subtrahend), and
whose scale is max(this.scale(), subtrahend.scale()).
BigDecimal  subtract(BigDecimal subtrahend, MathContext mc)
// Returns a BigDecimal whose value is (this - subtrahend),
with rounding according to the context settings.
BigInteger  toBigInteger()
// Converts this BigDecimal to a BigInteger.
BigInteger  toBigIntegerExact()
// Converts this BigDecimal to a BigInteger, checking for lost
information.
String  toEngineeringString()
// Returns a string representation of this BigDecimal, using
engineering notation if an exponent is needed.
String  toPlainString()
// Returns a string representation of this BigDecimal without
an exponent field.
String  toString()
// Returns the string representation of this BigDecimal, using
scientific notation if an exponent is needed.
BigDecimal  ulp()
// Returns the size of an ulp, a unit in the last place, of
this BigDecimal.
BigInteger  unscaledValue()
// Returns a BigInteger whose value is the unscaled value of
this BigDecimal.
static BigDecimal   valueOf(double val)
// Translates a double into a BigDecimal, using the double's
canonical string representation provided by the
Double.toString(double) method.
static BigDecimal   valueOf(long val)
// Translates a long value into a BigDecimal with a scale of
zero.
static BigDecimal   valueOf(long unscaledVal, int scale)
// Translates a long unscaled value and an int scale into a
BigDecimal.
```

Tree

Tree Traversal

Trie

后缀数组

Binary Indexed Tree

Segment Tree

```
const int MAX = 100000;

struct node {
    int left, right;
    int color;
    bool cover;
};

node nodes[3*MAX];

void build_tree(int left, int right, int u) {
    nodes[u].left = left;
    nodes[u].right = right;
    nodes[u].color = 1;
    nodes[u].cover = true;
    if (left == right) return;
    int mid = (left + right)/2;
    build_tree(left, mid, 2*u);
    build_tree(mid+1, right, 2*u + 1);
}

void get_down(int u) {
    int value = nodes[u].color;
    nodes[u].cover = false;
    nodes[2*u].color = value;
    nodes[2*u].cover = true;
    nodes[2*u + 1].color = value;
    nodes[2*u + 1].cover = true;
}

void update(int left, int right, int value, int u) {
    if (left <= nodes[u].left && nodes[u].right <= right) {
        nodes[u].color = value;
        nodes[u].cover = true;
        return;
    }
    if (nodes[u].color == value) return;  // optimize purpose
    //SHOW(u);
    if (nodes[u].cover) get_down(u);
    if (left <= nodes[2*u].right) {
        update(left, right, value, 2*u);
    }
    if (right >= nodes[2*u+1].left) {
        update(left, right, value, 2*u + 1);
    }
    nodes[u].color = nodes[2*u].color | nodes[2*u+1].color;
}

void query(int left, int right, int &sum, int u) {
    if (nodes[u].cover) {
        sum |= nodes[u].color;
        return;
    }
    if (left <= nodes[u].left && nodes[u].right <= right) {
        sum |= nodes[u].color;
        return;
    }
    if (left <= nodes[2*u].right) {
        query(left, right, sum, 2*u);
    }
    if (right >= nodes[2*u+1].left) {
        query(left, right, sum, 2*u + 1);
    }
}

// only for this question
int bit_count(int sum) {
    int ans = 0;
```

```
    while (sum) {
        if (sum%2) ans++;
        sum = sum >> 1;
    }
    return ans;
}

int main() {
    int L, T, O;
    cin >> L >> T >> O;
    build_tree(1, L, 1);
    while (O--) {
        char op;
        int a, b, c;
        cin >> op;
        if (op == 'C') {
            cin >> a >> b >> c;
            if (a > b) swap(a, b);
            update(a, b, 1<<(c-1), 1);
        } else {
            cin >> a >> b;
            if (a > b) swap(a, b);
            int sum = 0;
            query(a, b, sum, 1);
            cout << bit_count(sum) << endl;
        }
    }
    return 0;
}
```

Range Minimum Query RMQ

place holder

String

KMP

Longest palindromic substring (Manacher's algorithm)

Graph

Union-find Set

```
int father[n];
int get_father(int a) {
    if (father[a] != a)
        return father[a] = get_father(father[a]);
    return a;
}
void init() {
    for (int i = 0; i < n; i++)
        father[i] = i;
}
```

Union-find Set - application

place holder

Minimium Spanning Tree

Prim's

graph[][], time complexity: O(V^2)

```
void mst_prim() {
    int n_node, n_edge;
    ////////////////////////////////////////
    // read data
    ////////////////////////////////////////
    int graph[n_node][n_node];
    int min_dis[n_node];
    for (int i = 0; i < n_node; i++)
        min_dis[i] = INT_MAX; // initialize
    ////////////////////////////////////////
    // read graph[][]
    ////////////////////////////////////////

    int cur = 0; // the node just added
    for (int i = 1; i < n_node; i++) { // total need pick n-1 edges
        min_dis[cur] = -1; // add cur
        for (int j = 0; j < n_node; j++) // for all node
            if (graph[cur][j] < min_dis[j]) // if can reach from new node and nearer
                min_dis[j] = graph[cur][j]; // update the distance
        int next = -1; // the node to add
        int cur_min_dis = INT_MAX; // current distance of nearest node
        for (int j = 0; j < n_node; j++) // check all node
            if (min_dis[j] >= 0 && min_dis[j] < cur_min_dis) { // if j node is nearer
                next = j; // record
                cur_min_dis = min_dis[j];
            }
        // add edge: cur->next
        cur = next; // next node to add
    }
}
```

vector graph[], time complexity: (V + E)log(V)

```
struct Edge {
    int from;
    int to;
    int length;

    bool operator< (Edge b) const {
        return this->length > b.length;
    }
};

void mst_prim() {
    int n_node;
    int n_edge;
    ////////////////////////////////////////////
    // cin >> n_node >> n_edge;
    ////////////////////////////////////////////
    vector<Edge> graph[n_node];
    ////////////////////////////////////////////
```

```
    // read graph
    ////////////////////////////////////////

    priority_queue<Edge> discovered;
    int added[n_node];
    memset(added, 0, sizeof(added));

    int to_add = n_node;
    Edge temp = {0, 0, 0};
    discovered.push(temp); // 0 is first node
    while (to_add--) {
        Edge cur = discovered.top();
        discovered.pop();
        while (added[cur.to] == 1) {
            cur = discovered.top();
            discovered.pop();
        }
        // cur is the edge to add

        added[cur.to] = 1;
        for (int i = 0; i < graph[cur.to].size(); i++) {
            Edge& next = graph[cur.to][i];
            if (to != next.to && added[next.to] == 0) {
                discovered.push(next);
            }
        }
    }
    // should directly maintain the min distance for each node
to current tree
    // use heapfy...
}
```

Kruskal

Elog(E) + Elog(V)

```
struct Edge {
    int from;
    int to;
    int length;

    bool operator< (Edge b) const {
        return this->length < b.length;
    }
};

int get_father(int father[], int a) {
    if (father[a] != a)
        return father[a] = get_father(father, father[a]);
    return a;
}

void solve() {
    int n_node, n_edge;
    ////////////////////////////////////////////////////
    // initialize n_edge
    ////////////////////////////////////////////////////
    Edge e[n_edge];
    ////////////////////////////////////////////////////
    // initialize edge e
    ////////////////////////////////////////////////////
    int father[n_node];
    for (int i = 0; i < n_node; i++)
        father[i] = i; // initialize
    sort(e, e + n_edge);

    int to_add = n_node - 1;
    for (int cur = 0; to_add; cur++) {
        int fromSfather = get_father(father, e[cur].from);
        int toSfather = get_father(father, e[cur].to);
        if (fromSfather != toSfather) {
            father[fromSfather] = toSfather;
            to_add--;
            // add edge e[cur]
        }
    }
}
```

Shortest Path

任意两点

```
for ()
```

```
    for ()
        for ()
```

Bellman–Ford

Bellman–Ford algorithm is O(VE). Can be applied to situations when there is a maximun number of vertices in shortest path.

```
for (n times of relax)
    for (each node)
        relax each node
```

SPFA

Dijkstra

Dijkstra is good for graphs non-negative edges.

O(V^2)

Bipartite Graph 二分图

1. A graph is bipartite if and only if it does not contain an odd cycle.
2. A graph is bipartite if and only if it is 2-colorable, (i.e. its chromatic number is less than or equal to 2).
3. The spectrum of a graph is symmetric if and only if it's a bipartite graph.

Hungarian algorithm 匈牙利算法

O(E * V)

```
int n_node;
int graph[405][405];
int match[405];
int visited[405];

bool augment(int cur) {
    for (int i = 0; i < n_node; i++) { // for all node
        if (graph[cur][i] > 0 && visited[i] == 0) { // if have edge and not visited
            visited[i] = 1; // mark visited
            if (match[i] == -1 || augment(match[i])) { // if not matched, or its previous can find other match
                match[i] = cur; // match it
                return true;
            }
        }
    }
    return false; // cannot find any match
}

int match() {
    memset(graph, 0, sizeof(graph));
    ////////////////////////////////////////////
    // initialize n_node, graph
    ////////////////////////////////////////////
    int n_match = 0;
    memset(match, -1, sizeof(match));
    for (int i = 0; i < n_node; i++) { // for each node, find an augmented path
        memset(visited, 0, sizeof(visited)); // each node only visit once
        if (augment(i)) // if found
            n_match++; // maximum matching ++
    }
}
```

Maximum Flow Problem 最大流

Dinic

```
int graph[250][250];
int level[250];
```

```
int n_node;

int mark_level(int start, int end) {
    memset(level, -1, sizeof(level));
    queue<int> to_visit;

    level[start] = 0;
    to_visit.push(start);
    while (!to_visit.empty()) {
        int cur = to_visit.front();
        to_visit.pop();
        for (int i = 0; i < n_node; ++i) {
            if (graph[cur][i] != 0 && level[i] == -1) {
                level[i] = level[cur] + 1;
                to_visit.push(i);
            }
        }
    }

    if (level[end] == -1)
        return 0; // cannot reach the sink
    return 1; // can reach the sink
}

int augment(int cur, int end, int min_flow) {
    if (cur == end)
        return min_flow;

    int augmented_flow = 0;
    for (int i = 0; i < n_node; ++i) {
        if ((level[i] == level[cur] + 1 && graph[cur][i] > 0)
&&
            (augmented_flow = augment(i, end,
min(graph[cur][i], min_flow)))
        ) {
            graph[cur][i] -= augmented_flow;
            graph[i][cur] += augmented_flow;
            return augmented_flow;
        }
    }
    return 0;
}
```

```
int dinic(int start, int end) {
    int ans = 0;
    int temp = 0;
    while (mark_level(start, end))
        while (temp = augment(start, end, INT_MAX))
            ans += temp;
    return ans;
}
```

Minimum-Cost Maximum-Flow

```
// have not tested
int n_node;
int n_edge;

int cost[405][405]; // cost[i][j] = -cost[j][i]
int residual[405][405];

bool bellman_ford(int& flow_sum, int&cost_sum) { // 0: start,
n_node - 1: end
    int min_cost[405]; for (int i = 0; i < n_node; i++)
min_cost[i] = INT_MAX; min_cost[0] = 0;
    int pre_node[405]; pre_node[0] = 0;
    int max_flow[405];
    int in_queue[405]; memset(in_queue, 0, sizeof(in_queue));

    queue<int> q;
    q.push(0);
    while (q.size()) {
        int cur = q.front(); q.pop();
        in_queue[cur] = 0;

        for (int i = 0; i < n_node; i++) {
            if (residual[cur][i] > 0 && min_cost[i] >
min_cost[cur] + cost[cur][i]) {
                min_cost[i] = min_cost[cur] + cost[cur][i];
                pre_node[i] = cur;
                max_flow[i] = min(max_flow[cur],
residual[cur][i]);

                if (in_queue[i] == 0) {
```

```
                    in_queue[i] = 1;
                    q.push(i);
                }
            }
        }
    }
    if (min_cost[n_node - 1] == INT_MAX)
        return false;
    flow_sum += max_flow[n_node - 1];
    cost_sum += max_flow[n_node - 1] * min_cost[n_node - 1];
    for (int cur = n_node - 1; cur != 0; cur = pre_node[cur]) {
        residual[pre_node[cur]][cur] -= max_flow[n_node - 1];
        residual[cur][pre_node[cur]] += max_flow[n_node - 1];
    }
    return true;
}

void min_cost_max_flow() {
    int flow_sum = 0;
    int cost_sum = 0;
    while (bellman_ford(flow_sum, cost_sum));
    cout << flow_sum << " " << cost_sum << endl;
}
```

强连通分量 图的 割点, 桥, 双连通分支

https://www.byvoid.com/blog/biconnect

[点连通度与边连通度]

在一个无向连通图中，如果有一个顶点集合，删除这个顶点集合，以及这个集合中所有顶点相关联的边以后，原图变成多个连通块，就称这个点集为**割点集合**。一个图的**点连通度**的定义为，最小割点集合中的顶点数。

类似的，如果有一个边集合，删除这个边集合以后，原图变成多个连通块，就称这个点集为**割边集合**。一个图的**边连通度**的定义为，最小割边集合中的边数。

[双连通图、割点与桥]

如果一个无向连通图的点连通度大于 1，则称该图是**点双连通的(point biconnected)**，简称**双连通**或**重连通**。一个图有割点，当且仅当这个图的点连通度为 1，则割点集合的唯一元素被称为**割点(cut point)**，又叫**关节点(articulation point)**。

如果一个无向连通图的边连通度大于 1，则称该图是**边双连通的(edge biconnected)**，简称双连通或重连通。一个图有桥，当且仅当这个图的边连通度为 1，则割边集合的唯一元素被称为**桥(bridge)**，又叫**关节边(articulation edge)**。

可以看出，点双连通与边双连通都可以简称为双连通，它们之间是有着某种联系的，下文中提到的双连通，均既可指点双连通，又可指边双连通。

[双连通分支]

在图 G 的所有子图 G'中，如果 G'是双连通的，则称 G'为**双连通子图**。如果一个双连通子图 G'它不是任何一个双连通子图的真子集，则 G'为**极大双连通子图**。**双连通分支(biconnected component)**，或**重连通分支**，就是图的极大双连通子图。特殊的，点双连通分支又叫做**块**。

[求割点与桥]

该算法是 R.Tarjan 发明的。对图深度优先搜索，定义 DFS(u)为 u 在搜索树（以下简称为树）中被遍历到的次序号。定义 Low(u)为 u 或 u 的子树中能通过非父子边追溯到的最早的节点，即 DFS 序号最小的节点。根据定义，则有：

Low(u)=Min { DFS(u) DFS(v) (u,v)为后向边(返祖边) 等价于 DFS(v)<DFS(u) 且 v 不为 u 的父亲节点 Low(v) (u,v)为树枝边(父子边) }

一个顶点 u 是割点，当且仅当满足(1)或(2) (1) u 为树根，且 u 有多于一个子树。 (2) u 不为树根，且满足存在(u,v)为树枝边(或称父子边，即 u 为 v 在搜索树中的父亲)，使得 DFS(u)<=Low(v)。

一条无向边(u,v)是桥，当且仅当(u,v)为树枝边，且满足 DFS(u)<Low(v)。

[求双连通分支]

下面要分开讨论点双连通分支与边双连通分支的求法。

对于点双连通分支，实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈，存储当前双连通分支，在搜索图时，每找到一条树枝边或后向边(非横叉边)，就把这条边加入栈中。如果遇到某时满足 DFS(u)<=Low(v)，说明 u 是一个割点，同时把边从栈顶一个个取出，直到遇到了边(u,v)，取出的这些边与其关联的点，组成一个点双连通分支。割点可以属于多个点双连通分支，其余点和每条边只属于且属于一个点双连通分支。

对于边双连通分支，求法更为简单。只需在求出所有的桥以后，把桥边删除，原图变成了多个连通块，则每个连通块就是一个边双连通分支。桥不属于任何一个边双连通分支，其余的边和每个顶点都属于且只属于一个边双连通分支。

[构造双连通图]

一个有桥的连通图，如何把它通过加边变成边双连通图？方法为首先求出所有的桥，然后删除这些桥边，剩下的每个连通块都是一个双连通子图。把每个双连通子图收缩为一个顶点，再把桥边加回来，最后的这个图一定是一棵树，边连通度为 1。

统计出树中度为 1 的节点的个数，即为叶节点的个数，记为 leaf。则至少在树上添加(leaf+1)/2 条边，就能使树达到边二连通，所以至少添加的边数就是(leaf+1)/2。具体方法为，首先把两个最近公共祖先最远的两个叶节点之间连接一条边，这样可以把这两个点到祖先的路径上所有点收缩到一起，因为一个形成的环一定是双连通的。然后再找两个最近公共祖先最远的两个叶节点，这样一对一对找完，恰好是(leaf+1)/2 次，把所有点收缩到了一起。

拓扑排序

place holder

Euler Cycle/Path, Hamilton Cycle/Path

place holder

Mathematics

class/struct Matrix

```
operator+
operator*
```

Square matrix

```
struct Matrix {
    // int height;
    // int width;

    long long value[32][32];

    Matrix operator* (const Matrix& that);
    Matrix operator+ (const Matrix& that);
    Matrix mirror();
    void show() {
        cout << endl;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++)
                cout << this->value[i][j] << " ";
            cout << endl;
        }
    }
};

void mod_it(Matrix& temp) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            temp.value[i][j] %= m;
}

Matrix Matrix::operator* (const Matrix& that) {
    Matrix temp;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            temp.value[i][j] = 0;
            for (int k = 0; k < n; k++)
                temp.value[i][j] += this->value[i][k] *
that.value[k][j];
        }
    }
    mod_it(temp);
    return temp;
}

Matrix Matrix::operator+ (const Matrix& that) {
    Matrix temp;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            temp.value[i][j] = this->value[i][j] +
that.value[i][j];
    mod_it(temp);
    return temp;
}

Matrix Matrix::mirror() {
    Matrix temp;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            temp.value[i][j] = this->value[i][j];
    return temp;
}
```

欧拉函数？

欧几里得算法 / gcd

see next section

扩展欧几里得算法

http://www.cnblogs.com/frog112111/archive/2012/08/19/2646012.html

对于不完全为 0 的非负整数 a, b, 必然存在整数对 (x, y), 使得 gcd(a, b) = ax + by

suppose: a > b, we want to get (x1, y1)

(i) if b == 0, then gcd(a, b) = a = ax + 0, then x1 = 1, y1 = 0

(ii) if b != 0:

(1): a * x1 + b * y1 = gcd(a, b)

(2): b * x2 + (a % b) * y2 = gcd(b, a % b)

(1) == (2)

so: a * x1 + b * y1 = b * x2 + (a % b) * y2

so: a * x1 + b * y1 = b * x2 + (a - (int)(a / b) * b) * y2

so: a * x1 + b * y1 = a * y2 + b * (x2 - (int)(a / b) * y2)

so: x1 = y2, y1 = x2 - (int)(a / b) * y2, can get (x1, y1) from (x2, y2)

next:

```
  (1): b * x2 + (a % b) * y2 = gcd(b, a % b)

  (2): (a % b) * x3 + b % (a % b) * y3 = gcd(a % b, b % (a % b))

  so: can get (x2, y2) from (x3, y3)

  next: ... until in gcd(a, b), b == 0, then xi = 1, yi = 0, go back ...
```

```
long long ansx, ansy, ansd;

void euclidean(long long a, long long b) {
    if (b == 0) {
        ansx = 1;
        ansy = 0;
        ansd = a;
    } else {
        euclidean(b, a % b);
        long long temp = ansx;
        ansx = ansy;
        ansy = temp - a / b * ansy;
    }
}

int main(int argc, char const *argv[]) {
    long long a, b, c;
    cin >> a >> b >> c;

    ansx = 0;
    ansy = 0;
    ansd = 0;
    euclidean(a, b);

    // now (ansx, ansy) is the answer (x, y) for a * x1 + b * y1 = gcd(a, b)
    // ansd is the a when b == 0, which is just gcd(a, b)
}
```

求解不定方程

for: p * a + q * b = c

if c % gcd(a, b) == 0, then 有整数解 (p, q), else NO

if we get (p0, q0) for p0 * a + q0 * b = gcd(a, b)

then: for p * a + q * b = gcd(a, b) (k is any integer)

p = p0 + b / gcd(a, b) * k

q = q0 - a / gcd(a, b) * k

then: for p * a + q * b = c = c / gcd(a, b) * gcd(a, b) (k is any integer)

p = (p0 + b / gcd(a, b) * k) * c / gcd(a, b)

q = (q0 - a / gcd(a, b) * k) * c / gcd(a, b)

```
// after get ansx, ansy, ansd
// test if c % ansd == 0
// ansx = (ansx + b / gcd(a, b) * k) * c / gcd(a, b)
// ansy = (ansy - a / gcd(a, b) * k) * c / gcd(a, b)
// smallest: ansx % (b / gcd(a, b) + b / gcd(a, b)) % (b / gcd(a, b))
```

求解模线性方程（线性同余方程）

(a * x) % n = b % n, x = ?

same as: a * x + n * y= b

so: one answer for a * x + n * y= b is: x * b / gcd(a, n)

so: one answer for (a * x) % n = b % n is: x0 = (x * b / gcd(a, n)) % n

other answer xi = (x0 + i * (n / gcd(a, n))) % n, i = 0...gcd(a, n)-1

smallest answer is x0 % (n / gcd(a, n) + gcd(a, n)) % gcd(a, n)

求解模的逆元

(a * x) % n = 1, x = ?

if gcd(a, n) != 1, then NO answer

else:

same as: a * x + n * y = 1

can get only one answer x

```
// after get ansx, ansy, ansd
// if ansd != 1, then NO answer
// smallest ansx = (ansx % (n / gcd(a, n)) + (n / gcd(a, n))) % (n / gcd(a, n))
```

中国剩余定理

最小公倍数

```
a * b / gcd(a, b)
```

分解质因数

```
long long x;
cin >> x;
for (long long factor = 2; x != 1; factor++) {
    if (x % factor == 0)
        cout << factor << " is a prime factor" << endl;
    while (x % factor == 0)
        x = x / factor;
}
```

因数个数

```
n = p1 ^ x1 * p2 ^ x2 * ... * pn ^ xn
total = (x1 + 1) * (x2 + 1) * ... * (xn + 1)
```

素数判定

大于 3 的质数可以被表示为 6n - 1 或 6n + 1

```
bool is_prime(int n) {
    if (n == 1 || n % 2 == 0)
        return false;
    int t = sqrt(n);
    for (int i = 3; i <= t; i += 2)
        if (n % i == 0)
            return false;
    return true;
}
```

进制转换

```
void convert_dec_to_base(int n, const int base) {
    if (n == 0)
        cout << 0 << endl;
    while (n != 0) {
        int e = n % base;
        cout << e << " "; // printing in reverse order
        n /= base;
    } cout << endl;
}

int convert_base_to_dec(const int s[], const int len, const int
base) {
    int result = 0;
    for (int i = 0; i < len; i++)
        result = result * base + s[i];
    return result;
}
```

A / C

C(n, k) = C(n-1, k) + C(n-1, k-1) C(n, k) = C(n, n-k)

博弈论

place holder

Geometry

template class for Point?

```
struct point {
    int x, y;

    double length() {
        return sqrt(x*x + y*y);
    }

    long operator* (const point& b) {
        return x*b.y - y*b.x;
    }

    long cross_product(const point& b) {
      return x * b.x + y * b.y;
    }

    bool at_right_of(const point& a, const point& b) const {
        // a: relative point, b: base
        point vec_self = {x - b.x, y - b.y};
        point vec_that = {a.x - b.x, a.y - b.y};
        long product = vec_self * vec_that;
        if (product>0) return true;
        if (product==0 && vec_self.length()>vec_that.length())
return true;
        return false;
    }

    double to_point(const point& b) const {
        return sqrt(pow(x-b.x,2) + pow(y-b.y,2));
    }

    double to_segment(const point& a, const point& b) const {
        double len_ab = a.to_point(b);
        if (abs(len_ab)<E) return to_point(a);
        double r = ((a.y-y)*(a.y-b.y) - (a.x-x)*(a.x-
b.x))/pow(len_ab,2);
        if (r>1 || r<0) return min(to_point(a), to_point(b));
        // projection of p is on extension of AB
        r = ((a.y - y)*(b.y - y) - (a.x - x)*(b.y -
a.y))/pow(len_ab,2);
        return fabs(r*len_ab);
    }

    double to_segment_v2(const point& a, const point& b) const
{
        point vec_ab = {b.x - a.x, b.y - a.y};
        point vec_ia = {x - a.x, y - a.y};
        point vec_ib = {x - b.x, y - b.y};
        if (vec_ia.cross_product(ba) < 0 ||
vec_ib.cross_product(ba) > 0)
          return min(to_point(a), to_point(b));
        return abs(vec_ab * vec_ia) / vec_ab.length();
    } // same meaning with v1, need test
};
```

向量点乘 叉乘

a = (x1, y1) b = (x2, y2) i ... |i| = 1, vertical to a-b surface

dot product

a dot b = x1 * x2 + y1 * y2 = |a| * |b| * cos(angle)

if = 0: 90 degree

a dot b / |b| = a project to b

cross product

a x b = x1 * y2 - x2 * y1 = |a| * |b| * sin(angle) * i

if < 0: b is at left of a

if = 0: a, b in a line

if 0: b is at right of a

a x b = area of 平行四边形 a x b x c = area of 平行六面体, c = (x3, y3)

直线公式

(x, y) = (x1, y1) + k * ((x2, y2) - (x1, y1))

Convex Hull

Gift Wrapping

place holder

QuickHull

place holder

Graham scan

O(VlogV)

```
struct Point {
    long x;
    long y;

    bool at_right_of(Point& that, Point& base) {
        Point vec_self = {this->x - base.x, this->y - base.y};
        Point vec_that = {that.x - base.x, that.y - base.y};

        long product = vec_self * vec_that;
        if (product > 0)
            return true; // "this" is at right of "that"
        if (product == 0 && vec_self.length() >
vec_that.length())
            return true; // "this" is at right of "that"
        return false; // "this" is NOT at right of "that"
    };
    long operator* (Point& that) {
        return this->x * that.y - this->y * that.x;
    };
    double distance_to(Point& that) {
        long x_diff = this->x - that.x;
        long y_diff = this->y - that.y;
        return sqrt(x_diff * x_diff + y_diff * y_diff);
    };
    double length() {
        return sqrt(this->x * this->x + this->y * this->y);
    }
};

Point p[1005];
int my_stack[1005];
int n, l, my_stack_top = -1;

bool compare(Point p1, Point p2) {
    return p1.at_right_of(p2, p[0]);
}

void push(int index) {
    my_stack[++my_stack_top] = index;
}
int pop() {
    int temp = my_stack[my_stack_top--];
    return temp;
}

void graham_scan() {
    push(0);
    push(1);

    int pre;
    int prepre;
    for (int i = 2; i < n; i++) {
        pre = my_stack_top;
        prepre = my_stack_top - 1;
        while (p[i].at_right_of(p[my_stack[pre]],
p[my_stack[prepre]])) {
```

```
            pop();
            if (my_stack_top == 0)
                break;
            pre = my_stack_top;
            prepre = my_stack_top - 1;
        }
        push(i);
    }

    int last = my_stack_top;
    if (p[0].at_right_of(p[my_stack[last]], p[my_stack[pre]]))
        pop();
}

int main(int argc, char const *argv[]) {
    cin >> n >> l;

    int minimun = 0;
    for (int i = 0; i < n; ++i) {
        int temp_x, temp_y;
        cin >> temp_x >> temp_y;
        p[i] = {temp_x, temp_y};

        if ((p[i].y < p[minimun].y) || (p[i].y == p[minimun].y
&& p[i].x < p[minimun].x))
            minimun = i;
    }

    Point temp = {p[minimun].x, p[minimun].y}; // swap lowest
and most left point to p[0]
    p[minimun] = p[0];
    p[0] = temp;

    sort(p + 1, p + n, compare); // use p[0] as base, sort
according to polar angle
    graham_scan();
    // now all points in the stack is on Convex Hull // size of
stack = 1 + stack_top

    for (int i = 0; i <= my_stack_top; i++)
        cout << "point " << my_stack[i] << " is on Convex Hull"
<< endl;
}
```

Tricks / 分析方法

Recursive

Hanoi

```
void hanoi(int n, char x, char y, char z) { // 将 x 上编号 1 至 n 的圆盘移到 z, y 作辅助塔
    if (n == 1)
        printf("%d from %c to %c\n", n, x, z); // 将编号为 n 的圆盘从 x 移到 z
    else {
        hanoi(n-1, x, z, y); // 将 x 上编号 1 至 n-1 的圆盘移到 y, z 作辅助塔
        printf("%d from %c to %c\n", n, x, z); // 将编号为 n 的圆盘从 x 移到 z
        hanoi(n-1, y, x, z); // 将 y 上编号 1 至 n-1 的圆盘移到 z, x 作辅助塔
    }
}
```

Dynamic Programming

树上的

Divide and Conquer

迭代加深搜索 (binary increase/decrease)

```
int up_limit = ;
int down_limit = ;
int cur, pre;
```

```
while (true) {
    cur = (down_limit + up_limit) / 2;

    bool ok = search();

    if (ok)
        up_limit = cur;
    else
        down_limit = cur;
    pre = cur;
    cur = (down_limit + up_limit) / 2;
    if (pre == cur)
        return up_limit;
}
```

双向 BFS

从终点开始搜

Brute Force

子集生成

Userful Code Snippets

cantor_expansion / reverse_cantor_expansion

for hashing, or ...

```
long long factorial(int n) {
    if (n == 0)
        return 1;

    long long ans = n;
    for (int i = 1; i < n; i++)
        ans = ans * i;
    return ans;
}

long long cantor_expansion(int permutation[], int n) {
    // input: (m-th permutation of n numbers, n)
    // return: m
    int used[n + 1];
    memset(used, n, sizeof(used));

    long long ans = 0;
    for (int i = 0; i < n; i++) {
        int temp = 0;
        used[permutation[i]] = 1;
        for (int j = 1; j < permutation[i]; j++)
            if (used[j] != 1)
                temp += 1;
        ans += factorial(n - 1 - i) * temp;
    }

    return ans + 1;
}

void reverse_cantor_expansion(int n, long long m) {
    // m-th permutation of n numbers
    int ans[n + 1], used[n + 1];
    memset(ans, -1, sizeof (ans));
    memset(used, 0, sizeof (used));
    m = m - 1;
    for (int i = n - 1; i >= 0; i--) {
        long long fac = factorial(i);
        int temp = m / fac + 1;
        m = m - (temp - 1) * fac;
        for (int j = 1; j <= temp; j++)
            if (used[j] == 1)
                temp++;

        ans[n - i] = temp;
        used[temp] = 1;
    }

    for (int i = 1; i < n + 1; i++)
        cout << ans[i] << " ";
    cout << "\n";
}
```

Fast Exponention

To calculate n ^ p % M

```
int power_modulo(int n, int p, int M) {
    int result = 1;
    while (p > 0) {
        if (p % 2 == 1)
            result = (result*n) % M;
        p /= 2;
        n = (n*n) % M;
    }
    return result;
}
```

质数表

```
int is_prime[UP_LIMIT + 1];
for (int i = 1; i <= UP_LIMIT; i++) // init to 1
    is_prime[i] = 1;
for (int i = 4; i <= UP_LIMIT; i += 2) // even number is not
```

```
    is_prime[i] = 0;
for (int k = 3; k*k <= UP_LIMIT; k++) // start from 9, end at
sqrt
    if (is_prime[k])
        for(int i = k*k; i <= UP_LIMIT; i += 2*k) // every two
is not
            is_prime[i] = 0;
```