Bangladesh University of Engineering and Technology

# BUET_hotasha

Shafin Ahmed, Md. Ashraful Islam Fahim, Md. Tahmid Hossain

2025-06-20

1 Contest 1

2 Mathematics 2

3 Data structures 3

4 Dynamic-Programming 6

5 Numerical 7

6 Number theory 9

7 Combinatorial 12

8 Graph 13

9 Flow 15

10 Geometry 18

11 Strings 23


# Contest (1)

## template.cpp

21 lines

```cpp
#include <bits/stdc++.h>
using namespace std;

#ifdef LOCAL
#include "debug.h"
#else
#define debug(...)
#endif
#define endl "\n"

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit);
}
```

## sublimebuild

18 lines

```
{
"shell_cmd": "g++ \"${file}\" -o \"${file_path}/${
    file_base_name}\"",
"file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
"working_dir": "${file_path}",
"selector": "source.c, source.c++",

"variants":
[
    {
        "name": "General",
        "shell_cmd": "g++ \"${file}\" -o \"${file_path}/${
            file_base_name}\" && \"${file_path}/${
            file_base_name}\" <input.txt >output.txt && time
            \"${file_path}/${file_base_name}\" < input.txt"
    },
    {
        "name": "StressTesting",
        "shell_cmd": "g++ \"${file}\" -o \"${file_path}/${
            file_base_name}\" && g++ \"${file_path}/naive.cpp
            \" -o \"${file_path}/naive\" && g++ \"${file_path}
            /input_gen.cpp\" -o \"${file_path}/input_gen\" &&
            chmod +x \"${file_path}/script.sh\" && \"${
            file_path}/script.sh\" "
    }
]
}
```

## script.sh

9 lines

```sh
for i in `seq 1 10000`; do
    echo $i

    ./input_gen $i 4 5 > input.txt
    ./code < input.txt > output.txt
    ./naive < input.txt > answer.txt

    diff output.txt answer.txt || break
done
```

## FastIO-Pragmas

4 lines

```cpp
#pragma GCC optimize("O3", "unroll-loops")
#pragma GCC target("avx2", "bmi", "bmi2", "lzcnt", "popcnt")
ios_base::sync_with_stdio(false);
cin.tie(NULL);
```

## debug.h

58 lines

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/exception.hpp>
#include <ext/pb_ds/hash_policy.hpp>
#include <ext/pb_ds/list_update_policy.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/trie_policy.hpp>

using namespace std;

// Debug printer functions
void __print(int x)                { cerr << x; }
void __print(long x)               { cerr << x; }
void __print(long long x)          { cerr << x; }
void __print(unsigned x)           { cerr << x; }
void __print(unsigned long x)      { cerr << x; }
void __print(unsigned long long x) { cerr << x; }
void __print(float x)              { cerr << x; }
void __print(double x)             { cerr << x; }
void __print(long double x)        { cerr << x; }
void __print(char x)               { cerr << '\'' << x << '\''; }
void __print(const char *x)        { cerr << '\"' << x << '\"'; }
void __print(const string &x)      { cerr << '\"' << x << '\"'; }
void __print(bool x)               { cerr << (x ? "true" : "false
     "); }

template <typename T, typename V>
void __print(const pair<T, V> &x) {
    cerr << '{';
    __print(x.first);
    cerr << ", ";
    __print(x.second);
    cerr << '}';
}

template <typename T>
void __print(const T &x) {
    int f = 0;
    cerr << '{';
    for (auto &i : x) {
        if (f++) cerr << ", ";
        __print(i);
    }
    cerr << '}';
}

void _print() {
    cerr << "]\n";
}

template <typename T, typename... V>
void _print(T t, V... v) {
    __print(t);
    if (sizeof...(v)) cerr << ", ";
    _print(v...);
}

#define debug(x...) \
    cerr << "[" << #x << "] = ["; _print(x);
```

## hash.sh

3 lines

```sh
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]'| md5sum |cut -c-6
```

## troubleshoot.txt

52 lines

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it.

Runtime error:
Have you tested all corner cases locally?
```

```
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your teammates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
```

# Mathematics (2)

## 2.1 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The extremum is given by $x = -b/2a$.

$$\begin{matrix} ax + by = e \\ cx + dy = f \end{matrix} \Rightarrow \begin{matrix} x = \frac{ed - bf}{ad - bc} \\ y = \frac{af - ec}{ad - bc} \end{matrix}$$

In general, given an equation $Ax = b$, the solution to a variable $x_i$ is given by

$$x_i = \frac{\det A_i'}{\det A}$$

where $A_i'$ is $A$ with the $i$'th column replaced by $b$.

## 2.2 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k - c_1 x^{k-1} - \cdots - c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots $r$ become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

## 2.3 Trigonometry

$$\sin(v + w) = \sin v \cos w + \cos v \sin w$$
$$\cos(v + w) = \cos v \cos w - \sin v \sin w$$
$$\tan(v + w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$
$$\sin v + \sin w = 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2}$$
$$\cos v + \cos w = 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2}$$

$$(V + W) \tan(v - w)/2 = (V - W) \tan(v + w)/2$$

where $V, W$ are lengths of sides opposite angles $v, w$.

$$a \cos x + b \sin x = r \cos(x - \phi)$$
$$a \sin x + b \cos x = r \sin(x + \phi)$$

where $r = \sqrt{a^2 + b^2}, \phi = \text{atan2}(b, a)$.

## 2.4 Geometry

### 2.4.1 Triangles

Side lengths: $a, b, c$

Semiperimeter: $p = \frac{a + b + c}{2}$

Area: $A = \sqrt{p(p - a)(p - b)(p - c)}$

Circumradius: $R = \frac{abc}{4A}$

Inradius: $r = \frac{A}{p}$

Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two): $s_a = \sqrt{bc\left[1 - \left(\frac{a}{b + c}\right)^2\right]}$

Law of sines: $\frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} = \frac{1}{2R}$

Law of cosines: $a^2 = b^2 + c^2 - 2bc \cos \alpha$

Law of tangents: $\frac{a + b}{a - b} = \frac{\tan \frac{\alpha + \beta}{2}}{\tan \frac{\alpha - \beta}{2}}$

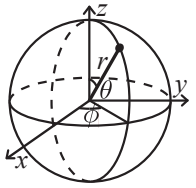
### 2.4.2 Quadrilaterals

With side lengths $a, b, c, d$, diagonals $e, f$, diagonals angle $\theta$, area $A$ and magic flux $F = b^2 + d^2 - a^2 - c^2$:

$$4A = 2ef \cdot \sin \theta = F \tan \theta = \sqrt{4e^2 f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is $180°$, $ef = ac + bd$, and $A = \sqrt{(p - a)(p - b)(p - c)(p - d)}$.

### 2.4.3 Spherical coordinates




$$\begin{matrix} x = r \sin \theta \cos \phi \\ y = r \sin \theta \sin \phi \\ z = r \cos \theta \end{matrix} \quad \begin{matrix} r = \sqrt{x^2 + y^2 + z^2} \\ \theta = \text{acos}(z/\sqrt{x^2 + y^2 + z^2}) \\ \phi = \text{atan2}(y, x) \end{matrix}$$

## 2.5 Derivatives/Integrals

$$\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1 - x^2}} \qquad \frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1 - x^2}}$$
$$\frac{d}{dx} \tan x = 1 + \tan^2 x \qquad \frac{d}{dx} \arctan x = \frac{1}{1 + x^2}$$
$$\int \tan ax = -\frac{\ln |\cos ax|}{a} \qquad \int x \sin ax = \frac{\sin ax - ax \cos ax}{a^2}$$
$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \text{erf}(x) \qquad \int x e^{ax} dx = \frac{e^{ax}}{a^2}(ax - 1)$$

Integration by parts:

$$\int_a^b f(x) g(x) dx = [F(x) g(x)]_a^b - \int_a^b F(x) g'(x) dx$$

## 2.6 Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1} - c^a}{c - 1}, c \neq 1$$

$$1 + 2 + 3 + \cdots + n = \frac{n(n + 1)}{2}$$
$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n + 1)(n + 1)}{6}$$
$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2 (n + 1)^2}{4}$$
$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n + 1)(2n + 1)(3n^2 + 3n - 1)}{30}$$

## 2.7 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, (-\infty < x < \infty)$$
$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, (-1 < x \leq 1)$$
$$\sqrt{1 + x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, (-\infty < x < \infty)$$

# Data structures (3)

## DSUwithRollbacks.cpp

96e2da, 32 lines

```
class DSUWithRollbacks{
public:
  int par[N],siz[N];
  stack<tuple<int,int,int,int>> history;
  int sets;
  DSUWithRollbacks(int n){
    for(int i=1;i<=n;i++) par[i]=i,siz[i]=1;
    sets=n;
  }
  int find(int x){
    while(x!=par[x]) x=par[x];
    return x;
  }
  void union_sets(int x,int y){
    int rootX=find(x);
    int rootY=find(y);
    if(rootX==rootY) return;
    if(siz[rootX]<siz[rootY]) swap(rootX,rootY);
    history.push({rootY,par[rootY],siz[rootX],sets});
    sets--;
    par[rootY]=rootX;
    siz[rootX]+=siz[rootY];
  }
  void rollback(){
    if(history.empty()) return;
    auto [child,oldPar,oldSiz,setno]=history.top();
    history.pop();
    sets++;
    siz[find(child)]=oldSiz;
    par[child]=oldPar;
  }
};
```

## OrderedSet.cpp

df9e80, 17 lines

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T> using o_set = tree<T, null_type, less<T>,
      rb_tree_tag, tree_order_statistics_node_update>;
template <typename T, typename R> using o_map = tree<T, R, less
     <T>, rb_tree_tag, tree_order_statistics_node_update>;
int main() {
  o_set<int>se;
  se.insert(1);
  se.insert(2);
  cout << *se.find_by_order(0) << endl;
  cout << se.order_of_key(2) << endl;
  o_map<int, int>mp;
  mp.insert({1, 10});
  mp.insert({2, 20});
  cout << mp.find_by_order(0)->second << endl;
  cout << mp.order_of_key(2) << endl;
}
```

## Sparse-Table.cpp

f5ed2a, 25 lines

```
#define N 200005
int sparse[N][25],a[N];
void build_sparse(int n){
    for(int i=1;i<=n;i++) sparse[i][0]=a[i];
    for(int j=1;j<25;j++){
        for(int i=1;i+(1<<j)-1<=n;i++){
            sparse[i][j]=sparse[i][j-1]+sparse[i+(1<<(j-1))][j
                -1];
        }
    }
}
int query_sum(int l,int r){
    long long sum=0;
    for(int j=25;j>=0;j--){
        if((1<<j)<=(r-l+1)){
            sum+=sparse[l][j];
            l+=(1<<j);
        }
    }
    return sum;
}
int query(int l,int r){
    int j=31 - __builtin_clz(r-l+1);
    int mini=min(sparse[l][j],sparse[r-(1<<j)+1][j]);
    return mini;
}
```

## SparseTable(2D).cpp

9e2627, 90 lines

```
const int N = 505, LG = 10;
int st[N][N][LG][LG];
int a[N][N], lg2[N];
int yo(int x1, int y1, int x2, int y2) {
  x2++;
  y2++;
  int a = lg2[x2 - x1], b = lg2[y2 - y1];
  return max(
         max(st[x1][y1][a][b], st[x2 - (1 << a)][y1][a][b]),
         max(st[x1][y2 - (1 << b)][a][b], st[x2 - (1 << a)][y2
             - (1 << b)][a][b])
       );
}
void build(int n, int m) { // 0 indexed
  for (int i = 2; i < N; i++) lg2[i] = lg2[i >> 1] + 1;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      st[i][j][0][0] = a[i][j];
    }
  }
  for (int a = 0; a < LG; a++) {
    for (int b = 0; b < LG; b++) {
      if (a + b == 0) continue;
      for (int i = 0; i + (1 << a) <= n; i++) {
        for (int j = 0; j + (1 << b) <= m; j++) {
          if (!a) {
            st[i][j][a][b] = max(st[i][j][a][b - 1], st[i][j +
                (1 << (b - 1))][a][b - 1]);
          } else {
            st[i][j][a][b] = max(st[i][j][a - 1][b], st[i + (1
                << (a - 1))][j][a - 1][b]);
          }
        }
      }
    }
  }
}
string s[N];
int l[N][N], u[N][N];
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  int n, m;
  cin >> n >> m;
  for (int i = 0; i < n; i++) {
    cin >> s[i];
  }
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      if (!j) l[i][j] = 1;
      else l[i][j] = 1 + (s[i][j - 1] <= s[i][j] ? l[i][j - 1]
          : 0);
    }
  }
  for (int j = 0; j < m; j++) {
    for (int i = 0; i < n; i++) {
      if (!i) u[i][j] = 1;
      else u[i][j] = 1 + (s[i - 1][j] <= s[i][j] ? u[i - 1][j]
          : 0);
    }
  }
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      int nw = 1, mnx = u[i][j], mny = l[i][j];
      for (int len = 1; len <= min(i, j); len++) {
        mnx = min(mnx, u[i][j - len]);
        mny = min(mny, l[i - len][j]);
        if (min(mnx, mny) >= len + 1) nw++;
        else break;
      }
      a[i][j] = nw;
    }
  }
  build(n, m);
  int q;
  cin >> q;
  while (q--) {
    int x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;
    x1--, y1--;
    x2--;
    y2--;
    int l = 1, r = min(x2 - x1 + 1, y2 - y1 + 1), ans = 0;
    while (l <= r) {
      int mid = l + r >> 1;
      if (yo(x1 + mid - 1, y1 + mid - 1, x2, y2) >= mid) ans =
          mid, l = mid + 1;
      else r = mid - 1;
    }
    cout << ans << '\n';
  }
  return 0;
}
// https://www.codechef.com/problems/CENS20B
```

## SegmentTree(LazyPropagation).cpp

610d9e, 69 lines

```
struct segtree{
  ll tree[4*N],rep[4*N],prop[4*N];
  void propagate(int node,int l,int r){
    rep[node*2]=rep[node];
    rep[node*2+1]=rep[node];
    prop[node*2]=0;
    prop[node*2+1]=0;
    int xx=rep[node];
    rep[node]=0;
```

```
    int mid=(l+r)/2;
    tree[node*2]=(mid-l+1)*xx;
    tree[node*2+1]=(r-(mid+1)+1)*xx;
  }
  void propagate2(int node,int l,int r){
    prop[node*2]+=prop[node];
    prop[node*2+1]+=prop[node];
    int xx=prop[node];
    prop[node]=0;
    int mid=(l+r)/2;
    tree[node*2]+=(mid-l+1)*xx;
    tree[node*2+1]+=(r-(mid+1)+1)*xx;
  }
  void update(int node,int l,int r,int a,int b,int u){
    if(b<l || a>r) return;
    if(l>=a && r<=b){
      tree[node]=(r-l+1)*u;
      rep[node]=u;
      prop[node]=0;
      return;
    }
    if(rep[node]!=0) propagate(node,l,r);
    propagate2(node,l,r);
    update(node*2,l,(l+r)/2,a,b,u);
    update(node*2+1,(l+r)/2 +1,r,a,b,u);
    tree[node]=tree[node*2]+tree[node*2+1];
  }
  void update2(int node,int l,int r,int a,int b,int u){
    if(b<l || a>r) return;
    if(l>=a && r<=b){
      tree[node]+=(r-l+1)*u;
      prop[node]+=u;
      return;
    }
    if(rep[node]!=0) propagate(node,l,r);
    propagate2(node,l,r);
    update2(node*2,l,(l+r)/2,a,b,u);
    update2(node*2+1,(l+r)/2 +1,r,a,b,u);
    tree[node]=tree[node*2]+tree[node*2+1];
  }
  int query(int node,int l,int r,int a,int b){
    if(r<a || l>b) return 0;
    if(l>=a && r<=b) return tree[node];
    if(rep[node]!=0) propagate(node,l,r);
    propagate2(node,l,r);
    return query(node*2,l,(l+r)/2,a,b)+query(node*2+1,(l+r)/2
        +1,r,a,b);
  }
}sg;
// Range Maximize
void propagate(int node,int l,int r){
    prop[node*2]=max(prop[node*2],prop[node]);
    prop[node*2+1]=max(prop[node*2+1],prop[node]);
    int u=prop[node];
    prop[node]=0;
    int mid=(l+r)/2;
    trmx[node*2]=max(trmx[node*2],u);
    trmn[node*2]=max(trmn[node*2],u);
    trmx[node*2+1]=max(trmx[node*2+1],u);
    trmn[node*2+1]=max(trmn[node*2+1],u);
}
```

## HashMap.cpp

6ef81c, 31 lines

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
const int RANDOM = chrono::high_resolution_clock::now().
    time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<int, int, chash> table;

struct custom_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM = chrono::steady_clock::
        now().time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};

gp_hash_table<int, int, custom_hash> mp;

//For Pair
struct chash {
    int operator()(pii x) const { return x.first* 31 + x.second
        ; }
};
gp_hash_table<pii, int, chash> table;
```

## FenwickTree.h

**Description:** Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.
**Time:** Both operations are $\mathcal{O}(\log N)$.

e62fac, 22 lines

```
struct FT {
  vector<ll> s;
  FT(int n) : s(n) {}
  void update(int pos, ll dif) { // a[pos] += dif
    for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
  }
  ll query(int pos) { // sum of values in [0, pos)
    ll res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos-1];
    return res;
  }
  int lower_bound(ll sum) {// min pos st sum of [0, pos] >= sum
    // Returns n if no sum is >= sum, or -1 if empty sum is.
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
      if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
        pos += pw, sum -= s[pos-1];
    }
    return pos;
  }
};
```

## FenwickTree2d.h

**Description:** Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).
**Time:** $\mathcal{O}\left(\log^2 N\right)$. (Use persistent segment trees for $\mathcal{O}(\log N)$.)

"FenwickTree.h" 157f07, 22 lines

```
struct FT2 {
  vector<vi> ys; vector<FT> ft;
  FT2(int limx) : ys(limx) {}
  void fakeUpdate(int x, int y) {
    for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
  }
  void init() {
    for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
  }
  int ind(int x, int y) {
    return (int)(lower_bound(all(ys[x]), y) - ys[x].begin()); }
  void update(int x, int y, ll dif) {
    for (; x < sz(ys); x |= x + 1)
      ft[x].update(ind(x, y), dif);
  }
  ll query(int x, int y) {
    ll sum = 0;
    for (; x; x &= x - 1)
      sum += ft[x-1].query(ind(x-1, y));
    return sum;
  }
};
```

## FenwickTreeRange.cpp

babb40, 77 lines

```
struct BIT {
  long long M[N], A[N];
  BIT() {
    memset(M, 0, sizeof M);
    memset(A, 0, sizeof A);
  }
  void update(int i, long long mul, long long add) {
    while (i < N) {
      M[i] += mul;
      A[i] += add;
      i |= (i + 1);
    }
  }
  void upd(int l, int r, long long x) {
    update(l, x, -x * (l - 1));
    update(r, -x, x * r);
  }
  long long query(int i) {
    long long mul = 0, add = 0;
    int st = i;
    while (i >= 0) {
      mul += M[i];
      add += A[i];
      i = (i & (i + 1)) - 1;
    }
    return (mul * st + add);
  }
  long long query(int l, int r) {
    return query(r) - query(l - 1);
  }
} t;

struct BIT2D {
  long long M[N][N][2], A[N][N][2];
  BIT2D() {
    memset(M, 0, sizeof M);
    memset(A, 0, sizeof A);
  }
  void upd2(long long t[N][N][2], int x, int y, long long mul,
      long long add) {
    for(int i = x; i < N; i += i & -i) {
      for(int j = y; j < N; j += j & -j) {
        t[i][j][0] += mul;
        t[i][j][1] += add;
      }
    }
  }
```

```
  void upd1(int x, int y1, int y2, long long mul, long long add
      ) {
    upd2(M, x, y1, mul, -mul * (y1 - 1));
    upd2(M, x, y2, -mul, mul * y2);
    upd2(A, x, y1, add, -add * (y1 - 1));
    upd2(A, x, y2, -add, add * y2);
  }
  void upd(int x1, int y1, int x2, int y2, long long val) {
    upd1(x1, y1, y2, val, -val * (x1 - 1));
    upd1(x2, y1, y2, -val, val * x2);
  }
  long long query2(long long t[N][N][2], int x, int y) {
    long long mul = 0, add = 0;
    for(int i = y; i > 0; i -= i & -i) {
      mul += t[x][i][0];
      add += t[x][i][1];
    }
    return mul * y + add;
  }
  long long query1(int x, int y) {
    long long mul = 0, add = 0;
    for(int i = x; i > 0; i -= i & -i) {
      mul += query2(M, i, y);
      add += query2(A, i, y);
    }
    return mul * x + add;
  }
  long long query(int x1, int y1, int x2, int y2) {
    return query1(x2, y2) - query1(x1 - 1, y2) - query1(x2, y1
        - 1) + query1(x1 - 1, y1 - 1);
  }
} t;
```

## PersistentSegmentTree.cpp

3291e2, 86 lines

```
const int N = 200005;

int n,q,a[N];

struct perseg{
  struct node{
    int l,r,val;
    node(){
      l=r=val=0;
    }
  }t[40*N];

  int id=0;
  int build(int b,int e){
    int cur=++id;
    if(b==e){
      t[cur].val=0;
      return cur;
    }
    int mid=(b+e)>>1;
    t[cur].l=build(b,mid);
    t[cur].r=build(mid+1,e);
    t[cur].val=t[t[cur].l].val+t[t[cur].r].val;
    return cur;
  }
  int upd(int pre,int b,int e,int i,int v){
    int cur=++id;
    t[cur]=t[pre];
    if(b==e){
      t[cur].val+=v;
      return cur;
    }
    int mid=(b+e)>>1;
    if(i<=mid){
      t[cur].l=upd(t[pre].l,b,mid,i,v);
    }
    else{
      t[cur].r=upd(t[pre].r,mid+1,e,i,v);
    }
    t[cur].val=t[t[cur].l].val+t[t[cur].r].val;
    return cur;
  }
  int qry(int cur,int b,int e,int p,int q){
    if(b>q || e<p) return 0;
    if(b>=p && e<=q) return t[cur].val;
    int mid=(b+e)>>1;
    return qry(t[cur].l,b,mid,p,q)+qry(t[cur].r,mid+1,e,p,q);
  }
  int findkth(int lef,int rig,int b,int e,int k){
    if(b==e) return b;
    int mid=(b+e)>>1;
    int cnt=t[t[rig].l].val-t[t[lef].l].val;
    if(cnt>=k) return findkth(t[lef].l,t[rig].l,b,mid,k);
    else return findkth(t[lef].r,t[rig].r,mid+1,e,k-cnt);
  }
}sg;

vector<int> ver;

void solve(){
  cin >> n >> q;
  for(int i=1;i<=n;i++) cin >> a[i];

  int root=sg.build(1,n);
  ver.push_back(root);

  while(q--){
    int t,k;
    cin >> t >> k;
    if(t==1){
      int a,x;
      cin >> a >> x;
      ver[k-1]=sg.upd(ver[k-1],1,n,a,x);
    }
    else if(t==2){
      int a,b;
      cin >> a >> b;
      cout << sg.qry(ver[k-1],1,n,a,b) << endl;
    }
    else{
      ver.push_back(ver[k-1]);
    }
  }
}

// CSES — Range Queries and Copies
```

## ImplicitSegmentTree.cpp

37efa4, 46 lines

```
struct node{
  int sum,prop;
  node *l,*r;
  node(): sum(0),prop(0),l(NULL),r(NULL) { }
};
struct segtree{

  void propagate(node *v,int l,int r){
    v->l->prop=v->prop;
    v->r->prop=v->prop;
    int xx=v->prop;
    v->prop=0;
    int mid=(l+r)/2;
    v->l->sum=(mid-l+1)*xx;
    v->r->sum=(r-(mid+1)+1)*xx;
  }
  void update(node *v,int l,int r,int a,int b,int u){
    if(b<l || a>r) return;
    if(l>=a && r<=b){
      v -> sum = (r-l+1)*u;
      v -> prop =u;
      return;
    }
    if(v->l==NULL) v->l=new node();
    if(v->r==NULL) v->r=new node();
    if(v->prop!=0) propagate(v,l,r);
    update(v->l,l,(l+r)/2,a,b,u);
    update(v->r,(l+r)/2 +1,r,a,b,u);
    v -> sum= v->l->sum + v->r->sum;
  }
  int query(node *v,int l,int r,int a,int b){
    if(r<a || l>b) return 0;
    if(l>=a && r<=b){
      return v -> sum;
    }
    if(v->prop!=0) propagate(v,l,r);
    return query(v->l,l,(l+r)/2,a,b)+query(v->r,(l+r)/2 +1,r,a,
        b);
  }
}sg;

void solve(){
  node *root =new node();
  sg.update(root,1,1000000005,a,b,1);
  sg.query(root,1,1000000000,1,1000000005);
}
```

## MergeSortTree.cpp

17d95e, 29 lines

```
struct mergesortree{
  vll tree[4*N];
  vll merge(vll x,vll y){
      int i=0,j=0,k;
      vll temp;
      while(temp.size()<(x.size()+y.size())){
      if(i==x.size()) temp.push_back(y[j++]);
      else if(j==y.size()) temp.push_back(x[i++]);
      else if(x[i]<y[j]) temp.push_back(x[i++]);
      else temp.push_back(y[j++]);
    }
      return temp;
  }

  void init(int node,int l,int r){
      if(l==r){
          tree[node].push_back(a[l]);
          return;
      }
      init(2*node,l,(l+r)/2);
      init(2*node+1,(l+r)/2+1,r);
      tree[node]=merge(tree[2*node],tree[2*node+1]);
  }
  int query(int node,int l,int r,int x,int y,int k){
      if(r<x || l>y) return 0;
      if(x<=l && r<=y) return upper_bound(all(tree[node]),k)-
          tree[node].begin();
      return query(2*node,l,(l+r)/2,x,y,k)+query(2*node+1,(l+r)
          /2+1,r,x,y,k);
  }
}sg;
```

## MO'sAlgo.cpp

962a45, 35 lines

```
struct query{
  int l, r, id;
} q[maxn];

const int k = 320; // As sqrt(100000) = ~320
                   // I recommand setting the max block size
                   // for the problem at the beginning.
                   // Somehow it fastens up runtime.

bool cmp(query &a, query &b) {
  int block_a = a.l / k, block_b = b.l / k;
  if(block_a == block_b) return a.r < b.r;
  return block_a < block_b;
}

int l = 0, r = -1, sum = 0, ans[maxn];

void add(int x) { sum += a[x]; }
void remove(int x) { sum -= a[x]; }

int main() {
  // do stuff, take input etc...
  for(int i = 0; i < Q; i++) {
    cin >> q[i].l >> q[i].r;
    q[i].id = i;
  }
  sort(q, q+Q, cmp);
  for(int i = 0; i < Q; i++) {
    while(l > q[i].l) add(--l);
    while(r < q[i].r) add(++r);
    while(l < q[i].l) remove(l++);
    while(r > q[i].r) remove(r--);
    ans[q[i].id] = sum;
  }
}
```

## Treap.cpp

e9b3fb, 110 lines

```
mt19937 rnd(chrono::steady_clock::now().time_since_epoch().
    count());
struct Node {
    int v, p, sz, sum; bool rev;
    Node *l, *r;
    Node(int _v):v(_v),p(rnd()),sz(1),sum(_v),rev(0),l(nullptr)
        ,r(nullptr){}
};
int sz(Node* t){ return t ? t->sz : 0; }
int sm(Node* t){ return t ? t->sum : 0; }
void upd(Node* t){
    t->sz = 1 + sz(t->l) + sz(t->r);
    t->sum = t->v + sm(t->l) + sm(t->r);
}
void push(Node* t){
    if (t && t->rev) {
        swap(t->l, t->r);
        if (t->l) t->l->rev ^= 1;
        if (t->r) t->r->rev ^= 1;
        t->rev = 0;
    }
}
// split t into [0..k-1] -> a, [k..] -> b
void split(Node* t, int k, Node*& a, Node*& b){
    if (!t) { a = b = nullptr; return; }
    push(t);
    if (sz(t->l) >= k) {
        split(t->l, k, a, t->l);
        b = t;
    } else {
        split(t->r, k - sz(t->l) - 1, t->r, b);
        a = t;
    }
    upd(t);
}
// merge a,b
Node* merge(Node* a, Node* b){
    if (!a || !b) return a ? a : b;
    push(a); push(b);
    if (a->p > b->p) {
        a->r = merge(a->r, b);
        upd(a);
        return a;
    } else {
        b->l = merge(a, b->l);
        upd(b);
        return b;
    }
}
void insert(Node*& t, int pos, int v){
    Node *a, *b;
    split(t, pos, a, b);
    t = merge(merge(a, new Node(v)), b);
}
void erase(Node*& t, int pos){
    Node *a, *m, *b;
    split(t, pos, a, m);
    split(m, 1, m, b);
    delete m;
    t = merge(a, b);
}
int get(Node* t, int pos){
    push(t);
    int l = sz(t->l);
    if (pos < l) return get(t->l, pos);
    if (pos == l) return t->v;
    return get(t->r, pos - l - 1);
}
void setv(Node* t, int pos, int v){
    push(t);
    int l = sz(t->l);
    if (pos < l) setv(t->l, pos, v);
    else if (pos == l) t->v = v;
    else setv(t->r, pos - l - 1, v);
    upd(t);
}
void reverse(Node*& t, int L, int R){
    Node *a, *b, *c;
    split(t, R+1, t, c);
    split(t, L, a, b);
    if (b) b->rev ^= 1;
    t = merge(merge(a, b), c);
}
int range_sum(Node*& t, int L, int R){
    Node *a, *b, *c;
    split(t, R+1, t, c);
    split(t, L, a, b);
    int res = sm(b);
    t = merge(merge(a, b), c);
    return res;
}
void print(Node* t) {
    if (!t) return;
    push(t);
    print(t->l);
    cout << t->v << ' ';
    print(t->r);
}
int main(){
    Node* root = nullptr;
    for(int i = 1; i <= 4; i++)
        insert(root, i-1, i);
    cout << "Initial: "; print(root); cout << "\n";
    cout << "get(2): " << get(root, 2) << "\n";   // 3
    setv(root, 2, 10);
    cout << "After setv(2,10): "; print(root); cout << "\n";
    reverse(root, 1, 3);
    cout << "After reverse(1,3): "; print(root); cout << "\n";
    cout << "sum[1,2]: " << range_sum(root, 1, 2) << "\n";
    erase(root, 2);
    cout << "After erase(2): "; print(root); cout << "\n";
}
```

# Dynamic-Programming (4)

## dncOptimization.cpp

7b22c3, 38 lines

```
int m, n;
vector<long long> dp_before, dp_cur;

long long C(int i, int j);

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;

    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};

    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {(k ? dp_before[k - 1] : 0) + C(k, mid
            ), k});
    }

    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}

long long solve() {
    dp_before.assign(n,0);
    dp_cur.assign(n,0);

    for (int i = 0; i < n; i++)
        dp_before[i] = C(0, i);

    for (int i = 1; i < m; i++) {
        compute(0, n - 1, 0, n - 1);
        dp_before = dp_cur;
    }

    return dp_before[n - 1];
}
```

## LineContainer.h

**Description:** Container where you can add lines of the form kx+m, and query maximum values at points x. Useful for dynamic programming ("convex hull trick").
**Time:** $\mathcal{O}(\log N)$

8ec1c7, 30 lines

```
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
```

```
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

## SOSDP.cpp

301b6f, 20 lines

```
vector<int> freq(SZ, 0);
    for(int x : a) freq[x]++;

    // g[mask] = sum of freq[submask] for submask (subset of)
        mask
    vector<int> g = freq;
    for(int i = 0; i < M; i++){
        for(int mask = 0; mask < SZ; mask++){
            if(mask & (1<<i))
                g[mask] += g[mask ^ (1<<i)];
        }
    }

    // h[mask] = sum of freq[supermask] for supermask (superset
        of) mask
    vector<int> h = freq;
    for(int i = 0; i < M; i++){
        for(int mask = 0; mask < SZ; mask++){
            if((mask & (1<<i)) == 0)
                h[mask] += h[mask ^ (1<<i)];
        }
    }
```

## N*sqrt(N)-SubsetSumOptimization.cpp

d2d864, 58 lines

```
const int N = 1e6 + 5; // Max possible sum or value
int n, g[N], r[N], cnt[N];
vector<pair<int, int>> nums; // {value, count}

// Solves bounded subset sum: find all possible sums with at
    most cnt[i] times value i
void solve() {
    cin >> n;

    int total_sum = 0;
    int overall_gcd = 0;
    bool has_zero = false;

    // Count frequencies of g[i], compute total sum and GCD of
        all r[i]
    for (int i = 1; i <= n; ++i) {
        cin >> g[i] >> r[i];
        cnt[g[i]]++;
        total_sum += g[i];
        overall_gcd = __gcd(overall_gcd, r[i]);
        if (g[i] == 0) has_zero = true;
    }

    // Collect unique {value, count} pairs
    for (int i = 1; i < N; ++i) {
        if (cnt[i]) nums.pb({i, cnt[i]});
    }

    vector<int> dp(N, 0); // dp[i] = 1 means sum 'i' is
        reachable
    dp[0] = 1;

    // SOS-style bounded knapsack using modular buckets
    for (auto [val, freq] : nums) {
        vector<int> new_dp = dp;

        for (int rem = 0; rem < val; ++rem) {
            int sum = 0, window = 0;

            for (int j = rem; j < N; j += val) {
                if (window > freq) {
                    sum -= dp[j - val * window];
                    window--;
                }
                if (sum > 0) new_dp[j] = 1;
                sum += dp[j];
                window++;
            }
        }
        swap(dp, new_dp);
    }

    // Example output: list all reachable sums
    // for (int i = 0; i <= total_sum; ++i)
    //     if (dp[i]) cout << i << " ";

    // You can now use dp[i] for queries like:
    // - Is sum 'S' achievable?
    // - Count total achievable sums
    // etc.
}
```

## LIS.cpp

ac57b7, 19 lines

```
int lis(vector<int> const& a) {
    int n = a.size();
    const int INF = 1e9;
    vector<int> d(n+1, INF);
    d[0] = -INF;

    for (int i = 0; i < n; i++) {
        int l = upper_bound(d.begin(), d.end(), a[i]) - d.begin
            ();
        if (d[l-1] < a[i] && a[i] < d[l])
            d[l] = a[i];
    }

    int ans = 0;
    for (int l = 0; l <= n; l++) {
        if (d[l] < INF)
            ans = l;
    }
    return ans;
}
```

# Numerical (5)

## FFT.cpp

**Description:** Iterative Implementation of FFT and FFTanymod. Complexity: O(N log N) 1. Whenever possible remove leading zeros. 2. Custom Complex class may slightly improve performance. 3. Use pairfft to do two ffts of real vectors at once, slightly less accurate than doing two ffts, but faster by about 304. FFT accuracy depends on answer. x <= 5e14 (double), x <= 1e18(long double) where x = max(ans[i]) for FFT, and x = N*mod for anymod

fd1b3b, 145 lines

```
//typedef complex<double> CD;
struct CD {
    double x, y;
    CD(double x=0, double y=0) :x(x), y(y) {}
    CD operator+(const CD& o) { return {x+o.x, y+o.y};}
    CD operator-(const CD& o) { return {x-o.x, y-o.y};}
    CD operator*(const CD& o) { return {x*o.x-y*o.y, x*o.y+o.x*
        y};}
    void operator /= (double d) { x/=d; y/=d;}
    double real() {return x;}
    double imag() {return y;}
};
CD conj(const CD &c) {return CD(c.x, -c.y);}

typedef long long LL;
const double PI = acos(-1.0L);

namespace FFT {
    int N;
    vector<int> perm;
    vector<CD> wp[2];

    void precalculate(int n) {
        assert((n & (n-1)) == 0);
        N = n;
        perm = vector<int> (N, 0);
        for (int k=1; k<N; k<<=1) {
            for (int i=0; i<k; i++) {
                perm[i] <<= 1;
                perm[i+k] = 1 + perm[i];
            }
        }

        wp[0] = wp[1] = vector<CD>(N);
        for (int i=0; i<N; i++) {
            wp[0][i] = CD( cos(2*PI*i/N),  sin(2*PI*i/N) );
            wp[1][i] = CD( cos(2*PI*i/N), -sin(2*PI*i/N) );
        }
    }

    void fft(vector<CD> &v, bool invert = false) {
        if (v.size() != perm.size())    precalculate(v.size());

        for (int i=0; i<N; i++)
            if (i < perm[i])
                swap(v[i], v[perm[i]]);

        for (int len = 2; len <= N; len *= 2) {
            for (int i=0, d = N/len; i<N; i+=len) {
                for (int j=0, idx=0; j<len/2; j++, idx += d) {
                    CD x = v[i+j];
                    CD y = wp[invert][idx]*v[i+j+len/2];
                    v[i+j] = x+y;
                    v[i+j+len/2] = x-y;
                }
            }
        }
```

```
        if (invert) {
            for (int i=0; i<N; i++) v[i]/=N;
        }
    }

     void pairfft(vector<CD> &a, vector<CD> &b, bool invert =
          false) {
        int N = a.size();
        vector<CD> p(N);
        for (int i=0; i<N; i++) p[i] = a[i] + b[i] * CD(0, 1);
        fft(p, invert);
        p.push_back(p[0]);

        for (int i=0; i<N; i++) {
            if (invert) {
                a[i] = CD(p[i].real(), 0);
                b[i] = CD(p[i].imag(), 0);
            }
            else {
                a[i] = (p[i]+conj(p[N-i]))*CD(0.5, 0);
                b[i] = (p[i]-conj(p[N-i]))*CD(0, -0.5);
            }
        }
    }

    vector<LL> multiply(const vector<LL> &a, const vector<LL> &
         b) {
        int n = 1;
        while (n < a.size()+ b.size())  n<<=1;

        vector<CD> fa(a.begin(), a.end()), fb(b.begin(), b.end
             ());
        fa.resize(n); fb.resize(n);

//          fft(fa); fft(fb);
        pairfft(fa, fb);
        for (int i=0; i<n; i++) fa[i] = fa[i] * fb[i];
        fft(fa, true);

        vector<LL> ans(n);
        for (int i=0; i<n; i++)     ans[i] = round(fa[i].real()
             );
        return ans;
    }

    const int M = 1e9+7, B = sqrt(M)+1;
    vector<LL> anyMod(const vector<LL> &a, const vector<LL> &b)
         {
        int n = 1;
        while (n < a.size()+ b.size())  n<<=1;
        vector<CD> al(n), ar(n), bl(n), br(n);

        for (int i=0; i<a.size(); i++)  al[i] = a[i]%M/B, ar[i]
             = a[i]%M%B;
        for (int i=0; i<b.size(); i++)  bl[i] = b[i]%M/B, br[i]
             = b[i]%M%B;

        pairfft(al, ar); pairfft(bl, br);
//          fft(al); fft(ar); fft(bl); fft(br);

        for (int i=0; i<n; i++) {
            CD ll = (al[i] * bl[i]), lr = (al[i] * br[i]);
            CD rl = (ar[i] * bl[i]), rr = (ar[i] * br[i]);
            al[i] = ll; ar[i] = lr;
            bl[i] = rl; br[i] = rr;
        }

        pairfft(al, ar, true); pairfft(bl, br, true);
//          fft(al, true); fft(ar, true); fft(bl, true); fft(br,
     true);

        vector<LL> ans(n);
        for (int i=0; i<n; i++) {
            LL right = round(br[i].real()), left = round(al[i].
                 real());;
            LL mid = round(round(bl[i].real()) + round(ar[i].
                 real()));
            ans[i] = ((left%M)*B*B + (mid%M)*B + right)%M;
        }
        return ans;
    }
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, m;
    cin>>n>>m;

    vector<LL> a(n), b(m);
    for (int i=0; i<n; i++) cin>>a[i];
    for (int i=0; i<m; i++) cin>>b[i];

    vector<LL> ans = FFT::anyMod(a, b);
    ans.resize(n+m-1);
    for (LL x: ans) cout<<x<<" ";
}
```

## NTT.cpp

9ac0fb, 88 lines

```
namespace NTT {
    vector<int> perm, wp[2];
    const int mod = 998244353, G = 3;
    int root, inv, N, invN;

    int power(int a, int p) {
        int ans = 1;
        while (p) {
            if (p & 1) ans = (1LL*ans*a)%mod;
            a = (1LL*a*a)%mod;
            p >>= 1;
        }
        return ans;
    }

    void precalculate(int n) {
        assert( (n&(n-1)) == 0 && (mod-1)%n==0);
        N = n;
        invN = power(N, mod-2);
        perm = wp[0] = wp[1] = vector<int>(N);

        perm[0] = 0;
        for (int k=1; k<N; k<<=1)
            for (int i=0; i<k; i++) {
                perm[i] <<= 1;
                perm[i+k] = 1 + perm[i];
            }

        root = power(G, (mod-1)/N);
        inv = power(root, mod-2);
        wp[0][0]=wp[1][0]=1;

        for (int i=1; i<N; i++) {
            wp[0][i] = (wp[0][i-1]*1LL*root)%mod;
            wp[1][i] = (wp[1][i-1]*1LL*inv)%mod;
        }
    }

    void fft(vector<int> &v, bool invert = false) {
        if (v.size() != perm.size())   precalculate(v.size());
        for (int i=0; i<N; i++)
            if (i < perm[i])
                swap(v[i], v[perm[i]]);

        for (int len = 2; len <= N; len *= 2) {
            for (int i=0, d = N/len; i<N; i+=len) {
                for (int j=0, idx=0; j<len/2; j++, idx += d) {
                    int x = v[i+j];
                    int y = (wp[invert][idx]*1LL*v[i+j+len/2])%
                         mod;
                    v[i+j] = (x+y>=mod ? x+y-mod : x+y);
                    v[i+j+len/2] = (x-y>=0 ? x-y : x-y+mod);
                }
            }
        }
        if (invert) {
            for (int &x : v) x = (x*1LL*invN)%mod;
        }
    }

    vector<int> multiply(vector<int> a, vector<int> b) {
        int n = 1;
        while (n < a.size()+ b.size())  n<<=1;
        a.resize(n);
        b.resize(n);

        fft(a);
        fft(b);
        for (int i=0; i<n; i++) a[i] = (a[i] * 1LL * b[i])%mod;
        fft(a, true);
        return a;
    }
};

const int M = 998244353;
int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, m;
    cin>>n>>m;

    vector<int> a(n), b(m);
    for (int i=0; i<n; i++)     cin>>a[i];
    for (int i=0; i<m; i++)     cin>>b[i];
    vector<int> c = NTT::multiply(a, b);
    c.resize(n+m-1);
    for (int x: c)   cout<<x<<" ";
}
```

## FastFourierTransform.h

**Description:** fft(a) computes $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all $k$. N must be a power of 2. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back. Rounding is safe if $(\sum a_i^2 + \sum b_i^2) \log_2 N < 9 \cdot 10^{14}$ (in practice $10^{16}$; higher for random inputs). Otherwise, use NTT/FFTMod.
**Time:** $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ ($\sim 1s$ for $N = 2^{22}$)

00ced6, 35 lines

```
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vector<complex<long double>> R(2, 1);
  static vector<C> rt(2, 1);  // (^ 10% faster if double)
```

```
  for (static int k = 2; k < n; k *= 2) {
    R.resize(n); rt.resize(n);
    auto x = polar(1.0L, acos(-1.0L) / k);
    rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
  }
  vi rev(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
  rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
      C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)
      a[i + j + k] = a[i + j] - z;
      a[i + j] += z;
    }
}
vd conv(const vd& a, const vd& b) {
  if (a.empty() || b.empty()) return {};
  vd res(sz(a) + sz(b) - 1);
  int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
  vector<C> in(n), out(n);
  copy(all(a), begin(in));
  rep(i,0,sz(b)) in[i].imag(b[i]);
  fft(in);
  for (C& x : in) x *= x;
  rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
  fft(out);
  rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
  return res;
}
```

## FastFourierTransformMod.h

**Description:** Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice $10^{16}$ or higher). Inputs must be in $[0, \text{mod})$.
**Time:** $\mathcal{O}(N \log N)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT)

"FastFourierTransform.h" b82773, 22 lines

```
typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  vl res(sz(a) + sz(b) - 1);
  int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
  vector<C> L(n), R(n), outs(n), outl(n);
  rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
  rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
  fft(L), fft(R);
  rep(i,0,n) {
    int j = -i & (n - 1);
    outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
  }
  fft(outl), fft(outs);
  rep(i,0,sz(res)) {
    ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
    ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
  }
  return res;
}
```

## FastSubsetTransform.h

**Description:** Transform to a basis with fast convolutions of the form $c[z] = \sum_{z=x \oplus y} a[x] \cdot b[y]$, where $\oplus$ is one of AND, OR, XOR. The size of $a$ must be a power of two.
**Time:** $\mathcal{O}(N \log N)$

464cf3, 16 lines

```
void FST(vi& a, bool inv) {
  for (int n = sz(a), step = 1; step < n; step *= 2) {
    for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step) {
      int &u = a[j], &v = a[j + step]; tie(u, v) =
        inv ? pii(v - u, u) : pii(v, u + v); // AND
        inv ? pii(v, u - v) : pii(u + v, u); // OR
        pii(u + v, u - v);                   // XOR
    }
  }
  if (inv) for (int& x : a) x /= sz(a); // XOR only
}
vi conv(vi a, vi b) {
  FST(a, 0); FST(b, 0);
  rep(i,0,sz(a)) a[i] *= b[i];
  FST(a, 1); return a;
}
```

# Number theory (6)

## LinearSieve.cpp

f222d8, 15 lines

```
const int N = 10000000;
vector<int> lp(N+1);
vector<int> pr;
for (int i=2; i <= N; ++i) {
    if (lp[i] == 0) {
        lp[i] = i;
        pr.push_back(i);
    }
    for (int j = 0; i * pr[j] <= N; ++j) {
        lp[i * pr[j]] = pr[j];
        if (pr[j] == lp[i]) {
            break;
        }
    }
}
```

## Congruence.cpp

<bits/stdc++.h> e1e12c, 175 lines

```
using namespace std;
typedef long long LL;
typedef pair<LL, LL> PLL;

inline LL modit(LL x, LL m) {
  LL z = x%m; return z<0 ? z+m : z;
}

LL gcd(LL u, LL v) {
    if (u == 0) return v;
    if (v == 0) return u;
    int shift = __builtin_ctzll(u | v);
    u >>= __builtin_ctzll(u);
    do {
        v >>= __builtin_ctzll(v);
        if (u > v) swap(u, v);
        v = v-u;
    } while (v);
    return u << shift;
}

LL power(LL a, LL b, LL m) {
    a = modit(a, m);
  LL ans = 1;
  while (b) {
    if (b & 1) ans = (ans*a)%m;
    a = (a*a)%m;
    b >>= 1;
  }
  return ans;
}

LL egcd(LL a, LL b, LL &x, LL &y) {
  LL xx = y = 0;
  LL yy = x = 1;
  while (b) {
    LL q = a/b;
    LL t = b; b = a%b; a = t;
    t = xx; xx = x-q*xx; x = t;
    t = yy; yy = y-q*yy; y = t;
  }
  return a;
}
LL inverse(LL a, LL m) {
  LL x, y;
  LL g = egcd(a, m, x, y);
  if (g > 1) return -1;
  return modit(x, m);
}

PLL CRT(LL m1, LL r1, LL m2, LL r2) {
  LL s, t;
  LL g = egcd(m1, m2, s, t);
  if (r1%g != r2%g) return PLL(0, -1);
  LL M = m1*m2;
  LL ss = ((s*r2)%m2)*m1;
  LL tt = ((t*r1)%m1)*m2;
  LL ans = modit(ss+tt, M);
  return PLL(ans/g, M/g);
}

pair<LL, LL> SolveCongruence(LL a, LL b, LL m) {
  LL x, y;
  LL g = egcd(a, m, x, y);
  if (b%g == 0) {
        LL d = m/g;
    x = modit(x*(b/g), d);
    return {x, d};
  }
  return {-1, -1};
}

bool LinearDiophantine(LL a, LL b, LL c, LL &x, LL &y) {
  if (!a && !b) {
    if (c)          return false;
    x = y = 0;      return true;
  }
  if (!a) {
    if (c%b)        return false;
    x = 0; y = c/b; return true;
  }
  if (!b) {
    if (c%a)        return false;
    x = c/a; y = 0; return true;
  }
  LL g = gcd(a, b);
  if (c%g) return false;
  x = c/g*inverse(a/g, b/g);
  y = (c-a*x)/b;
  return true;
}

LL primitive_root(LL p) {
    if (p == 2) return 1;
    LL phi = p-1, n = phi;

    vector<LL> factor;
    for (int i=2; i*i<=n; ++i)
        if (n%i == 0) {
            factor.push_back (i);
```

```
            while (n%i==0)  n/=i;
        }

    if (n>1)  factor.push_back(n);
    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (int i=0; i<factor.size() && ok; ++i)
            ok &= power(res, phi/factor[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}

int discreteLog(int a, int b, int M) {
    a %= M, b %= M;
    int k = 1, add = 0, g;
    while ((g = gcd(a, M)) > 1) {
        if (b == k)     return add;
        if (b%g)      return -1;
        b/=g, M/=g, ++add;
        k = (1LL*k*a/g)%M;
    }

    int RT = sqrt(M)+1, aRT = 1;
    for (int i=0; i<RT; i++) aRT = (aRT*1LL*a)%M;

    unordered_map<int, int> vals;
    for (int i=0, cur=b; i<=RT; i++) {
        vals[cur] = i;
        cur = (cur*1LL*a)%M;
    }

    for (int i=1, cur=k; i<=M/RT+1; i++) {
        cur = (cur*1LL*aRT)%M;
        if (vals.find(cur) != vals.end())    return RT*i-vals[
            cur]+add;
    }
    return -1;
}

int discreteRoot(int a, int b, int P) {
    if (b%P == 0)     return    a == 0 ? -1 : 0;
    int g = primitive_root(P);
    int y = discreteLog(power(g, a, P), b, P);
    return y == -1 ? -1 : power(g, y, P);
}

// Tonnelli Shanks: finds x st x^2 = a (mod p)
// p must be prime. returns -1 if none.
// complexity: O(log^2 p) worst case, O(log p) on average

int SQRT(int a, int p) {
  a = modit(a, p);
  if (a == 0) return 0;
  if (power(a, (p-1)/2, p) != 1)  return -1;
  if (p%4==3)           return power(a, (p+1)/4, p);
  int s = p-1, n = 2;
  int r = 0, m;
  while (s%2 == 0) ++r, s/=2;
  // find a non-square mod p
  while (power(n, (p-1)/2, p) != p-1) ++n;
  int x = power(a, (s+1)/2, p);
  int b = power(a, s, p), g = power(n, s, p);
  for (;; r = m) {
    int t = b;
    for (m = 0; m<r && t!=1; ++m) t = 1LL*t*t%p;
    if (m == 0) return x;
    int gs = power(g, 1LL << (r-m-1), p);
    g = 1LL*gs*gs%p;
    x = 1LL*x*gs%p;
    b = 1LL*b*g%p;
  }
}
```

## PollardRho.cpp

cd0d20, 53 lines

```
namespace Rho {
    ULL mult(ULL a, ULL b, ULL mod) {
        LL ret = a * b - mod * (ULL)(1.0L / mod * a * b);
        return ret + mod * (ret < 0) - mod * (ret >= (LL) mod);
    }

    ULL power(ULL x, ULL p, ULL mod){
        ULL s=1, m=x;
        while(p) {
            if(p&1) s = mult(s, m, mod);
            p>>=1;
            m = mult(m, m, mod);
        }
        return s;
    }

    vector<LL> bases = {2, 325, 9375, 28178, 450775, 9780504,
         1795265022};
    bool isprime(LL n) {
        if (n<2)    return 0;
        if (n%2==0)   return n==2;

        ULL s = __builtin_ctzll(n-1), d = n>>s;
        for (ULL x: bases) {
            ULL p = power(x%n, d, n), t = s;
            while (p!=1 && p!=n-1 && x%n && t--) p = mult(p, p,
                 n);
            if (p!=n-1 && t != s)               return 0;
        }
        return 1;
    }

    mt19937_64 rng(chrono::system_clock::now().time_since_epoch
         ().count());
    ULL FindFactor(ULL n) {
        if (n == 1 || isprime(n))    return n;
        ULL c = 1, x = 0, y = 0, t = 0, prod = 2, x0 = 1, q;
        auto f = [&](ULL X) { return mult(X, X, n) + c;};

        while (t++ % 128 or gcd(prod, n) == 1) {
            if (x == y) c = rng()%(n-1)+1, x = x0, y = f(x);
            if ((q = mult(prod, max(x, y) - min(x, y), n)))
                 prod = q;
            x = f(x), y = f(f(y));
        }
        return gcd(prod, n);
    }

    vector<ULL> factorize(ULL x) {
        if (x == 1)     return {};
        ULL a = FindFactor(x), b = x/a;
        if (a == x) return {a};
        vector<ULL> L = factorize(a), R = factorize(b);
        L.insert(L.end(), R.begin(), R.end());
        return L;
    }
}
```

## MillerRabin.h

**Description:** Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$; for larger numbers, use Python and extend A randomly.

**Time:** 7 times the complexity of $a^b \mod c$.

"ModMulLL.h" 60dcd1, 12 lines

```
bool isPrime(ull n) {
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
      s = __builtin_ctzll(n-1), d = n >> s;
  for (ull a : A) {   // ^ count trailing zeroes
    ull p = modpow(a%n, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n && i--)
      p = modmul(p, p, n);
    if (p != n-1 && i != s) return 0;
  }
  return 1;
}
```

## FloorSum.cpp

df6b13, 13 lines

```
typedef long long LL;
LL mod(LL a, LL m) {
    LL ans = a%m;
    return ans < 0 ? ans+m : ans;
}

LL floorSum(LL n, LL m, LL a, LL b) {
    LL ra = mod(a, m), rb = mod(b, m), k = (ra*n+rb);
    LL ans = ((a-ra)/m) * n*(n-1)/2 + ((b-rb)/m) * n;
    if (k < m)  return ans;
    return ans + floorSum(k/m, ra, m, k%m);

}
```

## Matrix.cpp

9fdad0, 60 lines

```
struct Mat {
  int n, m;
  vector<vector<int>> a;
  Mat() { }
  Mat(int _n, int _m) {n = _n; m = _m; a.assign(n, vector<int>(
       m, 0)); }
  Mat(vector< vector<int> > v) { n = v.size(); m = n ? v[0].
       size() : 0; a = v; }
  inline void make_unit() {
    assert(n == m);
    for (int i = 0; i < n; i++)  {
      for (int j = 0; j < n; j++) a[i][j] = i == j;
    }
  }
  inline Mat operator + (const Mat &b) {
    assert(n == b.n && m == b.m);
    Mat ans = Mat(n, m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < m; j++) {
        ans.a[i][j] = (a[i][j] + b.a[i][j]) % mod;
      }
    }
    return ans;
  }
  inline Mat operator - (const Mat &b) {
    assert(n == b.n && m == b.m);
    Mat ans = Mat(n, m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < m; j++) {
        ans.a[i][j] = (a[i][j] - b.a[i][j] + mod) % mod;
      }
    }
    return ans;
  }
  inline Mat operator * (const Mat &b) {
    assert(m == b.n);
```

```
    Mat ans = Mat(n, b.m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < b.m; j++) {
        for(int k = 0; k < m; k++) {
          ans.a[i][j] = (ans.a[i][j] + 1LL * a[i][k] * b.a[k][j
              ] % mod) % mod;
        }
      }
    }
    return ans;
  }
  inline Mat pow(long long k) {
    assert(n == m);
    Mat ans(n, n), t = a; ans.make_unit();
    while (k) {
      if (k & 1) ans = ans * t;
      t = t * t;
      k >>= 1;
    }
    return ans;
  }
  inline Mat& operator += (const Mat& b) { return *this = (*
      this) + b; }
  inline Mat& operator -= (const Mat& b) { return *this = (*
      this) - b; }
  inline Mat& operator *= (const Mat& b) { return *this = (*
      this) * b; }
  inline bool operator == (const Mat& b) { return a == b.a; }
  inline bool operator != (const Mat& b) { return a != b.a; }
};
```

## sieve(1e9).cpp

e09a32, 44 lines

```
const int MAXPR = (int)(1e4 + 5);
vector<int> primes;

void calcPrimes() {
    primes.push_back(2);
    auto sum = 2LL;
    int cnt = 1;

    const int S = round(sqrt(MAXPR));
    vector<char> sieve(S + 1, true);
    vector<array<int, 2>> cp;

    for (int i = 3; i < S; i += 2) {
        if (!sieve[i]) continue;
        cp.push_back({i, (i * i - 1) / 2});
        for (int j = i * i; j <= S; j += 2 * i)
            sieve[j] = false;
    }

    vector<char> block(S);
    int high = (MAXPR - 1) / 2;

    for (int low = 0; low <= high; low += S) {
        fill(block.begin(), block.end(), true);

        for (auto &i : cp) {
            int p = i[0], idx = i[1];
            for (; idx < S; idx += p)
                block[idx] = false;
            i[1] = idx - S;
        }

        if (low == 0)
            block[0] = false;

        for (int i = 0; i < S && low + i <= high; i++) {
            if (block[i]) {
                ++cnt;
                sum += (low + i) * 2 + 1;
                primes.push_back((low + i) * 2 + 1);
            }
        }
    }
}
```

## GaussianElimination.cpp

f94400, 35 lines

```
const int N = 3e5 + 9;

const double eps = 1e-9;
int Gauss(vector<vector<double>> a, vector<double> &ans) {
  int n = (int)a.size(), m = (int)a[0].size() - 1;
  vector<int> pos(m, -1);
  double det = 1; int rank = 0;
  for(int col = 0, row = 0; col < m && row < n; ++col) {
    int mx = row;
    for(int i = row; i < n; i++) if(fabs(a[i][col]) > fabs(a[mx
        ][col])) mx = i;
    if(fabs(a[mx][col]) < eps) {det = 0; continue;}
    for(int i = col; i <= m; i++) swap(a[row][i], a[mx][i]);
    if (row != mx) det = -det;
    det *= a[row][col];
    pos[col] = row;
    for(int i = 0; i < n; i++) {
      if(i != row && fabs(a[i][col]) > eps) {
        double c = a[i][col] / a[row][col];
        for(int j = col; j <= m; j++) a[i][j] -= a[row][j] * c;
      }
    }
    ++row; ++rank;
  }
  ans.assign(m, 0);
  for(int i = 0; i < m; i++) {
    if(pos[i] != -1) ans[i] = a[pos[i]][m] / a[pos[i]][i];
  }
  for(int i = 0; i < n; i++) {
    double sum = 0;
    for(int j = 0; j < m; j++) sum += ans[j] * a[i][j];
    if(fabs(sum - a[i][m]) > eps) return -1; //no solution
  }
  for(int i = 0; i < m; i++) if(pos[i] == -1) return 2; //
      infinte solutions
  return 1; //unique solution
}
```

## GaussianEliminationMod2.cpp

190541, 36 lines

```
using Matrix = vector<vector<bool>>;

vector<bool> gaussianEliminationMod2(Matrix& A, int n, int m) {
    vector<bool> solution(n, false);
    int row = 0;

    for (int col = 0; col < n && row < m; ++col) {
        int pivot = row;
        while (pivot < m && !A[pivot][col]) {
            ++pivot;
        }

        if (pivot == m) continue;

        swap(A[row], A[pivot]);

        for (int i = 0; i < m; ++i) {
            if (i != row && A[i][col]) {
                for (int j = col; j <= n; ++j) {
                    A[i][j] ^= A[row][j]; // XOR for GF(2)
                }
            }
        }
        ++row;
    }

    for (int i = row; i < m; ++i) {
        if (A[i][n]) return {}; // No solution
    }

    for (int i = 0; i < n; ++i) {
        solution[i] = A[i][n];
    }

    return solution;
}
```

## phiFunction.h

**Description:** *Euler's* $\phi$ function is defined as $\phi(n) :=$ # of positive integers $\leq n$ that are coprime with $n$. $\phi(1) = 1$, $p$ prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, $m, n$ coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1} p_2^{k_2} ... p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1 - 1} ... (p_r - 1)p_r^{k_r - 1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.
$\sum_{d|n} \phi(d) = n$, $\sum_{1 \leq k \leq n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$
**Euler's thm**: $a, n$ coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod{n}$.
**Fermat's little thm**: $p$ prime $\Rightarrow a^{p-1} \equiv 1 \pmod{p}$ $\forall a$.

cf7d6d, 8 lines

```
const int LIM = 5000000;
int phi[LIM];

void calculatePhi() {
  rep(i,0,LIM) phi[i] = i&1 ? i : i/2;
  for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
    for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
}
```

## XORBasis.cpp

**Description:** finds a xor basis such that each term is from v

862830, 23 lines

```
vector<int> xorBasis(const vector<int>& v) {
    const int BITS = 32;
    vector<int> basis(BITS, 0);
    vector<int> orig(BITS, 0);
    for (int x : v) {
        int cur = x;
        for (int i = BITS - 1; i >= 0; i--) {
            if (!(cur & (1 << i))) continue;
            if (!basis[i]) {
                basis[i] = cur;
                orig[i] = x;
                break;
            }
            cur ^= basis[i];
        }
    }
    vector<int> result;
    for (int i = 0; i < BITS; i++) {
        if (orig[i] != 0)
            result.push_back(orig[i]);
    }
    return result;
}
```

## 6.1 Divisibility

### 6.1.1 Bézout's identity

For $a \neq, b \neq 0$, then $d = gcd(a, b)$ is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If $(x, y)$ is one solution, then all solutions are given by

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), \quad k \in \mathbb{Z}$$

## 6.2 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2), \;\; b = k \cdot (2mn), \;\; c = k \cdot (m^2 + n^2),$$

with $m > n > 0$, $k > 0$, $m \perp n$, and either $m$ or $n$ even.

## 6.3 Primes

$p = 962592769$ is such that $2^{21} \mid p - 1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power $p^a$, except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $\mathbb{Z}_{2^a}^\times$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

## 6.4 Estimates

$\sum_{d|n} d = O(n \log \log n)$.

The number of divisors of $n$ is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

## 6.5 Mobius Function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g(n/d)$$

Other useful formulas/forms:

$\sum_{d|n} \mu(d) = [n = 1]$ (very useful)

$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n) g(d)$

$g(n) = \sum_{1 \le m \le n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \le m \le n} \mu(m) g(\lfloor \frac{n}{m} \rfloor)$

# Combinatorial (7)

## 7.1 Permutations

### 7.1.1 Factorial

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $n!$ | 1 | 2 | 6 | 24 | 120 | 720 | 5040 | 40320 | 362880 | 3628800 |
| $n$ | 11 | 12 | 13 | 14 | 15 | 16 | 17 | | | |
| $n!$ | 4.0e7 | 4.8e8 | 6.2e9 | 8.7e10 | 1.3e12 | 2.1e13 | 3.6e14 | | | |
| $n$ | 20 | 25 | 30 | 40 | 50 | 100 | 150 | 171 | | |
| $n!$ | 2e18 | 2e25 | 3e32 | 8e47 | 3e64 | 9e157 | 6e262 | >DBL_MAX | | |

### 7.1.2 Cycles

Let $g_S(n)$ be the number of $n$-permutations whose cycle lengths all belong to the set $S$. Then

$$\sum_{n=0}^{\infty} g_S(n) \frac{x^n}{n!} = \exp\left(\sum_{n \in S} \frac{x^n}{n}\right)$$

### 7.1.3 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$$

### 7.1.4 Burnside's lemma

Given a group $G$ of symmetries and a set $X$, the number of elements of $X$ *up to symmetry* equals

$$\frac{1}{|G|} \sum_{g \in G} |X^g|,$$

where $X^g$ are the elements fixed by $g$ ($g.x = x$).

If $f(n)$ counts "configurations" (of some sort) of length $n$, we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(\gcd(n, k)) = \frac{1}{n} \sum_{k|n} f(k) \phi(n/k).$$

## 7.2 Partitions and subsets

### 7.2.1 Partition function

Number of ways of writing $n$ as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1, \;\; p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

| $n$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 20 | 50 | 100 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p(n)$ | 1 | 1 | 2 | 3 | 5 | 7 | 11 | 15 | 22 | 30 | 627 | ∼2e5 | ∼2e8 |

### 7.2.2 Lucas' Theorem

Let $n, m$ be non-negative integers and $p$ a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod p$.

### 7.2.3 Binomials

multinomial.h

**Description:** Computes $\binom{k_1 + \cdots + k_n}{k_1, k_2, \ldots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \ldots k_n!}$. a0a312, 5 lines

```
ll multinomial(vi& v) {
  ll c = 1, m = v.empty() ? 1 : v[0];
  rep(i,1,sz(v)) rep(j,0,v[i]) c = c * ++m / (j+1);
  return c;
}
```

## 7.3 General purpose numbers

### 7.3.1 Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t - 1}$ (FFT-able).
$B[0, \ldots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \ldots]$

Sums of powers:

$$\sum_{i=1}^{n} n^m = \frac{1}{m+1} \sum_{k=0}^{m} \binom{m+1}{k} B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^{\infty} f(x)dx - \sum_{k=1}^{\infty} \frac{B_k}{k!} f^{(k-1)}(m)$$

$$\approx \int_m^{\infty} f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

### 7.3.2 Stirling numbers of the first kind

Number of permutations on $n$ items with $k$ cycles.

$$c(n, k) = c(n-1, k-1) + (n-1)c(n-1, k), \;\; c(0, 0) = 1$$
$$\sum_{k=0}^{n} c(n, k) x^k = x(x+1)\ldots(x+n-1)$$

$c(8, k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n, 2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$

### 7.3.3 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly $k$ elements are greater than the previous element. $k$ $j$:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ $j$:s s.t. $\pi(j) \geq j$, $k$ $j$:s s.t. $\pi(j) > j$.

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$$

$$E(n, 0) = E(n, n-1) = 1$$

$$E(n, k) = \sum_{j=0}^{k} (-1)^j \binom{n+1}{j} (k+1-j)^n$$

### 7.3.4 Stirling numbers of the second kind

Partitions of $n$ distinct elements into exactly $k$ groups.

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$

$$S(n,1) = S(n,n) = 1$$

$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

### 7.3.5 Bell numbers

Total number of partitions of $n$ distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$. For $p$ prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$$

### 7.3.6 Labeled unrooted trees

# on $n$ vertices: $n^{n-2}$
# on $k$ existing trees of size $n_i$: $n_1 n_2 \cdots n_k n^{k-2}$
# with degrees $d_i$: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

### 7.3.7 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2}C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with $n$ pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

# Graph (8)

## SCC.cpp

67120c, 44 lines

```
vector<int> g[N],r[N];
stack<int> st;
int vis[N];
vector<int> scc[N];
void dfs1(int x){
  if(vis[x]) return;
  vis[x]=1;
  for(auto u:g[x]){
    if(!vis[u]) dfs1(u);
  }
  st.push(x);
}
void dfs2(int x,int mark){
  if(vis[x]) return;
  vis[x]=1;
  scc[mark].pb(x);
  for(auto u:r[x]){
    if(!vis[u]) dfs2(u,mark);
  }
}
int a[N];
void solve(){
  cin >> n >> m;
  for(int i=1;i<=m;i++){
    int u,v;
    cin >> u >> v;
    g[u].pb(v);
    r[v].pb(u);
  }

  for(int i=1;i<=n;i++) dfs1(i);
  memset(vis,0,sizeof vis);
  int marker=0;
  while(!st.empty()){
    int x=st.top();
    st.pop();
    if(!vis[x]) dfs2(x,marker++);
  }
  cout << marker << endl;
  for(int i=0;i<marker;i++){
    for(int j=0;j<scc[i].size();j++){
    }
  }
}
```

## Bellman-Ford.cpp

b335d5, 17 lines

```
vector<pair<int,pii>> e;
int d[N],par[N];
for(int i=1;i<=n;i++) d[i]=inf;
d[v] = 0;
    while(1){
        bool chk=0;
        for (int j=0;j<m;j++){
          if (d[e[j].ss.ff] < inf){
                  if (d[e[j].ss.ss]>d[e[j].ss.ff]+e[j].ff){
                      d[e[j].ss.ss]=d[e[j].ss.ff]+e[j].ff;
                          chk=1;
                          par[e[j].ss.ss]=e[j].ss.ff;
                  }
                }
        }
        if (!chk) break;
    }
```

## Dijkstra.cpp

733fdf, 23 lines

```
#define mx 100005
vector<int> adj[mx];
vector<ll> cost[mx];
ll dis[mx];

void dijkstra(int src){
    fill(dis,dis+mx,1e16+9);
    dis[src]=0;
    priority_queue<pair<ll,int>> Q;
    Q.push({0,src});
    while(!Q.empty()){
        int u=Q.top().second,x=Q.top().first;
        Q.pop();
        if(dis[u]<-x) continue;
        for(int i=0;i<adj[u].size();i++){
            int v=adj[u][i],c=cost[u][i];
            if(dis[v]>dis[u]+c){
                dis[v]=dis[u]+c;
                Q.push({-dis[v],v});
            }
        }
    }
}
```

## LCA.cpp

c75caa, 79 lines

```
struct anc{
  int par[N][25],lvl[N];
  void construct(int n){
    lvl[0]=-1;
    memset(par,-1,sizeof par);
    dfs(1,0);
    par[1][0]=-1;
    for(int i=1;i<=n;i++){
      for(int j=1;(1<<j)<n;j++){
        if(par[i][j-1]!=-1) par[i][j]=par[par[i][j-1]][j-1];
      }
    }
  }
  void dfs(int x,int p){
    lvl[x]=lvl[p]+1;
    par[x][0]=p;
    for(auto u:g[x]){
      if(u==p) continue;
      dfs(u,x);
    }
  }

  int kthanc(int x,int k){
    for(int i=0;i<25;i++){
      if(k & (1<<i)){
        x=par[x][i];
      }
      if(x==-1) break;
    }
    return x;
  }

  int lca(int a,int b){
    if(lvl[a]<lvl[b]) swap(a,b);
    for(int i=24;i>=0;i--){
      if(lvl[a]-(1<<i)>=lvl[b]) a=par[a][i];
    }
    for(int i=24;i>=0;i--){
      if(par[a][i]!=-1 && par[a][i]!=par[b][i]){
        a=par[a][i];
        b=par[b][i];
      }

    }
    if(a==b) return a;
    return par[a][0];
  }

}p;

// Another Approach

const int N=200005,LG=20;
vector<int> g[N];
int e[2*N], d[2*N], f[N], st[2*N][LG+1], lg2[2*N+1], t;
void dfs(int u,int p,int h){
    f[u]=t; e[t]=u; d[t++]=h;
    for(int v:g[u]) if(v!=p){
        dfs(v,u,h+1);
        e[t]=u; d[t++]=h;
    }
```

```
}
void build(){
    t=0; dfs(1,0,0);
    int m=t; lg2[1]=0;
    for(int i=2;i<=m;i++) lg2[i]=lg2[i/2]+1;
    for(int i=0;i<m;i++) st[i][0]=i;
    for(int j=1;(1<<j)<=m;j++)
        for(int i=0;i+(1<<j)<=m;i++){
            int x=st[i][j-1], y=st[i+(1<<(j-1))][j-1];
            st[i][j]=d[x]<d[y]?x:y;
        }
}
int lca(int u,int v){
    int x=f[u], y=f[v];
    if(x>y) swap(x,y);
    int j=lg2[y-x+1], a=st[x][j], b=st[y-(1<<j)+1][j];
    return d[a]<d[b]?e[a]:e[b];
}
```

## HLD.h

1327ea, 83 lines

```
const int mod=1000000007;
const int N=200005;

int n,v[N],q;
vector<int> g[N];

struct segtree{
  int st[4*N]={0};
  void update(int idx, int val) {
    st[idx += n] = val;
    for (idx /= 2; idx; idx /= 2) st[idx] = max(st[2 * idx], st
        [2 * idx + 1]);
  }
  int query(int lo, int hi) {
    int ra = 0, rb = 0;
    for (lo += n, hi += n + 1; lo < hi; lo /= 2, hi /= 2) {
      if (lo & 1) ra = max(ra, st[lo++]);
      if (hi & 1) rb = max(rb, st[--hi]);
    }
    return max(ra, rb);
}
}sg;

int sz[N],dep[N],par[N];
void dfs(int x,int p){
  sz[x]=1;
  dep[x]=dep[p]+1;
  par[x]=p;
  for(auto u:g[x]){
    if(u==p) continue;
    dfs(u,x);
    sz[x]+=sz[u];
  }
}

int matha[N],konchain[N],koi[N];
int chain_count=1,idx=1;

void hld(int u,int p){
  if(!matha[chain_count]) matha[chain_count]=u;
  konchain[u]=chain_count;
  koi[u]=idx++;

  int heavy=-1,siz=-1;
  for(auto v:g[u]){
    if(v==p) continue;
    if(sz[v]>siz){
      siz=sz[v];
      heavy=v;
    }
  }
  if(heavy!=-1) hld(heavy,u);

  for(auto v:g[u]){
    if(v==p || v==heavy) continue;
    chain_count++;
    hld(v,u);
  }

}

int path_qry(int u,int v){
  int ans=0;
  while(konchain[u]!=konchain[v]){
    if(dep[matha[konchain[u]]]<dep[matha[konchain[v]]]) swap(u,
        v);
    ans=max(ans,sg.query(koi[matha[konchain[u]]],koi[u]));
    u=par[matha[konchain[u]]];
  }
  if(koi[u]>koi[v]) swap(u,v);
  ans=max(ans,sg.query(koi[u],koi[v]));
  return ans;
}
void node_upd(int u,int x){
  sg.update(koi[u],x);
}

void solve(){
    dfs(1,0);
    hld(1,0);
  node_upd(s,x);
  path_qry(u,v);
}
```

## DSUonTrees.cpp

40ee9e, 61 lines

```
int n,v[N];
int sz[N];
int st[N],ft[N],tym,koi[N];

void dfs(int u,int p){
    sz[u]=1;
    st[u]=tym;
    koi[tym++]=u;
    for(int v:g[u]){
        if(v==p) continue;
        dfs(v,u);
        sz[u]+=sz[v];
    }
    ft[u]=tym;
}

int cnt[N],tot,ans[N];

void add(int x){
    cnt[v[x]]++;
    if(cnt[v[x]]==1) tot++;
}
void rmv(int x){
    cnt[v[x]]--;
    if(cnt[v[x]]==0) tot--;
}

void dsu(int u,int p,int keep){
    int mxsz=-1,bigchild=-1;
    for(auto v:g[u]){
        if(v==p) continue;
        if(mxsz<sz[v]){
            mxsz=sz[v];
            bigchild=v;
        }
    }
    for(auto v:g[u]){
        if(v==bigchild || v==p) continue;
        dsu(v,u,0);
    }
    if(bigchild!=-1) dsu(bigchild,u,1);
    for(auto v:g[u]){
        if(v==bigchild || v==p) continue;
        for(int i=st[v];i<ft[v];i++){
            add(koi[i]);
        }
    }
    add(u);

    ans[u]=tot;

    if(!keep){
        for(int i=st[u];i<ft[u];i++) rmv(koi[i]);
    }

}
void solve(){
    dfs(1,0);
    dsu(1,0,1);
    for(int i=1;i<=n;i++) cout << ans[i] << " ";
}
```

## CentroidDecomposition.cpp

34618e, 34 lines

```
vector<vector<int>> adj;
vector<bool> is_removed;
vector<int> subtree_size;
int get_subtree_size(int node, int parent = -1) {
    subtree_size[node] = 1;
    for (int child : adj[node]) {
        if (child == parent || is_removed[child]) continue;
        subtree_size[node] += get_subtree_size(child, node);
    }
    return subtree_size[node];
}
int get_centroid(int node, int tree_size, int parent = -1) {
    for (int child : adj[node]) {
        if (child == parent || is_removed[child]) continue;
        if (subtree_size[child] * 2 > tree_size) {
            return get_centroid(child, tree_size, node);
        }
    }
    return node;
}
void build_centroid_decomp(int node = 0) {
    int tree_size = get_subtree_size(node);
    int centroid = get_centroid(node, tree_size);

    // Do something with the centroid (e.g., process it)

    is_removed[centroid] = true;

    for (int child : adj[centroid]) {
        if (!is_removed[child]) {
            build_centroid_decomp(child);
        }
    }
}
```

## ArticulationBridge.cpp

e0d5e4, 26 lines

```
vii adj[N];
int dt[N],low[N],vis[N];
int timer=0;
vector<pii> bridges;
void dfs(int v,int p){
  vis[v]=1;
  dt[v]=low[v]=timer++;
  for(auto u:adj[v]){
    if(u==p) continue;
    if(vis[u]) low[v]=min(low[v],dt[u]);
    else{
      dfs(u,v);
      low[v]=min(low[v],low[u]);
      if(low[u]>dt[v]) bridges.pb({v,u});
    }
  }
}

for(int i=1;i<=n;i++){
    vis[i]=0;
    dt[i]=-1;
    low[i]=-1;
  }
  for(int i=1;i<=n;i++){
    if(!vis[i]) dfs(i,-1);
  }
```

## ArticulationPoints.cpp

08bc87, 29 lines

```
vii adj[N];
int dt[N],low[N],vis[N];
int timer=0;
set<int> ap;
void dfs(int v,int p){
  vis[v]=1;
  dt[v]=low[v]=timer++;
  int cd=0;
  for(auto u:adj[v]){
    if(u==p) continue;
    if(vis[u]) low[v]=min(low[v],dt[u]);
    else{
      dfs(u,v);
      low[v]=min(low[v],low[u]);
      if(low[u]>=dt[v] && p!=-1) ap.insert(v);
      ++cd;
    }
  }
  if(p==-1 && cd>1) ap.insert(v);
}

for(int i=1;i<=n;i++){
    vis[i]=0;
    dt[i]=-1;
    low[i]=-1;
  }
  for(int i=1;i<=n;i++){
    if(!vis[i]) dfs(i,-1);
  }
```

## 2sat.h

**Description:** Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a||b)\&\&(!a||c)\&\&(d||!b)\&\&...$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (~x).

**Usage:** `TwoSat ts(number of boolean variables);`
`ts.either(0, ~3); // Var 0 is true or var 3 is false`
`ts.setValue(2); // Var 2 is true`
`ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true`
`ts.solve(); // Returns true iff it is solvable`
`ts.values[0..N-1] holds the assigned values to the vars`
**Time:** $\mathcal{O}(N+E)$, where N is the number of boolean variables, and E is the number of clauses.

5f9706, 56 lines

```
struct TwoSat {
  int N;
  vector<vi> gr;
  vi values; // 0 = false, 1 = true

  TwoSat(int n = 0) : N(n), gr(2*n) {}

  int addVar() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }

  void either(int f, int j) {
    f = max(2*f, -1-2*f);
    j = max(2*j, -1-2*j);
    gr[f].push_back(j^1);
    gr[j].push_back(f^1);
  }
  void setValue(int x) { either(x, x); }

  void atMostOne(const vi& li) { // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,sz(li)) {
      int next = addVar();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }

  vi val, comp, z; int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x; z.push_back(i);
    for(int e : gr[i]) if (!comp[e])
      low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
      x = z.back(); z.pop_back();
      comp[x] = low;
      if (values[x>>1] == -1)
        values[x>>1] = x&1;
    } while (x != i);
    return val[i] = low;
  }

  bool solve() {
    values.assign(N, -1);
    val.assign(2*N, 0); comp = val;
    rep(i,0,2*N) if (!comp[i]) dfs(i);
    rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1;
  }
};
```

## EulerWalk.h

**Description:** Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.
**Time:** $\mathcal{O}(V+E)$

780b64, 15 lines

```
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
  int n = sz(gr);
  vi D(n), its(n), eu(nedges), ret, s = {src};
  D[src]++; // to allow Euler paths, not just cycles
  while (!s.empty()) {
    int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
    if (it == end){ ret.push_back(x); s.pop_back(); continue; }
    tie(y, e) = gr[x][it++];
    if (!eu[e]) {
      D[x]--, D[y]++;
      eu[e] = 1; s.push_back(y);
    }}
  for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
  return {ret.rbegin(), ret.rend()};
}
```

## HamiltonianPath.cpp

ab1675, 41 lines

```
int n;
vector<vector<int>> adj;
vector<bool> visited;
vector<int> path;
bool foundHamiltonian = false;
void dfsHamiltonian(int u, int depth) {
    if (foundHamiltonian) return;
    visited[u] = true;
    path.push_back(u);
    if (depth == n) {
        foundHamiltonian = true;
        return;
    }
    for (int v : adj[u]) {
        if (!visited[v]) {
            dfsHamiltonian(v, depth + 1);
            if (foundHamiltonian) return;
        }
    }
    visited[u] = false;
    path.pop_back();
}
bool findHamiltonianPath() {
    // optional: sort neighbors for deterministic order
    for (int u = 1; u <= n; ++u)
        sort(adj[u].begin(), adj[u].end());
    for (int start = 1; start <= n; ++start) {
        fill(visited.begin(), visited.end(), false);
        path.clear();
        foundHamiltonian = false;
        dfsHamiltonian(start, 1);
        if (foundHamiltonian) {
            cout << "Hamiltonian Path found starting at " <<
                 start << ": ";
            for (int x : path) cout << x << ' ';
            cout << "\n";
            return true;
        }
    }
    cout << "No Hamiltonian Path exists.\n";
    return false;
}
```

# Flow (9)

## Dinic.cpp

**Description:** Complexity: $O(ans * E)$ or $O(V^2E)$ without scaling, $O(VElog(U))$ with scaling, Scaling performs much better in worst case, but has much higher constant factor To enable scaling, call maxFlow(true) Everything 0-indexed

d4b5ec, 140 lines

```
namespace Dinic {
    typedef long long LL;
    const int N = 5005, K = 60;
    const LL INF = 1e18;

    struct Edge { int frm, to; LL cap, flow; };

    int s, t, n;
    int level[N], ptr[N];
    vector<Edge> edges;
    vector<int> adj[N];

    void init(int nodes) {
        n = nodes;
        for (int i=0; i<n; i++)    adj[i].clear();
        edges.clear();
    }

    int addEdge(int a, int b, LL cap, LL revcap = 0) {
        edges.push_back({a, b, cap, 0});
        edges.push_back({b, a, revcap, 0});
        adj[a].push_back(edges.size()-2);
        adj[b].push_back(edges.size()-1);
        return edges.size()-2;
    }

    bool bfs(LL lim) {
        fill(level, level+n, -1);
        level[s] = 0;
        queue<int> q;
        q.push(s);

        while (!q.empty() && level[t] == -1) {
            int v = q.front();
            q.pop();
            for (int id: adj[v]) {
                Edge e = edges[id];
                if (level[e.to] == -1 && e.cap - e.flow >= lim)
                    {
                    q.push(e.to);
                    level[e.to] = level[v] + 1;
                }
            }
        }
        return level[t] != -1;
    }
    LL dfs(int v, LL flow) {
        if (v == t || !flow)        return flow;
        for (; ptr[v] < adj[v].size(); ptr[v]++) {
            int eid = adj[v][ptr[v]];
            Edge &e = edges[eid];
            if (level[e.to] != level[v] + 1)    continue;
            if (LL pushed = dfs(e.to, min(flow, e.cap - e.flow)
                )) {
                e.flow += pushed;
                edges[eid^1].flow -= pushed;
                return pushed;
            }
        }
        return 0;
    }
    LL maxFlow(int source, int sink, bool SCALING = false) {
        s = source, t = sink;

        long long flow = 0;
        for (LL lim = SCALING ? (1LL << K) : 1; lim > 0; lim
             >>= 1) {
            while (bfs(lim)) {
                fill(ptr, ptr+n, 0);
                while (LL pushed = dfs(s, INF)) flow += pushed;
            }
        }
        return flow;
    }

    bool leftOfMinCut(int x) {return level[x] != -1;}

    vector<vector<LL>> allPairMaxFlow(vector<Edge> &tree) {
        tree.clear();
        vector<vector<LL>> flow(n, vector<LL> (n, INF));

        vector<int> par(n);
        for (int i=1; i<n; i++) {
            for (auto &e: edges)    e.flow = 0;
            LL f = maxFlow(i, par[i]);
            tree.push_back({i, par[i], f});

            for (int j=i+1; j<n; j++)
                if (par[j] == par[i] && leftOfMinCut(j)) par[j]
                      = i;

            flow[i][par[i]] = flow[par[i]][i] = f;
            for (int j=0; j<i; j++)
                if (j != par[i])    flow[i][j] = flow[j][i] =
                     min(f, flow[par[i]][j]);
        }
        return flow;
    }
}

#define ALL_PAIR_FLOW_TEST
#ifndef ALL_PAIR_FLOW_TEST
int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, m;
    cin>>n>>m;
    int s = 1, t = n;

    Dinic::init(n+1);
    while (m--) {
        int a, b, c;
        cin>>a>>b>>c;
        Dinic::addEdge(a, b, c, c);
    }
    cout<<Dinic::maxFlow(s, t)<<endl;
}
#else

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    int n, m;
    cin>>n>>m;

    Dinic::init(n);

    for (int i=0; i<m; i++) {
        int u, v;
        cin>>u>>v;
        u--; v--;
        Dinic::addEdge(u, v, 1, 1);
    }

    vector<Dinic::Edge> e;
    auto f = Dinic::allPairMaxFlow(e);

    long long ans = 0;
    for (int i=0; i<n; i++)
        for (int j=i+1; j<n; j++)   ans += f[i][j];
    cout<<ans<<endl;
}
#endif
```

## MCMF.cpp

**Description:** Min Cost Max Flow Using Successive Shortest Path Complexity- SPFA: $O(ans * VE)$ , Dijkstra- $O(ans * ElogV)$ + Cost of Normalization Normalization- Sets potentials (pi) for Dijkstra() If all edges >= 0: you may comment out normalize() If graph is DAG, use DP in O(m) Otherwise, SPFA() is used in O(mn)

ec2bc6, 221 lines

```
namespace MCMF
{
    typedef long long F;
    typedef long long C;
    const F infF = 1e18;
    const C infC = 1e18;

    const int N = 5005;
    typedef pair<C, F> PCF;

    struct Edge
    {
        int frm, to;
        C cost;
        F cap, flow;
    };

    int n, s, t;
    vector<Edge> edges;
    vector<int> adj[N];
    C pi[N], dis[N];
    F fl[N];
    int prv[N], vis[N];

    void init(int nodes, int source, int sink)
    {
        n = nodes, s = source, t = sink;
        for (int i = 0; i < n; i++)
            pi[i] = 0, adj[i].clear();
        edges.clear();
    }

    void addEdge(int u, int v, F cap, C cost)
    {
        edges.push_back({u, v, cost, cap, 0});
        edges.push_back({v, u, -cost, 0, 0});
        adj[u].push_back(edges.size() - 2);
        adj[v].push_back(edges.size() - 1);
    }

    bool SPFA()
    {
        for (int i = 0; i < n; i++)
        {
            dis[i] = infC;
            fl[i] = 0;
```

```
        vis[i] = 0;
        prv[i] = -1;
    }
    queue<int> q;
    q.push(s);
    dis[s] = 0;
    fl[s] = infF;
    vis[s] = 1;

    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        vis[u] = 0;
        for (int eid : adj[u])
        {
            Edge &e = edges[eid];
            if (e.cap == e.flow)
                continue;

            if (dis[u] + e.cost < dis[e.to])
            {
                dis[e.to] = dis[u] + e.cost;
                fl[e.to] = min(fl[u], e.cap - e.flow);
                prv[e.to] = eid ^ 1;
                if (!vis[e.to])
                    q.push(e.to);
            }
        }
    }
    return fl[t] > 0;
}

PCF solveSPFA()
{
    C cost = 0;
    F flow = 0;
    while (SPFA())
    {
        C pathcost = dis[t];
        cost += pathcost * fl[t];
        flow += fl[t];
        for (int u = t, e = prv[u]; e != -1; u = edges[e].
             to, e = prv[u])
        {
            edges[e].flow -= fl[t];
            edges[e ^ 1].flow += fl[t];
        }
    }
    return {cost, flow};
}

void normalize()
{
    SPFA();
    for (int i = 0; i < n; i++)
        pi[i] = dis[i];
}

bool Dijkstra()
{
    for (int i = 0; i < n; i++)
    {
        dis[i] = infC;
        fl[i] = 0;
        vis[i] = 0;
        prv[i] = -1;
    }
    priority_queue<pair<C, int>> pq;
    pq.emplace(0, s);
    dis[s] = 0;
    fl[s] = infF;

    while (!pq.empty())
    {
        int u = pq.top().second;
        pq.pop();
        if (vis[u])
            continue;
        vis[u] = 1;

        for (int eid : adj[u])
        {
            Edge &e = edges[eid];
            if (vis[e.to] || e.cap == e.flow)
                continue;

            C nw = dis[u] + e.cost - pi[e.to] + pi[u];
            if (nw < dis[e.to])
            {
                dis[e.to] = nw;
                fl[e.to] = min(fl[u], e.cap - e.flow);
                prv[e.to] = eid ^ 1;
                pq.emplace(-dis[e.to], e.to);
            }
        }
    }
    return fl[t] > 0;
}

PCF solveDijkstra()
{
    normalize();
    C cost = 0;
    F flow = 0;
    while (Dijkstra())
    {
        for (int i = 0; i < n; i++)
            if (fl[i])
                pi[i] += dis[i];
        C pathcost = pi[t] - pi[s];
        cost += pathcost * fl[t];
        flow += fl[t];

        for (int u = t, e = prv[u]; e != -1; u = edges[e].
             to, e = prv[u])
        {
            edges[e].flow -= fl[t];
            edges[e ^ 1].flow += fl[t];
        }
    }
    return {cost, flow};
}

PCF kFlow(F k)
{
    normalize();
    C cost = 0;
    F flow = 0;

    while (flow < k && Dijkstra())
    {
        for (int i = 0; i < n; i++)
            if (fl[i])
                pi[i] += dis[i];

        F push = min(fl[t], k - flow);
        cost += pi[t] * push;
        flow += push;

        for (int u = t, e = prv[u]; e != -1; u = edges[e].
             to, e = prv[u])
        {
            edges[e].flow -= push;
            edges[e ^ 1].flow += push;
        }
    }

    if (flow < k)
        return {-1, flow}; // Not enough capacity
    return {cost, flow};
}

}

int main()
{
    int n, m, s, t;
    cin >> n >> m >> s >> t;

    MCMF::init(n + 1, s, t);

    for (int i = 0; i < m; i++)
    {
        int a, b, c, w;
        cin >> a >> b >> c >> w;
        MCMF::addEdge(a, b, c, w);
    }

    auto [c, f] = MCMF::solveDijkstra();
    cout << f << " " << c << endl;

    int k;
    cin >> k;
    auto [c, f] = MCMF::kFlow(k);
    if (f < k) cout << "Impossible to send " << k << " units of
        flow\n";
    else cout << f << " " << c << endl;
}
```

## HopcroftKarp.cpp

**Description:** Hopcroft Karp Bipartite Matching Algorithm Complexity- E sqrt(V) (much better in practice) Source- Foreverbell ICPC cheat sheet 1-indexed. Left and Right independently numbered. ml, mr contain matches for left and right parts. 1. You may pre-calculate a greedy-matching to speed up further 2. In order to convert to Kuhn remove bfs in matching() and (level[u] < level[v]) condition in dfs()

310401, 80 lines

```
namespace HopcroftKarp {
    const int maxN = 1e5+7, maxM = 1e5+7;
    int n, m, match;
    int vis[maxN], level[maxN], ml[maxN], mr[maxM];
    vector<int> edge[maxN];

    void init(int N, int M) {
        n = N, m = M;
        for (int i=1; i<=n; i++)    edge[i].clear(), ml[i] =
            -1;
        for (int i=1; i<=m; i++)    mr[i] = -1;
        match = 0;
    }

    void add(int u, int v) {
        edge[u].push_back(v);
        if (ml[u] == -1 && mr[v] == -1)     ml[u] = v, mr[v] =
            u, match++;
    }
```

```
    bool dfs(int u) {
        vis[u] = true;
        for (int x: edge[u]) {
            int v = mr[x];
            if (v == -1 || (!vis[v] && level[u] < level[v] &&
                 dfs(v))) {
                ml[u] = x; mr[x] = u;
                return true;
            }
        }
        return false;
    }

    int matching() {
        while (true) {
            queue<int> q;
            for (int i = 1; i <= n; ++i) {
                if (ml[i] == -1)    level[i] = 0, q.push(i);
                else                level[i] = -1;
            }
            while (!q.empty()) {
                int u = q.front();
                q.pop();
                for (int x: edge[u]) {
                    int v = mr[x];
                    if (v != -1 && level[v] < 0) {
                        level[v] = level[u] + 1;
                        q.push(v);
                    }
                }
            }
            for (int i = 1; i <= n; ++i) vis[i] = false;
            int d = 0;
            for (int i = 1; i <= n; ++i) if (ml[i] == -1 && dfs
                 (i)) ++d;
            if (d == 0) return match;
            match += d;
        }
    }
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, m, k;
    cin>>n>>m>>k;

    HopcroftKarp::init(n, m);

    while(k--) {
        int a, b;
        cin>>a>>b;
        a++; b++;
        HopcroftKarp::add(a, b);
    }

    cout<<HopcroftKarp::matching()<<endl;
    for (int i=1; i<=n; i++)
        if (HopcroftKarp::ml[i] != -1)
            cout<<i-1<<" "<<HopcroftKarp::ml[i]-1<<"\n";
}
```

Hungarian.cpp

**Description:** Hungarian algorithm for minimum weighted bipartite matching. (1-indexed) For max cost, negate cost matrix and negate output. Complexity- $O(n^2m)$. n must not be greater than m. Input- $(n+1)x(m+1)$ cost matrix. (0th row and column are useless) Output- $(ans, ml)$, where ml[i] = match for node i on the left.

63082d, 54 lines

```
template<typename T>
pair<T, vector<int>> Hungarian(const vector<vector<T>> &cost){
    const T INF = numeric_limits<T>::max();
    int n = cost.size()-1, m = cost[0].size()-1;
    vector<T> U(n+1), V(n+1);
    vector<int> mr(m+1), way(m+1), ml(n+1);

    for(int i = 1; i<=n; i++){
        mr[0] = i;
        int lastJ = 0;
        vector<T> minV(m+1, INF);
        vector<bool> used(m+1);
        do{
            used[lastJ] = true;
            int lastI = mr[lastJ], nextJ;
            T delta = INF;
            for(int j = 1; j<=m; j++){
                if(used[j]) continue;
                T diffCost = cost[lastI][j] - U[lastI] - V[j];
                if(diffCost < minV[j]) minV[j] = diffCost, way[
                     j] = lastJ;
                if(minV[j] < delta) delta = minV[j], nextJ = j;
            }
            for(int j = 0; j<=m; j++){
                if(used[j]) U[mr[j]] += delta, V[j] -= delta;
                else        minV[j] -= delta;
            }
            lastJ = nextJ;
        } while(mr[lastJ] != 0);
        do{
            int prevJ = way[lastJ];
            mr[lastJ] = mr[prevJ];
            lastJ = prevJ;
        } while(lastJ != 0);
    }
    for (int i=1; i<=m; i++)    ml[mr[i]] = i;
    return {-V[0], ml} ;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n;
    cin>>n;

    vector<vector<long long>> cost(n+1, vector<long long>(n+1))
         ;
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)    cin>>cost[i][j];

    auto [ans, match] = Hungarian(cost);
    cout<<ans<<endl;
    for (int i=1; i<=n; i++)    cout<<match[i]-1<<" ";
}
```

# Geometry (10)

## PickTheorem

4 lines

```
S = I + B/2 -1
Where, S= Area , I = he number of points with integer
coordinates lying strictly inside the polygon , B = the
number of points lying on polygon sides .
```

## MinkowskiSum.cpp

ec4bb5, 116 lines

```
const int N = 3e5 + 9;

const double inf = 1e100;
const double eps = 1e-9;
const double PI = acos(-1.0);

int sign(double x) {
    return (x > eps) - (x < -eps);
}

struct PT {
    double x, y;

    PT() : x(0), y(0) {}
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y) {}

    PT operator + (const PT &a) const {
        return PT(x + a.x, y + a.y);
    }

    PT operator - (const PT &a) const {
        return PT(x - a.x, y - a.y);
    }

    PT operator * (const double a) const {
        return PT(x * a, y * a);
    }

    friend PT operator * (const double &a, const PT &b) {
        return PT(a * b.x, a * b.y);
    }

    PT operator / (const double a) const {
        return PT(x / a, y / a);
    }

    bool operator == (PT a) const {
        return sign(a.x - x) == 0 && sign(a.y - y) == 0;
    }

    bool operator != (PT a) const {
        return !(*this == a);
    }

    bool operator < (PT a) const {
        return sign(a.x - x) == 0 ? y < a.y : x < a.x;
    }

    bool operator > (PT a) const {
        return sign(a.x - x) == 0 ? y > a.y : x > a.x;
    }

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

    double norm2() {
        return x * x + y * y;
    }

    PT perp() {
        return PT(-y, x);
    }
```

```
    double arg() {
        return atan2(y, x);
    }

    PT truncate(double r) {
        // returns a vector with norm r and having same
            direction
        double k = norm();
        if (!sign(k)) return *this;
        r /= k;
        return PT(x * r, y * r);
    }
};

// Rotate the polygon so that the (bottom, left)-most point is
    at the first position
void reorder_polygon(vector<PT> &p) {
    int pos = 0;
    for (int i = 1; i < p.size(); i++) {
        if (p[i].y < p[pos].y || (sign(p[i].y - p[pos].y) == 0
            && p[i].x < p[pos].x)) {
            pos = i;
        }
    }
    rotate(p.begin(), p.begin() + pos, p.end());
}

// Computes the Minkowski sum of two convex polygons 'a' and 'b
    '
// Precondition: min(a.size(), b.size()) >= 2
// Reference: https://cp-algorithms.com/geometry/minkowski.html
vector<PT> minkowski_sum(vector<PT> a, vector<PT> b) {
    reorder_polygon(a);
    reorder_polygon(b);

    int n = a.size(), m = b.size();
    int i = 0, j = 0;

    a.push_back(a[0]);
    a.push_back(a[1]);
    b.push_back(b[0]);
    b.push_back(b[1]);

    vector<PT> c;
    while (i < n || j < m) {
        c.push_back(a[i] + b[j]);

        double p = cross(a[i + 1] - a[i], b[j + 1] - b[j]);

        if (sign(p) >= 0) ++i;
        if (sign(p) <= 0) ++j;
    }

    return c;
}
```

## 10.1 Geometric primitives

### Point.h

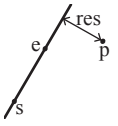
**Description:** Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes dist()=1
  P perp() const { return P(-y, x); } // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
  friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << "," << p.y << ")"; }
};
```

### lineDistance.h

**Description:**
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.

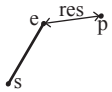



"Point.h" f6bf6b, 4 lines

```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
  return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

### SegmentDistance.h

**Description:**
Returns the shortest distance between point p and the line segment from point s to e.
**Usage:** `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

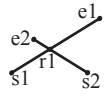



"Point.h" 5c88f4, 6 lines

```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
  return ((p-s)*d-(e-s)*t).dist()/d;
}
```

### SegmentIntersection.h

**Description:**
If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.
**Usage:** `vector<P> inter = segInter(s1,e1,s2,e2);`
`if (sz(inter)==1)`
`cout << "segments intersect at " << inter[0] << endl;`

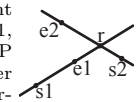



"Point.h", "OnSegment.h" 9d57f2, 13 lines

```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
  auto oa = c.cross(d, a), ob = c.cross(d, b),
       oc = a.cross(b, c), od = a.cross(b, d);
  // Checks if intersection is single non-endpoint point.
  if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
    return {(a * ob - b * oa) / (ob - oa)};
  set<P> s;
  if (onSegment(c, d, a)) s.insert(a);
  if (onSegment(c, d, b)) s.insert(b);
  if (onSegment(a, b, c)) s.insert(c);
  if (onSegment(a, b, d)) s.insert(d);
  return {all(s)};
}
```

### lineIntersection.h

**Description:**
If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.
**Usage:** `auto res = lineInter(s1,e1,s2,e2);`
`if (res.first == 1)`
`cout << "intersection point at " << res.second << endl;`




"Point.h" a01f81, 8 lines

```
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
  auto d = (e1 - s1).cross(e2 - s2);
  if (d == 0) // if parallel
    return {-(s1.cross(e1, s2) == 0), P(0, 0)};
  auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
  return {1, (s1 * p + e1 * q) / d};
}
```

### sideOf.h

**Description:** Returns where $p$ is as seen from $s$ towards $e$. 1/0/-1 $\Leftrightarrow$ left/on line/right. If the optional argument $eps$ is given 0 is returned if $p$ is within distance $eps$ from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.
**Usage:** `bool left = sideOf(p1,p2,q)==1;`

"Point.h" 3af81c, 9 lines

```
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

### OnSegment.h

**Description:** Returns true iff p lies on the line segment from s to e. Use `(segDist(s,e,p)<=epsilon)` instead when using Point<double>.

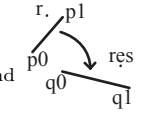
"Point.h" c597e8, 3 lines

```
template<class P> bool onSegment(P s, P e, P p) {
  return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

### linearTransformation.h

**Description:**

Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.




"Point.h" 03a306, 6 lines

```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
  P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
  return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

### Angle.h

**Description:** A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.

**Usage:** `vector<Angle> v = {w[0], w[0].t360() ...}; // sorted`
`int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }`
`// sweeps j such that (j-i) represents the number of positively oriented triangles with vertices at 0 and i`

0f0602, 35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
         make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

## 10.2 Circles

### CircleIntersection.h

**Description:** Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

"Point.h" 84d6d3, 11 lines

```
typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out) {
  if (a == b) { assert(r1 != r2); return false; }
  P vec = b - a;
  double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
         p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
  if (sum*sum < d2 || dif*dif > d2) return false;
  P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

### CircleTangents.h

**Description:** Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

"Point.h" b0153d, 13 lines

```
template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
  P d = c2 - c1;
  double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
  if (d2 == 0 || h2 < 0)  return {};
  vector<pair<P, P>> out;
  for (double sign : {-1, 1}) {
    P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
    out.push_back({c1 + v * r1, c2 + v * r2});
  }
  if (h2 == 0) out.pop_back();
  return out;
}
```

### CirclePolygonIntersection.h

**Description:** Returns the area of the intersection of a circle with a ccw polygon.

**Time:** $\mathcal{O}(n)$

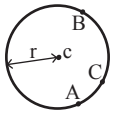
"../../content/geometry/Point.h" 19add1, 19 lines

```
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
  auto tri = [&](P p, P q) {
    auto r2 = r * r / 2;
    P d = q - p;
    auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = q + d * (t-1);
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i,0,sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
  return sum;
}
```

### circumcircle.h

**Description:**

The circumcirle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.




"Point.h" 1caa3a, 9 lines

```
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
      abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

### MinimumEnclosingCircle.h

**Description:** Computes the minimum circle that encloses a set of points.

**Time:** expected $\mathcal{O}(n)$

"circumcircle.h" 09dd0a, 17 lines

```
pair<P, double> mec(vector<P> ps) {
  shuffle(all(ps), mt19937(time(0)));
  P o = ps[0];
  double r = 0, EPS = 1 + 1e-8;
  rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
    o = ps[i], r = 0;
    rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
      o = (ps[i] + ps[j]) / 2;
      r = (o - ps[i]).dist();
      rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
      }
    }
  }
  return {o, r};
}
```

## 10.3 Polygons

### InsidePolygon.h

**Description:** Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.

**Usage:** `vector<P> v = {P{4,4}, P{1,2}, P{2,1}};`
`bool in = inPolygon(v, P{3, 3}, false);`

**Time:** $\mathcal{O}(n)$

"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines

```
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = sz(p);
  rep(i,0,n) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
  }
  return cnt;
}
```

### PolygonArea.h

**Description:** Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

"Point.h" f12300, 6 lines

```
template<class T>
T polygonArea2(vector<Point<T>>& v) {
  T a = v.back().cross(v[0]);
  rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
  return a;
}
```

### PolygonCenter.h

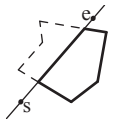
**Description:** Returns the center of mass for a polygon.

**Time:** $\mathcal{O}(n)$

"Point.h" 9706dc, 9 lines

```
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
  P res(0, 0); double A = 0;
  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
  }
  return res / A / 3;
}
```

### PolygonCut.h




**Description:**
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
**Usage:** `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`

"Point.h" d07181, 13 lines

```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  rep(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    auto a = s.cross(e, cur), b = s.cross(e, prev);
    if ((a < 0) != (b < 0))
      res.push_back(cur + (prev - cur) * (a / (a - b)));
    if (a < 0)
      res.push_back(cur);
  }
  return res;
}
```

### ConvexHull.h

**Description:**
Returns a vector of the points of the convex hull in counterclockwise order. Points on the edge of the hull between two other points are not considered part of the hull.
**Time:** $\mathcal{O}(n \log n)$

"Point.h" 310954, 13 lines

```
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
  if (sz(pts) <= 1) return pts;
  sort(all(pts));
  vector<P> h(sz(pts)+1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(all(pts)))
    for (P p : pts) {
      while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

### HullDiameter.h

**Description:** Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).
**Time:** $\mathcal{O}(n)$

"Point.h" c571b8, 12 lines

```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i,0,j)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
      if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
        break;
    }
  return res.second;
}
```

### PointInsideHull.h

**Description:** Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
**Time:** $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "OnSegment.h" 71446b, 14 lines

```
typedef Point<ll> P;
bool inHull(const vector<P>& l, P p, bool strict = true) {
  int a = 1, b = sz(l) - 1, r = !strict;
  if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
  if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
  if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<= -r)
    return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
  }
  return sgn(l[a].cross(l[b], p)) < r;
}
```

### LineHullIntersection.h

**Description:** Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon: $\bullet$ $(-1,-1)$ if no collision, $\bullet$ $(i,-1)$ if touching the corner $i$, $\bullet$ $(i,i)$ if along side $(i,i+1)$, $\bullet$ $(i,j)$ if crossing sides $(i,i+1)$ and $(j,j+1)$. In the last case, if a corner $i$ is crossed, this is treated as happening on side $(i,i+1)$. The points are returned in the same order as the line hits the polygon. `extrVertex` returns the point of a hull with the max projection onto a line.
**Time:** $\mathcal{O}(\log n)$

"Point.h" 7cf45b, 39 lines

```
#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
  int n = sz(poly), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
    int m = (lo + hi) / 2;
    if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
  }
  return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
  int endA = extrVertex(poly, (a - b).perp());
  int endB = extrVertex(poly, (b - a).perp());
  if (cmpL(endA) < 0 || cmpL(endB) > 0)
    return {-1, -1};
  array<int, 2> res;
  rep(i,0,2) {
    int lo = endB, hi = endA, n = sz(poly);
    while ((lo + 1) % n != hi) {
      int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
      (cmpL(m) == cmpL(endB) ? lo : hi) = m;
    }
    res[i] = (lo + !cmpL(hi)) % n;
    swap(endA, endB);
  }
  if (res[0] == res[1]) return {res[0], -1};
  if (!cmpL(res[0]) && !cmpL(res[1]))
    switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

## 10.4 Misc. Point Set Problems

### ClosestPair.h

**Description:** Finds the closest pair of points.
**Time:** $\mathcal{O}(n \log n)$

"Point.h" ac41a6, 17 lines

```
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set<P> S;
  sort(all(v), [](P a, P b) { return a.y < b.y; });
  pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for (P p : v) {
    P d{1 + (ll)sqrt(ret.first), 0};
    while (v[j].y <= p.y - d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
    for (; lo != hi; ++lo)
      ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
    S.insert(p);
  }
  return ret.second;
}
```

### kdTree.h

**Description:** KD-tree (2d, can be extended to 3d)

"Point.h" bac5b0, 63 lines

```
typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
  P pt; // if this is a leaf, the single point in it
  T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
  Node *first = 0, *second = 0;

  T distance(const P& p) { // min squared distance to a point
    T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x,y) - p).dist2();
  }

  Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
      x0 = min(x0, p.x); x1 = max(x1, p.x);
      y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
      // split on x if width >= height (not ideal...)
      sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
      // divide by taking half the array for each child (not
      // best performance with many duplicates in the middle)
      int half = sz(vp)/2;
      first = new Node({vp.begin(), vp.begin() + half});
      second = new Node({vp.begin() + half, vp.end()});
    }
  }
};

struct KDTree {
  Node* root;
  KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

  pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
      // uncomment if we should not find the point itself:
```

```
    // if (p == node->pt) return {INF, P()};
    return make_pair((p - node->pt).dist2(), node->pt);
  }

  Node *f = node->first, *s = node->second;
  T bfirst = f->distance(p), bsec = s->distance(p);
  if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

  // search closest side first, other side if needed
  auto best = search(f, p);
  if (bsec < best.first)
    best = min(best, search(s, p));
  return best;
 }

 // find nearest point to a point, and its squared distance
 // (requires an arbitrary operator< for Point)
 pair<T, P> nearest(const P& p) {
  return search(root, p);
 }
};
```

## FastDelaunay.h

**Description:** Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1], t[0][2], t[1][0], ... }, all counter-clockwise.
**Time:** $\mathcal{O}(n \log n)$

"Point.h" eefdf5, 88 lines

```
typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
  Q rot, o; P p = arb; bool mark;
  P& F() { return r()->p; }
  Q& r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r->o; r->r()->r() = r;
  rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
  r->p = orig; r->F() = dest;
  return r;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
  Q A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
         (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (circ(e->dir->F(), H(base), e->F())) { \
      Q t = e->dir; \
      splice(e, e->prev()); \
      splice(e->r(), e->r()->prev()); \
      e->o = H; H = e; e = t; \
    }
  for (;;) {
    DEL(LC, base->r(), o);  DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
      base = connect(RC, base->r());
    else
      base = connect(base->r(), LC->r());
  }
  return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
}
```

# 10.5 3D

## PolyhedronVolume.h

**Description:** Magic formula for the volume of a polyhedron. Faces should point outwards.

3058c3, 6 lines

```
template<class V, class L>
double signedPolyVolume(const V& p, const L& trilist) {
  double v = 0;
  for (auto i : trilist) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
  return v / 6;
}
```

## Point3D.h

**Description:** Class to handle points in 3D space. T can be e.g. double or long long.

8058ae, 32 lines

```
template<class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
  bool operator==(R p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z); }
  P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
  P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
  P operator*(T d) const { return P(x*d, y*d, z*d); }
  P operator/(T d) const { return P(x/d, y/d, z/d); }
  T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
  P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
  }
  T dist2() const { return x*x + y*y + z*z; }
  double dist() const { return sqrt((double)dist2()); }
  //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
  double phi() const { return atan2(y, x); }
  //Zenith angle (latitude) to the z-axis in interval [0, pi]
  double theta() const { return atan2(sqrt(x*x+y*y),z); }
  P unit() const { return *this/(T)dist(); } //makes dist()=1
  //returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  //returns point rotated 'angle' radians ccw around axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
  }
};
```

## 3dHull.h

**Description:** Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards.
**Time:** $\mathcal{O}(n^2)$

"Point3D.h" 5b45fc, 49 lines

```
typedef Point3D<double> P3;

struct PR {
  void ins(int x) { (a == -1 ? a : b) = x; }
  void rem(int x) { (a == x ? a : b) = -1; }
  int cnt() { return (a != -1) + (b != -1); }
  int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
  assert(sz(A) >= 4);
  vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.push_back(f);
  };
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);

  rep(i,4,sz(A)) {
    rep(j,0,sz(FS)) {
      F f = FS[j];
      if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
        E(a,b).rem(f.c);
        E(a,c).rem(f.b);
```

```
        E(b,c).rem(f.a);
        swap(FS[j--], FS.back());
        FS.pop_back();
      }
    }
    int nw = sz(FS);
    rep(j,0,nw) {
      F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
      C(a, b, c); C(a, c, b); C(b, c, a);
    }
  }
  for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
    A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
  return FS;
};
```

### sphericalDistance.h

**Description:** Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 ($\phi_1$) and f2 ($\phi_2$) from x axis and zenith angles (latitude) t1 ($\theta_1$) and t2 ($\theta_2$) from z axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx*radius is then the difference between the two points in the x direction and d*radius is the total distance between the points.

611f07, 8 lines

```
double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
  double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
  double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
  double dz = cos(t2) - cos(t1);
  double d = sqrt(dx*dx + dy*dy + dz*dz);
  return radius*2*asin(d/2);
}
```

# Strings (11)

### KMP.h

**Description:** pi[x] computes the length of the longest prefix of s that ends at x, other than s[0...x] itself (abacaba -> 0010123). Can be used to find all occurrences of a string.
**Time:** $\mathcal{O}(n)$

d4375c, 16 lines

```
vi pi(const string& s) {
  vi p(sz(s));
  rep(i,1,sz(s)) {
    int g = p[i-1];
    while (g && s[i] != s[g]) g = p[g-1];
    p[i] = g + (s[i] == s[g]);
  }
  return p;
}

vi match(const string& s, const string& pat) {
  vi p = pi(pat + '\0' + s), res;
  rep(i,sz(p)-sz(s),sz(p))
    if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
  return res;
}
```

### Zfunc.h

**Description:** z[i] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)
**Time:** $\mathcal{O}(n)$

ee09e2, 12 lines

```
vi Z(const string& S) {
  vi z(sz(S));
  int l = -1, r = -1;
  rep(i,1,sz(S)) {
    z[i] = i >= r ? 0 : min(r - i, z[i - l]);
    while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
      z[i]++;
    if (i + z[i] > r)
      l = i, r = i + z[i];
  }
  return z;
}
```

### Manacher.h

**Description:** For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).
**Time:** $\mathcal{O}(N)$

e7ad79, 13 lines

```
array<vi, 2> manacher(const string& s) {
  int n = sz(s);
  array<vi,2> p = {vi(n+1), vi(n)};
  rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
    int t = r-i+!z;
    if (i<r) p[z][i] = min(t, p[z][l+t]);
    int L = i-p[z][i], R = i+p[z][i]-!z;
    while (L>=1 && R+1<n && s[L-1] == s[R+1])
      p[z][i]++, L--, R++;
    if (R>r) l=L, r=R;
  }
  return p;
}
```

### MinRotation.h

**Description:** Finds the lexicographically smallest rotation of a string.
**Usage:** `rotate(v.begin(), v.begin()+minRotation(v), v.end());`
**Time:** $\mathcal{O}(N)$

d07a42, 8 lines

```
int minRotation(string s) {
  int a=0, N=sz(s); s += s;
  rep(b,0,N) rep(k,0,N) {
    if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
    if (s[a+k] > s[b+k]) { a = b; break; }
  }
  return a;
}
```

### SuffixArray.h

**Description:** Builds suffix array for a string. `sa[i]` is the starting index of the suffix which is $i$'th in the sorted suffix array. The returned vector is of size $n+1$, and `sa[0] = n`. The `lcp` array contains longest common prefixes for neighbouring strings in the suffix array: `lcp[i] = lcp(sa[i], sa[i-1])`, `lcp[0] = 0`. The input string must not contain any nul chars.
**Time:** $\mathcal{O}(n \log n)$

635552, 22 lines

```
struct SuffixArray {
  vi sa, lcp;
  SuffixArray(string s, int lim=256) { // or vector<int>
    s.push_back(0); int n = sz(s), k = 0, a, b;
    vi x(all(s)), y(n), ws(max(n, lim));
    sa = lcp = y, iota(all(sa), 0);
    for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
      p = j, iota(all(y), n - j);
      rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
      fill(all(ws), 0);
      rep(i,0,n) ws[x[i]]++;
      rep(i,1,lim) ws[i] += ws[i - 1];
      for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
      swap(x, y), p = 1, x[sa[0]] = 0;
      rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
        (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
    }
    for (int i = 0, j; i < n - 1; lcp[x[i++]] = k)
      for (k && k--, j = sa[x[i] - 1];
          s[i + k] == s[j + k]; k++);
  }
};
```

### Hashing.h

fa337f, 77 lines

```
const int N = 1e6 + 9;

int power(long long n, long long k, const int mod) {
  int ans = 1 % mod;
  n %= mod;
  if (n < 0) n += mod;
  while (k) {
    if (k & 1) ans = (long long) ans * n % mod;
    n = (long long) n * n % mod;
    k >>= 1;
  }
  return ans;
}

const int MOD1 = 127657753, MOD2 = 987654319;
const int p1 = 137, p2 = 277;
int ip1, ip2;
pair<int, int> pw[N], ipw[N];
void prec() {
  pw[0] =  {1, 1};
  for (int i = 1; i < N; i++) {
    pw[i].first = 1LL * pw[i - 1].first * p1 % MOD1;
    pw[i].second = 1LL * pw[i - 1].second * p2 % MOD2;
  }
  ip1 = power(p1, MOD1 - 2, MOD1);
  ip2 = power(p2, MOD2 - 2, MOD2);
  ipw[0] =  {1, 1};
  for (int i = 1; i < N; i++) {
    ipw[i].first = 1LL * ipw[i - 1].first * ip1 % MOD1;
    ipw[i].second = 1LL * ipw[i - 1].second * ip2 % MOD2;
  }

}
struct Hashing {
  int n;
  string s; // 0 - indexed
  vector<pair<int, int>> hs; // 1 - indexed
  Hashing() {}
  Hashing(string _s) {
    n = _s.size();
    s = _s;
    hs.emplace_back(0, 0);
    for (int i = 0; i < n; i++) {
      pair<int, int> p;
      p.first = (hs[i].first + 1LL * pw[i].first * s[i] % MOD1)
          % MOD1;
      p.second = (hs[i].second + 1LL * pw[i].second * s[i] %
          MOD2) % MOD2;
      hs.push_back(p);
    }
  }
  pair<int, int> get_hash(int l, int r) { // 1 - indexed
    assert(1 <= l && l <= r && r <= n);
    pair<int, int> ans;
    ans.first = (hs[r].first - hs[l - 1].first + MOD1) * 1LL *
        ipw[l - 1].first % MOD1;
    ans.second = (hs[r].second - hs[l - 1].second + MOD2) * 1LL
        * ipw[l - 1].second % MOD2;
    return ans;
  }
```

```
  pair<int, int> get_hash() {
    return get_hash(1, n);
  }
};
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  prec();
  int n;
  while (cin >> n) {
    string s, p;
    cin >> p >> s;
    Hashing h(s);
    auto hs = Hashing(p).get_hash();
    for(int i = 1; i + n - 1 <= s.size(); i++) {
      if (h.get_hash(i, i + n - 1) == hs) cout << i - 1 << '\n'
          ;
    }
    cout << '\n';
  }
  return 0;
}
```

## AhoCorasick.h

**Description:** Aho-Corasick automaton, used for multiple pattern matching. Initialize with AhoCorasick ac(patterns); the automaton start node will be at index 0. find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(−, word) finds all words (up to $N\sqrt{N}$ many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries.
**Time:** construction takes $\mathcal{O}(26N)$, where $N$ = sum of length of patterns. find(x) is $\mathcal{O}(N)$, where N = length of x. findAll is $\mathcal{O}(NM)$. f35677, 66 lines

```
struct AhoCorasick {
  enum {alpha = 26, first = 'A'}; // change this!
  struct Node {
    // (nmatches is optional)
    int back, next[alpha], start = -1, end = -1, nmatches = 0;
    Node(int v) { memset(next, v, sizeof(next)); }
  };
  vector<Node> N;
  vi backp;
  void insert(string& s, int j) {
    assert(!s.empty());
    int n = 0;
    for (char c : s) {
      int& m = N[n].next[c - first];
      if (m == -1) { n = m = sz(N); N.emplace_back(-1); }
      else n = m;
    }
    if (N[n].end == -1) N[n].start = j;
    backp.push_back(N[n].end);
    N[n].end = j;
    N[n].nmatches++;
  }
  AhoCorasick(vector<string>& pat) : N(1, -1) {
    rep(i,0,sz(pat)) insert(pat[i], i);
    N[0].back = sz(N);
    N.emplace_back(0);

    queue<int> q;
    for (q.push(0); !q.empty(); q.pop()) {
      int n = q.front(), prev = N[n].back;
      rep(i,0,alpha) {
        int &ed = N[n].next[i], y = N[prev].next[i];
        if (ed == -1) ed = y;
        else {
          N[ed].back = y;
          (N[ed].end == -1 ? N[ed].end : backp[N[ed].start])
            = N[y].end;
          N[ed].nmatches += N[y].nmatches;
          q.push(ed);
        }
      }
    }
  }
  vi find(string word) {
    int n = 0;
    vi res; // ll count = 0;
    for (char c : word) {
      n = N[n].next[c - first];
      res.push_back(N[n].end);
      // count += N[n].nmatches;
    }
    return res;
  }
  vector<vi> findAll(vector<string>& pat, string word) {
    vi r = find(word);
    vector<vi> res(sz(word));
    rep(i,0,sz(word)) {
      int ind = r[i];
      while (ind != -1) {
        res[i - sz(pat[ind]) + 1].push_back(ind);
        ind = backp[ind];
      }
    }
    return res;
  }
};
```

## Trie.h

5c364a, 104 lines

```
struct TrieNode {
    TrieNode* child[26];
    int wordCount;      // Number of times this word was
        inserted
    int prefixCount;    // Number of words that share this
        prefix

    TrieNode() {
        wordCount = 0;
        prefixCount = 0;
        for (int i = 0; i < 26; ++i)
            child[i] = nullptr;
    }
};

struct Trie {
    TrieNode* root;

    Trie() {
        root = new TrieNode();
    }

    // Inserts a word into the trie
    void insert(const string &word) {
        TrieNode* node = root;
        for (char c : word) {
            int idx = c - 'a';
            if (!node->child[idx])
                node->child[idx] = new TrieNode();
            node = node->child[idx];
            node->prefixCount++;
        }
        node->wordCount++;
    }

    // Returns true if the word exists in the trie
    bool search(const string &word) {
        TrieNode* node = root;
        for (char c : word) {
            int idx = c - 'a';
            if (!node->child[idx])
                return false;
            node = node->child[idx];
        }
        return node->wordCount > 0;
    }

    // Returns true if there is any word in the trie that
        starts with the given prefix
    bool startsWith(const string &prefix) {
        TrieNode* node = root;
        for (char c : prefix) {
            int idx = c - 'a';
            if (!node->child[idx])
                return false;
            node = node->child[idx];
        }
        return true;
    }

    // Returns how many times a word has been inserted
    int countWordsEqualTo(const string &word) {
        TrieNode* node = root;
        for (char c : word) {
            int idx = c - 'a';
            if (!node->child[idx]) return 0;
            node = node->child[idx];
        }
        return node->wordCount;
    }

    // Returns how many words start with the given prefix
    int countWordsStartingWith(const string &prefix) {
        TrieNode* node = root;
        for (char c : prefix) {
            int idx = c - 'a';
            if (!node->child[idx]) return 0;
            node = node->child[idx];
        }
        return node->prefixCount;
    }

    // Deletes one occurrence of the word from the trie
    void erase(const string &word) {
        if (!search(word)) return;

        TrieNode* node = root;
        for (char c : word) {
            int idx = c - 'a';
            node = node->child[idx];
            node->prefixCount--;
        }
        node->wordCount--;
    }

    // Optional: free memory
    void clear(TrieNode* node) {
        if (!node) return;
        for (int i = 0; i < 26; ++i)
            clear(node->child[i]);
        delete node;
    }
```

```
    ~Trie() {
        clear(root);
    }
};
```