Mục lục

1 Some definition **2**

2 Data structure **2**
2.1 Mo's algorithm 2
2.2 Set, map, multiset 2
2.3 Bitset 3
2.4 BIT 3
2.5 IT2D 3
2.6 Peristent IT 3
2.7 Li Chao Tree 4

3 Graph **5**
3.1 Dinic 5
3.2 Mincost 6
3.3 HLD 6
3.4 Tarjan 7
3.5 Monotone chain 7
3.6 MST 7
3.7 HopcroftKarp 7
3.8 Hungarian 8

4 String **9**
4.1 Aho Corasick 9
4.2 Manacher 10
4.3 Suffix Array 10
4.4 Z function 11
4.5 KMP 11
4.6 Lexicographically minimal string rotation 12
4.7 Hash 12
4.8 Hash 2D 13

5 Math **13**
5.1 Inverse of 2x2 matrix 13
5.2 Sum of divisors 13
5.3 Pisano period 13
5.4 Number Theory 13
5.5 Derivatives and integrals 14
5.6 Sum 14
5.7 Series 14
5.8 Trigonometric 14
5.9 Inverse 14
5.10 Gaussian elimination 15
5.11 Geometry 15
5.12 Discrete logarithm 16
5.13 Miller Rabin 16
5.14 Miller Rabin Deterministic 16
5.15 Chinese Remainer 17
5.16 Extended Euclid 18
5.17 FFT 18
5.18 PollardRho 19

6 Theorem **19**
6.1 Fermat's little theorem 19
6.2 Euler's theorem 19
6.3 Euler's totient function 19
6.4 Goldbach's conjecture 19
6.5 Dirichlet 19
6.6 Pythagorean triple 19
6.7 Legendre's formula 20
6.8 Stirling's approximation 20
6.9 Wilson's theorem 20

7 Other **20**
7.1 Matrix 20
7.2 Bignum mul 20
7.3 Random 21
7.4 Builtin bit function 21
7.5 Pythagorean triples 21
7.6 Sieve 21
7.7 Factorial mod 21
7.8 Catalan 22

7.9 Prime under 100 22
7.10 Pascal triangle 22
7.11 Fibo 22

8 Tips 22

1 Some definition

```
#include <bits/stdc++.h>
#include <random>
#include <chrono>
#include <ctime>

#define cross(A, B) (A.x * B.y - A.y * B.x)
#define dot(A, B) (A.x * B.x + A.y * B.y)
#define ccw(A, B, C) (-(A.x * (C.y - B.y) + B.x * (A.y - C.y)
   + C.x * (B.y - A.y))) // positive when ccw
#define CROSS(a, b, c, d) (a * d - b * c)

#define LL(x) (x << 1)
#define RR(x) ((x << 1) + 1)

using namespace std;
const int N = 1000005;
const int M = 30000;

const int Bases = 2;
const long long base[] = {137, 37};
const long long mod = 1000000007LL;

long long addi(long long a, long long b, long long m = mod) {
   a += b; if (a < 0) a += m; if (a >= m) a -= m; return a; }
long long subt(long long a, long long b, long long m = mod) {
   a -= b; if (a < 0) a += m; if (a >= m) a -= m; return a; }
long long mult(long long a, long long b, long long m = mod)
{
    if (b == 0) return 0;
    long long tmp = mult(a, b >> 1, m);
    tmp = addi(tmp, tmp, m);
    return (b & 1) ? addi(tmp, a, m) : tmp;
}
long long power(long long a, long long b, long long m = mod)
{
    long long tmp = 1;
    for (; b > 0; b >>= 1)
    {
        if (b & 1LL) tmp = mult(tmp, a, m);
        a = mult(a, a, m);
    }
    return tmp;
}
long long inv(long long a, long long m = mod) { return power(a
   , m - 2, m); }
```

2 Data structure

2.1 Mo's algorithm

$$O(N * \sqrt{N} + Q * \sqrt{N})$$

```
    S = sqrt(N);
    bool cmp(Query A, Query B) // compare 2 queries
    {
      if (A.l / S != B.l / S) {
        return A.l / S < B.l / S;
      }
      return A.r < B.r;
    }
```

2.2 Set, map, multiset

Use set.lower_bound() instead of lower_bound(set.begin(), set.end()) for better performance. The same is true for map.

Set and map is sorted internally, so *s.begin() will be the smallest element and *s.rbegin() will be the largest element.

If you want to remove 1 occurrence of 1 value in the multiset, use s.erase(s.find(value)). If we use s.erase(value), the multiset will erase all occurrence of that value in the multiset.

2.3 Bitset

Order positions are counted from the rightmost bit, which is order position 0.

Bitset can use bit operator like normal number.

If bitset is const-qualified then the return value of [] operator is bool, otherwise it is bitset::reference.

set() sets all value in bitset to 1. set(pos, val) sets value at pos to val (can throw out of bound).

reset() resets all value in bitset to 0. reset(pos) resets value at pos to 0 (can throw out of bound).

flip() flips all bits in bitset. flip(pos) flips bit at pos.

all() returns true if all bits are set. none() returns true if no bit is set. any() returns true if any of the bits is set.

count() returns the number of bits which are set.

test(pos) returns true if the bit at pos is set.

2.4 BIT

```
void update(int x, int val)
{
  for (; x <= n; x += x & ~x) BIT[x] = min(BIT[x], val);
}

int get(int x)
{
  int res = 1e9;
  for (; x > 0; x -= x & ~x) res = min(res, BIT[x]);
  return res;
}
```

2.5 IT2D

```
int Max[4096][4096];

struct dir {
  int ll, rr, id;
  dir (int L, int R, int X)
    { ll=L, rr=R, id=X; }
  dir left() const
    { return dir(ll, (ll+rr)/2, id*2); }
  dir right() const
    { return dir((ll+rr)/2+1, rr, id*2+1); }
  inline bool irrelevant(int L, int R) const
    { return ll>R || L>rr || L>R; }
};

void maximize(int &a, int b)
  { a=max(a, b); }

void maximize(const dir &dx, const dir &dy, int x, int y, int
   k, bool only_y) {
  if (dx.irrelevant(x, x) || dy.irrelevant(y, y)) return;
  maximize(Max[dx.id][dy.id], k);
  if (!only_y && dx.ll != dx.rr) {
    maximize(dx.left(), dy, x, y, k, false);
    maximize(dx.right(), dy, x, y, k, false);
  }
  if (dy.ll != dy.rr) {
    maximize(dx, dy.left(), x, y, k, true);
    maximize(dx, dy.right(), x, y, k, true);
  }
}

int max_range(const dir &dx, const dir &dy, int lx, int rx,
   int ly, int ry) {
  if (dx.irrelevant(lx, rx) || dy.irrelevant(ly, ry)) return
   0;
  if (lx<=dx.ll && dx.rr<=rx) {
    if (ly<=dy.ll && dy.rr<=ry) return Max[dx.id][dy.id];
    int Max1 = max_range(dx, dy.left(), lx, rx, ly, ry);
    int Max2 = max_range(dx, dy.right(), lx, rx, ly, ry);
    return max(Max1, Max2);
  } else {
    int Max1 = max_range(dx.left(), dy, lx, rx, ly, ry);
    int Max2 = max_range(dx.right(), dy, lx, rx, ly, ry);
    return max(Max1, Max2);
  }
}
```

2.6 Peristent IT

```
struct node{
    int _time, ans;
    node *l, *r;
```

```
    node(): l(r=nullptr), _time(0), ans(1){}
};

int last[N], Local_time;
vector <int> step[N];
node *root[N];
vector <pii> p[N];

void build(node *v, int tl, int tr){
    if(tl != tr){
        v -> l = new node();
        v -> r = new node();
        int tm = (tl + tr) >> 1;
        build(v -> l, tl, tm);
        build(v -> r, tm + 1, tr);
    }
}

void modify(node *v, int tl, int tr, int pos, pii fraction){
    if(tl == tr){
        v -> ans *= fraction.fi;
        v -> ans /= fraction.se;
    }else{
        int tm = (tl + tr) >> 1;
        if(pos <= tm){
            if((v -> l) -> _time != Local_time){
                node *old = v -> l;
                v -> l = new node();
                (v -> l) -> ans = old -> ans;
                (v -> l) -> l = old -> l;
                (v -> l) -> r = old -> r;
                (v -> l) -> _time = Local_time;
            }
            modify(v -> l, tl, tm, pos, fraction);
        }else{
            if((v -> r) -> _time != Local_time){
                node *old = v -> r;
                v -> r = new node();
                (v -> r) -> ans = old -> ans;
                (v -> r) -> l = old -> l;
                (v -> r) -> r = old -> r;
                (v -> r) -> _time = Local_time;
            }
            modify(v -> r, tm + 1, tr, pos, fraction);
        }
        v -> ans = mul((v -> l) -> ans, (v -> r) -> ans);
    }
}

int get(node *v, int tl, int tr, int l, int r){
    if(l > r)return 1;
    if(tl == l && tr == r)return v -> ans;
    int tm = (tl + tr) >> 1;
    return mul(get(v -> l, tl, tm, l, min(r, tm)), get(v -> r,
    tm + 1, tr, max(l, tm + 1), r));
}

//for first version:
root[0] = new node();
build(root[0], 1, n);
//when creating new version
Local_time++;
        root[i] = new node();
        root[i] -> ans = root[i - 1] -> ans;
        root[i] -> l = root[i - 1] -> l;
        root[i] -> r = root[i - 1] -> r;
        root[i] -> _time = Local_time;
//when calling: modify(root[i], 1, n, pos, {vals});
//when query: get(root[r], 1, n, l, r);
```

2.7 Li Chao Tree

```
typedef int ftype;
typedef complex<ftype> point;
#define x real
#define y imag

ftype dot(point a, point b) {
    return (conj(a) * b).x();
}

ftype f(point a, ftype x) {
    return dot(a, {x, 1});
}

const int maxn = 2e5;
```

```
point line[4 * maxn];

void add_line(point nw, int v = 1, int l = 0, int r = maxn) {
    int m = (l + r) / 2;
    bool lef = f(nw, l) < f(line[v], l);
    bool mid = f(nw, m) < f(line[v], m);
    if(mid) {
        swap(line[v], nw);
    }
    if(r - l == 1) {
        return;
    } else if(lef != mid) {
        add_line(nw, 2 * v, l, m);
    } else {
        add_line(nw, 2 * v + 1, m, r);
    }
}

int get(int x, int v = 1, int l = 0, int r = maxn) {
    int m = (l + r) / 2;
    if(r - l == 1) {
        return f(line[v], x);
    } else if(x < m) {
        return min(f(line[v], x), get(x, 2 * v, l, m));
    } else {
        return min(f(line[v], x), get(x, 2 * v + 1, m, r));
    }
}
```

3 Graph

3.1 Dinic

```
namespace Dinic // really fast, O(n^2 m) or O(sqrt(n)m) if
    bipartite
{
    vector<int> adj[N];
    long long c[N][N], f[N][N];
    int s = 0, t = 0, d[N], ptr[N];
    bool BFS()
    {
        queue<int> q;
        memset(d, -1, sizeof(d));
        d[s] = 0; q.push(s);
        while (!q.empty())
        {
            int u = q.front(); q.pop();
            for (int v : adj[u])
            {
                if (d[v] == -1 && c[u][v] > f[u][v])
                {
                    d[v] = d[u] + 1;
                    q.push(v);
                }
            }
        }
        return d[t] != -1;
    }
    long long DFS(int x, long long delta)
    {
        if (x == t) return delta;
        for (; ptr[x] < adj[x].size(); ++ptr[x]) // Skip the
used edge
        {
            int y = adj[x][ptr[x]];
            if (d[y] == d[x] + 1 && c[x][y] > f[x][y])
            {
                long long push = DFS(y, min(delta, c[x][y] - f
[x][y]));
                if (push)
                {
                    f[x][y] += push;
                    f[y][x] -= push;
                    return push;
                }
            }
        }
        return 0;
    }
    long long maxFlow(int x, int y) // From x to y
    {
        long long flow = 0;
        s = x; t = y;
        while (BFS())
        {
            memset(ptr, 0, sizeof(ptr));
```

```
            while (long long tmp = DFS(s, 1e9))
                flow += 1LL * tmp;
        }
        return flow;
    }
};
```

3.2 Mincost

```
int calc(int x, int y) { return (x >= 0) ? y : 0 - y; }

bool findpath()
{
  for (int i = 1; i <= n; i++) { trace[i] = 0; d[i] = inf; }
  q.push(n); d[n] = 0;
  while (!q.empty())
  {
    int u = q.front();
    q.pop();
    inq[u] = false;
    for (int i = 0; i < adj[u].size(); i++)
    {
      int v = adj[u][i];
      if (c[u][v] > f[u][v] && d[v] > d[u] + calc(f[u][v],
   cost[u][v]))
      {
        trace[v] = u;
        d[v] = d[u] + calc(f[u][v], cost[u][v]);
        if (!inq[v])
        {
          inq[v] = true;
          q.push(v);
        }
      }
    }
  }
  return d[t] != inf;
}

void incflow()
{
  int v = t, delta = inf;
  while (v != n)
  {
    int u = trace[v];
    if (f[u][v] >= 0)
      delta = min(delta, c[u][v] - f[u][v]);
    else
      delta = min(delta, 0 - f[u][v]);
    v = u;
  }
  v = t;
  while (v != n)
  {
    int u = trace[v];
    f[u][v] += delta;
    f[v][u] -= delta;
    v = u;
  }
}
```

3.3 HLD

```
void DFS(int x,int pa)
{
  DD[x]=DD[pa]+1; child[x]=1; int Max=0;
  for (int i=0; i<DSK[x].size(); i++)
  {
    int y=DSK[x][i].fi;
    if (y==pa) continue;
    p[y]=x;
    d[y]=d[x]+DSK[x][i].se;
    DFS(y,x);
    child[x]+=child[y];
    if (child[y]>Max)
    {
      Max=child[y];
      tree[x]=tree[y];
    }
  }
  if (child[x]==1) tree[x]=++nTree;
}

void init()
{
  nTree=0;
  DFS(1,1);
  DD[0]=long(1e9);
```

```
  for (int i=1; i<=n; i++) if (DD[i]<DD[root[tree[i]]]) root[
   tree[i]]=i;
}

int LCA(int u,int v)
{
  while (tree[u]!=tree[v])
  {
    if (DD[root[tree[u]]]<DD[root[tree[v]]]) v=p[root[tree[v
   ]]];
    else u=p[root[tree[u]]];
  }
  if (DD[u]<DD[v]) return u; else return v;
}
```

3.4 Tarjan

If u is articulation:
if (low[v] >= num[u]) arti[u] = arti[u] or p[u] != -1 or child[u] >= 2;
If (u, v) is bridge: $low[v] >= num[v]$

3.5 Monotone chain

```
void convex_hull (vector<pt> & a) {
  if (a.size() == 1) { // Only 1 point
    return;
  }

  // Sort with respect to x and then y
  sort(a.begin(), a.end(), &cmp);

  pt p1 = a[0],  p2 = a.back();

  vector<pt> up, down;
  up.push_back (p1);
  down.push_back (p1);

  for (size_t i=1; i<a.size(); ++i) {
    // Add to the upper chain

    if (i==a.size()-1 || cw (p1, a[i], p2)) {
      while (up.size()>=2 && !cw (up[up.size()-2], up[up.size
   ()-1], a[i]))
        up.pop_back();
      up.push_back (a[i]);
    }

    // Add to the lower chain
    if (i==a.size()-1 || ccw (p1, a[i], p2)) {
      while (down.size()>=2 && !ccw (down[down.size()-2], down
   [down.size()-1], a[i]))
        down.pop_back();
      down.push_back (a[i]);
    }
  }

  // Merge 2 chains
  a.clear();
  for (size_t i=0; i<up.size(); ++i)
    a.push_back (up[i]);
  for (size_t i=down.size()-2; i>0; --i)
    a.push_back (down[i]);
}
```

3.6 MST

Prim: remember to have visited array

3.7 HopcroftKarp

```
//N > n+m, variable n is number in both side (n+m)
//x<->y from n<->m should be adj[x].push_back(n+y);
namespace HopcroftKarp // O(sqrt(n) * m)
{
    vector<int> adj[N]; int match[N], d[N];
    bool BFS()
    {
        queue<int> q;
        memset(d, -1, sizeof(d));
        for (int i = 1; i <= n; ++i) if (!match[i])
        {
            d[i] = 0;
            q.push(i);
        }
        bool flag = false;
        while (!q.empty())
```

```
        {
            int u = q.front(); q.pop();
            for (int v : adj[u])
            {
                if (match[v] == 0)
                {
                    flag = true;
                    continue;
                }
                if (d[match[v]] == -1)
                {
                    d[match[v]] = d[u] + 1;
                    q.push(match[v]);
                }
            }
        }
        return flag;
    }
    bool DFS(int x)
    {
        for (int y : adj[x])
        {
            if (match[y] == 0 || (d[match[y]] == d[x] + 1 &&
DFS(match[y])))
            {
                match[y] = x;
                match[x] = y;
                return true;
            }
        }
        d[x] = -1;
        return false;
    }
    long long maxMatching() // From x to y
    {
        long long matching = 0;
        while (BFS())
        {
            for (int i = 1; i <= n; ++i) if (!match[i] && DFS(
i))
                ++matching;
        }
        return matching;
```

```
    }
};
```

3.8 Hungarian

```
struct Hungarian {
  long  c[N][N], fx[N], fy[N], d[N];
  int mx[N], my[N], trace[N], arg[N];
  queue<int> q;
  int start, finish, n, m;
  const long inf = 1e18;

  void Init(int _n, int _m) {
    n = _n, m = _m;
    FOR(i, 1, n) {
      mx[i] = my[i] = 0;
      FOR(j, 1, n) c[i][j] = inf;
    }
  }
  void addEdge(int u, int v, long cost) { c[u][v] = min(c[u][v
   ], cost); }
  inline long getC(int u, int v) { return c[u][v] - fx[u] - fy
   [v]; }

  void initBFS() {
    while (!q.empty()) q.pop();
    q.push(start);
    FOR(i, 0, n) trace[i] = 0;
    FOR(v, 1, n) {
      d[v] = getC(start, v), arg[v] = start;
    }
    finish = 0;
  }

  void findAugPath() {
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      FOR(v, 1, n) if (!trace[v]) {
        long w = getC(u, v);
        if (!w) {
          trace[v] = u;
          if (!my[v]) { finish = v; return; }
          q.push(my[v]);
```

```
      }
      if (d[v] > w) { d[v] = w; arg[v] = u; }
    }
  }
}

void subX_addY(){
  long delta = inf;
  FOR(v, 1, n) if (trace[v] == 0 && d[v] < delta) delta = d[
 v];
  fx[start] += delta;
  FOR(v, 1, n) if (trace[v]) {
    int u = my[v];
    fy[v] -= delta, fx[u] += delta;
  } else d[v] -= delta;

  FOR(v, 1, n) if (!trace[v] && !d[v]) {
    trace[v] = arg[v];
    if (!my[v]) { finish = v; return; }
    q.push(my[v]);
  }
}

void Enlarge() {
  do {
    int u = trace[finish], nxt = mx[u];
    mx[u] = finish, my[finish] = u, finish = nxt;
  } while (finish);
}

long minCost() {
  FOR(u, 1, n) {
    fx[u] = c[u][1];
    FOR(v, 1, n) fx[u] = min(fx[u], c[u][v]);
  }
  FOR(v, 1, n) {
    fy[v] = c[1][v] - fx[1];
    FOR(u, 1, n) fy[v] = min(fy[v], c[u][v] - fx[u]);
  }

  FOR(u, 1, n) {
    start = u;
    initBFS();
    while (finish == 0) {
      findAugPath();
      if (!finish) subX_addY();
    }
    Enlarge();
  }

  int res = 0;
  FOR(i, 1, n) res += c[i][mx[i]];
  return res;
 }
};
```

4 String

4.1 Aho Corasick

```
struct Node
{
  int nxt[26], go[26];
  bool leaf;
  long long val, sumVal;
  int p;
  int pch;
  int link;
};

Node t[N];
int sz;

void New(Node &x, int p, int link, int pch)
{
  x.p = p;
  x.link = link;
  x.pch = pch;
  x.val = 0;
  x.sumVal = -1;
  memset(x.nxt, -1, sizeof(x.nxt));
  memset(x.go, -1, sizeof(x.go));
}

void AddString(const string &s, int val)
{
```

```
  int v = 0;
  for (char c : s)
  {
    int id = c - 'A';
    if (t[v].nxt[id] == -1)
    {
      New(t[sz], v, -1, id);
      t[v].nxt[id] = sz++;
    }
    v = t[v].nxt[id];
  }
  t[v].leaf = true;
  t[v].val = val;
}

int Go(int u, int c);

int Link(int u)
{
  if (t[u].link == -1)
  {
    if (u == 0 || t[u].p == 0)
      t[u].link = 0;
    else
      t[u].link = Go(Link(t[u].p), t[u].pch);
  }
  return t[u].link;
}

int Go(int u, int c)
{
  if (t[u].go[c] == -1)
  {
    if (t[u].nxt[c] != -1)
      t[u].go[c] = t[u].nxt[c];
    else
      t[u].go[c] = (u == 0 ? 0 : Go(Link(u), c));
  }
  return t[u].go[c];
}
```

4.2 Manacher

```
void init() {
  cnt = 0;
  t[0] = '~';
  for (int i = 0; i<n; i++) {
    t[++cnt] = '#';t[++cnt] = s[i];
  }
  t[++cnt] = '#'; t[++cnt] = '-';
}

void manacher() {
  int n = cnt - 2;
  int r = 1; int C = 1;
  int ans = 0;
  for (int i = 2; i<n; i++) {
    int i_mirror = C * 2 - i;
    z[i] = (r > i) ? min(z[i_mirror], r - i) : 0;
    while (t[i + z[i] + 1] == t[i - z[i] - 1]) z[i]++;
    if (i + z[i] > r) {
      C = i;
      r = i + z[i];
    }
  }
}
```

4.3 Suffix Array

```
struct SuffixArray {
  string s;
  int n;
  vector<int> SA, RA, tempSA, tempRA, LCP;
  int L[N];

  void reset(string st) {
    s = st;
    RA.clear();
    s.push_back('$');
    n = s.size();
    RA.resize(n + 1, 0);
    SA = RA, tempSA = tempRA = LCP = RA;
  }

  void BuildSA() {
    REP(i, n) SA[i] = i, RA[i] = s[i];
    for (int k = 1; k < n; k <<= 1) {
      radix_sort(k);
```

```
      radix_sort(0);
      tempRA[SA[0]] = 0;
      for (int i = 1, r = 0; i < n; ++i) {
        if (getRA(SA[i - 1]) != getRA(SA[i]) || getRA(SA[i -
   1] + k) != getRA(SA[i] + k)) ++r;
        tempRA[SA[i]] = r;
      }
      REP(i, n) RA[i] = tempRA[i];
      if (RA[SA[n - 1]] == n - 1) break;
    }
  }

  void BuildLCP() {
    // kasai
    REP(i, n) RA[SA[i]] = i;
    int k = 0;
    REP(i, n) {
      if (RA[i] == n - 1) {
        k = 0; continue;
      }
      int j = SA[RA[i] + 1];
      while (i + k < n && j + k < n && s[i + k] == s[j + k])
   ++k;
      LCP[RA[i]] = k;
      if (k) k--;
    }
  }
private:
  inline int getRA(int i) { return (i < n ? RA[i] : 0); }
  void radix_sort(int k) {
    memset(L, 0, sizeof L);
    REP(i, n) L[getRA(i + k)]++;
    int p = 0;
    REP(i, N) {
      int x = L[i];
      L[i] = p;
      p += x;
    }
    REP(i, n) {
      int &x = L[getRA(SA[i] + k)];
      tempSA[x++] = SA[i];
    }
    REP(i, n) SA[i] = tempSA[i];
  }
};
```

4.4 Z function

```
vector<int> Zfunc(int n, vector<int> &a) {
  vector<int> z(n);
  z[0] = n;
  int l = 0, r = 0;
  FOR(i, 1, n - 1) {
    z[i] = (i <= r ? min(r - i + 1, z[i - l]) : 0);
    while (i + z[i] < n && a[z[i]] == a[i + z[i]]) ++z[i];
    if (i + z[i] > r) {
      r = i + z[i] - 1;
      l = i;
    }
  }
  return z;
}
```

4.5 KMP

```
// SUBSTR spoj
string s, t;int pos[N];
void build()
{
  pos[0] = -1;
  int pre = -1, cur = 0;
  while (cur < t.length())
  {
    while (pre >= 0 && t[cur] != t[pre])
    {
      pre = pos[pre];
    }
    pos[++cur] = ++pre;
  }
}

int main()
{
  cin >> s; cin >> t;
  build();
  int cur = 0;
  for (int i = 0; i < (int)s.length(); ++i)
```

```
  {
    while (cur >= 0 && s[i] != t[cur])
    {
      cur = pos[cur];
    }
    ++cur;
    if (cur == (int)t.length())
    {
      cout << i - (int)t.length() + 2 << ' ';
      cur = pos[cur];
    }
  }

  return 0;
}
```

4.6 Lexicographically minimal string rotation

```
def least_rotation(S: str) -> int:
  """Booth's algorithm."""
  S += S  // Concatenate string to it self to avoid modular
   arithmetic
  f = [-1] * len(S)  // Failure function
  k = 0  // Least rotation of string found so far
  for j in range(1, len(S)):
    sj = S[j]
    i = f[j - k - 1]
    while i != -1 and sj != S[k + i + 1]:
      if sj < S[k + i + 1]:
        k = j - i - 1
      i = f[i]
    if sj != S[k + i + 1]:  // if sj != S[k+i+1], then i == -1
      if sj < S[k]:  # k+i+1 = k
        k = j
      f[j - k] = -1
    else:
      f[j - k] = i + 1
  return k
```

4.7 Hash

```
long long POW[Bases][N];

struct Hash
{
  long long a[Bases];
  Hash operator+(const Hash& src)
  {
    Hash tmp;
    for (int i = 0; i < Bases; ++i) tmp.a[i] = addi(a[i], src.
   a[i]);
    return tmp;
  }
  Hash operator-(const Hash& src)
  {
    Hash tmp;
    for (int i = 0; i < Bases; ++i) tmp.a[i] = subt(a[i], src.
   a[i]);
    return tmp;
  }
  Hash operator*(int x)
  {
    Hash tmp;
    for (int i = 0; i < Bases; ++i) tmp.a[i] = mult(a[i], POW[
   i][x]);
    return tmp;
  }
  Hash operator+(char c)
  {
    Hash tmp;
    for (int i = 0; i < Bases; ++i) tmp.a[i] = addi(a[i], c);
    return tmp;
  }
  bool operator==(const Hash& src)
  {
    for (int i = 0; i < Bases; ++i) if (a[i] != src.a[i])
   return false;
    return true;
  }
};

bool operator<(const Hash& a, const Hash& b)
{
  for (int i = 0; i < Bases; ++i)
```

```
    if (a.a[i] < b.a[i]) return true;
    else if (a.a[i] > b.a[i]) return false;
    return false;
}

Hash hash1[N], hash2[N];
void initHash(int n)
{
  for (int j = 0; j < Bases; ++j) POW[j][0] = 1;
  for (int j = 0; j < Bases; ++j) for (int i = 1; i <= n; ++i)
    POW[j][i] = mult(POW[j][i - 1], base[j]);
}

void calcHash(int n)
{
  for (int j = 0; j < Bases; ++j) hash1[0].a[j] = 0;
  for (int i = 1; i <= n; ++i) hash1[i] = hash1[i - 1] * 1 + (
   s[i] - 'a');
}

void calcHashRev(int n)
{
  for (int j = 0; j < Bases; ++j) hash2[j].a[n + 1] = 0;
  for (int i = n; i >= 0; --i) hash2[i] = hash2[i + 1] * 1 + (
   s[i] - 'a');
}

Hash getHash(int l, int r) { return hash1[r] - hash1[l - 1] *
   (r - l + 1); }
Hash getHashRev(int l, int r) { return hash2[l] - hash2[r + 1]
    * (r - l + 1); }
```

4.8 Hash 2D

$$H[i][j] = H[i-1][j] * p + H[i][j-1] * q - H[i-1][j-1] * p * q + s[i][j] \tag{1}$$

$$\begin{aligned} Hash(a,b)(x,y) = H[x][y] - H[a-1][y] * p^{x-a+1} - H[x][b-1] \\ * q^{y-b+1} + H[a-1][b-1] * p^{x-a+1} * q^{y-b+1} \end{aligned} \tag{2}$$

5 Math

5.1 Inverse of 2x2 matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

5.2 Sum of divisors

If $n = p_1^{e_1} * p_2^{e_2} * \cdots * p_k^{e_k}$, then

$$sum = \frac{p_1^{e_1+1}-1}{p_1-1} * \frac{p_2^{e_2+1}-1}{p_2-1} * \cdots * \frac{p_k^{e_k+1}-1}{p_k-1}$$

5.3 Pisano period

$\pi(n) \le 6n$. $\pi(n) = 6n$ when $n = 2 \times 5^r$ for $r > 1$.

$\pi(2) = 3, \pi(5) = 20$.

If m and n are coprime, $\pi(mn) = LCM(\pi(m), \pi(n))$.

If p is prime, $\pi(p^k)$ divides $p^{k-1} \times \pi(p)$. It is conjectured that $\pi(p^k) = p^{k-1} \times \pi(p)$ for $k > 1$.

If $p \equiv 1 \ mod\ 10$ or $p \equiv 9 \ mod\ 10$, $\pi(p)$ is divisor of $p-1$.

If $p \equiv 3 \ mod\ 10$ or $p \equiv 7 \ mod\ 10$, $\pi(p)$ is divisor of $2(p+1)$.

5.4 Number Theory

$$a + b = a \oplus b + 2 \times (a \wedge b)$$

5.5 Derivatives and integrals

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

Integration by parts:

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

5.6 Sum

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

5.7 Series

$$e^x = 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\dots,\ (-\infty < x < \infty)$$
$$\ln(1+x) = x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\dots,\ (-1 < x \le 1)$$
$$\sqrt{1+x} = 1+\frac{x}{2}-\frac{x^2}{8}+\frac{2x^3}{32}-\frac{5x^4}{128}+\dots,\ (-1 \le x \le 1)$$
$$\sin x = x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\dots,\ (-\infty < x < \infty)$$
$$\cos x = 1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\dots,\ (-\infty < x < \infty)$$

5.8 Trigonometric

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

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

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

5.9 Inverse

Calculate inverse for [1, m - 1]. O(m). m is prime

```
inv[1] = 1;
for(int i = 2; i < m; ++i)
    inv[i] = m - (m/i) * inv[m%i] % m;
```

5.10 Gaussian elimination

```
// Gauss-Jordan elimination.
// Returns: number of solution (0, 1 or INF)
//   When the system has at least one solution, ans will
    contains
//   one possible solution
// Possible improvement when having precision errors:
//   - Divide i-th row by a(i, i)
//   - Choosing pivoting row with min absolute value (
    sometimes this is better that maximum, as implemented here)
// Tested:
// - https://open.kattis.com/problems/equationsolver
// - https://open.kattis.com/problems/equationsolverplus
int gauss (vector < vector<double> > a, vector<double> & ans)
    {
  int n = (int) a.size();
  int m = (int) a[0].size() - 1;

  vector<int> where (m, -1);
  for (int col=0, row=0; col<m && row<n; ++col) {
    int sel = row;
    for (int i=row; i<n; ++i)
      if (abs (a[i][col]) > abs (a[sel][col]))
        sel = i;
    if (abs (a[sel][col]) < EPS)
      continue;
    for (int i=col; i<=m; ++i)
      swap (a[sel][i], a[row][i]);
    where[col] = row;

    for (int i=0; i<n; ++i)
      if (i != row) {
        double c = a[i][col] / a[row][col];
        for (int j=col; j<=m; ++j)
          a[i][j] -= a[row][j] * c;
      }
    ++row;
  }

  ans.assign (m, 0);
  for (int i=0; i<m; ++i)
    if (where[i] != -1)
      ans[i] = a[where[i]][m] / a[where[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 (abs (sum - a[i][m]) > EPS)
      return 0;
  }

  // If we need any solution (in case INF solutions), we
     should be
  // ok at this point.
  // If need to solve partially (get which values are fixed/
     INF value):
//  for (int i=0; i<m; ++i)
//    if (where[i] != -1) {
//      REP(j,n) if (j != i && fabs(a[where[i]][j]) > EPS) {
//        where[i] = -1;
//        break;
//      }
//    }
  // Then the variables which has where[i] == -1 --> INF
     values

  for (int i=0; i<m; ++i)
    if (where[i] == -1)
      return INF;
  return 1;
}
```

5.11 Geometry

```
struct line
{
  double a,b,c;
  line() {}
  line(double A,double B,double C):a(A),b(B),c(C){}
  line(Point A,Point B)
  {
    a=A.y-B.y; b=B.x-A.x; c=-a*A.x-b*A.y;
  }
};

Point intersect(line AB,line CD)
```

```
{
  AB.c=-AB.c; CD.c=-CD.c;
  double D=CROSS(AB.a,AB.b,CD.a,CD.b);
  double Dx=CROSS(AB.c,AB.b,CD.c,CD.b);
  double Dy=CROSS(AB.a,AB.c,CD.a,CD.c);
  if (D==0.0) return Point(1e9,1e9);
  else return Point(Dx/D,Dy/D);
}
```

5.12 Discrete logarithm

```
// Returns minimum x for which a ^ x % m = b % m. O(sqrt(m))
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 = (k * 1ll * a / g) % m;
  }

  int n = sqrt(m) + 1;
  int an = 1;
  for (int i = 0; i < n; ++i)
    an = (an * 1ll * a) % m;

  unordered_map<int, int> vals;
  for (int q = 0, cur = b; q <= n; ++q) {
    vals[cur] = q;
    cur = (cur * 1ll * a) % m;
  }

  for (int p = 1, cur = k; p <= n; ++p) {
    cur = (cur * 1ll * an) % m;
    if (vals.count(cur)) {
      int ans = n * p - vals[cur] + add;
      return ans;
    }
  }
  return -1;
}
```

5.13 Miller Rabin

```
// n < 4,759,123,141        3 :  2, 7, 61
// n < 1,122,004,669,633    4 :  2, 13, 23, 1662803
// n < 3,474,749,660,383          6 :  pirmes <= 13
// n < 2^64                       7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
// test number in range [2, n - 2] to use magic
bool witness(long long a, long long n, long long u, int t) {
  long long x = power(a, u, n);
  for (int i = 0; i < t; i++) {
    long long nx = mult(x, x, n);
    if (nx == 1 && x != 1 && x != n - 1) return 1;
    x = nx;
  }
  return x != 1;
}
bool miller_rabin(long long n, int s = 20) {
  // iterate s times of witness on n
  // return 1 if prime, 0 otherwise
  if (n < 2) return 0;
  if (!(n & 1)) return n == 2;
  long long u = n - 1;
  int t = 0;
  // n-1 = u*2^t
  while (!(u & 1)) {
    u >>= 1;
    t++;
  }
  while (s--) {
    long long a = random(0, n - 2) + 1;
    if (witness(a, n, u, t)) return 0;
  }
  return 1;
}
```

5.14 Miller Rabin Deterministic

```
typedef long long ll;

// mulMod and powMod is same as Math/modulo.h.
// These 2 functions are duplicated here for easier copy-paste
    .
```

```
/**
 * When MOD < 2^63, use following mulMod:
 * Source: https://en.wikipedia.org/wiki/Modular_arithmetic#
   Example_implementations
 * On computer architectures where an extended precision
   format with at least 64 bits
 * of mantissa is available (such as the long double type of
   most x86 C compilers),
 * the following routine is faster than any algorithmic
   solution, by employing the
 * trick that, by hardware, floating-point multiplication
   results in the most
 * significant bits of the product kept, while integer
   multiplication results in the
 * least significant bits kept
 */
uint64_t mulMod(uint64_t a, uint64_t b, uint64_t m) {
    long double x;
    uint64_t c;
    int64_t r;

    if (a >= m) a %= m;
    if (b >= m) b %= m;

    x = a;
    c = x * b / m;
    r = (int64_t)(a * b - c * m) % (int64_t)m;
    return r < 0 ? r + m : r;
}

/** Calculates a^b % m */
uint64_t powMod(uint64_t a, uint64_t b, uint64_t m) {
    uint64_t r = m==1?0:1; // make it works when m == 1.
    while (b > 0) {
        if (b & 1) r = mulMod(r, a, m);
        b = b >> 1;
        a = mulMod(a, a, m);
    }
    return r;
}

bool suspect(ll a, ll s, ll d, ll n) {
    ll x = powMod(a, d, n);
    if (x == 1) return true;
    for (int r = 0; r < s; ++r) {
        if (x == n - 1) return true;
        x = mulMod(x, x, n);
    }
    return false;
}
// {2,7,61,-1}                      is for n < 4759123141 (=
   2^32)
// {2,3,5,7,11,13,17,19,23,-1} is for n < 10^15 (at least)
// add 29, 31, 37 for 64 bit integer
bool isPrime(int64_t n) {
    if (n <= 1 || (n > 2 && n % 2 == 0)) return false;
    ll test[] = {2,3,5,7,11,13,17,19,23,-1};
    ll d = n - 1, s = 0;
    while (d % 2 == 0) ++s, d /= 2;
    for (int i = 0; test[i] < n && test[i] != -1; ++i)
        if (!suspect(test[i], s, d, n)) return false;
    return true;
}
// Killer prime: 5555555557LL (fail when not used mulMod)
```

5.15 Chinese Remainer

```
// Solve linear congruences equation:
// - a[i] * x = b[i] MOD m[i] (mi don't need to be co-prime)
// Tested:
// - https://open.kattis.com/problems/generalchineseremainder
bool linearCongruences(const vector<ll> &a, const vector<ll> &
   b,
    const vector<ll> &m, ll &x, ll &M) {
  ll n = a.size();
  x = 0; M = 1;
  REP(i, n) {
    ll a_ = a[i] * M, b_ = b[i] - a[i] * x, m_ = m[i];
    ll y, t, g = extgcd(a_, m_, y, t);
    if (b_ % g) return false;
    b_ /= g; m_ /= g;
    x += M * (y * b_ % m_);
    M *= m_;
  }
  x = (x + M) % M;
  return true;
}
```

5.16 Extended Euclid

```
// other pairs are of the form:
// x' = x + k(b / gcd)
// y' = y - k(a / gcd)
// where k is an arbitrary integer.
// to minimize, set k to 2 closest integers near -x / (b / gcd
   )
// the algo always produce one of 2 small pairs.
int extgcd(int a, int b, int &x, int &y) {
  int g = a; x = 1; y = 0;
  if (b != 0) g = extgcd(b, a % b, y, x), y -= (a / b) * x;
  return g;
}
```

5.17 FFT

```
namespace FFT
{
  struct cd
  {
    double real, img;
    cd(double x = 0, double y = 0) : real(x), img(y) {}
    cd operator+(const cd& src) { return cd(real + src.real,
   img + src.img); }
    cd operator-(const cd& src) { return cd(real - src.real,
   img - src.img); }
    cd operator*(const cd& src) { return cd(real * src.real -
   img * src.img, real * src.img + src.real * img); }
  };
  cd conj(const cd& x) { return cd(x.real, -x.img); }
  const int MaxN = 1 << 15;
  const double PI = acos(-1);
  cd w[MaxN]; int rev[MaxN];

  void initFFT()
  {
    for (int i = 0; i < MaxN; ++i)
      w[i] = cd(cos(2 * PI * i / MaxN), sin(2 * PI * i / MaxN)
   );
  }
  void FFT(vector<cd>& a)
  {
    int n = a.size();
    for (int i = 0; i < n; ++i)
      if (rev[i] < i) swap(a[i], a[rev[i]]);

    for (int len = 2; len <= n; len <<= 1)
      for (int i = 0; i < n; i += len)
        for (int j = 0; j < (len >> 1); ++j)
        {
          cd u = a[i + j], v = a[i + j + (len >> 1)] * w[MaxN
   / len * j];
          a[i + j] = u + v;
          a[i + j + (len >> 1)] = u - v;
        }
  }
  void calcRev(int n)
  {
    rev[0] = 0;
    for (int i = 1; i < n; ++i)
      if (i & 1) rev[i] = rev[i - 1] + (n >> 1);
      else rev[i] = rev[i >> 1] >> 1;
  }
  vector<long long> polymul(const vector<int>& a, const vector
   <int>& b)
  {
    int n = a.size() + b.size() - 1;
    if (__builtin_popcount(n) != 1) n = 1 << (32 -
   __builtin_clz(n));

    vector<cd> pa(a.begin(), a.end()); pa.resize(n);
    vector<cd> pb(b.begin(), b.end()); pb.resize(n);

    calcRev(n); // Doesn't need to call multiple times

    FFT(pa); FFT(pb);
    for (int i = 0; i < n; ++i) pa[i] = conj(pa[i] * pb[i]);
    FFT(pa);
    //output of pa will be conj of the real answer
    vector<long long> res(n);
    for (int i = 0; i < n; ++i) res[i] = llround(pa[i].real /
   n);
    return res;
  }
};
```

5.18 PollardRho

```
// does not work when n is prime
long long modit(long long x, long long mod) {
  if (x >= mod) x -= mod;
  //if(x<0) x+=mod;
  return x;
}
long long mult(long long x, long long y, long long mod) {
  long long s = 0, m = x % mod;
  while (y) {
    if (y & 1) s = modit(s + m, mod);
    y >>= 1;
    m = modit(m + m, mod);
  }
  return s;
}
long long f(long long x, long long mod) {
  return modit(mult(x, x, mod) + 1, mod);
}
long long pollard_rho(long long n) {
  if (!(n & 1)) return 2;
  while (true) {
    long long y = 2, x = random() % (n - 1) + 1, res = 1;
    for (int sz = 2; res == 1; sz *= 2) {
      for (int i = 0; i < sz && res <= 1; i++) {
        x = f(x, n);
        res = __gcd(abs(x - y), n);
      }
      y = x;
    }
    if (res != 0 && res != n) return res;
  }
}
```

6 Theorem

6.1 Fermat's little theorem

If p is a prime number, then for any number a, $a^p - a$ is an integer multiple of p

$$a^p \equiv a \ (mod\ p)$$

If a is not divisible by p

$$a^{p-1} \equiv 1 \ (mod\ p)$$

6.2 Euler's theorem

If a and n are coprime, then

$$a^{\phi(n)} \equiv 1 \ (mod\ n)$$

6.3 Euler's totient function

The number of coprime $\leq n$

$$\phi(n) = n \prod (1 - \frac{1}{p})$$

With p is the prime divided by n

6.4 Goldbach's conjecture

Every even number greater than 2 is the sum of 2 primes. $\leq 4 * 10^{18}$

6.5 Dirichlet

Given n holes and $n+1$ pigeons to distribute evenly, then at least 1 hole must have 2 pigeons

6.6 Pythagorean triple

$$a = m^2 - n^2, \ b = 2mn, \ c = m^2 + n^2$$

Where m and n are positive integer with $m > n$, and with m and n are coprime and not both odd. When both m and n are odd, then a, b, and c will be even, and the triple will not be primitive; however, dividing a, b, and c by 2 will yield a primitive triple when m and n are coprime and both odd.

Despite generating all primitive triples, Euclid's formula does not produce all triples—for example, (9, 12, 15) cannot be generated using integer m and n. This can be remedied by inserting an additional parameter k to the formula. The following will generate all Pythagorean triples uniquely:

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

Where m, n, and k are positive integers with $m > n$, and with m and n coprime and not both odd.

6.7 Legendre's formula

Factor $n!$

$$v_p(n!) = \sum_{i=1}^{\infty} \left\lfloor \frac{n}{p^i} \right\rfloor$$

With p is prime

6.8 Stirling's approximation

$$n! \approx \sqrt{2\pi n} * (\frac{n}{e})^n$$

6.9 Wilson's theorem

$n > 1$ is a prime if and only if

$$(n-1)! \ mod \ n \equiv -1 \ mod \ n$$

7 Other

7.1 Matrix

```
struct matrix
{
  static const int MATRIX_SIZE = 2;
  long long a[MATRIX_SIZE][MATRIX_SIZE];
  matrix()
  {
    for (int i = 0; i < MATRIX_SIZE; ++i)
      for (int j = 0; j < MATRIX_SIZE; ++j)
        a[i][j] = 0;
  }
  matrix(bool x) : matrix()
  {
    for (int i = 0; i < MATRIX_SIZE; ++i) a[i][i] = 1;
  }
};

matrix matmul(const matrix& a, const matrix& b, long long m =
   mod)
{
  int n = a.MATRIX_SIZE;
  matrix res;
  for (int ii = 0; ii < n; ++ii) for (int jj = 0; jj < n; ++jj
   )
  {
    res.a[ii][jj] = 0;
    for (int kk = 0; kk < n; ++kk)
      res.a[ii][jj] = addi(res.a[ii][jj], mult(a.a[ii][kk], b.
   a[kk][jj], m), m);
  }
  return res;
}

matrix matpow(const matrix& a, long long n, long long m = mod)
{
  if (n == 0) return matrix(true);
  matrix tmp = matpow(a, n >> 1, m);
  return (n & 1) ? matmul(matmul(tmp, tmp, m), a, m) : matmul(
   tmp, tmp, m);
}
```

7.2 Bignum mul

```
string mul(string a,string b)
{
  int m=a.length(),n=b.length(),sum=0;
  string c="";
  for (int i=m+n-1; i>=0; i--)
  {
    for (int j=0; j<m; j++) if (i-j>0 && i-j<=n) sum+=(a[j]-'0
   ')*(b[i-j-1]-'0');
    c=(char)(sum%10+'0')+c;
    sum/=10;
  }
```

```
  while (c.length()>1 && c[0]=='0') c.erase(0,1);
  return c;
}
```

7.3 Random

```
// Random using mt19937
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());

// For random long long
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().
    count());

// Random shuffle using mt19937 as the generator
shuffle(a.begin(), a.end(), rng);

// Random range
int random(int a, int b)
{
  return uniform_int_distribution<int>(a, b)(rng);
}
```

7.4 Builtin bit function

```
__builtin_popcount(x); // number of bit 1 in x
__builtin_popcountll(x); // for long long
__builtin_clz(x); // number of leading 0
__builtin_clzll(x); // for long long
__builtin_ctz(x); // number of trailing 0
__builtin_ctzll(x); // for long long

(x & ~x) : the smallest bit 1 in x
floor(log2(x)) : 31 - __builtin_clz(x | 1);
floor(log2(x)) : 63 - __builtin_clzll(x | 1);
```

7.5 Pythagorean triples

c under 100 there are 16 triples: (3, 4, 5) (5, 12, 13) (8, 15, 17) (7, 24, 25) (20, 21, 29) (12, 35, 37) (9, 40, 41) (28, 45, 53) (11, 60, 61) (16, 63, 65) (33, 56, 65) (48, 55, 73) (13, 84, 85) (36, 77, 85) (39, 80, 89) (65, 72, 97)

$100 \le c \le 300$: (20, 99, 101) (60, 91, 109) (15, 112, 113) (44, 117, 125) (88, 105, 137) (17, 144, 145) (24, 143, 145) (51, 140, 149) (85, 132, 157) (119, 120, 169) (52, 165, 173) (19, 180, 181) (57, 176, 185) (104, 153, 185) (95, 168, 193) (28, 195, 197) (84, 187, 205) (133, 156, 205) (21, 220, 221) (140, 171, 221) (60, 221, 229) (105, 208, 233) (120, 209, 241) (32, 255, 257) (23, 264, 265) (96, 247, 265) (69, 260, 269) (115, 252, 277) (160, 231, 281) (161, 240, 289) (68, 285, 293)

7.6 Sieve

```
// faster for > 1e6
void sieve_new()
{
  for (int i = 2; i <= 1000000; ++i)
  {
    if (!notPrime[i]) prime.push_back(i);
    for (int j = 0; i * prime[j] <= 1000000 && j < prime.size
   (); ++j) {
      notPrime[i * prime[j]] = true;
      if (i % prime[j] == 0) break;
    }
  }
}
//
void sieve_old()
{
  for (long long i = 2; i <= 1000000; ++i)
  if (!notPrime[i]) {
    prime.push_back(i);
    for (long long j = i; j * i <= 1000000; ++j)
      notPrime[i * j] = true;
  }
}
```

7.7 Factorial mod

```
int factmod(int n, int p) {
    vector<int> f(p);
    f[0] = 1;
    for (int i = 1; i < p; i++)
        f[i] = f[i-1] * i % p;

    int res = 1;
    while (n > 1) {
```

```
        if ((n/p) % 2)
            res = p - res;
        res = res * f[n%p] % p;
        n /= p;
    }
    return res;
}
```

7.8 Catalan

$$\frac{(2n)!}{(n+1)!\,n!} = \prod_{k=2}^{n} \frac{n+k}{k}$$

The Catalan number Cn is the solution for

- Number of correct bracket sequence consisting of n opening and n closing brackets.
- The number of rooted full binary trees with $n+1$ leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
- The number of ways to completely parenthesize $n+1$ factors.
- The number of triangulations of a convex polygon with $n+2$ sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
- The number of ways to connect the $2n$ points on a circle to form n disjoint chords.
- The number of non-isomorphic full binary trees with n internal nodes (i.e. nodes having at least one son).
- The number of monotonic lattice paths from point (0,0) to point (n,n) in a square lattice of size $n \times n$, which do not pass above the main diagonal (i.e. connecting (0,0) to (n,n)).
- Number of permutations of length n that can be stack sorted (i.e. it can be shown that the rearrangement is stack sorted if and only if there is no such index $i < j < k$, such that $a_k < a_i < a_j$).
- The number of non-crossing partitions of a set of n elements.
- The number of ways to cover the ladder $1...n$ using n rectangles (The ladder consists of n columns, where i^{th} column has a height i).

7.9 Prime under 100

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

7.10 Pascal triangle

C(n,k)=number from line 0, column 0
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1

7.11 Fibo

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765

8 Tips

- 5' debug vẫn nhanh hơn 20' penalty
- Giả sử nó là số nguyên tố đi. Giả sử nó liên quan tới số nguyên tố đi.
- Giả sử nó là số có dạng 2^n đi.
- Giả sử chọn tối đa là 2, 3 số gì là có đáp án đi.
- Có liên quan gì tới Fibonacci hay tam giác pascal?
- Dãy này đơn điệu không em ei? Hay tổng của 2, 3 số fibonacci?

- Chia nhỏ ra xem.
- Random shuffe để AC
- Xoay mảng 45 độ (thường liên quan đến Manhattan)
- Tạo đỉnh ảo cho đồ thị (vd như Kruskal)
- Tìm t thỏa điều kiện nào đó thì chặt
- Merge set thì phải merge từ set nhỏ sang lớn ko thì TLE
- Xử lý ma trận cũng giống xử lý số bình thường, các phép nhân chia mod đều như cũ
- Làm luồng nhớ push cung ngược
- Nếu xử lý liên quan đến bit thì có thể nó liên quan tới Trie
- Số nguyên tố thường có dạng $6k \pm 1$ trừ 2 và 3
- Không biết làm thì dẹp, ngủ 30 phút rồi biết làm, đừng cố mò đường trong brain fog.