Team Bhagwan Bharose

Table of Contents

Trie

```
// Trie
struct trie_node {
    int word_count;
    int prefix_count;
    trie_node* edges[26];
    trie_node() {
        this->word_count = 0;
        this->prefix_count = 0;
        for(int i = 0; i < 26; i++) {
            this->edges[i] = NULL;
        }
    }
};
// Add string s using add(root,s,0)
void add_word(trie_node* &root, string
&str, int index) {
    if (index == str.length()) {
        root->word_count += 1;
        return;
    }
    root->prefix_count += 1;
    int i = int(str[index] - 'a');
    if(root->edges[i]==NULL){
        root->edges[i]=new trie_node();
    }
    add_word(root->edges[i], str, index+1);
}
// count_prefix(root,s,0) num. of matches
of s in trie prefix
int count_prefix(trie_node* root, string
&prefix, int index) {
    if (index == prefix.length()) {
        return root->prefix_count;
    }
    int i = int(prefix[index] - 'a');
    if (root->edges[i] == NULL) {
        return 0;
    }
    return count_prefix(root->edges[i],
prefix, index+1);
}
```

WeightedMatching

```
// Description: Min cost bipartite
matching. Negate costs for max cost.
// Time: O(N^3)

typedef vector<double> vd;
bool zero(double x) { return fabs(x) < 1e-
10; }
double MinCostMatching(const vector<vd>&
cost, vi& L, vi& R) {
    int n = sz(cost), mated = 0;
    vd dist(n), u(n), v(n);
    vi dad(n), seen(n);
    /// construct dual feasible solution
    rep(i,0,n) {
        u[i] = cost[i][0];
        rep(j,1,n) u[i] = min(u[i], cost[i]
[j]);
    }
    rep(j,0,n) {
        v[j] = cost[0][j] - u[0];
        rep(i,1,n) v[j] = min(v[j], cost[i]
[j] - u[i]);
```

AhoCorasick

```
/**
 * Description: Aho-Corasick tree is used
for multiple pattern matching.
 * Initialize the tree with
create(patterns). 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.
 * Time: Function create is $O(26N)$ where
$N$ is the sum of length of patterns.
 * find is $O(M)$ where $M$ is the length
of the word. findAll is $O(NM)$.
 * Status: lightly tested
 */
#pragma once

struct AhoCorasick {
    enum {alpha = 26, first = 'A'};
    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;
    vector<int> backp;
    void insert(string& s, int j) {
        assert(!s.empty());
        int n = 0;
        trav(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.emplace_back(-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()) {
```

```
    }
    /// find primal solution satisfying
complementary slackness
    L = R = vi(n, -1);
    rep(i,0,n) rep(j,0,n) {
        if (R[j] != -1) continue;
        if (zero(cost[i][j] - u[i] - v[j]))
{
            L[i] = j;
            R[j] = i;
            mated++;
            break;
        }
    }
    for (; mated < n; mated++) { // until
solution is feasible
        int s = 0;
        while (L[s] != -1) s++;
        fill(all(dad), -1);
        fill(all(seen), 0);
        rep(k,0,n)
            dist[k] = cost[s][k] - u[s] -
v[k];

        int j = 0;
        for (;;) { /// find closest
            j = -1;
            rep(k,0,n){
                if (seen[k]) continue;
                if (j == -1 || dist[k] <
dist[j]) j = k;
            }
            seen[j] = 1;
            int i = R[j];
            if (i == -1) break;
            rep(k,0,n) { /// relax
neighbors
                if (seen[k]) continue;
                auto new_dist = dist[j] +
cost[i][k] - u[i] - v[k];
                if (dist[k] > new_dist) {
                    dist[k] = new_dist;
                    dad[k] = j;
                }}    }
        /// update dual variables
        rep(k,0,n) {
            if (k == j || !seen[k])
continue;
            auto w = dist[k] - dist[j];
            v[k] += w, u[R[k]] -= w;
        }
        u[s] += dist[j];
        /// augment along path
        while (dad[j] >= 0) {
            int d = dad[j];
            R[j] = R[d];
            L[R[j]] = j;
            j = d;
        }
        R[j] = s;
        L[s] = j;
    }
    auto value = vd(1)[0];
    rep(i,0,n) value += cost[i][L[i]];
    return value;
}
```

XorSubSum

```
            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;
        trav(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;
    }
};
```

BinomialModPrime

```
//Lucas Theorem: Find (n Choose m) mod p
for prime p and large n,m. in O(log(m*n))
// nCm mod p by lucas theorem for large n,m
>=0
// p prime, require fact(factorial) &
invfact(inverse factorial)
v32 fact,invfact;
ll lucas(ll n,ll m,int p){
    ll res=1;
    while(n || m) {
        ll a=n%p,b=m%p;
        if(a<b) return 0;
        res=((res*fact[a]%p)*
(invfact[b]%p)%p)*(invfact[a-b]%p)%p;
        n/=p; m/=p;
    }
    return res;
}
```

EulerWalk

```
//Description: Eulerian undirected/directed
path/cycle algorithm. Returns a list of
```

```
// Basis for xor subsets
struct Gaussbase2{
    int numofbits=20;
    int rk=0;
    v32 Base;
    Gaussbase2(){
        Base.assign(numofbits,0);
    }
    Gaussbase2& operator = (Gaussbase2 &g){
        forn(i,numofbits)
Base[i]=g.Base[i];
        rk=g.rk;
    }
    bool canbemade(int x){
        rforn(i,numofbits-1)
x=min(x,x^Base[i]);
        return x==0;
    }
    void Add(int x){
        rforn(i,numofbits){
            if((x>>i)&1){
                if(!Base[i]){
                    Base[i]=x;
                    rk++;
                    return;
                }else x^=Base[i];   }  } }
};
```

Z

```
// Z Algorithm
// Z[i] is the length of the longest
substring starting from S[i]
// which is also a prefix of S
// O(n)
void z_func(v32 &s,v32 &z){
    int L=0,R=0;
    int sz=s.size();
    z.assign(sz,0);
    forsn(i,1,sz){
      if(i>R){
        L=R=i;
        while(R<sz && s[R-L]==s[R]) R++;
        z[i]=R-L; R--;
      }else{
        int k=i-L;
        if(z[k]<R-i+1) z[i]=z[k];
        else{
          L=i;
          while(R<sz && s[R-L]==s[R]) R++;
          z[i]=R-L; R--;    }}    }}
```

bsearch

```
// Range=[l,r] & monotonically inc
predicate P,
// find the smallest for which P(x) holds
true. return r if fail
while (l < r) {
  int mid = (l + r) / 2;
  if (P(mid)) r = mid;
  else l = mid + 1;
} // after the loop, l = r = x
// if monotonically decreasing predicate P,
for largest x. l if fail
while (l < r) {
  int mid = (l + r + 1) / 2;
  if (P(mid)) l = mid;
  else r = mid - 1;
```

```
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, also put it->second in s (and
then ret). Time: O(E)
struct V {
    vector<pii> outs; // (dest, edge index)
    int nins = 0;
};
vi euler_walk(vector<V>& nodes, int nedges,
int src=0) {
    int c = 0;
    trav(n, nodes) c += abs(n.nins -
sz(n.outs));
    if (c > 2) return {};
    vector<vector<pii>::iterator> its;
    trav(n, nodes)
        its.push_back(n.outs.begin());
    vector<bool> eu(nedges);
    vi ret, s = {src};
    while(!s.empty()) {
        int x = s.back();
        auto& it = its[x], end =
nodes[x].outs.end();
        while(it != end && eu[it->second])
++it;
        if(it == end) { ret.push_back(x);
s.pop_back(); }
        else { s.push_back(it->first);
eu[it->second] = true; }
    }
    if(sz(ret) != nedges+1)
        ret.clear(); // No Eulerian
cycles/paths.
    // else, non-cycle if ret.front() !=
ret.back()
    reverse(all(ret));
    return ret;
}
```

FFT

```
void fft(vector<cd> & a, bool invert) {
//invert = 1 for inverse FFT
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if (i < j)
            swap(a[i], a[j]); }
    for (int len = 2; len <= n; len <<= 1)
{
        double ang = 2 * PI / len * (invert
? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2;
j++) {
                cd u = a[i+j], v =
a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;      }   }   }
    if (invert) {
        for (cd & x : a)
            x /= n;   } }
```

```
} // after the loop, l = r = x
// search in sorted array
bool search(int x[], int n, int k) {
  int l = 0, r = n - 1;
  while (l < r) {
    int mid = (l + r) / 2;
    if (x[mid] >= k) r = mid;
    else l = mid + 1;
  }
  return x[l] == k;
}
```

centroid

```
vector<set<ll>> adj;
vector<ll> parent, subtree_size;
ll dfs(ll u, ll p) {
    subtree_size[u] = 1;
    for (auto v : adj[u]) {
        if (v != p) {
            subtree_size[u] += dfs(v,u);
        }
    }
    return subtree_size[u];
}
ll find_centroid(ll u, ll p, ll n) {
    for (auto v : adj[u]) {
        if (v != p and subtree_size[v] >
n/2) {
            return find_centroid(v, u, n);
        }
    }
    return u;
}
void decompose(ll u, ll p = 0) {
    ll n = dfs(u, p);
    ll centroid = find_centroid(u, p, n);
    if (p == 0) {p = centroid;}
    parent[centroid] = p;
    for (auto v : adj[centroid]) {
        adj[v].erase(centroid);
        decompose(v, centroid);
    }
    adj[centroid].clear();
}
```

chinese

```
// Chinese Remainder Theorem: x = a (mod m)
and x = b (mod n)
// Time: log(m + n) - extended euclidean is
euclid.h
template<class Z> Z chinese(Z a, Z m, Z b,
Z n) {
    Z x, y; euclid(m, n, x, y);
    Z ret = a * (y + m) % m * n + b * (x +
n) % n * m;
    if (ret >= m * n) ret -= m * n;
    return ret;
}
template<class Z> Z chinese_common(Z a, Z
m, Z b, Z n) {
    Z d = gcd(m, n);
    if (((b -= a) %= n) < 0) b += n;
    if (b % d) return -1; // No solution
    return d * chinese(Z(0), m/d, b/d, n/d)
+ a;
}
```

```
vector<int> multiply(vector<int> const& a,
vector<int> const& b) {
//function to multiply two polynomials
    vector<cd> fa(a.begin(), a.end()),
fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);
    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);
    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

FenwickTree

```
struct ft{
    vector<ll> s;
    int n;
    ft(int n) : s(n+10),n(n){}
    // Add zero based
    void add(int pos,ll val){
        ++pos;
        while(pos<s.size()){
            s[pos]+=val;
            pos+=pos&-pos;
        }
    }
    ll get(int pos){
        ++pos; ll ans=0;
        while(pos>0){
            ans+=s[pos];
            pos-=pos&-pos;
        }
        return ans;
    }
    int lower_bound(ll sum){
        int l=0,r=n-1;
        while(l<r){
              int mid=(l+r)/2;
              if (get(mid)>=sum) r=mid;
              else l=mid+1;
        } // after the loop, l = r = x
        if(get(l)>=sum) return l;
        else return n;
    }
};
struct FenwickTree2D {
    vector<vector<int>> bit;
    int n, m;
    // init(...) { ... }
    int sum(int x, int y) {
        int ret = 0;
        for (int i = x; i >= 0; i = (i & (i
+ 1)) - 1)
            for (int j = y; j >= 0; j = (j
& (j + 1)) - 1)
                ret += bit[i][j];
        return ret;
    }
    void add(int x, int y, int delta) {
        for (int i = x; i < n; i = i | (i +
```

cyclefind

```
p32 floydCycleFinding(int x0) { // function
int f(int x) is defined earlier
// 1st part: finding k*mu, hareâ€™s speed
is 2x tortoiseâ€™s
int tortoise = f(x0), hare = f(f(x0));
// f(x0) is the node next to x0
while (tortoise != hare) { tortoise =
f(tortoise); hare = f(f(hare)); }
// 2nd part: finding mu, hare and tortoise
move at the same speed
int mu = 0; hare = x0;
while (tortoise != hare) { tortoise =
f(tortoise); hare = f(hare); mu++;}
// 3rd part: finding lambda, hare moves,
tortoise stays
int lambda = 1; hare = f(tortoise);
while (tortoise != hare) { hare = f(hare);
lambda++; }
return p32(mu, lambda);
}
```

dij

```
v32 dist[LIM];
void dij(int s){
    priority_queue< p32, vector<p32>,
greater<p32> > pq;
    pq.push(mp(0, s));
    while (!pq.empty()) {
        p32 front = pq.top(); pq.pop(); int
d = front.first, u = front.second;
        if (d > dist[u]) continue;
        for (int j = 0; j <
(int)adj[u].size(); j++) {
            p32 v = AdjList[u][j];
            if (dist[u] + v.second <
dist[v.first]) {
                dist[v.first] = dist[u] +
v.second;
            pq.push(mp(dist[v.first],
v.first));
    } } }
}
```

editDist

```
int editDistance(char s1[],char s2[])  {
    int m=strlen(s1);
    int n=strlen(s2);
    int dp[m+1][n+1];
    for(int i=0;i<=m;i++)   {
        for(int j=0;j<=n;j++)   {
            if(i==0)
                dp[i][j]=j;
            else if(j==0)
                dp[i][j]=i;
            else if(s1[i-1]==s2[j-1])
                dp[i][j]=dp[i-1][j-1];
            else
                dp[i][j]=1+min(dp[i][j-
1],min(dp[i-1][j],dp[i-1][j-1]));
        }
    }
    return dp[m][n];
}
```

```
1))
            for (int j = y; j < m; j = j |
(j + 1))
                bit[i][j] += delta;
    }
};
```

HLD

```
// Decomposes a tree into vertex disjoint
heavy paths and light  edges such that the
path from any leaf to the root contains at
most log(n) light edges. The function of
the HLD can be changed by modifying T, LOW
and f. f is assumed to be associative and
commutative. Usage: HLD hld(G);
hld.update(index, value);  tie(value, lca)
= hld.query(n1, n2); SegTree required

typedef vector<pii> vpi;
struct Node {
    int d, par, val, chain = -1, pos = -1;
};
struct Chain {
    int par, val;
    vector<int> nodes;
    Tree tree; };
struct HLD {
    typedef int T;
    const T LOW = -(1<<29);
    void f(T& a, T b) { a = max(a, b); }
    vector<Node> V;
    vector<Chain> C;
    HLD(vector<vpi>& g) : V(sz(g)) {
        dfs(0, -1, g, 0);
        trav(c, C) {
            c.tree = {sz(c.nodes), 0};
            for (int ni : c.nodes)
                c.tree.update(V[ni].pos,
V[ni].val);}}
    void update(int node, T val) {
        Node& n = V[node]; n.val = val;
        if (n.chain != -1)
C[n.chain].tree.update(n.pos, val);}
    int pard(Node& nod) {
        if (nod.par == -1) return -1;
        return V[nod.chain == -1 ? nod.par
: C[nod.chain].par].d;}
    // query all *edges* between n1, n2
    pair<T, int> query(int i1, int i2) {
        T ans = LOW;
        while(i1 != i2) {
            Node n1 = V[i1], n2 = V[i2];
            if (n1.chain != -1 && n1.chain
== n2.chain) {
                int lo = n1.pos, hi =
n2.pos;
                if (lo > hi) swap(lo, hi);
                f(ans,
C[n1.chain].tree.query(lo, hi));
                i1 = i2 =
C[n1.chain].nodes[hi];
            } else {
                if (pard(n1) < pard(n2))
                    n1 = n2, swap(i1, i2);
                if (n1.chain == -1)
                    f(ans, n1.val), i1 =
n1.par;
                else {
```

euclid

```
// Extented GCD
inline ll gcd(ll a, ll b) { return __gcd(a,
b); }

ll euclid(ll a, ll b, ll &x, ll &y) {
    if (b) { ll d = euclid(b, a % b, y, x);
        return y -= a/b * x, d; }
    return x = 1, y = 0, a;
}
```

factor

```
/*Pollard's rho algorithm. It is a
probabilistic factorisation algorithm,
whose expected time complexity is good.
Before you start using it, run init(bits),
where bits is the length of the numbers you
use. Returns factors of the input without
duplicates. Time: Expected running time
should be good enough for 50-bit numbers.*/
 vector<ull> pr;
ull f(ull a, ull n, ull &has) {
    return (mod_mul(a, a, n) + has) % n;
}
vector<ull> factor(ull d) {
    vector<ull> res;
    for (int i = 0; i < sz(pr) &&
pr[i]*pr[i] <= d; i++)
        if (d % pr[i] == 0) {
            while (d % pr[i] == 0) d /=
pr[i];
            res.push_back(pr[i]);
        }
    //d is now a product of at most 2
primes.
    if (d > 1) {
        if (prime(d))
            res.push_back(d);
        else while (true) {
            ull has = rand() % 2321 + 47;
            ull x = 2, y = 2, c = 1;
            for (; c==1; c = __gcd((y > x ?
y - x : x - y), d)) {
                x = f(x, d, has);
                y = f(f(y, d, has), d,
has);
            }
            if (c != d) {
                res.push_back(c); d /= c;
                if (d != c)
res.push_back(d);
                break;}}}
    return res;
}
void init(int bits) {//how many bits do we
use?
    vi p = eratosthenes_sieve(1 << ((bits +
2) / 3));
    pr.assign(all(p));
}
```

floyd

```
int inf = MOD;
void floydWarshall(vv32 & m) {
    int n = (int)m.size();//m in inupt ->
dist i ,j and others as inf;
```

```
          Chain& c = C[n1.chain];
          f(ans, n1.pos ?
c.tree.query(n1.pos, sz(c.nodes))
                 :
c.tree.s[1]);
          i1 = c.par;    }    } }
    return make_pair(ans, i1);    }
  // query all *nodes* between n1, n2
  pair<T, int> query2(int i1, int i2) {
    pair<T, int> ans = query(i1, i2);
    f(ans.first, V[ans.second].val);
    return ans;    }
  pii dfs(int at, int par, vector<vpi>&
g, int d) {
    V[at].d = d; V[at].par = par;
    int sum = 1, ch, nod, sz;
    tuple<int,int,int> mx(-1,-1,-1);
    trav(e, g[at]){
      if (e.first == par) continue;
      tie(sz, ch) = dfs(e.first, at,
g, d+1);
      V[e.first].val = e.second;
      sum += sz;
      mx = max(mx, make_tuple(sz,
e.first, ch));}
    tie(sz, nod, ch) = mx;
    if (2*sz < sum) return pii(sum,
-1);
    if (ch == -1) { ch = sz(C);
C.emplace_back(); }
    V[nod].pos = sz(C[ch].nodes);
    V[nod].chain = ch;
    C[ch].par = at;
    C[ch].nodes.push_back(nod);
    return pii(sum, ch);    }};
```

Hashing

```
struct Hashtable{
  int sz;
  v64 hash1,hash2,inv1,inv2;
  ll MOD1=1e9+7,MOD2=1e9+9;
  ll pr1=31,pr2=37;
  void create(string &p){
    int len=p.size(); sz=len;

hash1.resize(len);hash2.resize(len);
    inv1.resize(len);inv2.resize(len);
    ll p1=1,p2=1;
    int i=0;
    while(p[i]){
      hash1[i]= (i==0)? 0:hash1[i-1];
      hash2[i]= (i==0)? 0:hash2[i-1];
      hash1[i]=
(hash1[i]+p[i]*p1)%MOD1;
      hash2[i]=
(hash2[i]+p[i]*p2)%MOD2;
      p1=p1*pr1%MOD1;
      p2=p2*pr2%MOD2;
      i++;
    }
    ll
iv1=inv(pr1,MOD1),iv2=inv(pr2,MOD2);
    inv1[0]=1,inv2[0]=1;
    forsn(i,1,len){
      inv1[i]=inv1[i-1]*iv1%MOD1;
      inv2[i]=inv2[i-1]*iv2%MOD2;
    }
  }
```

```
  forn(i,n) m[i][i] = min(m[i][i], {});
  forn(k,n) forn(i,n) forn(j,n)
    if (m[i][k] != inf && m[k][j] !=
inf){
      auto newDist = max(m[i][k] +
m[k][j], -inf);
    m[i][j] = min(m[i][j], newDist);
  }
  forn(k,0,n) if (m[k][k] < 0) forn(i,n)
rep(j,0,n)
  if (m[i][k] != inf && m[k][j] != inf)
m[i][j] = -inf;
}
```

geo

```
// 2D
typedef double T;
typedef complex<T> pt;
#define xc real()
#define yc imag()
const double PI=acos(-1);
pt operator *(pt p,T x){return
{p.xc*x,p.yc*x};}
pt operator /(pt p,T x){return
{p.xc/x,p.yc/x};}
//pt operator -(pt p,pt q){return {p.xc-
q.xc,p.yc-q.yc};}
T dot(pt v, pt w) {return (conj(v)*w).xc;}
T cross(pt v, pt w) {return
(conj(v)*w).yc;}
pt rot(pt p, double a) {return p *
polar(1.0, a);}
pt scale(pt c, double factor, pt p) {return
c + (p-c)*factor;}
pt translate(pt v, pt p) {return p+v;}
T sq(pt p) {return p.xc*p.xc + p.yc*p.yc;}
double abs(pt p) {return sqrt(sq(p));}
pt perp(pt p) {return {-p.yc, p.xc};}
bool isPerp(pt v, pt w) {return dot(v,w) ==
0;}
double angle(pt v, pt w) {
  double cosTheta = dot(v,w) / abs(v) /
abs(w);
  return acos(max(-1.0, min(1.0,
cosTheta)));
}
// AB to AC left for +ve
T orient(pt a, pt b, pt c) {return cross(b-
a,c-a);}
bool inAngle(pt a, pt b, pt c, pt p) {
  assert(orient(a,b,c) != 0);
  if (orient(a,b,c) < 0) swap(b,c);
  return orient(a,b,p) >= 0 &&
orient(a,c,p) <= 0;
}
double orientedAngle(pt a, pt b, pt c) {
  if (orient(a,b,c) >= 0) return angle(b-
a, c-a);
  else return 2*PI - angle(b-a, c-a);
}
bool isConvex(vector<pt> p) {
  bool hasPos=false, hasNeg=false;
  for (int i=0, n=p.size(); i<n; i++) {
    int o = orient(p[i], p[(i+1)%n],
p[(i+2)%n]);
    if (o > 0) hasPos = true;
    if (o < 0) hasNeg = true;
  }
```

```
    // Get hash of segment [l,r) l>=0,r>=1
based
    p64 gethash(int l,int r){
        ll ans1=hash1[r-1];
        if(l!=0) ans1+=MOD1-hash1[l-1];
        ll ans2=hash2[r-1];
        if(l!=0) ans2+=MOD2-hash2[l-1];
        ans1=ans1*inv1[l]%MOD1;
        ans2=ans2*inv2[l]%MOD2;
        return mp(ans1,ans2);
    }
    // Check if string q is in p :O(|p+q|)
    bool checkinside(Hashtable &q){
        int l=q.sz;
        auto ch=q.gethash(0,l);
        bool ans=0;
        forn(i,sz){
            if(i+l>sz) break;
            auto ch1=gethash(i,i+l);
            if(ch==ch1) return 1;}}};
```

IntPerm

```
// Permutaion V of 1..n mapped to
{0,1,..,n!-1} lexithographically
int permToInt(v32& v){
    int use=0,i=0,r=0;
    forstl(x,v) r=r*(++i)+bitcount(use&-
(1<<x)),use|=1<<x;
    return r;
}
```

IntervalContainer

```
/* Description: Add and remove intervals
from a set of disjoint intervals.
 * Will merge the added interval with any
overlapping intervals in the set when
adding.
 * Intervals are [inclusive, exclusive).
 * Time: O(log N)*/
#pragma once

set<pii>::iterator addInterval(set<pii>&
is, int L, int R) {
    if (L == R) return is.end();
    auto it = is.lower_bound({L, R}),
before = it;
    while (it != is.end() && it->first <=
R) {
        R = max(R, it->second);
        before = it = is.erase(it);
    }
    if (it != is.begin() && (--it)->second
>= L) {
        L = min(L, it->first);
        R = max(R, it->second);
        is.erase(it);
    }
    return is.insert(before, {L,R});
}

void removeInterval(set<pii>& is, int L,
int R) {
    if (L == R) return;
    auto it = addInterval(is, L, R);
    auto r2 = it->second;
    if (it->first == L) is.erase(it);
    else (int&)it->second = L;
```

```
    return !(hasPos && hasNeg);
}
bool half(pt p) { // true if in upper half
    if(p.xc==0 && p.yc==0) return 0;
    assert(p.xc != 0 || p.yc != 0); // the
argument of (0,0) is undefined
    return p.yc > 0 || (p.yc == 0 && p.xc <
0);
}
void polarSortAround(pt o, vector<pt> &v) {
    sort(v.begin(), v.end(), [&](pt v, pt
w) {
        return make_tuple(half(v-o),
0,sq(v-o)) < make_tuple(half(w-o), cross(v-
o, w-o),sq(w-o));
    });
}
vector<pt> convexHull(vector<pt> P){
    int i, j, n = (int)P.size();
    if (n <= 3) {
        if (!(P[0] == P[n-1]))
P.push_back(P[0]); // safeguard
        return P;
    }
    int P0 = 0;
    for (i = 1; i < n; i++)
        if (P[i].yc < P[P0].yc || (P[i].yc
== P[P0].yc && P[i].xc > P[P0].xc))
            P0 = i;
    pt temp = P[0];
    P[0] = P[P0]; P[P0] = temp;
    polarSortAround(P[0],P);
    forstl(it,P) cout<<it<<" ";
    cout<<ln;
    vector<pt> S;
    S.push_back(P[n-1]); S.push_back(P[0]);
S.push_back(P[1]);
    i = 2;
    while (i < n) {
        // note: N must be >= 3 for this
method to work
        if(S.empty()) break;
        j = (int)S.size()-1;
        if (orient(S[j-1], S[j], P[i])>0) {
            S.push_back(P[i++]);
        }
        // left turn, accept
        else S.pop_back();
    }
    // or pop the top of S until we have a
left turn
    // remove last elem as cycle formed
    if(!S.empty()) S.pop_back();
    return S;
}
struct line {
    pt v; T c;
    // From direction vector v and offset c
    line(pt v, T c) : v(v), c(c) {}
    // From equation ax+by=c
    line(T a, T b, T c) : v({b,-a}), c(c)
{}
    // From points P and Q
    line(pt p, pt q) : v(q-p),
c(cross(v,p)) {}
    T side(pt p) {return cross(v,p)-c;}
    double dist(pt p) {return abs(side(p))
/ abs(v);}
    double sqDist(pt p) {return
    if (R != r2) is.emplace(R, r2);
}
```

IntervalCover

```
/* Description: Compute indices of smallest
set of intervals covering another interval.
 * Intervals should be [inclusive,
exclusive). To support [inclusive,
inclusive],
 * change (A) to add {|| R.empty()}.
Returns empty set on failure (or if G is
empty).
 * Time: O(Nlog N)*/
template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>>
I) {
    vi S(sz(I)), R;
    iota(all(S), 0);
    sort(all(S), [&](int a, int b) { return
I[a] < I[b]; });
    T cur = G.first;
    int at = 0;
    while (cur < G.second) { // (A)
        pair<T, int> mx = make_pair(cur,
-1);
        while (at < sz(I) && I[S[at]].first
<= cur) {
            mx = max(mx,
make_pair(I[S[at]].second, S[at]));
            at++;}
        if (mx.second == -1) return {};
        cur = mx.first;
        R.push_back(mx.second);}
    return R;
}
```

KMP

```
//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: O(n)
v32 pi(const string& s){
    v32 p(s.size());
    forsn(i,1,s.size()){
        int g=p[i-1];
        while(g && s[i]!=s[g]) g=p[g-1];
        p[i]=g+(s[i]==s[g]);
    }
    return p;
}
// v32 of all ind. of occ. of pat in s
v32 match(const string& s,const string&
pat){
    v32 p=pi(pat+'\0'+s),res;
    forsn(i,(int)p.size()-
(int)s.size(),p.size())
        if(p[i]==pat.size()) res.pb(i-
2*pat.size());
    return res;
}
```

LCA

```
side(p)*side(p) / (double)sq(v);}
    line perpThrough(pt p) {return {p, p +
perp(v)};}
    bool cmpProj(pt p, pt q) {return
dot(v,p) < dot(v,q);}
    line translate(pt t) {return {v, c +
cross(v,t)};}
    line shiftLeft(double dist) {return {v,
c + dist*abs(v)};}
    pt proj(pt p) {return p -
perp(v)*side(p)/sq(v);}
    pt refl(pt p) {return p -
perp(v)*2*side(p)/sq(v);}
};
line bisector(line l1, line l2, bool
interior) {
    assert(cross(l1.v, l2.v) != 0); // l1
and l2 cannot be parallel!
    double sign = interior ? 1 : -1;
    return {l2.v/abs(l2.v) + l1.v/abs(l1.v)
* sign,
            l2.c/abs(l2.v) + l1.c/abs(l1.v)
* sign};
}
bool inter(line l1, line l2, pt &out) {
    T d = cross(l1.v, l2.v);
    if (d == 0) return false;
    out = (l2.v*l1.c - l1.v*l2.c) / d;
    return true;
}
bool inDisk(pt a, pt b, pt p) {return
dot(a-p, b-p) <= 0;}
bool onSegment(pt a, pt b, pt p) {return
orient(a,b,p) == 0 && inDisk(a,b,p);}
// Segment ab & cd intersection
bool properInter(pt a, pt b, pt c, pt d, pt
&out) {
    double oa = orient(c,d,a),
    ob = orient(c,d,b),
    oc = orient(a,b,c),
    od = orient(a,b,d);
    // Proper intersection exists iff
opposite signs
    if (oa*ob < 0 && oc*od < 0) {
        out = (a*ob - b*oa) / (ob-oa);
        return true;
    }
    return false;
}
struct cmpX {
    bool operator()(pt a, pt b) {
        return make_pair(a.xc, a.yc) <
make_pair(b.xc, b.yc);
    }
};
set<pt,cmpX> inters(pt a, pt b, pt c, pt d)
{
    pt out;
    if (properInter(a,b,c,d,out)) return
{out};
    set<pt,cmpX> 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 s;
}
double segPoint(pt a, pt b, pt p) {
    if (a != b) {
```

```
vv32 v;
v32 tin,tout,dist;
vv32 up;
void dfs(int i,int par,int lvl){
    tin[i]= ++t;
    dist[i]= lvl;
    up[i][0] = par;
    forsn(j,1,l+1) up[i][j]= up[up[i][j-1]]
[j-1];
    forstl(it,v[i]) if(it!=par)
dfs(it,i,lvl+1);
    tout[i] = ++t;
}
bool is_ancetor(int u, int v){
    return tin[u]<=tin[v] &&
tout[u]>=tout[v];
}
int lca(int u, int v){
    if (is_ancetor(u, v)) return u;
    if (is_ancetor(v, u)) return v;
    rforn(i,l) if(!is_ancetor(up[u][i], v))
u=up[u][i];
    return up[u][0];
}
int get_dis(int u,int v){
    int lcauv=lca(u,v);
    return dist[u]+dist[v]-2*dist[lcauv];
}
void preprocess(int root){
    tin.resize(n);
    tout.resize(n);
    dist.resize(n);
    t=0;
    l=ceil(log2((double)n));
    up.assign(n,v32(l+1));
    dfs(root,root,0);
}
```

LineContainer

```
/*Description: Container where you can add
lines of the form kx+m, and query maximum
values at points x.
 *Useful for dynamic programming.
 * Time: O(log N) */
const static int LX = -(1e9), RX = 1e9;
struct Dynamic_Hull { /* Max hull */
  struct Line{
    ll m, c; // slope, intercept
    Line(ll mm=0, ll cc=-1e18) { m = mm; c
= cc; }
    ll operator[](const int&x){ return
m*x+c; }  };
  struct node {
    node *lt,*rt; Line Ln;
    node(const Line &l){lt=rt=nullptr;
Ln=l;}};
  node *root=nullptr;
  void add(Line l,node*&it,int lx=LX,int
rx=RX){
    if(it==nullptr)it=new node(l);
    if(it->Ln[lx]>=l[lx] and it-
>Ln[rx]>=l[rx]) return;
    if(it->Ln[lx]<=l[lx] and it->Ln[rx]
<=l[rx]) {it->Ln=l; return;}
    int mid = (lx+rx)>>1;
    if(it->Ln[lx] < l[lx]) swap(it->Ln,l);
    if(it->Ln[mid] >= l[mid]) add(l,it-
>rt,mid+1,rx);
```

```
        line l(a,b);
        if (l.cmpProj(a,p) &&
l.cmpProj(p,b)) // if closest to projection
        return l.dist(p); // output
distance to line
    }
    return min(abs(p-a), abs(p-b)); //
otherwise distance to A or B
}
double segSeg(pt a, pt b, pt c, pt d) {
    pt dummy;
    if (properInter(a,b,c,d,dummy)) return
0;
    return min({segPoint(a,b,c),
segPoint(a,b,d),
    segPoint(c,d,a), segPoint(c,d,b)});
}
double areaTriangle(pt a, pt b, pt c)
{return abs(cross(b-a, c-a)) / 2.0;}
double areaPolygon(vector<pt> &p){
    double area = 0.0;
    for (int i = 0, n = p.size(); i < n;
i++){
        area += cross(p[i], p[(i+1)%n]);
    }
    return abs(area) / 2.0;
}
inline bool above(pt a, pt p) {return p.yc
>= a.yc;}
// check if [PQ] crosses ray from A
bool crossesRay(pt a, pt p, pt q) {
    return (above(a,q) - above(a,p)) *
orient(a,p,q) > 0;
}
// if strict, returns false when A is on
the boundary
bool inPolygon(vector<pt> p, pt a, bool
strict = true) {
    int numCrossings = 0;
    for (int i = 0, n = p.size(); i < n;
i++) {
        if (onSegment(p[i], p[(i+1)%n], a))
return !strict;
        numCrossings += crossesRay(a, p[i],
p[(i+1)%n]);
    }
    return numCrossings & 1; // inside if
odd number of crossings
}
bool inConvexPolygon(pt a,vector<pt> P){
    int n = (int) P.size();
    int P0 = 0;
    for(int i = 1; i < n; i++)
        if (P[i].yc < P[P0].yc || (P[i].yc
== P[P0].yc && P[i].xc > P[P0].xc))
            P0 = i;
    pt temp = P[0];
    P[0] = P[P0]; P[P0] = temp;
    polarSortAround(P[0],P);
    if((orient(P[0],P[1],a) < 0) ||
(orient(P[0],P[n-1],a) > 0)){
        return false;
    }
    if(orient(P[0],P[1],a) == 0){
        if(abs(P[0]-P[1]) >= abs(P[0]-a))
return true;
        else return false;
    }
    if(orient(P[0],P[n-1],a) == 0){
```

```
    else { swap(it->Ln,l); add(l,it-
>lt,lx,mid); }}
  void add(const ll &m,const ll &c) {
add(Line(m,c),root); }
  ll get(int &x,node*&it,int lx=LX,int
rx=RX){
    if(it==NULL) return -1e18; // Max hull
    ll ret = it->Ln[x];
    int mid = (lx+rx)>>1;
    if(x<=mid) ret = max(ret , get(x,it-
>lt,lx,mid));
    else ret = max(ret , get(x,it-
>rt,mid+1,rx));
    return ret;}
  ll get(int x){ return get(x,root); }};
// if In order
struct  Hull{
  struct line {
    ll m,c;
    ll eval(ll x){return m*x+c;}
    ld intersectX(line l){return (ld)(c-
l.c)/(l.m-m);}
    line(ll m,ll c): m(m),c(c){}
  };
  deque<line> dq;
  v32 ints;
  Hull(int n){ints.clear(); forn(i,n)
ints.pb(i); dq.clear();}
  // Dec order of slopes
  void add(line cur){
    while(dq.size()>=2 &&
cur.intersectX(dq[0])>=dq[0].intersectX(dq[1]))

      dq.pop_front();
    dq.push_front(cur);
  }
  void add(const ll &m,const ll &c)
{add(line(m,c));}
  // if query sorted dec.
  ll getval(ll x){
    while(dq.size()>=2 && dq.back().eval(x)
<=dq[dq.size()-2].eval(x))
      dq.pop_back();
    return dq.back().eval(x);}
  // if arbitary query
  ll getval(ll x,deque<line> &dq){
    auto cmp = [&dq](int idx,ll x){return
dq[idx].intersectX(dq[idx+1])<x;};
    int idx =
*lower_bound(ints.begin(),ints.begin()+dq.size()-1,x,cmp);

    return dq[idx].eval(x);}
  ll get(const ll &x){return getval(x,dq);
};
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; }};
```

LongestCommonSubsequence

```
//Longest Common Subsequence
//Time complexity O(mn)
//Space complexity O(mn)
void printLCS(char s1[], char s2[]) {
    int i,j;
    int m = strlen(s1);
```

```
        if(abs(P[0]-P[n-1]) >= abs(P[0]-a))
return true;
        else return false;
    }
    int l = 1,mid , r = n-1;
    while(r-l>1){
        mid = (l+r)/2;
        if(orient(P[0],P[mid],a) >= 0) l =
mid;
        else r = mid;
    }
    if(orient(P[l],P[l+1],a) >= 0) return
true;
    return false;
}
pt circumCenter(pt a, pt b, pt c) {
    b = b-a, c = c-a; // consider
coordinates relative to A
    assert(cross(b,c) != 0); // no
circumcircle if A,B,C aligned
    return a + perp(b*sq(c) -
c*sq(b))/cross(b,c)/2;
}
inline int sgn(int x){return (x==0)? 0:
(x>0)? 1:-1;}
int circleLine(pt o, double r, line l,
pair<pt,pt> &out) {
    double h2 = r*r - l.sqDist(o);
    if (h2 >= 0) { // the line touches the
circle
        pt p = l.proj(o); // point P
        pt h = l.v*sqrt(h2)/abs(l.v); //
vector parallel to l, of length h
        out = {p-h, p+h};
    }
    return 1 + sgn(h2);
}
int circleCircle(pt o1, double r1, pt o2,
double r2, pair<pt,pt> &out) {
    pt d=o2-o1; double d2=sq(d);
    if (d2 == 0) {assert(r1 != r2); return
0;} // concentric circles
    double pd = (d2 + r1*r1 - r2*r2)/2; //
= |O_1P| * d
    double h2 = r1*r1 - pd*pd/d2; // = hË†2
    if (h2 >= 0) {
        pt p = o1 + d*pd/d2, h =
perp(d)*sqrt(h2/d2);
        out = {p-h, p+h};
    }
    return 1 + sgn(h2);
}
int tangents(pt o1, double r1, pt o2,
double r2, bool inner, vector<pair<pt,pt>>
&out) {
    if (inner) r2 = -r2;
    pt d = o2-o1;
    double dr = r1-r2, d2 = sq(d), h2 = d2-
dr*dr;
    if (d2 == 0 || h2 < 0) {assert(h2 !=
0); return 0;}
    for (double sign : {-1,1}) {
        pt v = (d*dr +
perp(d)*sqrt(h2)*sign)/d2;
        out.push_back({o1 + v*r1, o2 +
v*r2});
    }
    return 1 + (h2 > 0);
}
```

```
    int n = strlen(s2);
    int LCS[m+1][n+1];
    for(i=0; i<=m; i++) {
        for(j=0; j<=n; j++) {
            if(i==0 || j==0)
                LCS[i][j] = 0;
            else if(s1[i-1] == s2[j-1])
                LCS[i][j] = 1 + LCS[i-1][j-
1];
            else
                LCS[i][j] = max(LCS[i-1]
[j], LCS[i][j-1]);
        }
    }
    int index = LCS[m][n];
    char lcs[index+1];
    lcs[index] = '\0';
    i = m; j = n;
    while(i > 0 && j > 0)   {
        if(s1[i-1] == s2[j-1])  {
            lcs[index-1] = s1[i-1];
            i--; j--; index--;
        }
        else if(LCS[i-1][j] > LCS[i][j-1])
            i--;
        else
            j--;
    }
    cout<<lcs<<endl;
}
```

LongestCommonSubstring

```
// Check for len=k common btw P & Q in
O(nlogn)
bool check(int k,Hashtable& p,Hashtable& q)
{
    if(k==0) return 1;
    int len1=p.sz,len2=q.sz,asz=len1+1-k;
    p64 a[asz];
    forn(i,asz) a[i]=p.gethash(i,i+k);
    sort(a,a+asz);
    forn(i,len2+1-k){
        auto f=q.gethash(i,i+k);
        auto in=lower_bound(a,a+asz,f)-a;
        if (in<asz && a[in]==f) return 1;
    }
    return 0;
}
// Longest common substring in O(n(logn)^2)
int LCSubstring(string &p,string &q){
    Hashtable P,Q;
    P.create(p),Q.create(q);
    int l=0,r=min(p.size(),q.size());
    while(l<r){
      int mid=(l+r+1)/2;
      if(check(mid,P,Q)) l=mid;
      else r=mid-1;
    }
    return l;
}
```

LongestIncreasingSeq

```
// O(n^2)
void printLIS(vector<ll> &a, vector<ll>
&prev, int pos) {
    if(pos<0)
        return;
```

```
// 3D
const double PI=acos(-1);
typedef double T;
struct p3 {
  T x,y,z;
  p3 operator+(p3 p) {return {x+p.x, y+p.y,
z+p.z};}
  p3 operator-(p3 p) {return {x-p.x, y-p.y,
z-p.z};}
  p3 operator*(T d) {return {x*d, y*d,
z*d};}
  p3 operator/(T d) {return {x/d, y/d,
z/d};}
  bool operator==(p3 p) {return tie(x,y,z)
== tie(p.x,p.y,p.z);}
  bool operator!=(p3 p) {return !operator==
(p);}
};
p3 zero{0,0,0};
// | is dot and * is cross  (priority * >
|)
T operator|(p3 v, p3 w) {return v.x*w.x +
v.y*w.y + v.z*w.z;}
T sq(p3 v) {return v|v;}
double abs(p3 v) {return sqrt(sq(v));}
p3 unit(p3 v) {return v/abs(v);}
double angle(p3 v, p3 w) {
  double cosTheta = (v|w) / abs(v) /
abs(w);
  return acos(max(-1.0, min(1.0,
cosTheta)));
}
p3 operator*(p3 v, p3 w) {
  return {v.y*w.z - v.z*w.y,v.z*w.x -
v.x*w.z,v.x*w.y - v.y*w.x};
}
//orient(P, Q, R, S)=(PQ*PR)|PS use to
determie side of s wrt pqr
T orient(p3 p, p3 q, p3 r, p3 s) {return
(q-p)*(r-p)|(s-p);}
// projection along normal or opposite
T orientByNormal(p3 p, p3 q, p3 r, p3 n)
{return (q-p)*(r-p)|n;}
struct plane {
  p3 n; T d;
  // From normal n and offset d (n|x=d)
  plane(p3 n, T d) : n(n), d(d) {}
  // From normal n and point P
  plane(p3 n, p3 p) : n(n), d(n|p) {}
  // From three non-collinear points P,Q,R
  plane(p3 p, p3 q, p3 r) : plane((q-p)*(r-
p), p) {}
  T side(p3 p) {return (n|p)-d;}
  double dist(p3 p) {return
abs(side(p))/abs(n);}
  plane translate(p3 t) {return {n, d+
(n|t)};}
  plane shiftUp(double dist) {return {n, d
+ dist*abs(n)};}
  p3 proj(p3 p) {return p -
n*side(p)/sq(n);}
  p3 refl(p3 p) {return p -
n*2*side(p)/sq(n);}
};
struct coords {
  p3 o, dx, dy, dz;
  // From three points P,Q,R on the plane:
  // build an orthonormal 3D basis
  coords(p3 p, p3 q, p3 r) : o(p) {
    printLIS(a, prev, prev[pos]);
    cout<<a[pos]<<" ";}
void LIS(vector<ll> &a) {
    int i,j,maxim = 0,n = a.size();
    vector<ll> LIS(n,1);
    vector<ll> prev(n,-1);
    for(i=0; i<n; i++) {
        for(j=0; j<i; j++) {
            if(a[i] > a[j] && LIS[i] <
LIS[j]+1) {
                LIS[i] = LIS[j] + 1;
                prev[i] = j;}}}
    int pos = 0;
    for(i=0; i<n; i++) {
        if(LIS[i] > maxim) {
            maxim = LIS[i];
            pos = i;}}
    cout<<"Length of LIS = "<<maxim<<endl;
    printLIS(a,prev,pos);
    cout<<endl;}
// O(nlogn)
v32 d;
forn(i,n){
    int x=a[i];
    vector<int>::iterator it =
lower_bound(d.begin(), d.end(), x);
    if (it==d.end()) d.push_back(x);
    else *it = x;}
ans=d.size();
```

LongestMaxcontsum

```
// Kadane Algo for maximum contiguous sum
int min_sum = 0, max_sum = 0; // max_sum =
Number[0] to not allow empty sub array
int sum = 0;
forn(i,N){
    sum += Number[i];
    if (sum - min_sum > max_sum) max_sum =
sum - min_sum;
    if (sum < min_sum) min_sum = sum;}
```

Manacher

```
// Given a string s of length N, finds all
palindromes as its substrings.
// p[0][i] = half length of longest even
palindrome around pos i
// p[1][i] = longest odd at i (half rounded
down i.e len 2*x+1).
//Time: O(N)
void manacher(const string& s){
    int n=s.size();
    v32 p[2]={v32(n+1),v32(n)};
    forn(z,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;}}
```

Matrix

```
int add(int a, int b){
    long long res = a + b;
    return res;}
int mult(int a, int b){
    long long res = a;
```

```
  dx = unit(q-p);
  dz = unit(dx*(r-p));
  dy = dz*dx;
  }
  // From four points P,Q,R,S:
  // take directions PQ, PR, PS as is
  coords(p3 p, p3 q, p3 r, p3 s) :
  o(p), dx(q-p), dy(r-p), dz(s-p) {}
  //pt pos2d(p3 p) {return {(p-o)|dx, (p-
o)|dy};}
  p3 pos3d(p3 p) {return {(p-o)|dx, (p-
o)|dy, (p-o)|dz};}
};
struct line3d {
  p3 d, o;
  // From two points P, Q
  line3d(p3 p, p3 q) : d(q-p), o(p) {}
  // From two planes p1, p2 (requires T =
double)
  line3d(plane p1, plane p2) {
    d = p1.n*p2.n;
    o = (p2.n*p1.d - p1.n*p2.d)*d/sq(d);
  }
  double sqDist(p3 p) {return sq(d*(p-
o))/sq(d);}
  double dist(p3 p) {return
sqrt(sqDist(p));}
  bool cmpProj(p3 p, p3 q) {return (d|p) <
(d|q);}
  p3 proj(p3 p) {return o + d*(d|(p-
o))/sq(d);}
  p3 refl(p3 p) {return proj(p)*2 - p;}
  p3 inter(plane p) {return o -
d*p.side(o)/(d|p.n);}
};
double dist(line3d l1, line3d l2) {
  p3 n = l1.d*l2.d;
  if (n == zero) // parallel
  return l1.dist(l2.o);
  return abs((l2.o-l1.o)|n)/abs(n);
}
p3 closestOnL1(line3d l1, line3d l2) {
  p3 n2 = l2.d*(l1.d*l2.d);
  return l1.o + l1.d*((l2.o-
l1.o)|n2)/(l1.d|n2);
}
inline int sgn(T x){return (x==0)? 0:(x>0)?
1:-1;}
bool isParallel(plane p1, plane p2) {return
p1.n*p2.n == zero;}
bool isPerpendicular(plane p1, plane p2)
{return (p1.n|p2.n) == 0;}
double smallAngle(p3 v, p3 w) {return
acos(min(abs(v|w)/abs(v)/abs(w), 1.0));}
double angle(plane p1, plane p2) {return
smallAngle(p1.n, p2.n);}
double angle(line3d l1, line3d l2) {return
smallAngle(l1.d, l2.d);}
bool isParallel(line3d l1, line3d l2)
{return l1.d*l2.d == zero;}
bool isPerpendicular(line3d l1, line3d l2)
{return (l1.d|l2.d) == 0;}
double angle(plane p, line3d l) {return
PI/2 - smallAngle(p.n, l.d);}
bool isParallel(plane p, line3d l) {return
(p.n|l.d) == 0;}
bool isPerpendicular(plane p, line3d l)
{return p.n*l.d == zero;}
line3d perpThrough(plane p, p3 o) {return
```

```
    res *= b;
    return res;}
struct matrix{
    int arr[SZ][SZ];
    void reset(){
        memset(arr, 0, sizeof(arr));}
    void makeiden(){
        reset();
        for(int i=0;i<SZ;i++){
            arr[i][i] = 1;}}
    matrix operator + (const matrix &o)
const {
        matrix res;
        for(int i=0;i<SZ;i++){
            for(int j=0;j<SZ;j++){
                res.arr[i][j] = add(arr[i]
[j], o.arr[i][j]);}}
        return res;}
    matrix operator * (const matrix &o)
const {
        matrix res;
        for(int i=0;i<SZ;i++){
            for(int j=0;j<SZ;j++){
                res.arr[i][j] = 0;
                for(int k=0;k<SZ;k++){
                    res.arr[i][j] =
add(res.arr[i][j] , mult(arr[i][k] ,
o.arr[k][j]));}}}
        return res;}};
matrix mpower(matrix a, int b){
    matrix res;
    res.makeiden();
    while(b){
        if(b & 1){
            res = res * a;}
        a = a * a;
        b >>= 1;}
    return res;}
matrix transpose(matrix a){
    matrix res;
    for(int i=0;i<SZ;i++){
        for(int j=0;j<SZ;j++){
            res.arr[i][j] = a.arr[j][i];}}
    return res;}
void getCofactor(matrix a, matrix b, int p,
int q, int n) //Cofactor of matrix a in b;
n is current dimension{
    int i = 0, j = 0;
    // Looping for each element of the
matrix
    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            //  Copying into temporary
matrix only those element
            //  which are not in given row
and column
            if (row != p && col != q) {
                b.arr[i][j++] = a.arr[row]
[col];
                // Row is filled, so
increase row index and
                // reset col index
                if (j == n - 1) {
                    j = 0; i++; } } } } }
long long determinantOfMatrix(matrix a, int
n) // n is current dimension{
    long long D = 0; // Initialize result
    //  Base case : if matrix contains
single element
```

```
line3d(o, o+p.n);}
plane perpThrough(line3d l, p3 o) {return
plane(l.d, o);}
p3 vectorArea2(vector<p3> p) { // vector
area * 2
  p3 S = zero;
  for (int i=0,n=p.size();i<n;i++)
S=S+p[i]*p[(i+1)%n];
  return S;
}
double area(vector<p3> p) {return
abs(vectorArea2(p)) / 2.0;}
double volume(vector<vector<p3>> fs) {
  double vol6 = 0.0;
  for (vector<p3> f : fs) vol6+=
(vectorArea2(f)|f[0]);
  return abs(vol6)/6.0;
}
// Create arbitrary comparator for map<>
bool operator<(p3 p, p3 q) {
  return tie(p.x, p.y, p.z) < tie(q.x, q.y,
q.z);
}
// On spheres
p3 sph(double r, double lat, double lon) {
  lat *= PI/180, lon *= PI/180;
  return {r*cos(lat)*cos(lon),
r*cos(lat)*sin(lon), r*sin(lat)};
}
int sphereLine(p3 o, double r, line3d l,
pair<p3,p3> &out) {
  double h2 = r*r - l.sqDist(o);
  if (h2 < 0) return 0; // the line
doesnâ€™t touch the sphere
  p3 p = l.proj(o); // point P
  p3 h = l.d*sqrt(h2)/abs(l.d); // vector
parallel to l, of length
  out = {p-h, p+h};
  return 1 + (h2 > 0);
}
double greatCircleDist(p3 o, double r, p3
a, p3 b) {
  return r * angle(a-o, b-o);
}
bool validSegment(p3 a, p3 b) {return a*b
!= zero || (a|b) > 0;}
bool properInter(p3 a, p3 b, p3 c, p3 d, p3
&out) {
  p3 ab = a*b, cd = c*d; // normals of
planes OAB and OCD
  int oa = sgn(cd|a),ob = sgn(cd|b),oc =
sgn(ab|c),od = sgn(ab|d);
  out = ab*cd*od; // overflow check!!!!!
  return (oa != ob && oc != od && oa !=
oc);
}
bool onSphSegment(p3 a, p3 b, p3 p) {
  p3 n = a*b;
  if (n == zero) return a*p == zero &&
(a|p) > 0;
  return (n|p) == 0 && (n|a*p) >= 0 &&
(n|b*p) <= 0;
}
struct directionSet : vector<p3> {
  using vector::vector; // import
constructors
  void insert(p3 p) {
    for (p3 q : *this) if (p*q == zero)
return;
```

```
    if (n == 1) return a.arr[0][0];
    matrix res; // To store cofactors
    int sign = 1;  // To store sign
multiplier
     // Iterate for each element of first
row
    for (int f = 0; f < n; f++) {
        // Getting Cofactor of mat[0][f]
        getCofactor(a, res, 0, f, n);
        D += sign * a.arr[0][f] *
determinantOfMatrix(res, n - 1);
        // terms are to be added with
alternate sign
        sign = -sign; }
    return D; }
void display(matrix a){
    forn(i,SZ){
        forn(j,SZ){
            cout<<a.arr[i][j]<<" ";}
        cout<<ln;}}
```

MillerRabin

```
/* Description: Miller-Rabin primality
probabilistic test.
 * Probability of failing one iteration is
at most 1/4. 15 iterations should be
 * enough for 50-bit numbers.
 * Time: 15 times the complexity of $a^b
\mod c$.
 */
bool prime(ull p) {
    if (p == 2) return true;
    if (p == 1 || p % 2 == 0) return false;
    ull s = p - 1;
    while (s % 2 == 0) s /= 2;
    rep(i,0,15) {
        ull a = rand() % (p - 1) + 1, tmp =
s;

        ull mod = mod_pow(a, tmp, p);
        while (tmp != p - 1 && mod != 1 &&
mod != p - 1) {
            mod = mod_mul(mod, mod, p);
            tmp *= 2;
        }
        if (mod != p - 1 && tmp % 2 == 0)
return false;
    }
    return true;
}
```

MinRotation

```
/*Finds the lexicographically smallest
rotation of a string.
 * Time: O(N) use:rotate(v.begin(),
v.begin()+min_rotation(v), v.end());*/
int min_rotation(string s) {
    int a=0, N=sz(s); s += s;
    rep(b,0,N) rep(i,0,N) {
        if (a+i == b || s[a+i] < s[b+i]) {b
+= max(0, i-1); break;}
        if (s[a+i] > s[b+i]) { a = b;
break; }
    }
    return a;
}
```

```
    push_back(p);
  }
};
directionSet intersSph(p3 a, p3 b, p3 c, p3
d) {
  assert(validSegment(a, b) &&
validSegment(c, d));
  p3 out;
  if (properInter(a, b, c, d, out)) return
{out};
  directionSet s;
  if (onSphSegment(c, d, a)) s.insert(a);
  if (onSphSegment(c, d, b)) s.insert(b);
  if (onSphSegment(a, b, c)) s.insert(c);
  if (onSphSegment(a, b, d)) s.insert(d);
  return s;
}
double angleSph(p3 a, p3 b, p3 c) {return
angle(a*b, a*c);}
double orientedAngleSph(p3 a, p3 b, p3 c) {
  if ((a*b|c) >= 0) return angleSph(a, b,
c);
  else return 2*PI - angleSph(a, b, c);
}
double areaOnSphere(double r, vector<p3> p)
{
  int n = p.size();
  double sum = -(n-2)*PI;
  for (int i = 0; i < n; i++)
    sum += orientedAngleSph(p[(i+1)%n],
p[(i+2)%n], p[i]);
  return r*r*sum;
}
int windingNumber3D(vector<vector<p3>> fs)
{
  double sum = 0;
  for (vector<p3> f : fs)
    sum += remainder(areaOnSphere(1, f),
4*PI);
  return round(sum / (4*PI));
}
```

karatsuba

```
class Karatsuba {
    public:
//m should be a power of 2
//lA = left start point of array A, rA =
ending, etc.
//eg Karatsuba::multiply(A,B,C,0,3,0,3)
stores ans in C
    static void multiply(int *A, int *B,
int *C, int lA, int rA, int lB, int rB){
        int m = rA-lA+1;
        if (m == 1) {
            C[0] = ((long long)A[lA]*B[lB])
% MOD;
            return;
        }

        int z0[m], z1[m], z2[m];

        int midA = (lA + rA) >> 1;
        int midB = (lB + rB) >> 1;

        multiply(A, B, z0, lA, midA, lB,
midB);
        multiply(A, B, z1, midA+1, rA,
midB+1, rB);
```

MinimumEnclosingCircle

```
/*Computes the minimum circle that encloses
a set of points.
 * Time: expected O(n)*/
pair<double, P> mec2(vector<P>& S, P a, P
b, int n) {
    double hi = INFINITY, lo = -hi;
    rep(i,0,n) {
        auto si = (b-a).cross(S[i]-a);
        if (si == 0) continue;
        P m = ccCenter(a, b, S[i]);
        auto cr = (b-a).cross(m-a);
        if (si < 0) hi = min(hi, cr);
        else lo = max(lo, cr);
    }
    double v = (0 < lo ? lo : hi < 0 ? hi :
0);
    P c = (a + b) / 2 + (b - a).perp() * v
/ (b - a).dist2();
    return {(a - c).dist2(), c};
}
pair<double, P> mec(vector<P>& S, P a, int
n) {
    random_shuffle(S.begin(), S.begin() +
n);
    P b = S[0], c = (a + b) / 2;
    double r = (a - c).dist2();
    rep(i,1,n) if ((S[i] - c).dist2() > r *
(1 + 1e-8)) {
        tie(r,c) = (n == sz(S) ?
            mec(S, S[i], i) : mec2(S, a,
S[i], i));
    }
    return {r, c};
}
pair<double, P> enclosingCircle(vector<P>
S) {
    assert(!S.empty()); auto r = mec(S,
S[0], sz(S));
    return {sqrt(r.first), r.second};
}
```

NTT

```
//root is primitive root, root_1 is inverse
of root_pw modulo mod
//root_pw = 1<<20 (or whatever pow of 2),
mod is the prime
//inverse is modular inverse
void ntt(vector<int> & a, bool invert) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if (i < j)
            swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len <<= 1)
{
        int wlen = invert ? root_1 : root;
        for (int i = len; i < root_pw; i
<<= 1)
            wlen = (int)(1LL * wlen * wlen
% mod);
        for (int i = 0; i < n; i += len) {
            int w = 1;
```

```
        int a[m], b[m];
        int shift = m>>1;
        int mid = m>>1;
        for (int i = lA, j = 0; i <= midA;
i++, j++) {
            a[j] = A[i] + A[i+shift];
            if (a[j] >= MOD) a[j] -= MOD;
        }
        for (int i = lB, j = 0; i <= midB;
i++, j++) {
            b[j] = B[i] + B[i+shift];
            if (b[j] >= MOD) b[j] -= MOD;
        }
        multiply(a, b, z2, 0, mid-1, 0,
mid-1);

        for (int i = 0; i <= m-2; i++) {
            C[i] = z0[i];
            if (C[i] >= MOD) C[i] -= MOD;
        }
        C[m-1] = 0;

        shift = m;
        for (int i = 0; i <= m-2; i++) {
            C[i+shift] = z1[i];
            if (C[i+shift] >= MOD)
C[i+shift] -= MOD;
        }

        shift = m>>1;
        for (int i = 0; i <= m-2; i++) {
            C[i+shift] += (z2[i] + (MOD-
z1[i]) + (MOD-z0[i])) % MOD;
            if (C[i+shift] >= MOD) {
                C[i+shift] -= MOD;
            }
        }
    }
};
```

kthPerm

```
// kth permutation of 1..n (k>=0)
v32 inttoPerm(int n,int k){
    ++k;
    v32 m(n,1);
    int tmp=1;
    forn(i,n-1){
        m[n-i-2]=tmp; tmp*=(i+2);}
    k=(k-1)%(m[0]*n);
    vector<bool> used(n,0);
    v32 ans;
    forn(i,n){
        int idx=k/m[i];
        k=k%m[i];
        forn(j,n){
            if(!used[j]){
                if(!idx){
                    ans.pb(j+1);
                    used[j]=true;
                    break; }
                idx--;}}}
    return ans;}
```

linsieve

```
// Linear Sieve and multiplication func
example(curr mu)
```

```
            for (int j = 0; j < len / 2; j++) {
                int u = a[i+j], v = (int) (1LL * a[i+j+len/2] * w % mod);
                a[i+j] = u + v < mod ? u + v : u + v - mod;
                a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
                w = (int)(1LL * w * wlen % mod);
            }
        }
    }
    if (invert) {
        int n_1 = inverse(n, mod);
        for (int & x : a)
            x = (int)(1LL * x * n_1 % mod);
    }
}
```

OrderStatisticTree

```
// * To get a map, change {null_type to some value}.
#include <bits/extc++.h> /** keep-include */
using namespace __gnu_pbds;
template<class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
      tree_order_statistics_node_update>;
void example() {
    Tree<int> t, t2; t.insert(8);
    auto it = t.insert(10).first;
    assert(it == t.lower_bound(9));
    assert(t.order_of_key(10) == 1);
    assert(t.order_of_key(11) == 2);
    assert(*t.find_by_order(0) == 8);
    t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

PolygonCenter

```
/*Returns the center of mass for a polygon.*/
typedef Point<double> P;
Point<double> polygonCenter(vector<P>& v) {
    auto i = v.begin(), end = v.end(), j = end-1;
    Point<double> res{0,0}; double A = 0;
    for (; i != end; j=i++) {
        res = res + (*i + *j) * j->cross(*i);
        A += j->cross(*i);
    }
    return res / A / 3;
}
```

PolygonCut

```
//Returns a vector with the vertices of a polygon with everything
//to the left of the line going from s to e cut away.
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
    vector<P> res;
```

```
int f[LIM],is_com[LIM];
v32 pr;
void sieve(){
    f[1]=1; // Assign f[1]
    forsn(i,2,LIM){
        if(!is_com[i]) pr.pb(i),mu[i]=-1;
        forstl(it,pr){
            if(it*i>=LIM) break;
            is_com[i*it]=1;
            if(i%it==0){
                f[i*it]=0; // f[i*it] based on it power
                break;
            }else{
                f[i*it]=f[i]*f[it]; // multiplicative property
            }}}}
//Erasthosthenes sieve
int lpd[LIM]; //largest prime divsor
v32 pr;
void sieve(){
    lpd[1]=1;
    forsn(i,2,LIM){
        if(!lpd[i]){
            pr.pb(i);
            for(int j=i;j<LIM;j+=i){
                lpd[j]=i; }}}}
```

mcmf

```
const int INT_MAX = MOD;
struct edge{int to, flow, cap, cost, rev;};
struct MinCostMaxFlow{
    int nodes;
    vector<int> prio, curflow, prevedge, prevnode, q, pot;
    vector<bool> inqueue;
    vector<vector<edge> > graph;
    MinCostMaxFlow() {}
    MinCostMaxFlow(int n): nodes(n), prio(n, 0), curflow(n, 0),
    prevedge(n, 0), prevnode(n, 0), q(n, 0), pot(n, 0), inqueue(n, 0), graph(n) {}
    void addEdge(int source, int to, int capacity, int cost){
        edge a = {to, 0, capacity, cost, (int)graph[to].size()};
        edge b = {source, 0, 0, -cost, (int)graph[source].size()};
        graph[source].push_back(a);
        graph[to].push_back(b);
    }
    void bellman_ford(int source, vector<int> &dist){
        fill(dist.begin(), dist.end(), INT_MAX);
        dist[source] = 0;
        int qt=0;
        q[qt++] = source;
        for(int qh=0;(qh-qt)%nodes!=0;qh++){
            int u = q[qh%nodes];
            inqueue[u] = false;
            for(auto &e : graph[u]){
                if(e.flow >= e.cap) continue;
                int v = e.to;
                int newDist = dist[u] + e.cost;
    rep(i,0,sz(poly)) {
        P cur = poly[i], prev = i ? poly[i-
1] : poly.back();
        bool side = s.cross(e, cur) < 0;
        if (side != (s.cross(e, prev) < 0))
{
            res.emplace_back();
            lineIntersection(s, e, cur,
prev, res.back());
        }
        if (side)
            res.push_back(cur);
    }
    return res;
}
```

PolygonDiameter

```
//Description: Calculates the max squared
distance of a set of points.
vector<pii> antipodal(const vector<P>& S,
vi& U, vi& L) {
    vector<pii> ret;
    int i = 0, j = sz(L) - 1;
    while (i < sz(U) - 1 || j > 0) {
        ret.emplace_back(U[i], L[j]);
        if (j == 0 || (i != sz(U)-1 &&
(S[L[j]] - S[L[j-1]])
                    .cross(S[U[i+1]] -
S[U[i]]) > 0)) ++i;
        else --j;
    }
    return ret;
}
pii polygonDiameter(const vector<P>& S) {
    vi U, L; tie(U, L) = ulHull(S);
    pair<ll, pii> ans;
    trav(x, antipodal(S, U, L))
        ans = max(ans, {(S[x.first] -
S[x.second]).dist2(), x});
    return ans.second;
}
```

PowerInverse

```
ll powm(ll x,ll pw,ll MOD){ //return x^pw %
MOD
    ll res=1;
    while(pw){
        if(pw&1LL) res=((res*x))%MOD;
        pw>>=1;
        x=((x*x))%MOD;
    }
    return res;
}
inline ll inv(ll x,ll MOD){
    return powm(x,MOD-2,MOD);
}
// Linear inverse table
// Assumption LIM<=MOD & MOD prime
ll inv[LIM];
inv[1] = 1;
forsn(i,2,LIM) inv[i]=mod-
(mod/i)*inv[mod%i]%mod;
```

SegTree

```
// Example Find maximum and number of times
it appears
```

```
                if(dist[v] > newDist){
                    dist[v] = newDist;
                    if(!inqueue[v]){
                        inqueue[v] = true;
                        q[qt++ % nodes] =
v;
    }}}}}
    p32 minCostFlow(int source, int dest,
int maxflow){
        bellman_ford(source, pot);
        int flow = 0;
        int flow_cost = 0;
        while(flow < maxflow){
            priority_queue<pair<int, int>,
vector<pair<int, int> >, greater<pair<int,
int> > > q;
            q.push({0, source});
            fill(prio.begin(), prio.end(),
INT_MAX);
            prio[source] = 0;
            curflow[source] = INT_MAX;
            while(!q.empty()){
                int d = q.top().first;
                int u = q.top().second;
                q.pop();
                if(d != prio[u]) continue;
                for(int
i=0;i<graph[u].size();i++){
                    edge &e=graph[u][i];
                    int v = e.to;
                    if(e.flow >=
e.cap)continue;
                    int newPrio = prio[u] +
e.cost + pot[u] - pot[v];
                    if(prio[v] > newPrio){
                        prio[v] = newPrio;
                        q.push({newPrio,
v});

                        prevnode[v] = u;
                        prevedge[v] = i;
                        curflow[v] =
min(curflow[u], e.cap - e.flow);
            }    }    }
            if(prio[dest] == INT_MAX)
break;
            for(int i=0;i<nodes;i++)
pot[i]+=prio[i];
            int df = min(curflow[dest],
maxflow - flow);
            flow += df;
            for(int
v=dest;v!=source;v=prevnode[v]){
                edge &e =
graph[prevnode[v]][prevedge[v]];
                e.flow += df;
                graph[v][e.rev].flow -= df;
                flow_cost += df * e.cost;
        }   }
        return {flow, flow_cost};
    }
};
```

multinomial

```
// Assumption No overflow
ll multinomial(v32& v) {
    ll c=1,m=(v.empty() ? 1:v[0]);
    forsn(i,1,v.size()) forn(j,v[i]) c=c*
(++m)/(j+1);
```

```
pair<int, int> t[4*MAXN];
pair<int, int> combine(pair<int, int> a,
pair<int, int> b) {
    if (a.first > b.first)
        return a;
    if (b.first > a.first)
        return b;
    return make_pair(a.first, a.second +
b.second);
}
void build(int a[], int v, int tl, int tr)
{
    if (tl == tr) {
        t[v] = make_pair(a[tl], 1);
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = combine(t[v*2], t[v*2+1]);
    }
}
pair<int, int> get_max(int v, int tl, int
tr, int l, int r) {
    if (l > r)
        return make_pair(-INF, 0);
    if (l == tl && r == tr)
        return t[v];
    int tm = (tl + tr) / 2;
    return combine(get_max(v*2, tl, tm, l,
min(r, tm)),
                   get_max(v*2+1, tm+1, tr,
max(l, tm+1), r));
}
void update(int v, int tl, int tr, int pos,
int new_val) {
    if (tl == tr) {
        t[v] = make_pair(new_val, 1);
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos,
new_val);
        else
            update(v*2+1, tm+1, tr, pos,
new_val);
        t[v] = combine(t[v*2], t[v*2+1]);
    }
}
// Example Finding subsegments with the
maximal sum
struct data{
    int sum, pref, suff, ans;
};
data t[4*MAXN];
data combine(data l, data r) {
    data res;
    res.sum = l.sum + r.sum;
    res.pref = max(l.pref, l.sum + r.pref);
    res.suff = max(r.suff, r.sum + l.suff);
    res.ans = max(max(l.ans, r.ans), l.suff
+ r.pref);
    return res;
}
data make_data(int val) {
    data res;
    res.sum = val;
    res.pref = res.suff = res.ans = max(0,
val);
    return res;
```

```
    return c;
}
```

prim

```
int dist[N], parent[N], MaxDist = MOD;
bool vis[N];
vector<pair<int, int> > g[N], mst[N];
int MST(int source){ //prim
    for(int i=1;i<=n;i++) dist[i]= MaxDist;
    priority_queue<p32, vp32, greater<p32>
> s;
    s.push({0, source});
    int cost=0;
    dist[source]=0;
    while(!s.empty()){
        x = s.top();
        s.pop();
        vis[x.se]=1; cost+=x.fi;
        mst[x.se].push_back({parent[x,se],
x.fi});
        mst[parent[x.se]].push_back({x.se,
x.fi});
        forstl(it,g[x.second])
if(!vis[it.fi]){
            if(dist[it.fi] > it.se){
                s.erase({dist[it.fi],
it.fi});
                dist[it.fi]=it.se;
                s.insert({dist[it.fi],
it.fi});
                parent[it.fi]=x.se;
    }    }    }
    return cost;
}
```

suffixArray

```
struct ranking {
    ll index,first,second;};
// P[i][j] holds the ranking of the j-th
suffix
// after comparing the first 2^i characters
// of that suffix
ll P[20][1000000];
bool comp(ranking a, ranking b) {
    if (a.first == b.first) {
        return a.second < b.second;}
    return a.first < b.first;}
vector<ll> build_suffix_array(string s) {
    const ll n = s.length();
    const ll lgn = ceil(log2(n));
    // vector to hold final suffix array
result
    vector<ll> sa(n);
    // vector to store ranking tuples of
suffixes
    vector<ranking> ranks(n);
    ll i, j, step = 1;
    for(j = 0; j < n; j++) {
        P[0][j] = s[j] - 'a'; }
    for(i = 1; i <= lgn; i++, step++) {
        for(j = 0; j < n; j++) {
            ranks[j].index = j;
            ranks[j].first = P[i-1][j];
            ranks[j].second = (j + pow(2,i-
1) < n) ? P[i-1][j + (ll)(pow(2,i-1))] :
-1;
        }
```

```
}
void build(int a[], int v, int tl, int tr)
{
    if (tl == tr) {
        t[v] = make_data(a[tl]);
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = combine(t[v*2], t[v*2+1]);
    }
}
void update(int v, int tl, int tr, int pos,
int new_val) {
    if (tl == tr) {
        t[v] = make_data(new_val);
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos,
new_val);
        else
            update(v*2+1, tm+1, tr, pos,
new_val);
        t[v] = combine(t[v*2], t[v*2+1]);
    }
}
// find_kth (Use idea for sum also,num of
zero,etc)
int find_kth(int v, int tl, int tr, int k)
{
    if (k > t[v])
        return -1;
    if (tl == tr)
        return tl;
    int tm = (tl + tr) / 2;
    if (t[v*2] >= k)
        return find_kth(v*2, tl, tm, k);
    else
        return find_kth(v*2+1, tm+1, tr, k
- t[v*2]);
}
// Lazy Example for finding maximum
void push(int v) {
    t[v*2] += lazy[v];
    lazy[v*2] += lazy[v];
    t[v*2+1] += lazy[v];
    lazy[v*2+1] += lazy[v];
    lazy[v] = 0;
}
void update(int v, int tl, int tr, int l,
int r, int addend) {
    if (l > r)
        return;
    if (l == tl && tr == r) {
        t[v] += addend;
        lazy[v] += addend;
    } else {
        push(v);
        int tm = (tl + tr) / 2;
        update(v*2, tl, tm, l, min(r, tm),
addend);
        update(v*2+1, tm+1, tr, max(l,
tm+1), r, addend);
        t[v] = max(t[v*2], t[v*2+1]);
    }
}
int query(int v, int tl, int tr, int l, int
r) {
```

```
        sort(ranks.begin(), ranks.end(),
comp);
        for(j = 0; j < n; j++) {
            P[i][ranks[j].index] = (j > 0
&& ranks[j].first == ranks[j-1].first &&
ranks[j].second == ranks[j-1].second) ?
P[i][ranks[j-1].index] : j;}}
    step -= 1;
    for(i = 0; i < n; i++) {
        sa[P[step][i]] = i;}
    return sa;}
vector<ll> build_lcp(vector<ll> &sa, ll n)
{
    vector<ll> lcp(n,0);
    ll k, i, j , x, delta, step =
ceil(log2(n));
    for (k = 1; k < n; k++) {
        i = k;
        j = k - 1;
        for (x = step; x >= 0; x--) {
            if (P[x][sa[i]] == P[x][sa[j]])
{
                delta = pow(2,x);
                lcp[i] += delta;
                i += delta;
                j += delta;}}    }
    return lcp;}
// Can give longest common substring using
max{lcp[i]}
```

tarjan

```
v32 dfs_num, dfs_low, S, visited;
int numSCC = 0,dfsCounter = 0;
void dfsSCC(int u,vv32 & AdjList,vv32 &
SCC) {
    dfs_low[u] = dfs_num[u] = dfsCounter++;
    S.push_back(u);
// stores u, order of visitation
    visited[u] = 1;
    for (int j = 0; j <
(int)AdjList[u].size(); j++) {
        int v = AdjList[u][j];
        if (dfs_num[v] == -1)
            tarjanSCC(v.first);
        if (visited[v])  // condition for
update
            dfs_low[u] = min(dfs_low[u],
dfs_low[v.first]); }

    if (dfs_low[u] == dfs_num[u]) {
        // if this is a root (start) of an
SCC
        while (1) {
            int v = S.back(); S.pop_back();
visited[v] = 0;
            SCC[numSCC].pb(v);
            if (u == v) break;
        }
        ++numSCC;
    }
}
void tarjan(vv32 & AdjList,vv32 & SCC){
    dfs_num.assign(V,-1); dfs_low.assign(V,
0); visited.assign(V, 0);
    dfsCounter = numSCC = 0;
    for (int i = 0; i < V; i++)
        if (dfs_num[i] == UNVISITED)
    if (l > r)
        return -INF;
    if (l <= tl && tr <= r)
        return t[v];
    push(v);
    int tm = (tl + tr) / 2;
    return max(query(v*2, tl, tm, l, min(r,
tm)),
               query(v*2+1, tm+1, tr,
max(l, tm+1), r));
}
// 2D Seg Tree Example sum on 2D
void build_y(int vx, int lx, int rx, int
vy, int ly, int ry) {
    if (ly == ry) {
        if (lx == rx)
            t[vx][vy] = a[lx][ly];
        else
            t[vx][vy] = t[vx*2][vy] +
t[vx*2+1][vy];
    } else {
        int my = (ly + ry) / 2;
        build_y(vx, lx, rx, vy*2, ly, my);
        build_y(vx, lx, rx, vy*2+1, my+1,
ry);
        t[vx][vy] = t[vx][vy*2] + t[vx]
[vy*2+1];
    }
}
void build_x(int vx, int lx, int rx) {
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        build_x(vx*2, lx, mx);
        build_x(vx*2+1, mx+1, rx);
    }
    build_y(vx, lx, rx, 1, 0, m-1);
}
int sum_y(int vx, int vy, int tly, int
try_, int ly, int ry) {
    if (ly > ry)
        return 0;
    if (ly == tly && try_ == ry)
        return t[vx][vy];
    int tmy = (tly + try_) / 2;
    return sum_y(vx, vy*2, tly, tmy, ly,
min(ry, tmy))
         + sum_y(vx, vy*2+1, tmy+1, try_,
max(ly, tmy+1), ry);
}
int sum_x(int vx, int tlx, int trx, int lx,
int rx, int ly, int ry) {
    if (lx > rx)
        return 0;
    if (lx == tlx && trx == rx)
        return sum_y(vx, 1, 0, m-1, ly,
ry);
    int tmx = (tlx + trx) / 2;
    return sum_x(vx*2, tlx, tmx, lx,
min(rx, tmx), ly, ry)
         + sum_x(vx*2+1, tmx+1, trx,
max(lx, tmx+1), rx, ly, ry);
}
void update_y(int vx, int lx, int rx, int
vy, int ly, int ry, int x, int y, int
new_val) {
    if (ly == ry) {
        if (lx == rx)
            t[vx][vy] = new_val;
        else
```

```
            dfsSCC(i,AdjList,SCC);
}
```

techniques

```
Tech
Recursion
Divide and conquer
    Finding interesting points in N log N
Algorithm analysis
    Master theorem
    Amortized time complexity
Greedy algorithm
    Scheduling
    Max contiguous subvector sum
    Invariants
    Huffman encoding
Graph theory
    Dynamic graphs (extra book-keeping)
    Breadth first search
    Depth first search
    * Normal trees / DFS trees
    Dijkstra's algorithm
    MST: Prim's algorithm
    Bellman-Ford
    Konig's theorem and vertex cover
    Min-cost max flow
    Lovasz toggle
    Matrix tree theorem
    Maximal matching, general graphs
    Hopcroft-Karp
    Hall's marriage theorem
    Graphical sequences
    Floyd-Warshall
    Euler cycles
    Flow networks
    * Augmenting paths
    * Edmonds-Karp
    Bipartite matching
    Min. path cover
    Topological sorting
    Strongly connected components
    2-SAT
    Cut vertices, cut-edges and biconnected
components
    Edge coloring
    * Trees
    Vertex coloring
    * Bipartite graphs (=> trees)
    * 3^n (special case of set cover)
    Diameter and centroid
    K'th shortest path
    Shortest cycle
Dynamic programming
    Knapsack
    Coin change
    Longest common subsequence
    Longest increasing subsequence
    Number of paths in a dag
    Shortest path in a dag
    Dynprog over intervals
    Dynprog over subsets
    Dynprog over probabilities
    Dynprog over trees
    3^n set cover
    Divide and conquer
    Knuth optimization
    Convex hull optimizations
    RMQ (sparse table a.k.a 2^k-jumps)
```

```
            t[vx][vy] = t[vx*2][vy] +
t[vx*2+1][vy];
    } else {
        int my = (ly + ry) / 2;
        if (y <= my)
            update_y(vx, lx, rx, vy*2, ly,
my, x, y, new_val);
        else
            update_y(vx, lx, rx, vy*2+1,
my+1, ry, x, y, new_val);
        t[vx][vy] = t[vx][vy*2] + t[vx]
[vy*2+1];
    }
}
void update_x(int vx, int lx, int rx, int
x, int y, int new_val) {
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        if (x <= mx)
            update_x(vx*2, lx, mx, x, y,
new_val);
        else
            update_x(vx*2+1, mx+1, rx, x,
y, new_val);
    }
    update_y(vx, lx, rx, 1, 0, m-1, x, y,
new_val);
}
// Persistent Seg Tree
struct Vertex {
    Vertex *l,*r;
    int sum;
    Vertex(int val) :
l(nullptr),r(nullptr),sum(val) {}
    Vertex(Vertex *l, Vertex *r) : l(l),
r(r), sum(0) {
        if (l) sum += l->sum;
        if (r) sum += r->sum;
    }
};
Vertex* build(int a[], int tl, int tr) {
    if (tl == tr)
        return new Vertex(a[tl]);
    int tm = (tl + tr) / 2;
    return new Vertex(build(a, tl, tm),
build(a, tm+1, tr));
}
int get_sum(Vertex* v, int tl, int tr, int
l, int r) {
    if (l > r)
        return 0;
    if (l == tl && tr == r)
        return v->sum;
    int tm = (tl + tr) / 2;
    return get_sum(v->l, tl, tm, l, min(r,
tm))
         + get_sum(v->r, tm+1, tr, max(l,
tm+1), r);
}
Vertex* update(Vertex* v, int tl, int tr,
int pos, int new_val) {
    if (tl == tr)
        return new Vertex(new_val);
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return new Vertex(update(v->l, tl,
tm, pos, new_val), v->r);
    else
        return new Vertex(v->l, update(v-
```

```
    Bitonic cycle
    Log partitioning (loop over most
restricted)
Combinatorics
    Computation of binomial coefficients
    Pigeon-hole principle
    Inclusion/exclusion
    Catalan number
    Pick's theorem
Number theory
    Integer parts
    Divisibility
    Euclidean algorithm
    Modular arithmetic
    * Modular multiplication
    * Modular inverses
    * Modular exponentiation by squaring
    Chinese remainder theorem
    Fermat's little theorem
    Euler's theorem
    Phi function
    Frobenius number
    Quadratic reciprocity
    Pollard-Rho
    Miller-Rabin
    Hensel lifting
    Vieta root jumping
Game theory
    Combinatorial games
    Game trees
    Mini-max
    Nim
    Games on graphs
    Games on graphs with loops
    Grundy numbers
    Bipartite games without repetition
    General games without repetition
    Alpha-beta pruning
Probability theory
Optimization
    Binary search
    Ternary search
    Unimodality and convex functions
    Binary search on derivative
Numerical methods
    Numeric integration
    Newton's method
    Root-finding with binary/ternary search
    Golden section search
Matrices
    Gaussian elimination
    Exponentiation by squaring
Sorting
    Radix sort
Geometry
    Coordinates and vectors
    * Cross product
    * Scalar product
    Convex hull
    Polygon cut
    Closest pair
    Coordinate-compression
    Quadtrees
    KD-trees
    All segment-segment intersection
Sweeping
    Discretization (convert to events and
sweep)
    Angle sweeping
```

```
>r, tm+1, tr, pos, new_val));
}
```

SuffixTree

```
/* Description: Ukkonen's algorithm for
online suffix tree construction.
 *  Each node contains indices [l, r) into
the string, and a list of child nodes.
 *  Suffixes are given by traversals of
this tree, joining [l, r) substrings.
 *  The root is 0 (has l = -1, r = 0), non-
existent children are -1.
 *  To get a complete tree, append a dummy
symbol -- otherwise it may contain
 *  an incomplete path (still useful for
substring matching, though).
 * Time: $O(26N)$
 */
struct SuffixTree {
    enum { N = 200010, ALPHA = 26 }; // N ~
2*maxlen+10
    int toi(char c) { return c - 'a'; }
    string a; // v = cur node, q = cur
position
    int t[N]
[ALPHA],l[N],r[N],p[N],s[N],v=0,q=0,m=2;
    void ukkadd(int i, int c) { suff:
        if (r[v]<=q) {
            if (t[v][c]==-1) { t[v][c]=m;
l[m]=i;
                p[m++]=v; v=s[v]; q=r[v];
goto suff; }
            v=t[v][c]; q=l[v];
        }
        if (q==-1 || c==toi(a[q])) q++;
else {
            l[m+1]=i;  p[m+1]=m;
l[m]=l[v];  r[m]=q;
            p[m]=p[v];  t[m][c]=m+1;  t[m]
[toi(a[q])]=v;
            l[v]=q;  p[v]=m;  t[p[m]]
[toi(a[l[m]])]=m;
            v=s[p[m]];  q=l[m];
            while (q<r[m]) { v=t[v]
[toi(a[q])];  q+=r[v]-l[v]; }
            if (q==r[m])  s[m]=v;  else
s[m]=m+2;
            q=r[v]-(q-r[m]);  m+=2;  goto
suff;
        }
    }
    SuffixTree(string a) : a(a) {
        fill(r,r+N,sz(a));
        memset(s, 0, sizeof s);
        memset(t, -1, sizeof t);
        fill(t[1],t[1]+ALPHA,0);
        s[0] = 1; l[0] = l[1] = -1; r[0] =
r[1] = p[0] = p[1] = 0;
        forn(i,a.size()) ukkadd(i,
toi(a[i]));
    }
    // example: find longest common
substring (uses ALPHA = 28)
    p32 best;
    int lcs(int node, int i1, int i2, int
olen) {
        if (l[node] <= i1 && i1 < r[node])
return 1;
```

```
    Line sweeping
    Discrete second derivatives
Strings
    Longest common substring
    Palindrome subsequences
    Knuth-Morris-Pratt
    Tries
    Rolling polynomial hashes
    Suffix array
    Suffix tree
    Aho-Corasick
    Manacher's algorithm
    Letter position lists
Combinatorial search
    Meet in the middle
    Brute-force with pruning
    Best-first (A*)
    Bidirectional search
    Iterative deepening DFS / A*
Data structures
    LCA (2^k-jumps in trees in general)
    Pull/push-technique on trees
    Heavy-light decomposition
    Centroid decomposition
    Lazy propagation
    Self-balancing trees
    Convex hull trick
(wcipeg.com/wiki/Convex_hull_trick)
    Monotone queues / monotone stacks /
sliding queues
    Sliding queue using 2 stacks
    Persistent segment tree
```

topsort

```
v32 ts;//in reverse order
int vists[LIM] = {0};
void tsdfs(int u){
    vists[u] = 1;
    forstl(j,adj[u]){
        if(vists[j] == 0) tsdfs(j);
    }
    ts.pb(u);
}
void topsort(){
    ts.clear();
    forn(i,(int) adj.size()){
        if(vists[i] == 0) tsdfs(i);
    }
}
```

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 datastructures 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.

```
        if (l[node] <= i2 && i2 < r[node])
return 2;
        int mask = 0, len = node ? olen +
(r[node] - l[node]) : 0;
        forn(c,ALPHA) if (t[node][c] != -1)
            mask |= lcs(t[node][c], i1, i2,
len);
        if (mask == 3)
            best = max(best, {len, r[node]
- len});
        return mask;
    }
    static p32 LCS(string s, string t) {
        SuffixTree st(s + (char)('z' + 1) +
t + (char)('z' + 2));
        st.lcs(0, sz(s), sz(s) + 1 + sz(t),
0);
        return st.best;
    }
};
```

Template

```
#pragma GCC optimize("-Ofast")
#include <bits/stdc++.h>
using namespace std;
#define fastio
ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0)

#define pb push_back
#define mp make_pair
#define fi first
#define se second
#define memreset(a) memset(a,0,sizeof(a))
#define forstl(i,v) for(auto &i: v)
#define forn(i,e) for(int i = 0; i < e;
i++)
#define forsn(i,s,e) for(int i = s; i < e;
i++)
#define rforsn(i,s,e) for(int i = s; i >=
e; i--)
#define leadzero(a) __builtin_clz(a) //
count leading zeroes
#define trailzero(a) __builtin_ctz(a) //
count trailing zeroes
#define bitcount(a) __builtin_popcount(a)
// count set bits (add ll)
#define ln '\n'
#define dbg(args...) { string _s = #args;
replace(_s.begin(), _s.end(), ',', ' '); \
stringstream _ss(_s);
istream_iterator<string> _it(_ss); err(_it,
args); }
void err(istream_iterator<string> it) {
cerr<<endl; }
template<typename T, typename... Args>
void err(istream_iterator<string> it, T a,
Args... args) {
    cerr << *it << " = " << a << "\t";
err(++it, args...);
}
template<typename T1,typename T2>
ostream& operator <<(ostream& c,pair<T1,T2>
&v){
    c<<"("<<v.fi<<","<<v.se<<")";
}
template <template <class...> class TT,
class ...T>
ostream& operator<<(ostream& out,TT<T...>&
```

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 team mate. Ask the team mate 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 team mate 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 team mates think about your algorithm?

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

Primes - 10001st prime is 1299721, 100001st prime is 15485867 Large primes - 999999937, 1e9+7, 987646789, 987101789 78498 primes less than 10^6 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 7! 5040, 8! 40320, 9! 362880, 10! 362880, 11! 4.0e7, 12! 4.8e8, 15! 1.3e12, 20! 2e18

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.

Articulation points and bridges articulation point:- there exist child : dfslow[child] >= dfsnum[curr] bridge :- tree ed: dfslow[ch] > dfsnum[par];

A connected multigraph has an Euler path but not an Euler circuit if and only if it has exactly two vertices of odd degree.

Binomial coefficients - base case ncn and nc0 = 1; recursion is nCk = (n-1)C(k-1)+(n-1)Ck

Catalan numbers - used in valid paranthesis expressions - formula is Cn = summation{i=0 to n-1} (CiCn-i-1); Another formula is Cn = 2nCn/(n+1). There are Cn binary trees of n nodes and Cn-1 rooted trees of n nodes

Derangements - D(n) = (n-1)(D(n-1)+D(n-2))

Burnsideâ€™s Lemma - number of equivalence classes = (summation I(pi))/n : I(pi) are number of fixed points. Usual formula: [summation {i=0 to n-1} k^gcd(i,n)]/n

Stirling numbers - first kind - permutations of n elements with k disjoint cycles. s(n+1,k) = ns(n,k)+s(n,k-1). s(0,0) = 1, s(n,0) = 0 if n>0. Summation {k=0 to n} s(n,k) = n!

Stirling numbers - Second kind - partition n objects into k non empty subsets. S(n+1,k) = kS(n,k) + S(n,k-1). S(0,0) = 1, S(n,0) = 0 if n>0. S(n,k) = (summation{j=0 to k} [(-1)^(k-j)*kCj*j^n])/k!

Hermite identity - summation{k=0 to n-1} floor[(x+k)/n] = floor[nx]

```
c){
    out<<"{ ";
    forstl(x,c) out<<x<<" ";
    out<<"}";
}
typedef long long int ll;
const int LIM = 1e5+5,MOD = 1e9+7;
int main(){
    fastio;
    return 0;
}
```

TernarySearch

```
/* Find the smallest i in [a,b] that
maximizes f(i), assuming that f(a) < .. <
f(i) >= .. >= f(b)
 * To reverse which of the sides allows
non-strict inequalities, change the <
marked with (A) to <=, and reverse the loop
at (B).
 * To minimize f, change it to >, also at
(B).
 * Usage: int ind = ternSearch(0,n-1,[&]
(int i){return a[i];});
 * Time: O(log(b-a))
 */
template<class F>
int ternSearch(int a, int b, F f) {
    assert(a <= b);
    while (b - a >= 5) {
        int mid = (a + b) / 2;
        if (f(mid) < f(mid+1)) // (A)
            a = mid;
        else
            b = mid+1;
    }
    forsn(i,a+1,b+1) if (f(a) < f(i)) a =
i; // (B)
    return a;
}
```

Kirchoff matrix tree theorem - number of spanning trees in a graph is determinant of Laplacian Matrix with one row and column removed, where L = degree matrix - adjacency matrix

Expected value tricks: 1. Linearity of Expectation: E(X+Y) = E(X)+E(Y) 2. Contribution to the sum - If we want to find the sum over many ways/possibilities, we should consider every element (maybe a number, or a pair or an edge) and count how many times it will be added to the answer. 3. For independent events - E(XY) = E(X)E(Y) 4. Ordered pairs (Super interpretation of square) - The square of the size of a set is equal to the number of ordered pairs of elements in the set. So we iterate over pairs and for each we compute the contribution to the answer. Similarly, the k-th power is equal to the number of sequences (tuples) of length k. 5. Powers technique - If you want to maintain the sum of k-th powers, it might help to also maintain the sum of smaller powers. For example, if the sum of 0-th, 1-th and 2-nd powers is S0, S1 and S2, and we increase every element by x, the new sums are S0, S1â€‰+â€‰S0Â·x and S2â€‰+â€‰2Â·xÂ·S1â€‰+â€‰x2Â·S0.

For Compile alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++14 \ -fsanitize=undefined,address'