Team Note of Winter_Is_Coming

Adid, Sowrav, Shuvro

Compiled on December 12, 2025

Contents

1 Data Structure

1.1 Segment Tree

```
const int N = 3e5 + 9;
int a[N];
struct ST {
  int t[4 * N];
  static const int inf = 1e9;
  ST() {
    memset(t, 0, sizeof t);
  }
  void build(int n, int b, int e) {
    if (b == e) {
      t[n] = a[b];
      return;
    }
    int mid = (b + e) >> 1, l = n << 1, r = l | 1;
    build(l, b, mid);
    build(r, mid + 1, e);
    t[n] = max(t[l], t[r]);
  }
  void upd(int n, int b, int e, int i, int x) {
    if (b > i || e < i) return;
    if (b == e && b == i) {
      t[n] = x;
      return;
    }
    int mid = (b + e) >> 1, l = n << 1, r = l | 1;
    upd(l, b, mid, i, x);
    upd(r, mid + 1, e, i, x);
    t[n] = max(t[l], t[r]);
  }
  int query(int n, int b, int e, int i, int j) {
    if (b > j || e < i) return -inf;
    if (b >= i && e <= j) return t[n];
    int mid = (b + e) >> 1, l = n << 1, r = l | 1;
    int L = query(l, b, mid, i, j);
    int R = query(r, mid + 1, e, i, j);
    return max(L, R);
  }
}t;
```

1.2 Lazy Segment Tree

```
const int N = 5e5 + 9;
int a[N];
struct ST {
  #define lc (n << 1)
  #define rc ((n << 1) | 1)
  long long t[4 * N], lazy[4 * N];
  ST() {
    memset(t, 0, sizeof t);
    memset(lazy, 0, sizeof lazy);
  }
  inline void push(int n, int b, int e) {
    if (lazy[n] == 0) return;
    t[n] = t[n] + lazy[n] * (e - b + 1);
    if (b != e) {
      lazy[lc] = lazy[lc] + lazy[n];
      lazy[rc] = lazy[rc] + lazy[n];
    }
    lazy[n] = 0;
  }
  inline long long combine(long long a,long long b) {
    return a + b;
  }
  inline void pull(int n) {
    t[n] = t[lc] + t[rc];
  }
  void build(int n, int b, int e) {
    lazy[n] = 0;
    if (b == e) {
      t[n] = a[b];
      return;
    }
    int mid = (b + e) >> 1;
    build(lc, b, mid);
    build(rc, mid + 1, e);
    pull(n);
  }
  void upd(int n, int b, int e, int i, int j, long long v)
  {
    push(n, b, e);
    if (j < b || e < i) return;
    if (i <= b && e <= j) {
      lazy[n] = v; //set lazy
      push(n, b, e);
      return;
    }
    int mid = (b + e) >> 1;
    upd(lc, b, mid, i, j, v);
    upd(rc, mid + 1, e, i, j, v);
    pull(n);
  }
  long long query(int n, int b, int e, int i, int j) {
    push(n, b, e);
    if (i > e || b > j) return 0; //return null
    if (i <= b && e <= j) return t[n];
    int mid = (b + e) >> 1;
    return combine(query(lc, b, mid, i, j), query(rc, mid +
    1, e, i, j));
  }
};
```

1.3 Persistent Segment Tree

```
int ar[100005];
struct node{
  node *left, *right;
  int val;
  node(int a = 0, node *b = NULL, node *c = NULL) :
    val(a), left(b), right(c) {}// ** Constructor
  int merge(int x, int y) {return x + y;}
  void build(int l, int r) {  // We are not  initializing
  values for now.
    if(l == r) {
        val = ar[l];
        return; // We reached leaf node, No need more links
    }
    left = new node(); // Create new node for Left child
    right = new node();// We are creating nodes only when
    necessary!
    int mid = l + r >> 1;
    left -> build(l, mid);
    right -> build(mid+1, r);
    val = merge(left -> val, right -> val);
  }
  node *update(int l, int r, int idx, int v) {
    if(r < idx || l > idx) return this; // Out of range,
    use this node.
    if(l == r) {  // Leaf Node, create new node and return
    that.
      node *ret = new node(val, left, right);
      ret -> val += v;
      return ret;
      // we first cloned our current node, then added v to
      the value.
    }
    int mid = l + r >> 1;
    node *ret = new node(val); //Create a new node, as idx
    in in [l, r]
    ret -> left = left -> update(l, mid, idx, v);
    ret -> right = right -> update(mid+1, r, idx, v);
    // Note that 'ret -> left' is new node's left child,
    // But 'left' is current old node's left child.
    // So we call to update idx in left child of old node.
```

```
    // And use it's return node as new node's left child.
    Same for right.
    ret -> val = merge( ret -> left -> val, ret -> right ->
    val); // Update value.
    return ret; // Return the new node to parent.
  } // [l, r] node range, [i, j] query range.
  int query(int l, int r, int i, int j) {
    if(r < i || l > j) return 0; // out of range
    if(i <= l && r <= j) { // completely inside
      return val;  // return value stored in this node
    } int mid = l + r >> 1;
    return merge(left -> query(l, mid, i, j), right ->
    query(mid+1, r, i, j));
  }
}*root[100005];
int main() {
  root[0] = new node();
  root[0] -> build(0, n - 1);
}
```

1.4 Sparse table

```
int t[N][18], a[N];
void build(int n) {
  for(int i = 1; i <= n; ++i) t[i][0] = a[i];
  for(int k = 1; k < 18; ++k) {
    for(int i = 1; i + (1 << k) - 1 <= n; ++i) {
      t[i][k] = min(t[i][k - 1], t[i + (1 << (k - 1))][k -
      1]);
    }
  }
}
int query(int l, int r) {
  int k = 31 - __builtin_clz(r - l + 1);
  return min(t[l][k], t[r - (1 << k) + 1][k]);
}
// For idempodent functions, we can calculate it in O(n).
then, ans = gcd(spt[K-1][l], spt[K-1][l+(1<<(K-1))]);
```

1.5 BIT

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

1.6 MO's algo

```
const int MX = 2e5;   // query size . . .
const int MXI = 1e6;  // maximum value in array . . .
ll cnt[MXI+5];
ll block_size, range_ans;
struct Query{
    int id, l, r;
    Query() {}
    Query(int _id, int _l, int _r){
     id = _id;
     l = _l;
     r = _r; }
    bool operator<(Query &other) const{
     int curr_size = l/block_size;
     int other_size = other.l/block_size;
     if(curr_size == other_size) return r < other.r;
     return curr_size < other_size;
    }} query[MX+5];
void add(ll x){
    if(cnt[x]) range_ans -= cnt[x]*cnt[x]*x;
    cnt[x]++;
    range_ans += cnt[x]*cnt[x]*x; }
void rmv(ll x){
    range_ans -= cnt[x]*cnt[x]*x;
    cnt[x]--;
    if(cnt[x]) range_ans += cnt[x]*cnt[x]*x; }
ll get_solution(){
    return range_ans; }
void mos_algo(){
    block_size = sqrt(n);
    int ara[n];
    for(int K = 0; K < n; K++) cin >> ara[K];
    for(int K = 0; K < t; K++){
     cin >> l >> r;
     query[K] = Query(K, l-1, r-1); // 0-based indexing .
     .}
    sort(query, query+t);
    int L = 0, R = -1;
    ll v[t];
    for(int K = 0; K < t; K++){
     l = query[K].l;
     r = query[K].r;
     while(L < l){
      rmv(ara[L]);
      L++;}
     while(L > l){
      L--;
      add(ara[L]); }
     while(R < r){
      R++;
      add(ara[R]); }
     while(R > r){
      rmv(ara[R]);
      R--;}
     v[query[K].id] = get_solution; }
    for(int K = 0; K < t; K++) cout << v[K] << "\n"; }}
```

1.7 MO's on Tree

```
const int N = 3e5 + 9;
//unique elements on the path from u to v
vector<int> g[N];
int st[N], en[N], T, par[N][20], dep[N], id[N * 2];
void dfs(int u, int p = 0) {
  st[u] = ++T;
  id[T] = u;
  dep[u] = dep[p] + 1;
  par[u][0] = p;
  for(int k = 1; k < 20; k++) par[u][k] = par[par[u][k -
  1]][k - 1];
  for(auto v : g[u]) if(v != p) dfs(v, u);
  en[u] = ++T;
  id[T] = u;
}
int lca(int u, int v) {
  if(dep[u] < dep[v]) swap(u, v);
  for(int k = 19; k >= 0; k--) if(dep[par[u][k]] >= dep[v])
  u = par[u][k];
  if(u == v) return u;
  for(int k = 19; k >= 0; k--) if(par[u][k] != par[v][k]) u
  = par[u][k], v = par[v][k];
  return par[u][0];
}
int cnt[N], a[N], ans;
inline void add(int u) {
  int x = a[u];
  if(cnt[x]++ == 0) ans++;
}
inline void rem(int u) {
  int x = a[u];
  if(--cnt[x] == 0) ans--;
}
bool vis[N];
inline void yo(int u) {
  if(!vis[u]) add(u);
  else rem(u);
  vis[u] ^= 1;
}
const int B = 320;
struct query {
  int l, r, id;
  bool operator < (const query &x) const {
 if(l / B == x.l / B) return r < x.r;
 return l / B < x.l / B;
  }
} Q[N];
int res[N];
int main() {
  int n, q;
  while(cin >> n >> q) {
 for(int i = 1; i <= n; i++) cin >> a[i];
 map<int, int> mp;
 for(int i = 1; i <= n; i++) {
   if(mp.find(a[i]) == mp.end()) mp[a[i]], mp[a[i]] =
   mp.size();
   a[i] = mp[a[i]];
 }
 for(int i = 1; i < n; i++) {
   int u, v;
   cin >> u >> v;
   g[u].push_back(v);
   g[v].push_back(u);
 }
 T = 0;
 dfs(1);
 for(int i = 1; i <= q; i++) {
   int u, v ;
   cin >> u >> v;
   if(st[u] > st[v]) swap(u, v);
   int lc = lca(u, v);
   if(lc == u) Q[i].l = st[u], Q[i].r = st[v];
   else Q[i].l = en[u], Q[i].r = st[v];
   Q[i].id = i;
 }
 sort(Q + 1, Q + q + 1);
 ans = 0;
 int l = 1, r = 0;
 for(int i = 1; i <= q; i++) {
   int L = Q[i].l, R = Q[i].r;
   if(R < l) {
     while (l > L) yo(id[--l]);
     while (l < L) yo(id[l++]);
     while (r < R) yo(id[++r]);
     while (r > R) yo(id[r--]);
   } else {
     while (r < R) yo(id[++r]);
     while (r > R) yo(id[r--]);
     while (l > L) yo(id[--l]);
     while (l < L) yo(id[l++]);
   }
   int u = id[l], v = id[r], lc = lca(u, v);
   if(lc != u && lc != v) yo(lc); //take care of the lca
   separately
   res[Q[i].id] = ans;
   if(lc != u && lc != v) yo(lc);
 }
 for(int i = 1; i <= q; i++) cout << res[i] << '\n';
 for(int i = 0; i <= n; i++) {
   g[i].clear();
   vis[i] = cnt[i] = 0;
   for(int k = 0; k < 20; k++) par[i][k] = 0;
 }
  }
  return 0;
}
```

1.8 Trie

```
struct Trie{
 const int A = 26;
 int N;
 vector<vector<int> > next;
 vector<int> cnt;
 Trie() : N(0) { node(); }
 int node(){
  next.emplace_back(A, -1);
  cnt.emplace_back(0);
  return N++; }
 inline int get(char c){ return c-'a'; }
 inline void insert(string s){
  int cur = 0;
  for(char c : s){
   int to = get(c);
   if(next[cur][to] == -1) next[cur][to] = node();
   cur = next[cur][to];
  }
  Cnt[cur]++; }
 inline bool find(string s){
  int cur = 0;
  for(char c : s){
   int to = get(c);
   if(next[cur][to] == -1) return false;
   cur = next[cur][to];
  }
  return cnt[cur]!=0;
 }
 // Doesn't check for existance
 inline void erase(string s){
  int cur = 0;
  for(char c : s){
   int to =get(c);
   cur = next[cur][to];
  }
  cnt[cur]--;
 }
 vector<string> dfs(){
 stack<pair<int, int> > st;
 string s;
 vector<string> ret;
     for(st.push({0, -1}), s.push_back('$'); !st.empty();
     ){
```

```
  auto [u,  c] = st.top();
  st.pop();
  s.pop_back();
  c++;
  if(c <  A){
   st.push({u, c});
   s.push_back(c+'a');
   int v = next[u][c];
   if(~v){
    if(cnt[v]) ret.emplace_back(s);
    st.push({v, -1});
    s.push_back('$');
  }}}
    return ret;} };
////////// for bits
struct Trie {
   static const int B = 31;
 // Node structure of Trie...
   struct Node {
     Node* links[2]; // two possibility to go next.
     int sz;   // how many prefixes
  Node() {
   links[0] = links[1] = NULL;
   sz = 0;
  }
  bool containsKey(int ind) {
        return (links[ind] != NULL);
  }
  Node* get(int ind) {
   return links[ind];
  }
  void put(int ind, Node* Node) {
   links[ind] = Node;
  }
  } *root;
 Trie() {
  root = new Node();
 }
 // insert value in trie.
 void insert(int val) {
  Node* cur = root;
  cur->sz++;
  for (int i = B - 1; i >= 0; i--) {
   int b = val >> i & 1;
   if(!cur->containsKey(b)) {
               cur->put(b, new Node());
            }
   cur = cur->links[b];
   cur->sz++;
  }
 }
 // delete node from trie...
 void del(Node* cur) {
     for (int i = 0; i < 2; i++) if (cur -> links[i])
     del(cur -> links[i]);
     delete(cur);
   }
 // number of values in trie (formally given in array)
 // such that val ^ x < k
   int query(int x, int k) {
     Node* cur = root;
     int ans = 0;
     for (int i = B - 1; i >= 0; i--) {
        if (cur == NULL) break;
        int b1 = x >> i & 1, b2 = k >> i & 1;
        if (b2 == 1) {
          if (cur->containsKey(b1)) ans +=
          cur->links[b1]->sz;
          cur = cur->links[!b1];
        } else {
    cur = cur->links[b1];
   }
    }
     return ans;
   }
 // get ans such that ans ^ x is maximized where x is in
 trie(formally given in array).
 // complexity O(32)
 int get_max(int x) {
  Node* cur = root;
  int ans = 0;
  for (int i = B - 1; i >= 0; i--) {
   int k = x >> i & 1;
   if (cur->containsKey(!k)){
    cur = cur->links[!k];
    ans <<= 1;
    ans++;
   } else {
    cur = cur->links[k];
    ans <<= 1;
   }
  }
  return ans;
 }
 // get ans such that ans ^ x is minimized where x is in
 trie(formally given in array).
 // complexity O(32)
   int get_min(int x) {
     Node* cur = root;
     int ans = 0;
     for (int i = B - 1; i >= 0; i--) {
        int k = x >> i & 1;
    if (cur->containsKey(k)){
     cur = cur->links[k];
     ans <<= 1;
    } else {
     cur = cur->links[!k];
     ans <<= 1;
     ans++;
    }
     }
     return ans;
   }
} trie;
```

1.9 DSU rollback

```
struct DSUrollback {
    int n, cmp, cur = -1;
    vector<int> par, rank, cmpsz;
    vector<pii> stack;
    void init(int n) {
        this->n = cmp = n;
        par.resize(n + 5), rank.resize(n + 5),
        cmpsz.resize(n + 5);
        for(int i = 0; i <= n; i++)
        par[i]=i, rank[i] = 1;
    }
    int find(int x) {
        if(x == par[x]) return x;
        return find(par[x]);
    }
    bool merge(int x, int y) {
        int xroot = find(x), yroot = find(y);
        if(xroot != yroot) {
            if(rank[xroot] < rank[yroot]) swap(xroot,
            yroot);
            par[yroot] = xroot;
            rank[xroot] += rank[yroot];
            stack.PB({yroot, xroot});
            cur++;
            cmpsz[cur] = cmp;
            cmp--;
            return true;
        }
        else {
            stack.PB({-1, -1});
            return false;
        }
    }
    void rollback() {
        if(stack.back().fi == -1) {
            stack.pop_back(); return; // no change in last
            operation
        }
        par[stack.back().fi] = stack.back().fi;
        rank[stack.back().se] -= rank[stack.back().fi];
        cmp = cmpsz[cur];
        cur--;
        stack.pop_back();
    }
};
```

1.10 DSU on Tree Smaller to larger (DSU on tree)

Time Complexity: $O(n \log^2 n)$

```
const int N = 2e5;
vector<int> edge[N+5], vec[N+5], color(N+5), sz(N+5),
cnt(N+5);
void dfs_size(int u, int p){
 sz[u] = 1;
 for(auto v: edge[u]){
     if(v != p){
         dfs_size(v, u);
         sz[u] += sz[v];
     }}}
void dfs(int u, int p, bool keep){
 int mx = -1, bigchild = -1;
 for(auto v: edge[u]){
     if(v != p && mx < sz[v]){
         mx = sz[v];
         bigchild = v;}}
 for(auto v: edge[u]){
     if(v != p && v != bigchild){
         dfs(v, u, 0);}}
 if(bigchild != -1){
     dfs(bigchild, u, 1);
     swap(vec[u], vec[bigchild]);}
 vec[u].push_back(u);
 cnt[color[u]]++;
 for(auto v: edge[u]){
     if(v != p && v != bigchild){
         for(auto x: vec[v]){
             vec[u].push_back(x);
             cnt[color[x]]++;
         }}}
 // ans area of a subtree...
 if(keep == 0){
     for(auto v: vec[u]){
         cnt[color[v]]--;}}}
```

1.11 Centroid Decomposition

```
const int N= 2e5+5; // check
int sub[N], par[N], lvl[N], vis[N], cnt[N];
vector<int> Tree[N];
int get_sub(int u,int p) {
    sub[u]=1;
    for(auto v : Tree[u]){
        if(v != p && vis[v] == 0) sub[u]+=get_sub(v, u);
    }
    return sub[u];
}
int get_centroid(int u,int p,int n) {
    for(auto v:Tree[u]){
        if(vis[v]) continue;
        if(v != p && sub[v] > n/2)
        return get_centroid(v, u, n);
    }
    return u;
}
void add_centroid(int x, int y) {
    par[y] = x;
    lvl[y] = lvl[x] + 1;
    vis[y] = 1;
}
void build_centroid(int u,int p = -1) {
    int n=get_sub(u,p); // subtree size
    int centroid = get_centroid(u,p,n);
    if(p == -1) p = centroid;
    add_centroid(p, centroid);
    for(auto v : Tree[centroid]){
        if(vis[v]) continue;
        build_centroid(v, centroid);
    }
}
```

1.12 HLD

```
int val[N];
vector<int> adj[N];
int sub[N], dep[N], par[N];
int head[N] , st[N] , en[N] , clk;
int dfsarr[N];
/* Segment Tree for Euler Tour Technique */
struct LazySeg {
  int n;
  vector<int> t, lz;
  LazySeg(int n){
    this->n = 1;
    while(this->n < n) this->n <<= 1;
    t.assign(2*this->n, -inf);
    lz.assign(this->n, 0);
  }
  void apply(int p, int val){
    t[p] += val;
    if(p < n) lz[p] += val;
  }
  void push(int p){
    apply(p<<1, lz[p]);
    apply(p<<1|1, lz[p]);
    lz[p] = 0;
  }
  void push_path(int p){
    for(int h = __lg(n); h > 0; h--){
      int x = p >> h;
      if(lz[x]) push(x);
    }
  }
  void pull(int p){
    while(p > 1){
      p >>= 1;
      t[p] = max(t[p<<1], t[p<<1|1]) + lz[p];
    }
  }
  // set a leaf to a value (used during
  build/initialization)
  void set_point(int pos, int value){
```

```
    int p = pos + n;
    push_path(p);
    t[p] = value;
    // no lazy on leaf
    p >>= 1;
    while(p){
      t[p] = max(t[p<<1], t[p<<1|1]) + lz[p];
      p >>= 1;
    }
  }
  void range_add(int l, int r, int val){
    l += n; r += n;
    int L = l, R = r;
    push_path(L);push_path(R);
    while(L <= R){
      if(L & 1) apply(L++, val);
      if(!(R & 1)) apply(R--, val);
      L >>= 1; R >>= 1;
    }
    pull(l);pull(r);
  }
  int range_max(int l, int r){
    l += n; r += n;
    push_path(l); push_path(r);
    int res = -inf;
    while(l <= r){
      if(l & 1) res = max(res, t[l++]);
      if(!(r & 1)) res = max(res, t[r--]);
      l >>= 1; r >>= 1;
    }
    return res;
  }
};
LazySeg seg(N);
// compute size of subtree and heavy child at adj[u][0]
void dfs_sz(int u , int p = 0){
 sub[u] = 1 , par[u] = p;
 dep[u] = (p == 0) ? 0 : dep[p] + 1;
 int mxv = -1;
 for(auto &x : adj[u]){
     if(x == p) continue;
     dfs_sz(x , u);
     sub[u] += sub[x];
     if(sub[x] > mxv){
         mxv = sub[x];
         swap(x , adj[u][0]);
     }
 }
}
// compute head , st , en arrays
void dfs_hld(int u , int p = 0){
 st[u] = ++clk;
 if(p == 0) head[u] = u;
 else if(adj[p][0] == u) head[u] = head[p];
 else head[u] = u;
 for(auto &x : adj[u]){
     if(x == p) continue;
     dfs_hld(x , u);
 }
 en[u] = clk;
}
int lca(int a , int b){
 while(head[a] != head[b]){
     if(dep[head[a]] > dep[head[b]]) swap(a , b);
     b = par[head[b]];
 }
 if(dep[a] > dep[b]) swap(a , b);
 return a;
}
// process path query from a to b (excl = whether to
exclude lca node)
// write same for path updates
int path_process(int a , int b , bool excl = false){
 int ret = 0;
 while(head[a] != head[b]){
        if(dep[head[a]] > dep[head[b]]) swap(a , b);
        // make the query on range query data structure
        ret = max(ret, seg.range_max(st[head[b]], st[b]));
        b = par[head[b]];
 }

 if(dep[a] > dep[b]) swap(a , b);
    // make the query on range query data structure
    // st[a]  to st[b] on the same chain (excl = whether to
    exclude lca node)
    ret = max(ret, seg.range_max(st[a] + excl, st[b]));
 return ret;
} // remember chain are consecutive in dfsarr
// dfs_sz(1);
// dfs_hld(1);
// dfsarr[st[i]] = val[i];
// seg.set_point(st[i], val[i]);
// int ans = path_process(u, v, false);
```

1.13 Treap

```
using treap_val = char; // treap value data type
struct node {
    node *L, *R; int W, S;
    treap_val V; bool F;
    node(char x) {
L = R = 0; W = rand();
S = 1; V = x; F = 0; }
};
int size(node *treap) {
return (treap == 0 ? 0 : treap->S);}
void push(node *treap) {
    if (treap && treap->F) {
treap->F = 0;
swap(treap->L, treap->R);
if (treap->L) treap->L->F ^= 1;
if (treap->R) treap->R->F ^= 1;
}}
// k -> 1-indexed || K index in left tree
void split(node *treap, node *&left, node *&right, int k) {
if (treap == 0) left = right = 0;
else {
    push(treap);
    if (size(treap->L) < k) {
split(treap->R, treap->R, right, k - size(treap->L) - 1);
left = treap;}
    else {
split(treap->L, left, treap->L, k);
right = treap;}
treap->S = size(treap->L) + size(treap->R) + 1;}
}
// root - left root - right root -> merge tree root is
treap
void merge(node *&treap, node *left, node *right) {
    if (left == 0) treap = right;
    else if (right == 0) treap = left;
    else {
push(left);push(right);
        if (left->W < right->W) {
merge(left->R, left->R, right);
treap = left;}
        else {
merge(right->L, left, right->L);
treap = right;}
treap->S = size(treap->L) + size(treap->R) + 1;}}
void add(node *&treap, treap_val x) {
merge(treap, treap, new node(x));
}
// insert at position upper_bound(x)
void insert(node *&treap, int x) {
if(treap == 0) {
treap = new node(x); return;}
node *a, *b; split(treap, a, b, x);
add(a, x); merge(treap, a, b);}
// pos -> 1-indexed
void remove(node *&treap, int pos) {
node *A, *B, *C;
split(treap, A, B, pos - 1);
split(B, B, C, 1);
merge(treap, A, C);}
void reverse(node *&treap, int x, int y) {
node *A, *B, *C;
split(treap, A, B, x - 1);
split(B, B, C, y - x + 1);
B->F ^= 1; merge(treap, A, B);
merge(treap, treap, C);}
void print(node *treap) {
if (treap == NULL) return;
push(treap); print(treap->L);
cout << treap->V; print(treap->R);}
```

1.14 Implicit Treap

```
mt19937
rnd(chrono::steady_clock::now().time_since_epoch().count());
const int N = 3e5 + 9, mod = 1e9 + 7;
struct treap { //This is an implicit treap which
investigates here on an array
  struct node {
    int val, sz, prior, lazy, sum, mx, mn, repl;
    bool repl_flag, rev;
    node *l, *r, *par;
    node() {
      lazy = 0;rev = 0;sum = 0;val = 0;sz = 0;
      mx = 0;mn = 0;repl = 0;repl_flag = 0;
      prior = 0;l = NULL;r = NULL;par = NULL;
    }
    node(int _val) {
      val = _val;sum = _val;mx = _val;mn = _val;
      repl = 0;repl_flag = 0;rev = 0;lazy = 0;
      sz = 1;prior = rnd();l = NULL;r = NULL;
      par = NULL;
    }
  };
  typedef node* pnode;
  pnode root;
  map<int, pnode> position;//positions of all the values
  //clearing the treap
void clear() {
root = NULL;position.clear();}
treap() {clear();}
  int size(pnode t) {
    return t ? t->sz : 0;
  }
  void update_size(pnode &t) {
    if(t) t->sz = size(t->l) + size(t->r) + 1;
  }
  void update_parent(pnode &t) {
    if(!t) return;
    if(t->l) t->l->par = t;
    if(t->r) t->r->par = t;
  }
  //add operation
  void lazy_sum_upd(pnode &t) {
    if( !t or !t->lazy )    return;
    t->sum += t->lazy * size(t);
    t->val += t->lazy;
    t->mx += t->lazy;
    t->mn += t->lazy;
    if( t->l )  t->l->lazy += t->lazy;
    if( t->r )  t->r->lazy += t->lazy;
    t->lazy = 0;
  }
  //replace update
  void lazy_repl_upd(pnode &t) {
    if( !t or !t->repl_flag )   return;
    t->val = t->mx = t->mn = t->repl;
    t->sum = t->val * size(t);
    if( t->l ) {
      t->l->repl = t->repl;
      t->l->repl_flag = true;
    }
    if( t->r ) {
      t->r->repl = t->repl;
      t->r->repl_flag = true;
    }
    t->repl_flag = false;
    t->repl = 0;
  }
  //reverse update
  void lazy_rev_upd(pnode &t) {
    if( !t or !t->rev ) return;
    t->rev = false;
    swap(t->l, t->r);
    if( t->l )  t->l->rev ^= true;
    if( t->r )  t->r->rev ^= true;
  }
  //reset the value of current node assuming it now
  //represents a single element of the array
  void reset(pnode &t) {
    if(!t) return;
    t->sum = t->val;
    t->mx = t->val;
    t->mn = t->val;
  }
  //combine node l and r to form t by updating
  corresponding queries
  void combine(pnode &t, pnode l, pnode r) {
    if(!l) {
      t = r;
      return;
    }
    if(!r) {
      t = l;
      return;
    }
    //Beware!!!Here t can be equal to l or r anytime
    //i.e. t and (l or r) is representing same node
    //so operation is needed to be done carefully
    //e.g. if t and r are same then after
    t->sum=l->sum+r->sum operation,
    //r->sum will be the same as t->sum
    //so BE CAREFUL
    t->sum = l->sum + r->sum;
    t->mx = max(l->mx, r->mx);
    t->mn = min(l->mn, r->mn);
  }
  //perform all operations
  void operation(pnode &t) {
    if( !t )    return;
    reset(t);
    lazy_rev_upd(t->l);
    lazy_rev_upd(t->r);
    lazy_repl_upd(t->l);
    lazy_repl_upd(t->r);
    lazy_sum_upd(t->l);
```

```
  lazy_sum_upd(t->r);
  combine(t, t->l, t);
  combine(t, t, t->r);
}
//split node t in l and r by key k
//so first k+1 elements(0,1,2,...k) of the array from
node t
//will be split in left node and rest will be in right
node
void split(pnode t, pnode &l, pnode &r, int k, int add =
0) {
  if(t == NULL) {
    l = NULL;
    r = NULL;
    return;
  }
  lazy_rev_upd(t);
  lazy_repl_upd(t);
  lazy_sum_upd(t);
  int idx = add + size(t->l);
  if(t->l) t->l->par = NULL;
  if(t->r) t->r->par = NULL;
  if(idx <= k)
    split(t->r, t->r, r, k, idx + 1), l = t;
  else
    split(t->l, l, t->l, k, add), r = t;
  update_parent(t);
  update_size(t);
  operation(t);
}
//merge node l with r in t
void merge(pnode &t, pnode l, pnode r) {
  lazy_rev_upd(l);
  lazy_rev_upd(r);
  lazy_repl_upd(l);
  lazy_repl_upd(r);
  lazy_sum_upd(l);
  lazy_sum_upd(r);
  if(!l) {
    t = r;
    return;
  }
  if(!r) {
    t = l;
    return;
  }
  if(l->prior > r->prior)
    merge(l->r, l->r, r), t = l;
  else
    merge(r->l, l, r->l), t = r;
  update_parent(t);
  update_size(t);
  operation(t);
}
//insert val in position a[pos]
//so all previous values from pos to last will be right
shifted
void insert(int pos, int val) {
  if(root == NULL) {
    pnode to_add = new node(val);
    root = to_add;
    position[val] = root;
    return;
  }
  pnode l, r, mid;
  mid = new node(val);
  position[val] = mid;
  split(root, l, r, pos - 1);
  merge(l, l, mid);
  merge(root, l, r);
}
//erase from qL to qR indexes
//so all previous indexes from qR+1 to last will be left
shifted qR-qL+1 times
void erase(int qL, int qR) {
  pnode l, r, mid;
  split(root, l, r, qL - 1);
  split(r, mid, r, qR - qL);
  merge(root, l, r);
}
//returns answer for corresponding types of query
int query(int qL, int qR) {
  pnode l, r, mid;
  split(root, l, r, qL - 1);
  split(r, mid, r, qR - qL);
  int answer = mid->sum;
  merge(r, mid, r);
  merge(root, l, r);
  return answer;
}
//add val in all the values from a[qL] to a[qR] positions
void update(int qL, int qR, int val) {
  pnode l, r, mid;
  split(root, l, r, qL - 1);
  split(r, mid, r, qR - qL);
  lazy_repl_upd(mid);
  mid->lazy += val;
  merge(r, mid, r);
  merge(root, l, r);
}
//reverse all the values from qL to qR
void reverse(int qL, int qR) {
  pnode l, r, mid;
  split(root, l, r, qL - 1);
  split(r, mid, r, qR - qL);
  mid->rev ^= 1;
  merge(r, mid, r);
  merge(root, l, r);
}
//replace all the values from a[qL] to a[qR] by v
void replace(int qL, int qR, int v) {
  pnode l, r, mid;
  split(root, l, r, qL - 1);
  split(r, mid, r, qR - qL);
  lazy_sum_upd(mid);
  mid->repl_flag = 1;
  mid->repl = v;
  merge(r, mid, r);
  merge(root, l, r);
}
//it will cyclic right shift the array k times
//so for k=1, a[qL]=a[qR] and all positions from ql+1 to
qR will
//have values from previous a[qL] to a[qR-1]
//if you make left_shift=1 then it will to the opposite
void cyclic_shift(int qL, int qR, int k, bool left_shift
= 0) {
  if(qL == qR) return;
  k %= (qR - qL + 1);
  pnode l, r, mid, fh, sh;
  split(root, l, r, qL - 1);
  split(r, mid, r, qR - qL);
  if(left_shift == 0) split(mid, fh, sh, (qR - qL + 1) -
  k - 1);
  else split(mid, fh, sh, k - 1);
  merge(mid, sh, fh);
  merge(r, mid, r);
  merge(root, l, r);
}
bool exist;
//returns index of node curr
int get_pos(pnode curr, pnode son = nullptr) {
  if(exist == 0) return 0;
  if(curr == NULL) {
    exist = 0;
    return 0;
  }
  if(!son) {
    if(curr == root) return size(curr->l);
    else return size(curr->l) + get_pos(curr->par, curr);
  }
  if(curr == root) {
    if(son == curr->l) return 0;
    else return size(curr->l) + 1;
  }
  if(curr->l == son) return get_pos(curr->par, curr);
  else return get_pos(curr->par, curr) + size(curr->l) +
  1;
}
//returns index of the value
//if the value has multiple positions then it will
//return the last index where it was added last time
//returns -1 if it doesn't exist in the array
int get_pos(int value) {
  if(position.find(value) == position.end()) return -1;
  exist = 1;
  int x = get_pos(position[value]);
  if(exist == 0) return -1;
  else return x;
}
//returns value of index pos
int get_val(int pos) {
  return query(pos, pos);
}
//returns size of the treap
int size() {
  return size(root);
}
//inorder traversal to get indexes chronologically
void inorder(pnode cur) {
  if(cur == NULL) return;
  operation(cur);
  inorder(cur->l);
  cout << cur->val << ' ';
  inorder(cur->r);
}
//print current array values serially
void print_array() {
//      for(int i=0;i<size();i++) cout<<get_val(i)<<' ';
//      cout<<nl;
    inorder(root);
    cout << nl;
  }
  bool find(int val) {
    if(get_pos(val) == -1) return 0;
    else return 1;
  }
};
treap t;
//Beware!!!here treap is 0-indexed
```

2 Game Theory

- Bogus Nim: Pile e stone add kora jabe. But opponent sei add kora stone abr soriye nullify korte parbe. Ebhabe oponent move mirror kore nullify kora jay.
- Grundy Number: Sob impartial game ke ekta nim pile e convert kora jay. Fully independent state gula ke combine korte xor kora lage. Ekta state theke transition diye je je state e jawa possible , sobar grundy valuer mex hobe amr current stater grundy. Dhoro ekta transition diye onno state p te gesi , but p ta current mover karone multiple state e vag hoye gese, tahole p er independent state gula xor diye combine kore then oita consider korbo. Winning move print korte bolle dekhte hobe kon kon move diye oponent ke 0 grundy value te falano jay. Losing state grundy number is 0.
- Misere Nim Last stone je player nibe se harbe . ekhaneo xor==0 hole losing state. Corner case holo sob pile e jodi odd number of stone thake.
- Nim Game N ta pile , proti ta theke stone neya jabe . Je nite parbena se looser. pile gular xor ! = 0 hole 1st player jitbe.
- Staircase Nim Proti siri te ekta pile. ek sirir pile theke stone niye ek step nicher siri te rakha jay. ebhabe even siri gula useless karon opponent mirror korte pare.

3 Math

3.1 Prime Numbers

999995713, 999998173, 999997571, 999997793, 999999193, 999999883, 999998789

3.2 Application of Catalan Numbers

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845,..

- 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 × 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 ith column has a height i)

3.3 Catalan Numbers

$C_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1}\binom{2n}{n}, n \geq 0$

3.4 Stirling Numbers

Falling factorial:

$$x^{\underline{n}} = x(x-1)\cdots(x-n+1) = \prod_{i=0}^{n-1}(x-i) = \sum_{k=0}^{n} s(n,k)x^k.$$

Rising factorial:

$$x^{\overline{n}} = x(x+1)\cdots(x+n-1) = \prod_{i=0}^{n-1}(x+i) = \sum_{k=0}^{n} S(n,k)x^{\underline{k}}.$$

i. Stirling Numbers of the First Kind

$s(n,k)$ = number of permutations of n elements with k cycles.

Example: $s(3,2) = 3$: $(1)(23), (2)(13), (3)(12)$

Recurrence:

$$|s(n,k)| = |s(n-1,k-1)| + (n-1)|s(n-1,k)|, \quad s(n,k) = (-1)^{n-k}|s(n,k)|.$$

Bases: $s(0,0) = 1$, $s(n,0) = s(0,k) = 0$, $s(n,n) = 1$

A. Fixed n (all k)

$$x^{\underline{n}} = \prod_{i=0}^{n-1}(x-i) = \sum_{k=0}^{n} s(n,k)x^k.$$

NTT D&C: Build $P(x) = \prod_{i=0}^{n-1}(x-i)$ recursively; coefficients give $s(n,k)$ Complexity: $O(n \log^2 n)$

B. Fixed k ($n = k..N$)

$$G_k(x) = \sum_{n\geq k} \frac{s(n,k)}{n!}x^n = \frac{(\log(1+x))^k}{k!}, \quad s(n,k) = n!\,[x^n]\,(\log(1+x))^k/k!.$$

Steps: Compute $g(x) = \log(1+x)$, then $g(x)^k/k!$, recover $s(n,k)$ from coefficient of x^n Complexity: $O(N \log N)$

ii. Stirling Numbers of the Second Kind

$S(n,k)$ = number of partitions of n elements into k nonempty subsets.

Example: $S(3,2) = 3$: $\{1\}\{2,3\}, \{2\}\{1,3\}, \{3\}\{1,2\}$

Recurrence:

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

S(0,0)=1, S(n,0)=S(0,k)=0, S(n,n)=1

A. Fixed n (all k)

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

NTT convolution: Let $a_i = (-1)^i/i!$, $b_i = i^n/i!$, then $S(n,k) = [x^k]\sum a_i x^i \cdot \sum b_i x^i$ Complexity: $O(n \log n)$

B. Fixed k ($n = k..N$)

GF: $\sum_{n\geq k} S(n,k)x^n = x^k/\prod_{i=1}^{k}(1-ix)$ Method:

1. $P(x) = \prod_{i=1}^{k}(1-ix)$ via D&C + NTT
2. $Q(x) = 1/P(x) \bmod x^{N-k+1}$
3. $S(n,k) = [x^n](x^k Q(x))$

Complexity: $O(N \log N)$

3.5 Bell Numbers

Bell numbers count the number of ways to partition an n-element set into non-empty subsets, and they appear in combinatorics, DP over set partitions, and problems involving grouping indistinguishable structures. They satisfy the recurrence $B_0 = 1$, $B_{n+1} = \sum_{k=0}^{n}\binom{n}{k}B_k$, and relate to Stirling numbers via $B_n = \sum_{k=0}^{n} S(n,k)$. A useful exponential generating function identity is $\sum_{n\geq 0}\frac{B_n x^n}{n!} = \exp(e^x - 1)$. To compute all $B_0 \ldots B_N$ up to $N = 5\cdot 10^5$ efficiently, we use polynomial/NTT techniques: (1) build the series $f(x) = e^x - 1 = \sum_{i\geq 1} x^i/i!$; (2) compute its exponential $g(x) = \exp(f(x))$ modulo x^{N+1} using polynomial Newton iteration with NTT-based multiplication; (3) multiply coefficients back by factorials to convert the exponential generating function into ordinary Bell numbers. This gives a full $O(n \log^2 n)$ solution fast enough for $N \leq 5\cdot 10^5$.

3.6 Expected Value & Power Technique

i. Expected Value Basics

The expected value (EV) of a random variable X is the average outcome if the experiment is repeated infinitely:

$$\mathbb{E}[X] = \sum_i x_i \cdot \Pr[X = x_i]$$

Linearity of Expectation: For any random variables $X_1, X_2, \ldots, X_n$:

$$\mathbb{E}[X_1 + X_2 + \cdots + X_n] = \mathbb{E}[X_1] + \mathbb{E}[X_2] + \cdots + \mathbb{E}[X_n]$$

This holds even if the variables are dependent.

Card Game Example *Problem:* n cards numbered 1 to n. Each turn pick a random card. Let X = number of turns until card 1 is picked.

Solution: Let $X_i = 1$ if the i-th turn picks card 1. Probability = $1/n$ per turn.

Using linearity and geometric series:

$$\mathbb{E}[X] = 1\cdot\frac{1}{n} + 2\cdot\frac{1}{n}\Big(1-\frac{1}{n}\Big) + \cdots = n$$

ii. Power Technique for Higher Moments

$$\mathbb{E}[X^k] = \sum_{x=0}^{n} x^k \cdot \Pr(X = x)$$

A coin is tossed n times. Let X = number of heads.

Each possible outcome $x \in \{0,1,2,\ldots,n\}$ has probability

$$\Pr(X = x) = 2^{-n}\binom{n}{x}.$$

The general formula for the k-th moment using the power technique is:

$$\mathbb{E}[X^k] = \sum_{x=0}^{n} x^k \cdot \Pr(X = x) = \sum_{x=0}^{n} x^k \cdot 2^{-n}\binom{n}{x}.$$

Example: For $n = 4$ and $k = 2$,

$$\mathbb{E}[X^2] = \sum_{x=0}^{4} x^2 \cdot 2^{-4}\binom{4}{x} = 5.$$

Interpretation: - x = number of good rounds - x^2 = number of ordered pairs of good rounds - x^k = number of ordered k-tuples of good rounds

So $\mathbb{E}[X^k]$ = expected number of good k-tuples:

$$\mathbb{E}[X^k] = \sum_{\text{all } k\text{-tuples } t} \Pr(t \text{ all good})$$

Coin Toss Example: $n = 4$, $k = 2$ - Total ordered pairs: $4^2 = 16$ - Tuples with repeated round: 4, probability $1/2$ each - Tuples with different rounds: 12, probability $(1/2)^2 = 1/4$ each

$$\mathbb{E}[X^2] = 4\cdot\frac{1}{2} + 12\cdot\frac{1}{4} = 2 + 3 = 5$$

iii. Joker Card Game Example

Problem: There are m cards, exactly 1 is a joker. A game has n rounds. In each round, a card is drawn, observed, and replaced. Let X = number of rounds where the joker is seen. Compute $\mathbb{E}[X^k]$.

Direct probability approach: Each round, $\Pr(\text{joker}) = p = 1/m$. Then $X \in \{0,1,\ldots,n\}$, and

$$\Pr(X = x) = \binom{n}{x}p^x(1-p)^{n-x}, \quad \text{so } \mathbb{E}[X^k] = \sum_{x=0}^{n} x^k\binom{n}{x}p^x(1-p)^{n-x}.$$

For large n, this is infeasible.

Power technique: Interpret x^k as the number of ordered tuples of length k, where each element corresponds to a "good round" (joker seen). Then

$$\mathbb{E}[X^k] = \sum_{\text{all } k\text{-tuples } t} \Pr(t \text{ all good}) = \sum_t p^{\#\text{distinct indices in } t}.$$

Let d = number of distinct indices in a tuple t. Then

$$\mathbb{E}[X^k] = \sum_{d=0}^{k}(\tfrac{1}{m})^d \cdot D_d,$$

where D_d = number of tuples of length k with d distinct indices.

DP computation: Define $dp[i][j]$ = number of tuples of length i with j distinct indices.

$$dp[i+1][j] += dp[i][j]\cdot j \qquad \text{(choose an existing index)}$$
$$dp[i+1][j+1] += dp[i][j]\cdot(n-j) \qquad \text{(choose a new index)}$$

Finally, the expected value is

$$\mathbb{E}[X^k] = \sum_{d=0}^{k} dp[k][d]\cdot p^d.$$

Complexity: $O(k^2)$, feasible even for $k \sim 5000$.

Stirling Number of Second Kind: Let d = number of distinct indices in a k-length tuple. - **Choose:** d distinct rounds from n rounds $\Rightarrow \binom{n}{d}$. - **Assign:** Distribute k positions in the tuple to d chosen rounds onto k positions (onto function) $\Rightarrow S(k,d)$ (Stirling number of the second kind). - **Permute:** Each of d rounds can be ordered $\Rightarrow d!$.

Number of k-tuples with d distinct indices:

$$\text{Tuples}_d = \binom{n}{d}\cdot S(k,d)\cdot d!$$

$$\mathbb{E}[X^k] = \sum_{d=0}^{k}\text{Tuples}_d \cdot p^d = \sum_{d=0}^{k}\binom{n}{d}S(k,d)d!\,p^d, \quad p = \frac{1}{m}.$$

Complexity: $O(k^2)$ using DP to compute $S(k,d)$, or $O(k \log k)$ using NTT for fixed k.

3.7 Burnside's Lemma (Orbit Counting Theorem)

Burnside's Lemma is used to count **distinct objects under symmetry** (e.g., rotations, reflections). Two objects that can be transformed into each other by a symmetry are considered the **same**.

Burnside's Lemma

If a symmetry group has N operations and $c(i)$ is the number of objects **fixed** (unchanged) by the i-th symmetry, then the number of distinct objects is: Distinct $= \frac{1}{N}\sum_{i=1}^{N} c(i)$

Necklaces / Cyclic Strings (Rotations Only)

For a string of length N using M colors, a rotation by i positions fixes exactly $c(i) = M^{\gcd(i,N)}$ because a rotation creates $\gcd(i,N)$ cycles, and each cycle must have the same color. Thus, Distinct $= \frac{1}{N}\sum_{i=1}^{N} M^{\gcd(i,N)}$ Here M denote in how many way a single unit can be colored. if a cube like shape with $k*k$ grid in each face is rotated then $M = C^{k*k}$

Divisor Grouping

Group all rotations that have the same gcd value. Let C_d be the number of integers i with $\gcd(i,N) = d$. Then: Distinct $= \frac{1}{N}\sum_{d|N} C_d\, M^d$

A number theory result gives: $C_d = \varphi\big(\frac{N}{d}\big)$

so the final fast formula is: $\boxed{\text{Distinct} = \frac{1}{N}\sum_{d|N}\varphi\left(\frac{N}{d}\right)M^d}$

3.8 Matrix Exponentiation

```
struct mat {
 ll a[3][3];
 mat() { mem(a, 0); }
 mat operator*(const mat &b) const {
        mat ret;
        rep(i, 3) rep(j, 3) rep(k, 3) ret.a[i][j] =
        add(ret.a[i][j], mult(a[i][k], b.a[k][j]));
        return ret; } };
mat power(mat a, ll b) {
 mat ret;
 rep(i, 3) ret.a[i][i] = 1;
 while (b) {
        if (b & 1) ret = ret * a;
        b >>= 1;
        a = a * a; }
 return ret; }
const int MOD = 998244353;
typedef vector<int> row;
typedef vector<row> matrix;
inline int add(const int &a, const int &b) {
  int c = a + b;
  if (c >= MOD) c -= MOD;
  return c; }
inline int mult(const int &a, const int &b) {
  return (long long)a * b % MOD; }
matrix operator+(const matrix &m1, const matrix &m2) {
  int r = m1.size();
  int c = m1.back().size();
  matrix ret(r, row(c));
  for (int i = 0; i < r; i++) {
 for (int j = 0; j < c; j++) {
   ret[i][j] = add(m1[i][j], m2[i][j]); } }
  return ret; }
matrix operator*(const matrix &m1, const int m2) {
  int r = m1.size();
  int c = m1.back().size();
  matrix ret(r, row(c));
  for (int i = 0; i < r; i++) {
 for (int j = 0; j < c; j++) {
   ret[i][j] = mult(m1[i][j], m2); } }
  return ret; }
matrix operator*(const matrix &m1, const matrix &m2) {
  int r = m1.size();
  int m = m1.back().size();
  int c = m2.back().size();
  matrix ret(r, row(c, 0));
  for (int i = 0; i < r; i++) {
 for (int k = 0; k < m; k++) {
   for (int j = 0; j < c; j++) {
     ret[i][j] = add(ret[i][j], mult(m1[i][k],
     m2[k][j]));}}}
  return ret;}
matrix one(int dim) {
  matrix ret(dim, row(dim, 0));
  for (int i = 0; i < dim; i++) {
 ret[i][i] = 1; }
  return ret; }
matrix operator^(const matrix &m, const int &e) {
  if (e == 0) return one(m.size());
  matrix sqrtm = m ^ (e / 2);
  matrix ret = sqrtm * sqrtm;
  if (e & 1) ret = ret * m;
  return ret; }
```

3.9 Inclusion Exclusion Principle

Usage:

$$|A_1 \cup A_2 \cup \cdots \cup A_n| = \sum_i |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \cdots + (-1)^{n+1} |A_1 \cap A_2 \cap \cdots \cap A_n|.$$

```
for (int mask = 1; mask < (1 << k); mask++) {
    int product = 1;
    int bits = 0;
    for (int i = 0; i < k; i++) {
        if (mask & (1 << i)) {
            product *= primes[i];
            bits++;
        }
    }
    result -= (bits % 2 == 1 ? 1 : -1) * count(a, b,
    product);
}
```

3.10 Lagrange interpolation

```
const int N = 3e5 + 9, mod = 1e9 + 7;
//p = first at least n + 1 points: a, a+d, ..., a+n*d of
the n degree polynomial, returns f(x)
mint Lagrange(const vector<mint> &p, mint x, mint a = 0,
mint d = 1) {
  int n = p.size() - 1;
  if (a == 0 and d == 1 and x.value <= n) return
  p[x.value];
  vector<mint> pref(n + 1, 1), suf(n + 1, 1);
  for (int i = 0; i < n; i++) pref[i + 1] = pref[i] * (x -
  (a + d * i));
  for (int i = n; i > 0; i--) suf[i - 1] = suf[i] * (x - (a
  + d * i));
  vector<mint> fact(n + 1, 1),finv(n + 1, 1);
  for (int i = 1; i <= n; i++) fact[i] = fact[i - 1] * d *
  i;
  finv[n] /= fact[n];
  for (int i = n; i >= 1; i--) finv[i - 1] = finv[i] * d *
  i;
  mint ans = 0;
  for (int i = 0; i <= n; i++) {
    mint tmp = p[i] * pref[i] * suf[i] * finv[i] *
    finv[n-i];
    if ((n - i) & 1) ans -= tmp;
    else ans += tmp;
  }
  return ans;
}
// int n, k; cin >> n >> k;
// vector<mint> p; mint sum = 0; p.push_back(0);
// for (int i = 1; i <= k + 1; i++) {
// sum += mint(i).pow(k);
//  p.push_back(sum);
//  }
// cout << Lagrange(p, n) << '\n';
```

3.11 FFT

```
const int N = 3e5 + 9;
const double PI = acos(-1);
struct base {
    double a, b;
    base(double a = 0, double b = 0): a(a), b(b) {}
    const base operator + (const base & c) const {
     return base(a + c.a, b + c.b); }
    const base operator - (const base & c) const {
     return base(a - c.a, b - c.b); }
    const base operator * (const base & c) const {
     return base(a * c.a - b * c.b, a * c.b + b * c.a); }
    };
void fft(vector < base > & p, bool inv = 0) {
    int n = p.size(), i = 0;
    for(int j = 1; j < n - 1; ++j) {
     for(int k = n >> 1; k > (i ^= k); k >>= 1);
     if(j < i) swap(p[i], p[j]); }
    for(int l = 1, m;
     (m = l << 1) <= n; l <<= 1) {
     double ang = 2 * PI / m;
   base wn = base(cos(ang), (inv ? 1. : -1.) * sin(ang)),
   w;
     for(int i = 0, j, k; i < n; i += m) {
      for(w = base(1, 0), j = i, k = i + l; j < k; ++j, w =
      w * wn) {
       base t = w * p[j + l];
       p[j + l] = p[j] - t;
       p[j] = p[j] + t; }}}
    if(inv) for(int i = 0; i < n; ++i) p[i].a /= n, p[i].b
    /= n;}
vector < long long > multiply(vector < int > & a, vector <
int > & b) {
    int n = a.size(), m = b.size(), t = n + m - 1, sz = 1;
    while(sz < t) sz <<= 1;
    vector < base > x(sz), y(sz), z(sz);
    for(int i = 0; i < sz; ++i) {
     x[i] = i < (int) a.size() ? base(a[i], 0) : base(0,
     0);
     y[i] = i < (int) b.size() ? base(b[i], 0) :
     base(0,0);}
    fft(x), fft(y);
    for(int i = 0; i < sz; ++i) z[i] = x[i] * y[i];
    fft(z, 1);
    vector < long long > ret(sz);
    for(int i = 0; i < sz; ++i) ret[i] = (long long)
    round(z[i].a);
    while((int) ret.size() > 1 && ret.back() == 0)
    ret.pop_back();
    return ret; }
long long ans[N];
int32_t main()
    int n, x;
    cin >> n >> x;
    int nw = 0;
    a[0]++;
    b[n]++;
    long long z = 0;
    for(int i = 1; i <= n; i++) {
     int k; cin >> k;
     nw += k < x;
     a[nw]++;
     b[-nw + n]++;
     z += c[nw] + !nw;
     c[nw]++; }
    auto res = multiply(a, b);
    for(int i = n + 1; i < res.size(); i++) {
     ans[i - n] += res[i]; }
    ans[0] = z;
    for(int i = 0; i <= n; i++) cout << ans[i] << ' ';
```

3.12 NTT

Usage: Polynomial multiplication in mod

```
const int N = 1 << 20;
const int mod = 998244353;
const int root = 3;
int lim, rev[N], w[N], wn[N], inv_lim;
void reduce(int &x) { x = (x + mod) % mod; }
/*  Calculate a^b % mod  */
int POW(int x, int y, int ans = 1) {
    while(y){
        if (y & 1) ans = ans * x % mod;
        x = x * x % mod; y >>= 1;
    } return ans;
}
void precompute(int len) {
    lim = wn[0] = 1; int s = -1;
    while (lim < len) lim <<= 1, ++s;
    for (int i = 0; i < lim; ++i) rev[i] = rev[i >> 1] >> 1
    | (i & 1) << s;
    const int g = POW(root, (mod - 1) / lim);
    inv_lim = POW(lim, mod - 2);
    for (int i = 1; i < lim; ++i) wn[i] = (wn[i - 1] * g) %
    mod;
}
void ntt(vector<int> &a, int typ) {
    for(int i = 0; i < lim; ++i) if(i < rev[i]) swap(a[i],
    a[rev[i]]);
    for(int i = 1; i < lim; i <<= 1){
        for (int j = 0, t = lim / i / 2; j < i; ++j) w[j] =
        wn[j * t];
        for (int j = 0; j < lim; j += i << 1){
            for (int k = 0; k < i; ++k){
                const int x = a[k + j], y = a[k + j + i] *
                w[k] % mod;
                reduce(a[k + j] += y - mod), reduce(a[k + j
                + i] = x - y);
            }
        }
    }
    if (!typ) {
        reverse(a.begin() + 1, a.begin() + lim);
        for (int i = 0; i < lim; ++i) a[i] = (long long)
        a[i] * inv_lim % mod;
    }
}
vector<int> multiply(vector<int> &f, vector<int> &g) {
    if (f.empty() or g.empty()) return {};
    int n = (int)f.size() + (int)g.size() - 1;
    if (n == 1) return {(int)((long long) f[0] * g[0] %
    mod)};
    precompute(n);
    vector<int> a = f, b = g;
    a.resize(lim); b.resize(lim);
    ntt(a, 1), ntt(b, 1);
    for (int i = 0; i < lim; ++i) a[i] = (long long) a[i] *
    b[i] % mod;
    ntt(a, 0);
    a.resize(n + 1);
    return a;
}
```

3.13 NTT any mod

Usage: Polynomial multiplication in any mod

```
const int N = 3e5 + 9, mod = 998244353;
struct base {
  double x, y;
  base() { x = y = 0; }
  base(double x, double y): x(x), y(y) { }
};
inline base operator + (base a, base b) { return base(a.x + b.x, a.y + b.y); }
inline base operator - (base a, base b) { return base(a.x - b.x, a.y - b.y); }
inline base operator * (base a, base b) { return base(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
inline base conj(base a) { return base(a.x, -a.y); }
int lim = 1;
vector<base> roots = {{0, 0}, {1, 0}};
vector<int> rev = {0, 1};
const double PI = acosl(-1.0);
void ensure_base(int p) {
  if(p <= lim) return;
  rev.resize(1 << p);
  for(int i = 0; i < (1 << p); i++) rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (p - 1));
  roots.resize(1 << p);
  while(lim < p) {
    double angle = 2 * PI / (1 << (lim + 1));
    for(int i = 1 << (lim - 1); i < (1 << lim); i++) {
      roots[i << 1] = roots[i];
      double angle_i = angle * (2 * i + 1 - (1 << lim));
      roots[(i << 1) + 1] = base(cos(angle_i),
      sin(angle_i));
    }
    lim++;
  }
}
void fft(vector<base> &a, int n = -1) {
  if(n == -1) n = a.size();
  assert((n & (n - 1)) == 0);
  int zeros = __builtin_ctz(n);
  ensure_base(zeros);
  int shift = lim - zeros;
  for(int i = 0; i < n; i++) if(i < (rev[i] >> shift))
  swap(a[i], a[rev[i] >> shift]);
  for(int k = 1; k < n; k <<= 1) {
    for(int i = 0; i < n; i += 2 * k) {
      for(int j = 0; j < k; j++) {
        base z = a[i + j + k] * roots[j + k];
        a[i + j + k] = a[i + j] - z;
        a[i + j] = a[i + j] + z;
      }
    }
  }
}
//eq = 0: 4 FFTs in total
//eq = 1: 3 FFTs in total
vector<int> multiply(vector<int> &a, vector<int> &b, int eq = 0) {
  int need = a.size() + b.size() - 1;
  int p = 0;
  while((1 << p) < need) p++;
  ensure_base(p);
  int sz = 1 << p;
  vector<base> A, B;
  if(sz > (int)A.size()) A.resize(sz);
  for(int i = 0; i < (int)a.size(); i++) {
    int x = (a[i] % mod + mod) % mod;
    A[i] = base(x & ((1 << 15) - 1), x >> 15);
  }
  fill(A.begin() + a.size(), A.begin() + sz, base{0, 0});
  fft(A, sz);
  if(sz > (int)B.size()) B.resize(sz);
  if(eq) copy(A.begin(), A.begin() + sz, B.begin());
  else {
    for(int i = 0; i < (int)b.size(); i++) {
      int x = (b[i] % mod + mod) % mod;
      B[i] = base(x & ((1 << 15) - 1), x >> 15);
    }
    fill(B.begin() + b.size(), B.begin() + sz, base{0, 0});
    fft(B, sz);
  }
  double ratio = 0.25 / sz;
  base r2(0, -1), r3(ratio, 0), r4(0, -ratio), r5(0, 1);
  for(int i = 0; i <= (sz >> 1); i++) {
    int j = (sz - i) & (sz - 1);
    base a1 = (A[i] + conj(A[j])), a2 = (A[i] - conj(A[j])) * r2;
    base b1 = (B[i] + conj(B[j])) * r3, b2 = (B[i] - conj(B[j])) * r4;
    if(i != j) {
      base c1 = (A[j] + conj(A[i])), c2 = (A[j] - conj(A[i])) * r2;
      base d1 = (B[j] + conj(B[i])) * r3, d2 = (B[j] - conj(B[i])) * r4;
      A[i] = c1 * d1 + c2 * d2 * r5;
      B[i] = c1 * d2 + c2 * d1;
    }
    A[j] = a1 * b1 + a2 * b2 * r5;
    B[j] = a1 * b2 + a2 * b1;
  }
  fft(A, sz); fft(B, sz);
  vector<int> res(need);
  for(int i = 0; i < need; i++) {
    long long aa = A[i].x + 0.5;
    long long bb = B[i].x + 0.5;
    long long cc = A[i].y + 0.5;
    res[i] = (aa + ((bb % mod) << 15) + ((cc % mod) << 30))%mod;
  }
  return res;
}
vector<int> pow(vector<int>& a, int p) {
  vector<int> res;
  res.emplace_back(1);
  while(p) {
    if(p & 1) res = multiply(res, a);
    a = multiply(a, a, 1);
    p >>= 1;
  }
  return res;
}
int main() {
  int n, k; cin >> n >> k;
  vector<int> a(10, 0);
  while(k--) {
    int m; cin >> m;
    a[m] = 1;
  }
  vector<int> ans = pow(a, n / 2);
  int res = 0;
  for(auto x: ans) res = (res + 1LL * x * x % mod) % mod;
  cout << res << '\n';
  return 0;
}
```

3.14 Polynomial

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1 << 20, mod = 998244353;
struct base {
  double x, y;
  base() { x = y = 0; }
  base(double x, double y): x(x), y(y) { }
};
inline base operator + (base a, base b) { return base(a.x + b.x, a.y + b.y); }
inline base operator - (base a, base b) { return base(a.x - b.x, a.y - b.y); }
inline base operator * (base a, base b) { return base(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
inline base conj(base a) { return base(a.x, -a.y); }

int lim = 1;
vector<base> roots = {{0, 0}, {1, 0}};
vector<int> rev = {0, 1};
const double PI = acosl(-1.0);
void ensure_base(int p) {
  if(p <= lim) return;
  rev.resize(1 << p);
  for(int i = 0; i < (1 << p); i++) rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (p - 1));
  roots.resize(1 << p);
  while(lim < p) {
    double angle = 2 * PI / (1 << (lim + 1));
    for(int i = 1 << (lim - 1); i < (1 << lim); i++) {
      roots[i << 1] = roots[i];
      double angle_i = angle * (2 * i + 1 - (1 << lim));
      roots[(i << 1) + 1] = base(cos(angle_i),
      sin(angle_i));
    }
    lim++;
  }
}
void fft(vector<base> &a, int n = -1) {
  if(n == -1) n = a.size();
  assert((n & (n - 1)) == 0);
  int zeros = __builtin_ctz(n);
  ensure_base(zeros);
  int shift = lim - zeros;
  for(int i = 0; i < n; i++) if(i < (rev[i] >> shift))
  swap(a[i], a[rev[i] >> shift]);
  for(int k = 1; k < n; k <<= 1) {
    for(int i = 0; i < n; i += 2 * k) {
      for(int j = 0; j < k; j++) {
        base z = a[i + j + k] * roots[j + k];
        a[i + j + k] = a[i + j] - z;
        a[i + j] = a[i + j] + z;
      }
    }
  }
}
//eq = 0: 4 FFTs in total
//eq = 1: 3 FFTs in total
vector<int> multiply(vector<int> &a, vector<int> &b, int eq = 0) {
  int need = a.size() + b.size() - 1;
  int p = 0;
  while((1 << p) < need) p++;
  ensure_base(p);
  int sz = 1 << p;
  vector<base> A, B;
  if(sz > (int)A.size()) A.resize(sz);
  for(int i = 0; i < (int)a.size(); i++) {
    int x = (a[i] % mod + mod) % mod;
    A[i] = base(x & ((1 << 15) - 1), x >> 15);
  }
  fill(A.begin() + a.size(), A.begin() + sz, base{0, 0});
  fft(A, sz);
  if(sz > (int)B.size()) B.resize(sz);
  if(eq) copy(A.begin(), A.begin() + sz, B.begin());
  else {
    for(int i = 0; i < (int)b.size(); i++) {
      int x = (b[i] % mod + mod) % mod;
      B[i] = base(x & ((1 << 15) - 1), x >> 15);
    }
    fill(B.begin() + b.size(), B.begin() + sz, base{0, 0});
    fft(B, sz);
  }
  double ratio = 0.25 / sz;
  base r2(0, -1), r3(ratio, 0), r4(0, -ratio), r5(0, 1);
  for(int i = 0; i <= (sz >> 1); i++) {
    int j = (sz - i) & (sz - 1);
    base a1 = (A[i] + conj(A[j])), a2 = (A[i] - conj(A[j])) * r2;
    base b1 = (B[i] + conj(B[j])) * r3, b2 = (B[i] - conj(B[j])) * r4;
    if(i != j) {
      base c1 = (A[j] + conj(A[i])), c2 = (A[j] - conj(A[i])) * r2;
      base d1 = (B[j] + conj(B[i])) * r3, d2 = (B[j] - conj(B[i])) * r4;
      A[i] = c1 * d1 + c2 * d2 * r5;
      B[i] = c1 * d2 + c2 * d1;
    }
    A[j] = a1 * b1 + a2 * b2 * r5;
    B[j] = a1 * b2 + a2 * b1;
  }
  fft(A, sz); fft(B, sz);
  vector<int> res(need);
  for(int i = 0; i < need; i++) {
    long long aa = A[i].x + 0.5;
    long long bb = B[i].x + 0.5;
    long long cc = A[i].y + 0.5;
    res[i] = (aa + ((bb % mod) << 15) + ((cc % mod) << 30))%mod;
  }
  return res;
}

template <int32_t MOD>
struct modint {
  int32_t value;
  modint() = default;
  modint(int32_t value_) : value(value_) {}
  inline modint<MOD> operator + (modint<MOD> other) const {
  int32_t c = this->value + other.value; return
  modint<MOD>(c >= MOD ? c - MOD : c); }
  inline modint<MOD> operator - (modint<MOD> other) const {
  int32_t c = this->value - other.value; return
  modint<MOD>(c < 0 ? c + MOD : c); }
  inline modint<MOD> operator * (modint<MOD> other) const {
  int32_t c = (int64_t)this->value * other.value % MOD;
  return modint<MOD>(c < 0 ? c + MOD : c); }
  inline modint<MOD> & operator += (modint<MOD> other) {
  this->value += other.value; if (this->value >= MOD)
  this->value -= MOD; return *this; }
  inline modint<MOD> & operator -= (modint<MOD> other) {
  this->value -= other.value; if (this->value < 0)
  this->value += MOD; return *this; }
  inline modint<MOD> & operator *= (modint<MOD> other) {
  this->value = (int64_t)this->value * other.value % MOD;
  if (this->value < 0) this->value += MOD; return *this; }
  inline modint<MOD> operator - () const { return
  modint<MOD>(this->value ? MOD - this->value : 0); }
```

```
  modint<MOD> pow(uint64_t k) const {
    modint<MOD> x = *this, y = 1;
    for (; k; k >>= 1) {
      if (k & 1) y *= x;
      x *= x;
    }
    return y;
  }
  modint sqrt() const {
    if (value == 0) return 0;
    if (MOD == 2) return 1;
    if (pow((MOD - 1) >> 1) == MOD - 1)  return 0; // does
    not exist, it should be -1, but kept as 0 for this
    program
    unsigned int Q = MOD - 1, M = 0, i;
    modint zQ; while (!(Q & 1)) Q >>= 1, M++;
    for (int z = 1;; z++) {
      if (modint(z).pow((MOD - 1) >> 1) == MOD - 1) {
        zQ = modint(z).pow(Q); break;
      }
    }
    modint t = pow(Q), R = pow((Q + 1) >> 1), r;
    while (true) {
      if (t == 1) { r = R; break; }
      for (i = 1; modint(t).pow(1 << i) != 1; i++);
      modint b = modint(zQ).pow(1 << (M - 1 - i));
      M = i, zQ = b * b, t = t * zQ, R = R * b;
    }
    return min(r, - r + MOD);
  }
  modint<MOD> inv() const { return pow(MOD - 2); }  // MOD
  must be a prime
  inline modint<MOD> operator /  (modint<MOD> other) const
  { return *this *  other.inv(); }
  inline modint<MOD> operator /= (modint<MOD> other)
  { return *this *= other.inv(); }
  inline bool operator == (modint<MOD> other) const {
  return value == other.value; }
  inline bool operator != (modint<MOD> other) const {
  return value != other.value; }
  inline bool operator < (modint<MOD> other) const { return
  value < other.value; }
  inline bool operator > (modint<MOD> other) const { return
  value > other.value; }
};
template <int32_t MOD> modint<MOD> operator * (int64_t
value, modint<MOD> n) { return modint<MOD>(value) * n; }
template <int32_t MOD> modint<MOD> operator * (int32_t
value, modint<MOD> n) { return modint<MOD>(value % MOD) *
n; }
template <int32_t MOD> ostream & operator << (ostream &
out, modint<MOD> n) { return out << n.value; }

using mint = modint<mod>;
struct poly {
  vector<mint> a;
  inline void normalize() {
    while((int)a.size() && a.back() == 0) a.pop_back();
  }
  template<class...Args> poly(Args...args): a(args...) { }
  poly(const initializer_list<mint> &x): a(x.begin(),
  x.end()) { }
  int size() const { return (int)a.size(); }
  inline mint coef(const int i) const { return (i <
  a.size() && i >= 0) ? a[i]: mint(0); }
  mint operator[](const int i) const { return (i < a.size()
  && i >= 0) ? a[i]: mint(0); } //Beware!! p[i] = k won't
  change the value of p.a[i]
  bool is_zero() const {
    for (int i = 0; i < size(); i++) if (a[i] != 0) return
    0;
    return 1;
  }
  poly operator + (const poly &x) const {
    int n = max(size(), x.size());
    vector<mint> ans(n);
    for(int i = 0; i < n; i++) ans[i] = coef(i) +
    x.coef(i);
    while ((int)ans.size() && ans.back() == 0)
    ans.pop_back();
    return ans;
  }
  poly operator - (const poly &x) const {
    int n = max(size(), x.size());
    vector<mint> ans(n);
    for(int i = 0; i < n; i++) ans[i] = coef(i) -
    x.coef(i);
    while ((int)ans.size() && ans.back() == 0)
    ans.pop_back();
    return ans;
  }
  poly operator * (const poly& b) const {
    if(is_zero() || b.is_zero()) return {};
    vector<int> A, B;
    for(auto x: a) A.push_back(x.value);
    for(auto x: b.a) B.push_back(x.value);
    auto res = multiply(A, B, (A == B));
    vector<mint> ans;
    for(auto x: res) ans.push_back(mint(x));
    while ((int)ans.size() && ans.back() == 0)
    ans.pop_back();
    return ans;
  }
  poly operator * (const mint& x) const {
    int n = size();
    vector<mint> ans(n);
    for(int i = 0; i < n; i++) ans[i] = a[i] * x;
    return ans;
  }
  poly operator / (const mint &x) const{ return (*this) *
  x.inv(); }
  poly& operator += (const poly &x) { return *this =
  (*this) + x; }
  poly& operator -= (const poly &x) { return *this =
  (*this) - x; }
  poly& operator *= (const poly &x) { return *this =
  (*this) * x; }
  poly& operator *= (const mint &x) { return *this =
  (*this) * x; }
  poly& operator /= (const mint &x) { return *this =
  (*this) / x; }
  poly mod_xk(int k) const { return {a.begin(), a.begin() +
  min(k, size())}; } //modulo by x^k
  poly mul_xk(int k) const { // multiply by x^k
    poly ans(*this);
    ans.a.insert(ans.a.begin(), k, 0);
    return ans;
  }
  poly div_xk(int k) const { // divide by x^k
    return vector<mint>(a.begin() + min(k, (int)a.size()),
    a.end());
  }
  poly substr(int l, int r) const { // return
  mod_xk(r).div_xk(l)
    l = min(l, size());
    r = min(r, size());
    return vector<mint>(a.begin() + l, a.begin() + r);
  }
  poly reverse_it(int n, bool rev = 0) const { // reverses
  and leaves only n terms
    poly ans(*this);
    if(rev) { // if rev = 1 then tail goes to head
      ans.a.resize(max(n, (int)ans.a.size()));
    }
    reverse(ans.a.begin(), ans.a.end());
    return ans.mod_xk(n);
  }
  poly differentiate() const {
    int n = size(); vector<mint> ans(n);
    for(int i = 1; i < size(); i++) ans[i - 1] = coef(i) *
    i;
    return ans;
  }
  poly integrate() const {
    int n = size(); vector<mint> ans(n + 1);
    for(int i = 0; i < size(); i++) ans[i + 1] = coef(i) /
    (i + 1);
    return ans;
  }
  poly inverse(int n) const {  // 1 / p(x) % x^n, O(nlogn)
    assert(!is_zero()); assert(a[0] != 0);
    poly ans{mint(1) / a[0]};
    for(int i = 1; i < n; i *= 2) {
        ans = (ans * mint(2) - ans * ans * mod_xk(2 *
        i)).mod_xk(2 * i);
    }
    return ans.mod_xk(n);
  }
  poly log(int n) const { //ln p(x) mod x^n
    assert(a[0] == 1);
    return (differentiate().mod_xk(n) *
    inverse(n)).integrate().mod_xk(n);
  }
  poly exp(int n) const { //e ^ p(x) mod x^n
    if(is_zero()) return {1};
    assert(a[0] == 0);
    poly ans({1});
    int i = 1;
    while(i < n) {
      poly C = ans.log(2 * i).div_xk(i) - substr(i, 2 * i);
      ans -= (ans * C).mod_xk(i).mul_xk(i);
      i *= 2;
    }
    return ans.mod_xk(n);
  }
};

struct Combi{
  int n; vector<mint> facts, finvs, invs;
  Combi(int _n): n(_n), facts(_n), finvs(_n), invs(_n){
    facts[0] = finvs[0] = 1;
    invs[1] = 1;
      for (int i = 2; i < n; i++) invs[i] =  invs[mod % i]
      * (-mod / i);
    for(int i = 1; i < n; i++){
      facts[i] = facts[i - 1] * i;
      finvs[i] = finvs[i - 1] * invs[i];
    }
  }
  inline mint fact(int n) { return facts[n]; }
  inline mint finv(int n) { return finvs[n]; }
  inline mint inv(int n) { return invs[n]; }
  inline mint ncr(int n, int k) { return n < k ? 0 :
  facts[n] * finvs[k] * finvs[n-k]; }
};
Combi C(N);
// mul (1 - ix)
void yo(int l, int r, poly &ans){
  if (l == r) {
    ans = poly({1, mod - l});
    return;
  }
  int mid = (l + r) >> 1;
  poly a, b;
  yo(l ,mid, a);
  yo(mid + 1, r, b);
  ans = a * b;
}
// stirling2nd(i, k) for 0 <= i <= n
// O(n log^2 n)
vector<mint> stirling(int n, int k) {
  poly p;
  yo(1, k, p);
  p = p.inverse(n + 2);
  auto ans = p.a;
  ans.insert(ans.begin(), k, 0);
  ans.resize(n + 2);
  return ans;
}

vector<mint> bell(int n) { // e^(e^x - 1)
  poly p(n + 1);
  mint f = 1;
  for (int i = 0; i <= n; i++) {
    p.a[i] = mint(1) / f;
    f *= i + 1;
  }
  p.a[0] -= 1;
  p = p.exp(n + 1);
  vector<mint> ans(n + 1);
  f = 1;
  for (int i = 0; i <= n; i++) {
    ans[i] = p[i] * f;
    f *= i + 1;
  }
  return ans;
}

// get (x)(x-1)(x-2)...(x-n+1) by build(0, n-1)
vector<int> build(int l, int r){
  if(l == r){
    return { (mod - (l % mod)) % mod, 1 }; // represents x
    - l modulo mod
  }

  int mid = (l + r) / 2;
  vector<int> left_poly = build(l, mid);
  vector<int> right_poly = build(mid + 1, r);
  return multiply(left_poly, right_poly);
}
// compute polynomial^power with polynomial multiplication
under degree limit
vector<int> poly_power_trunc(vector<int> &a, int p, int
dl){
 vector<int> res = {1};
 while(p > 0){
  if(p & 1){
    res = multiply_trunc(res, a, dl + 1);
    if((int)res.size() > dl + 1){
      res.resize(dl + 1); // Truncate to degree limit
    }
  }
  a = multiply_trunc(a, a, dl + 1);
  if((int)a.size() > dl + 1){
    a.resize(dl + 1); // Truncate to degree limit
  }
  p >>= 1;
 }
 return res;
}
```

3.15 Online NTT

```
void solve(int l, int r){
    if(l == r)return;
    int mid = l + r >> 1;
    solve(l, mid);vector<LL> a, b;
    for(int i=l;i<=mid;i++){a.PB(A[i]);}
    for(int i = 0; i < B.size() && i <= r - l;
    i++)b.push_back(B[i]);
    vector<LL> temp = FFT::anyMod(a, b);
    for(int i = mid + 1; i <= r && i - l < temp.size();
    i++){
        A[i] += temp[i - l];
        if(A[i] >= mod)A[i] -= mod;
    }
    solve(mid + 1, r);}
```

3.16 FWHT

Usage: AND, OR works for any modulo, XOR works for only prime. Size must be a power of two

Time Complexity: O(nlogn)

```
const ll mod = 998244353;
int add (int a, int b) {
  return a + b < mod? a + b: a + b - mod;
}
int sub (int a, int b) {
  return a - b >= 0? a - b: a - b + mod;
}
ll poww (ll a, ll p, ll mod){
  a %= mod;
  ll ret = 1;
  while (p){
    if (p & 1) {
      ret = ret * a % mod;
    }
    a = a * a % mod;
    p >>= 1;
  }
  return ret;
}
void fwht(vector<int> &a, int inv, int f) {
  int sz = a.size();
  for (int len = 1; 2 * len <= sz; len <<= 1) {
    for (int i = 0; i < sz; i += 2 * len) {
      for (int j = 0; j < len; j++) {
        int x = a[i + j];
        int y = a[i + j + len];
        if (f == 0) {
          if (!inv)  a[i + j] = y, a[i + j + len] = add(x,
          y);
          else  a[i + j] = sub(y, x), a[i + j + len] = x;
        }
        else if (f == 1) {
          if (!inv)  a[i + j + len] = add(x, y);
          else  a[i + j + len] = sub(y, x);
        }
        else {
          a[i + j] = add(x, y);
          a[i + j + len] = sub(x, y);
        }
      }
    }
  }
}
vector<int> mul(vector<int> a, vector<int> b, int f) { //
0:AND, 1:OR, 2:XOR
  int sz = a.size();
  fwht(a, 0, f);  fwht(b, 0, f);
  vector<int> c(sz);
  for (int i = 0; i < sz; ++i) {
    c[i] = 1ll * a[i] * b[i] % mod;
  }
  fwht(c, 1, f);
  if (f) {
    int sz_inv = poww(sz, mod - 2, mod);
    for (int i = 0; i < sz; ++i) {
      c[i] = 1ll * c[i] * sz_inv % mod;
    }
  }
  return c;
}
```

3.17 Gauss solving linear eqn

Usage: Given a system of n linear algebraic equations (SLAE) with m unknowns variables. You are asked to solve the system: to determine if it has no solution, exactly one solution or infinite number of solutions. And in case it has at least one solution, find any of them.

```
const int N = 3e5 + 9;
const double eps = 1e-9;
int Gauss(vector<vector<double>> a, vector<double> &ans) {
  int n = (int)a.size(), m = (int)a[0].size() - 1;
  vector<int> pos(m, -1);
  double det = 1; int rank = 0;
  for(int col = 0, row = 0; col < m && row < n; ++col) {
    int mx = row;
    for(int i = row; i < n; i++) if(fabs(a[i][col]) >
    fabs(a[mx][col])) mx = i;
    if(fabs(a[mx][col]) < eps) {det = 0; continue;}
    for(int i = col; i <= m; i++) swap(a[row][i],
    a[mx][i]);
    if (row != mx) det = -det;
    det *= a[row][col];
    pos[col] = row;
    for(int i = 0; i < n; i++) {
      if(i != row && fabs(a[i][col]) > eps) {
        double c = a[i][col] / a[row][col];
        for(int j = col; j <= m; j++) a[i][j] -= a[row][j]
        * c;
      } }
    ++row; ++rank; }
  ans.assign(m, 0);
  for(int i = 0; i < m; i++) {
    if(pos[i] != -1) ans[i] = a[pos[i]][m] / a[pos[i]][i];}
  for(int i = 0; i < n; i++) {
    double sum = 0;
    for(int j = 0; j < m; j++) sum += ans[j] * a[i][j];
    if(fabs(sum - a[i][m]) > eps) return -1; //no solution
    }
  for(int i = 0; i < m; i++) if(pos[i] == -1) return 2;
  //infinte solutions
  return 1; //unique solution
}}
```

3.18 Xor Basis vector

```
const int N = 3e5 + 9;
struct Basis {
  vector<int> a;
  void insert(int x) {
 for (auto &i: a) x = min(x, x ^ i);
 if (!x) return;
 for (auto &i: a) if ((i ^ x) < i) i ^= x;
 a.push_back(x);
 sort(a.begin(), a.end()); }
  bool can(int x) {
 for (auto &i: a) x = min(x, x ^ i);
 return !x; }
  int maxxor(int x = 0) {
 for (auto &i: a) x = max(x, x ^ i);
 return x; }
  int minxor(int x = 0) {
 for (auto &i: a) x = min(x, x ^ i);
 return x; }
  int kth(int k) { // 1st is 0
 int sz = (int)a.size();
 if (k > (1LL << sz)) return -1;
 k--; int ans = 0;
 for (int i = 0; i < sz; i++) if (k >> i & 1) ans ^= a[i];
 return ans; }
}t;
int ty, k; cin >> ty >> k;
if (ty == 1) t.insert(k);
else cout << t.kth(k) << '\n';
```

3.19 Extended GCD

Usage: While the Euclidean algorithm calculates only the greatest common divisor (GCD) of two integers x and y, the extended version also finds a way to represent GCD in terms of a and b, i.e. coefficients x and y for which:

$a.x + b.y = GCD(x, y)$

Now, we know Diophantine Equations.These are the polynomial equations for which integral solutions exist.Ex: 3x+7y=1 or $x^2 - y^2 = z^3$.

For CP, we only need to deal with ax+by=c . Here, solutions exist only if $gcd(a, b)$ divides c.

Now, how to find x and y if we have to find the value of x and y if we are given the equation or we can come in a situation where we need to solve this equation. ax+by=gcd(a,b)

$gcd(a, b) = gcd(b, a\%b)$

$gcd(b, a\%b) = bx_1 + (a\%b)y_1$

Now, a%b=a-(a/b)*b

From above, $ax + by = bx_1 + (a\%b)y_1$

$ax + by = bx_1 + (a - (a/b) * b)y_1$

$ax + by = ay_1 + b(x_1 - (a/b) * y_1)$

So, comparing the coefficients of a and b, x=y1 and y=x1-(a/b)*y1

```
int egcd(int a, int b, int &x, int &y) { if (a == 0) { x =
0; y = 1; return b; } int x1, y1; int d = egcd(b%a, a, x1,
y1); x = y1 - (b / a) * x1; y = x1; return d; }
bool find_any_solution(int a, int b, int c, int &x0, int
&y0, int &g) {
    g = egcd(abs(a), abs(b), x0, y0); if (c % g) return
    false; x0 *= c / g; y0 *= c / g; if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0; return true; }
```

3.20 Chinese Remainder Theorem

```
// egcd code
class ChineseRemainderTheorem{
    typedef long long vlong;
    typedef pair<vlong,vlong> pll;
    /** CRT Equations stored as pairs of vectors. See
    addEquation()*/
    vector<pll> equations;
public:
    void clear(){
        equations.clear(); }
    /** Add equation of the form x = r (mod m)*/
    void addEquation( vlong r, vlong m ){
        equations.push_back({r, m});}
    pll solve(){
        if (equations.size() == 0)
            return {-1,-1}; /// No
        equations to solve
        vlong a1 = equations[0].first;
        vlong m1 = equations[0].second;
        a1 %= m1;
        /** Initially x = a_0 (mod m_0)*/
        /** Merge the solution with remaining equations */
        for ( int i = 1; i < equations.size(); i++ ){
                vlong a2 = equations[i].first;
                vlong m2 = equations[i].second;
                vlong g = __gcd(m1, m2);
                if ( a1 % g != a2 % g )
                    return {-1,-1}; ///Conflict in
                    equations
                /** Merge the two equations*/
                vlong p, q;
                ext_gcd(m1/g, m2/g, &p, &q);
                vlong mod = m1 / g * m2;
                vlong x = ( (__int128)a1 * (m2/g) % mod *q
                % mod + (__int128)a2 * (m1/g) % mod * p %
                mod ) % mod;
                /** Merged equation*/
                a1 = x;
                if ( a1 < 0 )
                    a1 += mod;
                m1 = mod; }
        return {a1, m1};}}
```

3.21 Euler Totient Function

```
/*  O(sqrt(n)) */
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}
/*  O(nloglogn) */
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

3.22 Mobius

```
int mobius[M];
bool sv[M+3];
void cal_mobius()
{
  sv[0]=true; sv[1]=true;
  for(int i=0;i<M;i++) mobius[i]=1;
    mobius[0]=0;mobius[1]=0;
  mobius[2]=-1;
  for(int i=4;i<M;i+=2){
    sv[i]=true;
    mobius[i]*=(i%4)? -1:0;
  }
  for(int i=3;i<M;i+=2){
    if(!sv[i]){
      mobius[i]=-1;
      for(int j=2*i;j<M;j+=i){
        sv[j]=true;
        mobius[j]*=(j%(i*i))? -1:0;
      }
    }
  }
    return;
}
```

3.23 Pollard Rho O(n^1/4)

Usage: `is_prime` For primarily test, `factor` for factorization ($n < 2^{62}$)

```
namespace PollardRho{
Mt19937
rnd(chrono::steady_clock::now().time_since_epoch().count());
  const int P = 1e6 + 9;
  ll seq[P];
  int primes[P], spf[P];
  inline ll add_mod(ll x, ll y, ll m){return (x+=y)<m?x:x -
  m; }
  inline ll mul_mod(ll x, ll y, ll m){
 ll res = __int128(x) *y % m;
 return res;
    //ll res = x *y - (ll)((long double)x *y / m + 0.5) *m;
    // return res < 0 ? res + m : res;}
  inline ll pow_mod(ll x, ll n, ll m){
 ll res = 1 % m;
 for (; n; n >>= 1){
    if (n & 1) res = mul_mod(res, x, m);
    x = mul_mod(x, x, m); }
 return res;}
// O(it *(logn)^3), it = number of rounds performed
  inline bool miller_rabin(ll n) {
 if (n <= 2 || (n &1 ^ 1)) return (n == 2);
 if (n < P) return spf[n] == n;
 ll c, d, s = 0, r = n - 1;
 for (; !(r & 1); r >>= 1, s++) {}
    // each iteration is a round
 for (int i = 0; primes[i] < n && primes[i] < 32; i++){
    c = pow_mod(primes[i], r, n);
    for (int j = 0; j < s; j++){
      d = mul_mod(c, c, n);
     if (d == 1 && c != 1 && c != (n - 1)) return false;
     c = d;}
    if (c != 1) return false;}
 return true; }
  void init(){
 int cnt = 0;
 for (int i = 2; i < P; i++){
   if (!spf[i]) primes[cnt++] = spf[i] = i;
   for (int j = 0, k; (k = i *primes[j]) < P; j++){
     spf[k] = primes[j];
     if (spf[i] == spf[k]) break; }}}
 ll pollard_rho(ll n){ // returns O(n^(1/4))
 while (1){
   ll x = rnd() % n, y = x, c = rnd() % n, u = 1, v, t = 0;
   ll *px = seq, *py = seq;
   while (1){
     *py++ = y = add_mod(mul_mod(y, y, n), c, n); *py++ = y
     = add_mod(mul_mod(y, y, n), c, n);
     if ((x = *px++) == y) break;
     v = u;
     u = mul_mod(u, abs(y - x), n);
     if (!u) return __gcd(v, n);
     if (++t == 32){
       t = 0;
       if ((u = __gcd(u, n)) > 1 && u < n) return u;}}
    if (t && (u = __gcd(u, n)) > 1 && u < n) return u; }}
 vector<ll> factorize(ll n){
 if (n == 1) return vector<ll> ();
 if (miller_rabin(n)) return vector < ll> { n };
 vector<ll> v, w;
 while (n > 1 && n < P){
   v.push_back(spf[n]);
   n /= spf[n];}
 if (n >= P){
   ll x = pollard_rho(n);
   v = factorize(x);
   w = factorize(n / x);
   v.insert(v.end(), w.begin(), w.end());}
 return v;}}
# auto f = PollardRho::factorize(n);
sort(f.begin(), f.end());
```

3.24 Co-Primes

Usage: **If p is a prime number, then gcd$(p, q) = 1$ for all $1 \le q < p$. Therefore we have: $\phi(p) = p - 1$. If p is a prime number and $k \ge 1$, then there are exactly $\frac{p^k}{p}$ numbers between 1 and p^k that are divisible by p. Which gives us $\phi(p^k) = p^k - p^{k-1}$. If a and b are relatively prime, then: $\phi(ab) = \phi(a) \cdot \phi(b)$. In general, for not co-prime a and b, the equation $\phi(ab) = \phi(a) \cdot \phi(b) \cdot \frac{d}{\phi(d)}$ with $d =$ gcd(a, b) holds. Sum of coprimes of a number n is $n \cdot \phi(n)/2$.**

3.25 Divisors

Usage: Maximal number of divisors of any n-digit number: First 18 numbers: 4, 12, 32, 64, 128, 240, 448, 768, 1344, 2304, 4032, 6720, 10752, 17280, 26880, 41472, 64512, 103680

4 String Algorithm

4.1 Kmp

```
const int N = 3e5 + 9;
// returns the longest proper prefix array of pattern p
// where lps[i]=longest proper prefix which is also suffix
of p[0...i]
vector<int> build_lps(string p) {
  int sz = p.size();
  vector<int> lps;
  lps.assign(sz + 1, 0);
  int j = 0;
  lps[0] = 0;
  for(int i = 1; i < sz; i++) {
    while(j >= 0 && p[i] != p[j]) {
      if(j >= 1) j = lps[j - 1];
      else j = -1;
    }
    j++;
    lps[i] = j;
  }
  return lps;}
vector<int>ans;
// returns matches in vector ans in 0-indexed
void kmp(vector<int> lps, string s, string p) {
  int psz = p.size(), sz = s.size();
  int j = 0;
  for(int i = 0; i < sz; i++) {
    while(j >= 0 && p[j] != s[i])
      if(j >= 1) j = lps[j - 1];
      else j = -1;
    j++;
    if(j == psz) {
      j = lps[j - 1];
      // pattern found in string s at position i-psz+1
      ans.push_back(i - psz + 1);
    }
    // after each loop we have j=longest common suffix of
    s[0..i] which is also prefix of p
  }
}
int aut[N][26];
void compute_automaton(string s){
    s += '#';
    int n = (int)s.size();
    vector<int> pi = prefix_function(s);
    for (int i = 0; i < n; i++) {
        for (int c = 0; c < 26; c++) {
            if (i > 0 && 'a' + c != s[i]) aut[i][c] =
            aut[pi[i-1]][c];
            else aut[i][c] = i + ('a' + c == s[i]);} }}
```

4.2 Palindromic Tree

```
const int N = 3e5 + 9;
/*
-> diff(v) = len(v) - len(link(v))
-> series link will lead from the vertex v to the vertex u
corresponding
   to the maximum suffix palindrome of v which satisfies
   diff(v) != diff(u)
-> path within series links to the root contains only O(log
n) vertices
-> cnt contains the number of palindromic suffixes of the
node
*/
struct PalindromicTree {
  struct node {
    int nxt[26], len, st, en, link, diff, slink, cnt, oc;
  };
  string s;
  vector<node> t;
  int sz, last;
  PalindromicTree() {}
  PalindromicTree(string _s) {
    s = _s;
    int n = s.size();
    t.clear();
    t.resize(n + 9);
    sz = 2, last = 2;
    t[1].len = -1, t[1].link = 1;
    t[2].len = 0, t[2].link = 1;
    t[1].diff = t[2].diff = 0;
    t[1].slink = 1;
    t[2].slink = 2;
  }
  int extend(int pos) { // returns 1 if it creates a new
  palindrome
    int cur = last, curlen = 0;
    int ch = s[pos] - 'a';
    while (1) {
      curlen = t[cur].len;
      if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] ==
      s[pos]) break;
      cur = t[cur].link;
    }
    if (t[cur].nxt[ch]) {
      last = t[cur].nxt[ch];
      t[last].oc++;
      return 0;
    }
    sz++;
    last = sz;
    t[sz].oc = 1;
    t[sz].len = t[cur].len + 2;
    t[cur].nxt[ch] = sz;
    t[sz].en = pos;
    t[sz].st = pos - t[sz].len + 1;
    if (t[sz].len == 1) {
      t[sz].link = 2;
      t[sz].cnt = 1;
      t[sz].diff = 1;
      t[sz].slink = 2;
      return 1;
    }
    while (1) {
      cur = t[cur].link;
      curlen = t[cur].len;
      if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] ==
      s[pos]) {
        t[sz].link = t[cur].nxt[ch];
        break;
      }
    }
    t[sz].cnt = 1 + t[t[sz].link].cnt;
    t[sz].diff = t[sz].len - t[t[sz].link].len;
    if (t[sz].diff == t[t[sz].link].diff) t[sz].slink =
    t[t[sz].link].slink;
    else t[sz].slink = t[sz].link;
    return 1;
  }
  void calc_occurrences() {
    for (int i = sz; i >= 3; i--) t[t[i].link].oc +=
    t[i].oc;
  }
  vector<array<int, 2>> minimum_partition() { //(even,
  odd), 1 indexed
    int n = s.size();
    vector<array<int, 2>> ans(n + 1, {0, 0}), series_ans(n
    + 5, {0, 0});
    ans[0][1] = series_ans[2][1] = 1e9;
    for (int i = 1; i <= n; i++) {
      extend(i - 1);
      for (int k = 0; k < 2; k++) {
        ans[i][k] = 1e9;
        for (int v = last; t[v].len > 0; v = t[v].slink) {
          series_ans[v][!k] = ans[i - (t[t[v].slink].len +
          t[v].diff)][!k];
          if (t[v].diff == t[t[v].link].diff)
          series_ans[v][!k] = min(series_ans[v][!k],
          series_ans[t[v].link][!k]);
```

```
            ans[i][k] = min(ans[i][k], series_ans[v][!k] +
            1);
          }
        }
      }
      return ans;
    }
} t;
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  string s;
  cin >> s;
  PalindromicTree t(s);
  for (int i = 0; i < s.size(); i++) t.extend(i);
  t.calc_occurrences();
  long long ans = 0;
  for (int i = 3; i <= t.sz; i++) ans += t.t[i].oc;
  cout << ans << '\n';
  return 0;
  //auto ans = t.minimum_partition();
  // for (int i = 1; i <= s.size(); i++) {
  //     cout << (ans[i][1] == 1e9 ? -1 : ans[i][1]) << '
  ';
  //     cout << (ans[i][0] == 1e9 ? -2 : ans[i][0]) <<
  '\n';
  // }
  return 0;
}
```

4.3 Manacher

Usage: Returns palindromic radius of S. To calculate even length palindromes, insert $ between each character.

Time Complexity: $\mathcal{O}(N)$

```
string s;
  cin >> s;
  n = s.size();
  vector<int> d1(n);  // maximum odd length palindrome
  centered at i
  // here d1[i]=the palindrome has d1[i]-1 right characters
  from i
  // e.g. for aba, d1[1]=2;
  for (int i = 0, l = 0, r = -1; i < n; i++) {
    int k = (i > r) ? 1 : min(d1[l + r - i], r - i);
    while (0 <= i - k && i + k < n && s[i - k] == s[i + k])
    {
      k++;
    }
    d1[i] = k--;
    if (i + k > r) {
      l = i - k;
      r = i + k;
    }
  }
  vector<int> d2(n);  // maximum even length palindrome
  centered at i
  // here d2[i]=the palindrome has d2[i]-1 right characters
  from i
  // e.g. for abba, d2[2]=2;
  for (int i = 0, l = 0, r = -1; i < n; i++) {
    int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i +
    1);
    while (0 <= i - k - 1 && i + k < n && s[i - k - 1] ==
    s[i + k]) {
      k++;
    }
    d2[i] = k--;
    if (i + k > r) {
      l = i - k - 1;
      r = i + k ;
    }
  }
int get(int l, int r) {
  return r * (r + 1) / 2 - (l - 1) * l / 2;
}
wavelet_tree oddl, oddr;
int odd(int l, int r) {
  int m = (l + r) / 2;
  int c = l - l;
  int less_ = oddl.LTE(l, m, c);
  int ansl = get(l, m) + oddl.sum(l, m, c) + (m - l + 1 -
  less_) * c;
  c = l + r;
  less_ = oddr.LTE(m + 1, r, c);
  int ansr = -get(m + 1, r) + oddr.sum(m + 1, r, c) + (r -
  m - less_) * c;
  return ansl + ansr;
}
wavelet_tree evenl, evenr;
int even(int l, int r) {
  int m = (l + r) / 2;
  int c = -l;
  int less_ = evenl.LTE(l, m, c);
  int ansl = get(l, m) + evenl.sum(l, m, c) + (m - l + 1 -
  less_) * c;
  c = l + r;
  less_ = evenr.LTE(m + 1, r, c);
  int ansr = -get(m + 1, r) + evenr.sum(m + 1, r, c) + (r -
  m - less_) * c;
  return ansl + ansr;
}
int a[N], b[N], c[N], d[N];
for(i = 1; i <= n; i++) a[i] = d1[i - 1] - i;
  oddl.init(a + 1, a + n + 1, -MAXV, MAXV);
  for(i = 1; i <= n; i++) b[i] = d1[i - 1] + i;
  oddr.init(b + 1, b + n + 1, -MAXV, MAXV);
  for(i = 1; i <= n; i++) c[i] = d2[i - 1] - i;
  evenl.init(c + 1, c + n + 1, -MAXV, MAXV);
  for(i = 1; i <= n; i++) d[i] = d2[i - 1] + i;
  evenr.init(d + 1, d + n + 1, -MAXV, MAXV);
```

4.4 Hashing

For Multiset Hashing:

Way 1: choosing a random number r

$$\text{Hash} = (r + a_1) \times (r + a_2) \times (r + a_3) \ldots\ldots$$

Complexity: $O(n)$

Way 2: choosing a random base B

$$\text{Hash} = B^{a_1} + B^{a_2} + B^{a_3} + B^{a_4} \ldots$$

Complexity: $O(n \log n)$

For Rooted Tree Isomerism:

Way 1: Using map of vector Hash of a node will be a sorted vector which has all the hashes of its children. We will map every vector to a number.

Way 2: For every height of a tree we will assign a random number x except for leaf nodes whose hash will be 1. Then it will be product of $(r + \text{hash(child)})$. Multiplying by p^i gives:

$$\text{hash}(s[i \ldots j]) \cdot p^i = \sum_{k=i}^{j} s[k] \cdot p^k \mod m$$

$$= \text{hash}(s[0 \ldots j]) - \text{hash}(s[0 \ldots i-1]) \mod m$$

4.5 Rolling Hash

```
// define mods and calculate powers and inverse_powers...
const int MOD1 = 127657753, MOD2 = 987654319;
const int p1 = 137, p2 = 277;
int ip1, ip2;
pair<int, int> pw[N], ipw[N];
void prec() {
    pw[0] =  {1, 1};
    for (int i = 1; i < N; i++) {
        pw[i].first = 1LL * pw[i - 1].first * p1 % MOD1;
        pw[i].second = 1LL * pw[i - 1].second * p2 % MOD2;
    }
    ip1 = power(p1, MOD1 - 2, MOD1); // write pow function
    ip2 = power(p2, MOD2 - 2, MOD2);
    ipw[0] =  {1, 1};
    for (int i = 1; i < N; i++) {
        ipw[i].first = 1LL * ipw[i - 1].first * ip1 % MOD1;
        ipw[i].second = 1LL * ipw[i - 1].second * ip2 %
        MOD2;
    }
}
struct Hashing {
    int n;
    string s; // 0 - indexed
    vector<pair<int, int>> hs; // 1 - indexed
    Hashing() {}
    Hashing(string _s) {
        n = _s.size();
        s = _s;
        hs.emplace_back(0, 0);
        for (int i = 0; i < n; i++) {
            pair<int, int> p;
            p.first = (hs[i].first + 1LL * pw[i].first *
            s[i] % MOD1) % MOD1;
            p.second = (hs[i].second + 1LL * pw[i].second *
            s[i] % MOD2) % MOD2;
            hs.push_back(p);
        }
    }
    // get hash from (l...r)
    pair<int, int> get_hash(int l, int r) { // 1 - indexed
        assert(1 <= l && l <= r && r <= n);
        pair<int, int> ans;
        ans.first = (hs[r].first - hs[l - 1].first + MOD1)
        * 1LL * ipw[l - 1].first % MOD1;
        ans.second = (hs[r].second - hs[l - 1].second +
        MOD2) * 1LL * ipw[l - 1].second % MOD2;
        return ans;
    }
    // get hash from (1...n)
    pair<int, int> get_hash() {
        return get_hash(1, n);
    }
};
```

4.6 Hash Table

```
mt19937
rng((uint32_t)chrono::steady_clock::now().time_since_epoch().coun
struct chash {  /// use most bits rather than just the
lowest ones
 const uint64_t C = (long long int)(2e18 * acos((long
 double)-1)) + 71;  // large odd number
 const int RANDOM = rng();
 long long int
 operator()(long long int x) const {
  return __builtin_bswap64((x ^ RANDOM) * C);
 }
};
template <class K, class V> using ht = gp_hash_table<K, V,
chash>;
ht<int, pair<int,int>> hm; // use
```

4.7 Suffix array

```
const int N = 3e5 + 9;
const int LG = 18;
void induced_sort(const vector<int> &vec, int val_range,
vector<int> &SA, const vector<bool> &sl, const vector<int>
&lms_idx) {
  vector<int> l(val_range, 0), r(val_range, 0);
  for (int c : vec) {
    if (c + 1 < val_range) ++l[c + 1];
    ++r[c];
  }
  partial_sum(l.begin(), l.end(), l.begin());
  partial_sum(r.begin(), r.end(), r.begin());
  fill(SA.begin(), SA.end(), -1);
  for (int i = lms_idx.size() - 1; i >= 0; --i)
    SA[--r[vec[lms_idx[i]]]] = lms_idx[i];
  for (int i : SA)
    if (i >= 1 && sl[i - 1]) {
      SA[l[vec[i - 1]]++] = i - 1;
    }
  fill(r.begin(), r.end(), 0);
  for (int c : vec)
    ++r[c];
  partial_sum(r.begin(), r.end(), r.begin());
  for (int k = SA.size() - 1, i = SA[k]; k >= 1; --k, i =
  SA[k])
    if (i >= 1 && !sl[i - 1]) {
      SA[--r[vec[i - 1]]] = i - 1;
    }
}
vector<int> SA_IS(const vector<int> &vec, int val_range) {
  const int n = vec.size();
  vector<int> SA(n), lms_idx;
  vector<bool> sl(n);
  sl[n - 1] = false;
  for (int i = n - 2; i >= 0; --i) {
    sl[i] = (vec[i] > vec[i + 1] || (vec[i] == vec[i + 1]
    && sl[i + 1]));
    if (sl[i] && !sl[i + 1]) lms_idx.push_back(i + 1);
  }
  reverse(lms_idx.begin(), lms_idx.end());
  induced_sort(vec, val_range, SA, sl, lms_idx);
  vector<int> new_lms_idx(lms_idx.size()),
  lms_vec(lms_idx.size());
  for (int i = 0, k = 0; i < n; ++i)
    if (!sl[SA[i]] && SA[i] >= 1 && sl[SA[i] - 1]) {
      new_lms_idx[k++] = SA[i];
    }
  int cur = 0;
  SA[n - 1] = cur;
  for (size_t k = 1; k < new_lms_idx.size(); ++k) {
    int i = new_lms_idx[k - 1], j = new_lms_idx[k];
    if (vec[i] != vec[j]) {
      SA[j] = ++cur;
      continue;
    }
    bool flag = false;
    for (int a = i + 1, b = j + 1;; ++a, ++b) {
      if (vec[a] != vec[b]) {
```

```
        flag = true;
        break;
      }
      if ((!sl[a] && sl[a - 1]) || (!sl[b] && sl[b - 1])) {
        flag = !((!sl[a] && sl[a - 1]) && (!sl[b] && sl[b - 1]));
        break;
      }
    }
    SA[j] = (flag ? ++cur : cur);
  }
  for (size_t i = 0; i < lms_idx.size(); ++i)
    lms_vec[i] = SA[lms_idx[i]];
  if (cur + 1 < (int)lms_idx.size()) {
    auto lms_SA = SA_IS(lms_vec, cur + 1);
    for (size_t i = 0; i < lms_idx.size(); ++i) {
      new_lms_idx[i] = lms_idx[lms_SA[i]];
    }
  }
  induced_sort(vec, val_range, SA, sl, new_lms_idx);
  return SA;
}
vector<int> suffix_array(const string &s, const int LIM = 128) {
  vector<int> vec(s.size() + 1);
  copy(begin(s), end(s), begin(vec));
  vec.back() = '$';
  auto ret = SA_IS(vec, LIM);
  ret.erase(ret.begin());
  return ret;
}
struct SuffixArray {
  int n;
  string s;
  vector<int> sa, rank, lcp;
  vector<vector<int>> t;
  vector<int> lg;
  SuffixArray() {}
  SuffixArray(string _s) {
    n = _s.size();
    s = _s;
    sa = suffix_array(s);
    rank.resize(n);
    for (int i = 0; i < n; i++) rank[sa[i]] = i;
    costruct_lcp();
    prec();
    build();
  }
  void costruct_lcp() {
    int k = 0;
    lcp.resize(n - 1, 0);
    for (int i = 0; i < n; i++) {
      if (rank[i] == n - 1) {
        k = 0;
        continue;
      }
      int j = sa[rank[i] + 1];
      while (i + k < n && j + k < n && s[i + k] == s[j + k])  k++;
      lcp[rank[i]] = k;
      if (k)  k--;
    }
  }
  void prec() {
    lg.resize(n, 0);
    for (int i = 2; i < n; i++) lg[i] = lg[i / 2] + 1;
  }
  void build() {
    int sz = n - 1;
    t.resize(sz);
    for (int i = 0; i < sz; i++) {
      t[i].resize(LG);
      t[i][0] = lcp[i];
    }
    for (int k = 1; k < LG; ++k) {
      for (int i = 0; i + (1 << k) - 1 < sz; ++i) {
        t[i][k] = min(t[i][k - 1], t[i + (1 << (k - 1))][k - 1]);
      }
    }
  }
  int query(int l, int r) { // minimum of lcp[l], ..., lcp[r]
    int k = lg[r - l + 1];
    return min(t[l][k], t[r - (1 << k) + 1][k]);
  }
  int get_lcp(int i, int j) { // lcp of suffix starting from i and j
    return query(i, j - 1);
    if (i == j) return n - i;
    int l = rank[i], r = rank[j];
    if (l > r) swap(l, r);
    return query(l, r - 1);
  }
  int lower_bound(string &t) {
    int l = 0, r = n - 1, k = t.size(), ans = n;
    while (l <= r) {
      int mid = l + r >> 1;
      if (s.substr(sa[mid], min(n - sa[mid], k)) >= t) ans = mid, r = mid - 1;
      else l = mid + 1;
    }
    return ans;
  }
  int upper_bound(string &t) {
    int l = 0, r = n - 1, k = t.size(), ans = n;
    while (l <= r) {
      int mid = l + r >> 1;
      if (s.substr(sa[mid], min(n - sa[mid], k)) > t) ans = mid, r = mid - 1;
      else l = mid + 1;
    }
    return ans;
  }
  // occurrences of s[p, ..., p + len - 1]
  pair<int, int> find_occurrence(int p, int len) {
    p = rank[p];
    pair<int, int> ans = {p, p};
    int l = 0, r = p - 1;
    while (l <= r) {
      int mid = l + r >> 1;
      if (query(mid, p - 1) >= len) ans.first = mid, r = mid - 1;
      else l = mid + 1;
    }
    l = p + 1, r = n - 1;
    while (l <= r) {
      int mid = l + r >> 1;
      if (query(p, mid - 1) >= len) ans.second = mid, l = mid + 1;
      else r = mid - 1;
    }
    return ans;
  }
};
```

4.8 Suffix array(short)

```
vector<int> sort_cyclic_shifts(string const& s) {
    int n = s.size();
    const int alphabet = 256;
    vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
    for (int i = 0; i < n; i++)
        cnt[s[i]]++;
    for (int i = 1; i < alphabet; i++)
        cnt[i] += cnt[i-1];
    for (int i = 0; i < n; i++)
        p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i-1]])
            classes++;
        c[p[i]] = classes - 1;
    }
     vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
            pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h)) % n]};
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
        if(classes==n)break;//jodi logn iteration er aghei sort hoye jay
    }
    return p;
}
vector<int> suffix_array_construction(string s) {
    s += "$";
    vector<int> sortedShift = sort_cyclic_shifts(s);
    sortedShift.erase(sortedShift.begin());
    return sortedShift;
}
//longest common prefix of string suffix
vector<int> lcp_construction(string const& s, vector<int> const& p) {
    int n = s.size();
    vector<int> rank(n, 0);
    for (int i = 0; i < n; i++)
        rank[p[i]] = i;
    int k = 0;
    vector<int> lcp(n-1, 0);
    for (int i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
        while (i + k < n && j + k < n && s[i+k] == s[j+k])
            k++;
        lcp[rank[i]] = k;
        if (k)
            k--;
    }
    return lcp;}
// Here We need to build sparse table of lcp array and write query function
int get_lcp(int i, int j) {// lcp of suffix starting from i and j
    if (i == j) return n - i;
    int l = rank[i], r = rank[j];
    if (l > r) swap(l, r);
    return query(l, r - 1);}
  int lower_bound(string &t) {
    int l = 0, r = n - 1, k = t.size(), ans = n;
    while (l <= r) {
      int mid = l + r >> 1;
      if (s.substr(sa[mid], min(n - sa[mid], k)) >= t) ans = mid, r = mid - 1;
      else l = mid + 1;
    }
    return ans;
  }
  int upper_bound(string &t) {
    int l = 0, r = n - 1, k = t.size(), ans = n;
    while (l <= r) {
      int mid = l + r >> 1;
      if (s.substr(sa[mid], min(n - sa[mid], k)) > t) ans = mid, r = mid - 1;
      else l = mid + 1;
    }
    return ans; }
  // occurrences of s[p, ..., p + len - 1]
  pair<int, int> find_occurrence(int p, int len) {
    p = rank[p];
    pair<int, int> ans = {p, p};
    int l = 0, r = p - 1;
    while (l <= r) {
      int mid = l + r >> 1;
      if (query(mid, p - 1) >= len) ans.first = mid, r = mid - 1;
      else l = mid + 1;
    }
    l = p + 1, r = n - 1;
    while (l <= r) {
      int mid = l + r >> 1;
      if (query(p, mid - 1) >= len) ans.second = mid, l = mid + 1;
      else r = mid - 1;
    }
    return ans;
  }
```

4.9 Aho-Corasick

Usage: `MAXC`: size of alphabet, `F`, `FG`: failure (parent), failure graph, `ftrans`: state transition function.

```
struct AC {
  int N, P;
  const int A = 26;
  vector <vector <int>> next;
  vector <int> link, out_link;
  vector <vector <int>> out;
  AC(): N(0), P(0) {node();}
  int node() {
    next.emplace_back(A, 0);
    link.emplace_back(0);
    out_link.emplace_back(0);
    out.emplace_back(0);
    return N++; }
```

```
  inline int get (char c) {
    return c - 'a'; }
  int add_pattern (const string T) {
    int u = 0;
    for (auto c : T) {
      if (!next[u][get(c)]) next[u][get(c)] = node();
      u = next[u][get(c)];
    }
    out[u].push_back(P);
    return P++; }
  void compute() {
    queue <int> q;
    for (q.push(0); !q.empty();) {
      int u = q.front(); q.pop();
      for (int c = 0; c < A; ++c) {
        int v = next[u][c];
        if (!v) next[u][c] = next[link[u]][c];
        else {
          link[v] = u ? next[link[u]][c] : 0;
          out_link[v] = out[link[v]].empty() ?
          out_link[link[v]] : link[v];
          q.push(v);
        } } }}
  int advance (int u, char c) {
    while (u && !next[u][get(c)]) u = link[u];
    u = next[u][get(c)];
    return u;
  }};
```

4.10 Suffix Automaton

Usage: search for all occurrences of one string in another, or count the amount of different substrings of a given string in linear time.

```
const int N = 3e5 + 9;
// len -> largest string length of the corresponding
endpos-equivalent class
// link -> longest suffix that is another endpos-equivalent
class.
// firstpos -> 1 indexed end position of the first
occurrence of the largest string of that node
// minlen(v) -> smallest string of node v = len(link(v)) +
1
// terminal nodes -> store the suffixes
struct SuffixAutomaton {
    struct node {
        int len, link, firstpos;
        map<char, int> nxt;
    };
    int sz, last;
    vector<node> t;
    vector<int> terminal;
    vector<long long> dp;
    vector<vector<int>> g;
    SuffixAutomaton() {}
    SuffixAutomaton(int n) {
        t.resize(2 * n); terminal.resize(2 * n, 0);
        dp.resize(2 * n, -1); sz = 1; last = 0;
        g.resize(2 * n);
        t[0].len = 0; t[0].link = -1; t[0].firstpos = 0;
    }
    void extend(char c) {
        int p = last;
        if (t[p].nxt.count(c)) {
            int q = t[p].nxt[c];
            if (t[q].len == t[p].len + 1) {
                last = q;
                return;
            }
            int clone = sz++;
            t[clone] = t[q];
            t[clone].len = t[p].len + 1;
            t[q].link = clone;
            last = clone;
            while (p != -1 && t[p].nxt[c] == q) {
                t[p].nxt[c] = clone;
                p = t[p].link;
            }
            return;
        }
        int cur = sz++;
        t[cur].len = t[last].len + 1;
        t[cur].firstpos = t[cur].len;
        p = last;
        while (p != -1 && !t[p].nxt.count(c)) {
            t[p].nxt[c] = cur;
            p = t[p].link;
        }
        if (p == -1) t[cur].link = 0;
        else {
            int q = t[p].nxt[c];
            if (t[p].len + 1 == t[q].len) t[cur].link = q;
            else {
                int clone = sz++;
                t[clone] = t[q];
                t[clone].len = t[p].len + 1;
                while (p != -1 && t[p].nxt[c] == q) {
                    t[p].nxt[c] = clone;
                    p = t[p].link;
                }
                t[q].link = t[cur].link = clone;
            }
        }
        last = cur;
    }
    void build_tree() {
        for (int i = 1; i < sz; i++)
        g[t[i].link].push_back(i);
    }
    void build(string &s) {
        for (auto x: s) {
            extend(x);
            terminal[last] = 1;
        }
        build_tree();
    }
    long long cnt(int i) { //number of times i-th node
    occurs in the string
        if (dp[i] != -1) return dp[i];
        long long ret = terminal[i];
        for (auto &x: g[i]) ret += cnt(x);
        return dp[i] = ret;
    }
};
int32_t main() {
    int t; cin >> t;
    while (t--) {
        string s; cin >> s;
        int n = s.size();
        SuffixAutomaton sa(n);
        sa.build(s);
        long long ans = 0; //number of unique substrings
        for (int i = 1; i < sa.sz; i++) ans += sa.t[i].len
        - sa.t[sa.t[i].link].len;
        cout << ans << '\n';}
}
```

4.11 String Matching Bitset

Time Complexity: $O(|P| \cdot \frac{|T|}{K})$

```
const int N = 1e5 + 9;
vector<int> v;
bitset<N>bs[26], oc;
int main() {
  int i, j, k, n, q, l, r;
  string s, p;
  cin >> s;
  for(i = 0; s[i]; i++) bs[s[i] - 'a'][i] = 1;
  cin >> q;
  while(q--) {
    cin >> p;
    oc.set();
    for(i = 0; p[i]; i++) oc &= (bs[p[i] - 'a'] >> i);
    cout << oc.count() << endl; // number of occurences
    int ans = N, sz = p.size();
    int pos = oc._Find_first();
    v.push_back(pos);
    pos = oc._Find_next(pos);
    while(pos < N) {
      v.push_back(pos);
      pos = oc._Find_next(pos);
    }
    for(auto x : v) cout << x << ' '; // position of
    occurences
    cout << endl;
    v.clear();
    cin >> l >> r; // number of occurences from l to
    r,where l and r is 1-indexed
    if(sz > r - l + 1) cout << 0 << endl;
    else cout << (oc >> (l - 1)).count() - (oc >> (r - sz +
    1)).count() << endl;
  }
  return 0;
}
```

5 Dynamic Programming

5.1 Knuth Optimization

```
const int N = 1010;
using ll = long long;
/* Knuths optimization works for optimization over sub
arrays
for which optimal middle point depends monotonously on the
end points.
Let mid[l,r] be the first middle point for (l,r) sub array
which gives optimal result.
It can be proven that mid[l,r-1] <= mid[l,r] <= mid[l+1,r]
- this means monotonicity of mid by l and r.
Applying this optimization reduces time complexity from
O(k^3) to O(k^2)
because with fixed s (sub array length) we have m_right(l)
= mid[l+1][r] = m_left(l+1).
That's why nested l and m loops require not more than 2k
iterations overall. */
int n, k;
int a[N], mid[N][N];
ll res[N][N];
ll solve() {
  for (int s = 0; s <= k; s++) {       // s - length of the
  subarray
 for (int l = 0; l + s <= k; l++) {  // l - left point
    int r = l + s;                    // r - right point
    if (s < 2) {
      res[l][r] = 0;            // base case- nothing to
      break
      mid[l][r] = l;            // mid is equal to left
      border
      continue;
    }
    int mleft = mid[l][r - 1];
    int mright = mid[l + 1][r];
    res[l][r] = 2e18;
    for (int m = mleft; m <= mright; m++) { // iterating for
    m in the bounds only
      ll tmp = res[l][m] + res[m][r] + (a[r] - a[l]);
      if (res[l][r] > tmp) {              // relax current
      solution
        res[l][r] = tmp;
        mid[l][r] = m;
      }}} }
  ll ans = res[0][k];
  return ans;
}
int main() {
  int i, j, m;
  while(cin >> n >> k) {
 for(i = 1; i <= k; i++) cin >> a[i];
 a[0] = 0;
 a[k + 1] = n;
 k++;
 cout << solve() << endl;
  }}
```

5.2 D&Q Dp

```
//Used to calculate Dp[i][j] = min (dp[i-1][k-1] + c[k][j])
// here k range from 0 to j
int m, n;
vector<long long> dp_before, dp_cur;
long long C(int i, int j);
// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;
    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};
    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {(k ? dp_before[k - 1] : 0) + C(k,
        mid), k});
    }
    dp_cur[mid] = best.first;
    int opt = best.second;
    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}
long long solve() {
    dp_before.assign(n,0);
    dp_cur.assign(n,0);
    for (int i = 0; i < n; i++)
        dp_before[i] = C(0, i);
    for (int i = 1; i < m; i++) {
        compute(0, n - 1, 0, n - 1);
        dp_before = dp_cur;
    }
    return dp_before[n - 1];
}
```

5.3 SOS Dp

Time Complexity: $\mathcal{O}(N.2^N)$

```
for(int i = 0; i<(1<<N); ++i)
 F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N);
++mask){
 if(mask & (1<<i)) // for supermask check unset
  F[mask] += F[mask^(1<<i)];
}
```

5.4 Digit Dp

```
vector<int>num;
// leftmost is the lsb and rightmost is msb
void makeDigit(int a){
    num.clear();
    n = a;
    while(a > 0){
        num.push_back(a % 10);
        a /= 10;
    }   // no reverse!!!
}
int recur(int i, bool f, int mask){
    if(i == -1){
        int cnt = __builtin_popcount(mask);
        int righmost = 31 - __builtin_clz(mask);
        if(cnt == righmost){
            return 1;
        } else {
            return 0;
        }
    }
    if(dp[i][mask] != -1 && f) return dp[i][mask]; // see
    the f!!!
    int ans = 0;
    int limit;
    if(f) limit = 9;
    else limit = num[i];
    for(int it=0;it<=limit;it++){
        bool nf = f;
        if(it < num[i]) nf = 1;
        int nmask;
        if(it == 0 and mask == 0) nmask = mask;
        else  nmask = mask | (1 << it);
        ans += recur(i-1, nf, nmask);
    }
    return (f ? dp[i][mask] = ans : ans);
}
int n; cin >> n; makeDigit(n);
cout << recur(num.size()-1, 0, 0) - 1 << endl;
memset(dp, -1, sizeof(dp)); // once...
```

5.5 MCM

```
const int mx = 100 + 9;
const ll inf = 9e18;
struct Matrix {
    int row, col;
    Matrix(int _row, int _col) {
        row = _row;
        col = _col;
    }
};
vector <Matrix> mats;
ll dp[mx][mx];
int mergeCost(int i, int j, int k) {
    return mats[i].row * mats[k].col * mats[j].col;
}
ll solve(int l, int r) {
    if (l >= r) return 0;
    ll& ans = dp[l][r];
    if (ans != -1) return ans;
    ans = inf;
    for (int i = l; i < r; i++) {
        ll left = solve(l, i);
        ll right = solve(i + 1, r);
        ll cost = mergeCost(l, r, i) + (left + right);
        ans = min(ans, cost);
    }
    return ans;
}
int evaluate(int i, int j) {
    if (i >= j) {
        return 0;
    }
    return dp[i][j];
}

ll solve() {
    int n = mats.size();
    for (int sz = 1; sz <= n; sz++) {
        for (int i = 0; i < n; i++) {
            int j = i + sz - 1;
            if (j >= n) break;
            ll ans = inf;
            for(int k = i; k < j; k++) {
                ll res_left = evaluate(i, k);
                ll res_right = evaluate(k + 1, j);
                ll cost = (res_left + res_right) +
                mergeCost(i, j, k);
                ans = min(ans, cost);
            }
            dp[i][j] = ans;
        }
    }
    return dp[0][n-1];
}
int main() {
    int n; cin >> n;
    for (int i = 0; i < n; i++) {
        int row, col;
        cin >> row >> col;
        mats.emplace_back(row, col);
    }
    //iterative version
    cout << solve() << '\n';
    //recursive version
    memset(dp, -1, sizeof dp);
    cout << solve(0, n - 1) << '\n';
}
```

5.6 CHT

```
struct CHT {
  vector<ll> m, b;
  int ptr = 0;
  bool bad(int l1, int l2, int l3) {
    return 1.0 * (b[l3] - b[l1]) * (m[l1] - m[l2])  <= 1.0
    * (b[l2] - b[l1]) * (m[l1] - m[l3]); //(slope dec+query
    min),(slope inc+query max)
    return 1.0 * (b[l3] - b[l1]) * (m[l1] - m[l2])  > 1.0 *
    (b[l2] - b[l1]) * (m[l1] - m[l3]); //(slope dec+query
    max), (slope inc+query min)
  }
  void add(ll _m, ll _b) {
    m.push_back(_m);
    b.push_back(_b);
    int s = m.size();
    while(s >= 3 && bad(s - 3, s - 2, s - 1)) {
      s--;
      m.erase(m.end() - 2);
      b.erase(b.end() - 2);
    }
  }
  ll f(int i, ll x) {
    return m[i] * x + b[i];
  }
  //(slope dec+query min), (slope inc+query max) -> x
  increasing
  //(slope dec+query max), (slope inc+query min) -> x
  decreasing
  ll query(ll x) {
    if(ptr >= m.size()) ptr = m.size() - 1;
    while(ptr < m.size() - 1 && f(ptr + 1, x) < f(ptr, x))
    ptr++;
    return f(ptr, x);
  }
  ll bs(int l, int r, ll x) {
    int mid = (l + r) / 2;
    if(mid + 1 < m.size() && f(mid + 1, x) < f(mid, x))
    return bs(mid + 1, r, x); // > for max
    if(mid - 1 >= 0 && f(mid - 1, x) < f(mid, x)) return
    bs(l, mid - 1, x); // > for max
    return f(mid, x);
  }
};
```

5.7 Dynamic CHT

```
//add lines with -m and -b and return -ans to
//make this code work for minimums.(not -x)
const ll inf = -(1LL << 62);
struct line {
  ll m, b;
  mutable function<const line*() > succ;
  bool operator < (const line& rhs) const {
    if (rhs.b != inf) return m < rhs.m;
    const line* s = succ();
    if (!s) return 0;
    ll x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};
struct CHT : public multiset<line> {
  bool bad(iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end()) return 0;
      return y -> m == z -> m && y -> b <= z -> b;
    }
    auto x = prev(y);
    if (z == end()) return y -> m == x -> m && y -> b <= x
    -> b;
    return 1.0 * (x -> b - y -> b) * (z -> m - y -> m) >=
    1.0 * (y -> b - z -> b) * (y -> m - x -> m);
  }
  void add(ll m, ll b) {
    auto y = insert({ m, b });
    y->succ = [=, this] { return next(y) == end() ? 0 :
    &*next(y); };
    if (bad(y)) {
      erase(y);
      return;
    }
    while (next(y) != end() && bad(next(y)))
    erase(next(y));
    while (y != begin() && bad(prev(y))) erase(prev(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound((line) {
      x, inf
    });
    return l.m * x + l.b;
  }
};
CHT* cht;
```

5.8 LiChao Tree

```
struct Line {
  ll m, c;
  ll eval(ll x) {
    return m * x + c;
  }
};
struct node {
  Line line;
  node* left = nullptr;
  node* right = nullptr;
  node(Line line) : line(line) {}
  void add_segment(Line nw, int l, int r, int L, int R) {
    if (l > r || r < L || l > R) return;
    int m = (l + 1 == r ? l : (l + r) / 2);
    if (l >= L and r <= R) {
      bool lef = nw.eval(l) < line.eval(l);
      bool mid = nw.eval(m) < line.eval(m);
      if (mid) swap(line, nw);
      if (l == r) return;
      if (lef != mid) {
        if (left == nullptr) left = new node(nw);
        else left -> add_segment(nw, l, m, L, R);
      }
      else {
        if (right == nullptr) right = new node(nw);
        else right -> add_segment(nw, m + 1, r, L, R);
      }
      return;
    }
    if (max(l, L) <= min(m, R)) {
      if (left == nullptr) left = new node({0, inf});
      left -> add_segment(nw, l, m, L, R);
    }
    if (max(m + 1, L) <= min(r, R)) {
      if (right == nullptr) right = new node ({0, inf});
      right -> add_segment(nw, m + 1, r, L, R);
    }
  }
  ll query_segment(ll x, int l, int r, int L, int R) {
    if (l > r || r < L || l > R) return inf;
    int m = (l + 1 == r ? l : (l + r) / 2);
    if (l >= L and r <= R) {
      ll ans = line.eval(x);
      if (l < r) {
        if (x <= m && left != nullptr) ans = min(ans, left
        -> query_segment(x, l, m, L, R));
        if (x > m && right != nullptr) ans = min(ans, right
        -> query_segment(x, m + 1, r, L, R));
      }
      return ans;
    }
    ll ans = inf;
    if (max(l, L) <= min(m, R)) {
      if (left == nullptr) left = new node({0, inf});
      ans = min(ans, left -> query_segment(x, l, m, L, R));
    }
    if (max(m + 1, L) <= min(r, R)) {
      if (right == nullptr) right = new node ({0, inf});
      ans = min(ans, right -> query_segment(x, m + 1, r, L,
      R));
    }
    return ans;
```

```
  }
};
struct LiChaoTree {
  int L, R;
  node* root;
  LiChaoTree() : L(numeric_limits<int>::min() / 2),
  R(numeric_limits<int>::max() / 2), root(nullptr) {}
  LiChaoTree(int L, int R) : L(L), R(R) {
    root = new node({0, inf});
  }
  void add_line(Line line) {
    root -> add_segment(line, L, R, L, R);
  }
  // y = mx + b: x in [l, r]
  void add_segment(Line line, int l, int r) {
    root -> add_segment(line, L, R, l, r);
  }
  ll query(ll x) {
    return root -> query_segment(x, L, R, L, R);
  }
  ll query_segment(ll x, int l, int r) {
    return root -> query_segment(x, l, r, L, R);
  }
};
```

6 Graph

6.1 Bridge and Articulation Point

```
struct TECC { // 0 indexed
  int n, k;
  vector<vector<int>> g, t;
  vector<bool> used;
  vector<int> comp, ord, low;
  using edge = pair<int, int>;
  vector<edge> br;
  void dfs(int x, int prv, int &c) {
    used[x] = 1; ord[x] = c++; low[x] = n;
    bool mul = 0;
    for (auto y:g[x]) {
      if (used[y]) {
        if (y != prv || mul) low[x] = min(low[x], ord[y]);
        else mul = 1;
//if(low[v] >= dis[u] && pre != 0) art[u] = 1;
        continue;}
      dfs(y, x, c);
      low[x] = min(low[x], low[y]);
    }
 //if(pre == 0 && child > 1) art[u] = 1;
 }
  void dfs2(int x, int num) {
    comp[x] = num;
    for (auto y: g[x]) {
      if (comp[y] != -1) continue;
      if (ord[x] < low[y]) {
        br.push_back({x, y});
        k++;
        dfs2(y, k);
      } else dfs2(y, num);
    }}
  TECC(const vector<vector<int>> &g): g(g), n(g.size()),
  used(n), comp(n, -1), ord(n), low(n), k(0) {
    int c = 0;
    for (int i = 0; i < n; i++) {
      if (used[i]) continue;
      dfs(i, -1, c);
      dfs2(i, k);
      k++;
    } }
  void build_tree() {
    t.resize(k);
    for (auto e: br) {
      int x = comp[e.first], y = comp[e.second];
      t[x].push_back(y);
      t[y].push_back(x); } }};
```

6.2 LCA

```
const int N= 1e5+5,LOG= 20;
int depth[N],up[N][LOG];
vector<int> v[N];
void dfs(int pos,int pre){
   for(auto it:v[pos]){
      if(it==pre) continue;
      depth[it]=depth[pos]+1;
      up[it][0] = pos;
      for(int j = 1; j < LOG; j++){
        up[it][j] = up[up[it][j-1]][j-1]; }
      dfs(it,pos); }
   return ;}
int kthancestor(int pos,int k){
    for(int i=LOG-1;i>=0;i--){
        if(k&(1<<i))
        pos=up[pos][i];}
    return pos;}
int get_lca(int a, int b) {
    if(depth[a] < depth[b]) {
        swap(a, b);}
    // 1) Get same depth.
    int k = depth[a] - depth[b];
    a=kthancestor(a,k);
    // 2) if b was ancestor of a then now a==b
    if(a == b) { return a; }
    // 3) move both a and b with powers of two
    for(int j = LOG - 1; j >= 0; j--) {
        if(up[a][j] != up[b][j]) {
            a = up[a][j];
            b = up[b][j]; } }
    return up[a][0]; }
```

6.3 Max Flow Dinic

```
const int N = 5010;
const long long inf = 1LL << 61;
struct Dinic {
  struct edge {
 int to, rev;
 long long flow, w;
 int id; };
  int n, s, t, mxid;
  vector<int> d, flow_through;
  vector<int> done;
  vector<vector<edge>> g;
  Dinic() {}
  Dinic(int _n) {
 n = _n + 10;
 mxid = 0;
 g.resize(n); }
  void add_edge(int u, int v, long long w, int id = -1) {
 edge a = {v, (int)g[v].size(), 0, w, id};
 edge b = {u, (int)g[u].size(), 0, 0, -2};//for
 bidirectional edges cap(b) = w
 g[u].emplace_back(a);
 g[v].emplace_back(b);
 mxid = max(mxid, id); }
  bool bfs() {
 d.assign(n, -1);
 d[s] = 0;
 queue<int> q;
 q.push(s);
 while (!q.empty()) {
   int u = q.front();
   q.pop();
   for (auto &e : g[u]) {
     int v = e.to;
     if (d[v] == -1 && e.flow < e.w) d[v] = d[u] + 1,
     q.push(v);
   }}
 return d[t] != -1; }
  long long dfs(int u, long long flow) {
 if (u == t) return flow;
 for (int &i = done[u]; i < (int)g[u].size(); i++) {
   edge &e = g[u][i];
   if (e.w <= e.flow) continue;
   int v = e.to;
   if (d[v] == d[u] + 1) {
     long long nw = dfs(v, min(flow, e.w - e.flow));
     if (nw > 0) {
       e.flow += nw;
       g[v][e.rev].flow -= nw;
       return nw;
     }}}
 return 0;}
  long long max_flow(int _s, int _t) {
 s = _s;
 t = _t;
 long long flow = 0;
 while (bfs()) {
   done.assign(n, 0);
   while (long long nw = dfs(s, inf)) flow += nw;
 }
 flow_through.assign(mxid + 10, 0);
 for(int i = 0; i < n; i++) for(auto e : g[i]) if(e.id >=
 0) flow_through[e.id] = e.flow;
 return flow; } };
int main() {
  int n, m;
  cin >> n >> m;
  Dinic F(n + 1);
  for (int i = 1; i <= m; i++) {
 int u, v, w;
 cin >> u >> v >> w;
 F.add_edge(u, v, w);}
  cout << F.max_flow(1, n) << '\n';}
```

6.4 Maximum Bipartite Matching

```
/// maximum independent set = n - max matching
/// minimum vertex cover = max matching
/// minimum edge cover = n - max matching
/// minimum path cover(vertex disjoint) on DAG: Take each
node twice to construct a bipartite graph.
/// If the DAG has edge u to v the bipartite graph has edge
from u of left side to v of right side.
/// minimum path cover = n - max matching where n is the
number of nodes in DAG
/// Minimum Path Cover (Vertex not Disjoint) in General
Graph: Create SCC graph. Take each node twice to construct
a bipartite graph.
/// If in the DAG(SCC graph) vertex v is reachable from u
then add edge from u of left side to v of right side in the
bipartite graph
/// Reachability can be checked using by calling bfs n
times where n is the number of nodes in SCC graph.
/// minimum path cover = n - max matching where n is the
number of nodes in SCC graph
// 1 indexed Hopcroft-Karp Matching in O(E sqrtV)
// Add edge from left side to right side
struct Hopcroft_Karp
{
    static const int inf = 1e9;
    int n;
    vector<int> matchL, matchR, dist;
    vector<vector<int> > g;
    Hopcroft_Karp(int n, int m) : n(n),
    matchL(n+1),matchR(m+1), dist(n+1), g(n+1) {}
    void addEdge(int u, int v) {
        g[u].push_back(v);
    }
    bool bfs()
    {
        queue<int> q;
        for(int u=1;u<=n;u++) {
            if(!matchL[u]) {
                dist[u]=0;
                q.push(u);
            }
            else
            dist[u]=inf;
        }
        dist[0]=inf;
        while(!q.empty())
        {
            int u=q.front();
            q.pop();
            for(auto v:g[u])
            {
                if(dist[matchR[v]] == inf)
                {
                    dist[matchR[v]] = dist[u] + 1;
                    q.push(matchR[v]);
                }
            }
        }
        return (dist[0]!=inf);
    }
    bool dfs(int u)
    {
        if(!u) return true;
        for(auto v:g[u])
        {
            if(dist[matchR[v]] == dist[u]+1
            &&dfs(matchR[v])) {
                matchL[u]=v;
                matchR[v]=u;
                return true;
            }
        }
        dist[u]=inf;
        return false;
    }
    int max_matching()
    {
        int matching=0;
        while(bfs()) {
            for(int u=1;u<=n;u++) {
                if(!matchL[u]) if(dfs(u))
                matching++;
            }
        }
        return matching;
    }
};
```

6.5 Hungarian , Maximum Weighted Matching

```
/* Complexity: O(n^3) but optimized
It finds minimum cost maximum matching.
For finding maximum cost maximum matching
add -cost and return -matching()
1-indexed */
struct Hungarian {
  long long c[N][N], fx[N], fy[N], d[N];
  int l[N], r[N], arg[N], trace[N];
  queue<int> q;
  int start, finish, n;
  const long long inf = 1e18;
  Hungarian() {}
  Hungarian(int n1, int n2): n(max(n1, n2)) {
    for (int i = 1; i <= n; ++i) {
      fy[i] = l[i] = r[i] = 0;
      for (int j = 1; j <= n; ++j) c[i][j] = inf; // make
      it 0 for maximum cost matching (not necessarily with
      max count of matching)
    }
  }
  void add_edge(int u, int v, long long cost) {
    c[u][v] = min(c[u][v], cost);
  }
  inline long 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 (int i = 0; i <= n; ++i) trace[i] = 0;
    for (int v = 1; v <= n; ++v) {
      d[v] = getC(start, v);
      arg[v] = start;
    }
    finish = 0;
  }
  void findAugPath() {
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (int v = 1; v <= n; ++v) if (!trace[v]) {
          long long w = getC(u, v);
          if (!w) {
            trace[v] = u;
            if (!r[v]) {
              finish = v;
              return;
            }
            q.push(r[v]);
          }
          if (d[v] > w) {
            d[v] = w;
            arg[v] = u;
          }
        }
    }
  }
  void subX_addY() {
    long long delta = inf;
    for (int v = 1; v <= n; ++v) if (trace[v] == 0 && d[v]
    < delta) {
        delta = d[v];
    }
    // Rotate
    fx[start] += delta;
    for (int v = 1; v <= n; ++v) if(trace[v]) {
        int u = r[v];
        fy[v] -= delta;
        fx[u] += delta;
    } else d[v] -= delta;
    for (int v = 1; v <= n; ++v) if (!trace[v] && !d[v]) {
        trace[v] = arg[v];
        if (!r[v]) {
          finish = v;
          return;
        }
        q.push(r[v]);
    }
  }
  void Enlarge() {
    do {
      int u = trace[finish];
      int nxt = l[u];
      l[u] = finish;
      r[finish] = u;
      finish = nxt;
    } while (finish);
  }
  long long maximum_matching() {
    for (int u = 1; u <= n; ++u) {
      fx[u] = c[u][1];
      for (int v = 1; v <= n; ++v) {
        fx[u] = min(fx[u], c[u][v]);
      }
    }
    for (int v = 1; v <= n; ++v) {
      fy[v] = c[1][v] - fx[1];
      for (int u = 1; u <= n; ++u) {
        fy[v] = min(fy[v], c[u][v] - fx[u]);
      }
    }
    for (int u = 1; u <= n; ++u) {
      start = u;
      initBFS();
      while (!finish) {
        findAugPath();
        if (!finish) subX_addY();
      }
      Enlarge();
    }
    long long ans = 0;
    for (int i = 1; i <= n; ++i) {
      if (c[i][l[i]] != inf) ans += c[i][l[i]];
      else l[i] = 0;
    }
    return ans;
  }
};
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n1, n2, m;
  cin >> n1 >> n2 >> m;
  Hungarian M(n1, n2);
  for (int i = 1; i <= m; i++) {
    int u, v, w;
    cin >> u >> v >> w;
    M.add_edge(u, v, -w);
  }
  cout << -M.maximum_matching() << '\n';
  for (int i = 1; i <= n1; i++) cout << M.l[i] << ' ';
  return 0;
}
```

6.6 Min cost Max Flow

```
const int N = 3e5 + 9;
//Works for both directed, undirected and with negative
cost too
//doesn't work for negative cycles
//for undirected edges just make the directed flag false
//Complexity: O(min(E^2 *V log V, E logV * flow))
using T = long long;
const T inf = 1LL << 61;
struct MCMF {
  struct edge {
 int u, v;
 T cap, cost;
 int id;
 edge(int _u, int _v, T _cap, T _cost, int _id) {
   u = _u;
   v = _v;
   cap = _cap;
   cost = _cost;
   id = _id;
 }};
  int n, s, t, mxid;
  T flow, cost;
  vector<vector<int>> g;
  vector<edge> e;
  vector<T> d, potential, flow_through;
  vector<int> par;
  bool neg;
  MCMF() {}
  MCMF(int _n) { // 0-based indexing
 n = _n + 10;
 g.assign(n, vector<int> ());
 neg = false;
 mxid = 0;}
  void add_edge(int u, int v, T cap, T cost, int id = -1,
  bool directed = true) {
 if(cost < 0) neg = true;
 g[u].push_back(e.size());
 e.push_back(edge(u, v, cap, cost, id));
 g[v].push_back(e.size());
 e.push_back(edge(v, u, 0, -cost, -1));
 mxid = max(mxid, id);
 if(!directed) add_edge(v, u, cap, cost, -1, true);}
  bool dijkstra() {
 par.assign(n, -1);
 d.assign(n, inf);
 priority_queue<pair<T, T>, vector<pair<T, T>>,
 greater<pair<T, T>> > q;
 d[s] = 0;
 q.push(pair<T, T>(0, s));
 while (!q.empty()) {
   int u = q.top().second;
   T nw = q.top().first;
   q.pop();
   if(nw != d[u]) continue;
   for (int i = 0; i < (int)g[u].size(); i++) {
     int id = g[u][i];
     int v = e[id].v;
     T cap = e[id].cap;
     T w = e[id].cost + potential[u] - potential[v];
     if (d[u] + w < d[v] && cap > 0) {
       d[v] = d[u] + w;
       par[v] = id;
       q.push(pair<T, T>(d[v], v));
     }}}
 for (int i = 0; i < n; i++) {
   if (d[i] < inf) d[i] += (potential[i] - potential[s]);}
 for (int i = 0; i < n; i++) {
   if (d[i] < inf) potential[i] = d[i];}
 return d[t] != inf; // for max flow min cost
 // return d[t] <= 0; // for min cost flow}
  T send_flow(int v, T cur) {
 if(par[v] == -1) return cur;
 int id = par[v];
 int u = e[id].u;
 T w = e[id].cost;
 T f = send_flow(u, min(cur, e[id].cap));
 cost += f * w;
 e[id].cap -= f;
 e[id ^ 1].cap += f;
 return f;}
  //returns {maxflow, mincost}
  pair<T, T> solve(int _s, int _t, T goal = inf) {
 s = _s;
 t = _t;
 flow = 0, cost = 0;
 potential.assign(n, 0);
 if (neg) {
   // Run Bellman-Ford to find starting potential on the
   starting graph
   // If the starting graph (before pushing flow in the
   residual graph) is a DAG,
   // then this can be calculated in O(V + E) using DP:
   // potential(v) = min({potential[u] + cost[u][v]}) for
   each u -> v and potential[s] = 0
   d.assign(n, inf);
   d[s] = 0;
   bool relax = true;
   for (int i = 0; i < n && relax; i++) {
     relax = false;
     for (int u = 0; u < n; u++) {
       for (int k = 0; k < (int)g[u].size(); k++) {
         int id = g[u][k];
         int v = e[id].v;
         T cap = e[id].cap, w = e[id].cost;
         if (d[v] > d[u] + w && cap > 0) {
           d[v] = d[u] + w;
           relax = true;
         }}}}
   for(int i = 0; i < n; i++) if(d[i] < inf) potential[i] =
   d[i];}
 while (flow < goal && dijkstra()) flow += send_flow(t,
 goal - flow);
 flow_through.assign(mxid + 10, 0);
 for (int u = 0; u < n; u++) {
   for (auto v : g[u]) {
     if (e[v].id >= 0) flow_through[e[v].id] = e[v ^
     1].cap;
   }}
 return make_pair(flow, cost);
  }};
int main() {
  int n;
  cin >> n;
  assert(n <= 10);
  MCMF F(2 * n);
  for (int i = 0; i < n; i++) {
 for (int j = 0; j < n; j++) {
   int k;
   cin >> k;
   F.add_edge(i, j + n, 1, k, i * 20 + j); }}
  int s = 2 * n + 1, t = s + 1;
  for (int i = 0; i < n; i++) {
 F.add_edge(s, i, 1, 0);
 F.add_edge(i + n, t, 1, 0);}
  auto ans = F.solve(s, t).second;
  long long w = 0;
  set<int> se;
  for (int i = 0; i < n; i++) {
 for (int j = 0; j < n; j++) {
   int p = i * 20 + j;
   if (F.flow_through[p] > 0) {
     se.insert(j);
     w += F.flow_through[p]; }}}
  assert(se.size() == n && w == n);
  cout << ans << '\n';}
```

6.7 2-SAT

Time Complexity: O(n+m) where n is he number of variables and m is the number of clauses.

```
// TwoSat two_sat(n);
// two_sat.either(i, j); // i, j both 1-indexed
// two_sat.either(i, -j); // i, j both 1-indexed
// two_sat.solve(); // solves and returns true if solution exists
// two_sat.values; // get assignments in the solution
struct TwoSat {
  int n;
  vector<vector<int>> adj;
  vector<int> values; // 0 = false, 1 = true
  TwoSat(int n = 0) : n(n), adj(2 * n) {}
  void either(int i, int j) {
 i = 2 * (abs(i) - 1) + (i < 0);
 j = 2 * (abs(j) - 1) + (j < 0);
 adj[i ^ 1].push_back(j);
 adj[j ^ 1].push_back(i);
  }
  vector<int> enter, comp, curr_comp;
  int time = 0;
  int dfs(int at) {
 int low = enter[at] = ++time;
 curr_comp.push_back(at);
 for (int to : adj[at]) {
    if (!comp[to]) low = min(low, enter[to] ? enter[to] :
    dfs(to));
 }
 if (low == enter[at]) {
    int v;
    do {
       v = curr_comp.back();
       curr_comp.pop_back();
       comp[v] = low;
       if (values[v >> 1] == -1) values[v >> 1] = !(v & 1);
    } while (v != at);
 }
 return enter[at] = low;
  }
  bool solve() {
 values.assign(n, -1);
 enter.assign(2 * n, 0);
 comp = enter;
 for (int i = 0; i < 2 * n; i++) {
    if (!comp[i]) dfs(i);}
 for (int i = 0; i < n; i++) {
   if (comp[2 * i] == comp[2 * i + 1]) return false; }
 return true; }};
```

6.8 SCC

```
const int N = 3e5 + 9;
// given a directed graph return the minimum number of
edges to be added so that the whole graph  become an SCC
bool vis[N];
vector<int> g[N], r[N], G[N], vec; //G is the condensed
graph
void dfs1(int u) {
  vis[u] = 1;
  for(auto v: g[u]) if(!vis[v]) dfs1(v);
  vec.push_back(u);
}
vector<int> comp;
void dfs2(int u) {
  comp.push_back(u);
  vis[u] = 1;
  for(auto v: r[u]) if(!vis[v]) dfs2(v);
}
int idx[N], in[N], out[N];
int main() {
  int n, m;
  cin >> n >> m;
  for(int i = 1; i <= m; i++) {
    int u, v;
    cin >> u >> v;
    g[u].push_back(v);
    r[v].push_back(u);
  }
  for(int i = 1; i <= n; i++) if(!vis[i]) dfs1(i);
  reverse(vec.begin(), vec.end());
  memset(vis, 0, sizeof vis);
  int scc = 0;
  for(auto u: vec) {
    if(!vis[u]) {
      comp.clear();
      dfs2(u);
      scc++;
      for(auto x: comp) idx[x]=scc;
    }
  }
  for(int u = 1; u <= n; u++) {
    for(auto v: g[u]) {
      if(idx[u] != idx[v]) {
        in[idx[v]]++, out[idx[u]]++;
        G[idx[u]].push_back(idx[v]);
      }
    }
  }
  int needed_in=0, needed_out=0;
  for(int i = 1; i <= scc; i++) {
    if(!in[i]) needed_in++;
    if(!out[i]) needed_out++;
  }
  int ans = max(needed_in, needed_out);
  if(scc == 1) ans = 0;
  cout << ans << '\n';
  return 0;
}
```

6.9 Max clique

```
const int N = 42;
int g[N][N], res, edges[N]; //3 ^ (n / 3)
void BronKerbosch(int n, long long R, long long P, long
long X) {
  if (P == 0LL && X == 0LL) { // found max clique
    int t = __builtin_popcountll(R);
    res = max(res, t);
    return;
  }
  int u = 0;
  while (!(((1LL << u) & (P | X))) u ++;
  for (int v = 0; v < n; v++) {
    if (((1LL << v) & P) && !((1LL << v) & edges[u])) {
      BronKerbosch(n, R | (1LL << v), P & edges[v], X &
      edges[v]);
      P -= (1LL << v);
      X |= (1LL << v);
    }
  }
}
int max_clique (int n) {
  res = 0;
  for (int i = 1; i <= n; i++) {
    edges[i - 1] = 0;
    for (int j = 1; j <= n; j++)  if (g[i][j]) edges[i - 1]
    |= ( 1LL << (j - 1) );
  }
  BronKerbosch(n, 0, (1LL << n) - 1, 0);
  return res;
}
```

6.10 Virtual Tree

Usage: clique is a complete subgraph of a given graph

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

vector<int> g[N];
int par[N][20], dep[N], sz[N], st[N], en[N], T;
void dfs(int u, int pre) {
  par[u][0] = pre;
  dep[u] = dep[pre] + 1;
  sz[u] = 1;
  st[u] = ++T;
  for (int i = 1; i <= 18; i++) par[u][i] = par[par[u][i -
  1]][i - 1];
  for (auto v : g[u]) {
    if (v == pre) continue;
    dfs(v, u);
    sz[u] += sz[v];
  }
  en[u] = T;
}
int lca(int u, int v) {
  if (dep[u] < dep[v]) swap(u, v);
  for (int k = 18; k >= 0; k--) if (dep[par[u][k]] >=
  dep[v]) u = par[u][k];
  if (u == v) return u;
  for (int k = 18; k >= 0; k--) if (par[u][k] != par[v][k])
  u = par[u][k], v = par[v][k];
  return par[u][0];
}
int kth(int u, int k) {
  for (int i = 0; i <= 18; i++) if (k & (1 << i)) u =
  par[u][i];
  return u;
}
int dist(int u, int v) {
  int lc = lca(u, v);
  return dep[u] + dep[v] - 2 * dep[lc];
}
int isanc(int u, int v) {
  return (st[u] <= st[v]) && (en[v] <= en[u]);
}
vector<int> t[N];
// given specific nodes, construct a compressed directed
tree with these vertices(if needed some other nodes
included)
// returns the nodes of the tree
// nodes.front() is the root
// t[] is the specific tree
vector<int> buildtree(vector<int> v) {
  // sort by entry time
  sort(v.begin(), v.end(), [](int x, int y) {
    return st[x] < st[y];
  });
  // finding all the ancestors, there are few of them
  int s = v.size();
  for (int i = 0; i < s - 1; i++) {
    int lc = lca(v[i], v[i + 1]);
    v.push_back(lc);
  }
  // removing duplicated nodes
  sort(v.begin(), v.end());
  v.erase(unique(v.begin(), v.end()), v.end());
  // again sort by entry time
  sort(v.begin(), v.end(), [](int x, int y) {
    return st[x] < st[y];
  });
  stack<int> st;
  st.push(v[0]);
  for (int i = 1; i < v.size(); i++) {
    while (!isanc(st.top(), v[i])) st.pop();
    t[st.top()].push_back(v[i]);
    st.push(v[i]);
  }
  return v;
}
int ans;
int imp[N];
int yo(int u) {
  vector<int> nw;
  for (auto v : t[u]) nw.push_back(yo(v));
  if (imp[u]) {
    for (auto x : nw) if (x) ans++;
    return 1;
  } else {
    int cnt = 0;
    for (auto x : nw) cnt += x > 0;
    if (cnt > 1) {
      ans++;
      return 0;
    }
    return cnt;
  }
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int i, j, k, n, m, q, u, v;
  cin >> n;
  for (i = 1; i < n; i++) cin >> u >> v, g[u].push_back(v),
  g[v].push_back(u);
  dfs(1, 0);
  cin >> q;
  while (q--) {
    cin >> k;
    vector<int> v;
    for (i = 0; i < k; i++) cin >> m, v.push_back(m),
    imp[m] = 1;
    int fl = 1;
    for (auto x : v) if (imp[par[x][0]]) fl = 0;
    ans = 0;
    vector<int> nodes;
    if (fl) nodes = buildtree(v);
    if (fl) yo(nodes.front());
    if (!fl) ans = -1;
    cout << ans << '\n';
    // clear the tree
    for (auto x : nodes) t[x].clear();
    for (auto x : v) imp[x] = 0;
  }
  return 0;
}
```

6.11 Euler path & circuit

```
/* conditins :
all edges should be in same connected component
#directed graph:
euler path: for all-> indeg=outdeg || 1 node->
(indeg-outdeg=1) and one node-> (outdeg-indeg=1) and
others-> in=out
euler circuit: for all -> indeg = outdeg
#undirected graph:
euler path: all degrees are even or exactly 2 of them are
odd
euler circuit: all degrees are even */
vector<pii> g[N];//nxtNode-EdgeNum
```

```
vector<int> ans;
int done[N], vis[M];
void dfs(int u) {
  while (done[u] < g[u].size()) {
    auto e = g[u][done[u]++];
    if (vis[e.second]) continue;
    vis[e.second] = 1;
    dfs(e.first);
  }
  ans.push_back(u);
}
int solve(int n) {
// check conditions, return 0 if dont exist
/* for dirGraph root: if any node (outdeg != indeg) -> node
with outdeg > indeg
 else any node with outdeg >= 1 */
/* for undirGraph root: if all even deg -> any with deg>=1
else node with deg odd */
  if (root == 0) return 1; //empty graph
  dfs(root);
  if (ans.size() != M + 1) return 0; //connectivity
  reverse(ans.begin(), ans.end());
  return 1;
}
```

6.12 Inverse graph

```
const int N = 2e5;
vector<int> g[N+5], par(N+5), vis(N+5, 0);
int Find(int x){return par[x] == x ? x : par[x] =
Find(par[x]);}
void dfs(int u, int n){
 if(vis[u]) return;
 vis[u] = 1;
 par[u] = Find(u+1);
 int v = 0;
 for(auto it: g[u]){
  v = Find(v+1);
  while(v < it){
   dfs(v, n);
   v = Find(v+1);
  }
  v = it;
 }
 v = Find(v+1);
 while(v <= n){
  dfs(v, n);
  v = Find(v+1);
 }
}
ininialize with:
for(int K = 1; K <= n+1; K++) par[K] = K; // n+1 is
important
for(int K = 1; K <= n; K++) vis[K] = 0;
call with:
for(int K = 1; K <= n; K++) if(!vis[K]) dfs(K, n);
```

6.13 Shortest Cycle

```
#define N 100200
vector<int> gr[N];
void Add_edge(int x, int y)
{gr[x].PB(y);gr[y].PB(x);}
int shortest_cycle(int n){
    int ans = INT_MAX;
for (int i = 0; i < n; i++) {
    vector<int> dist(n,(int)(1e9));
    vector<int> par(n, -1);
    dist[i] = 0;
    queue<int> q;
    q.push(i);
    while (!q.empty()) {
        int x = q.front();
        q.pop();
        for (int it : gr[x]) {
            if (dist[it] == (int)(1e9)) {
                dist[it] = 1 + dist[x];
                par[it] = x;
                q.push(it);
            }
            else if(par[x]!=it & par[it]!= x)
            ans=min(ans,dist[x]+dist[it]+1);
        }
    }
}
if(ans == INT_MAX)return -1;
else return ans;
}
```

7 Geometry

7.1 All 2D Functions

```
const double inf = 1e100;
const double eps = 1e-9;
const double PI = acos((double)-1.0);
int sign(double x) { return (x > eps) - (x < -eps); }
struct PT {
    double x, y;
    PT() { x = 0, y = 0; }
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y)    {}
    PT operator + (const PT &a) const { return PT(x + a.x,
    y + a.y); }
    PT operator - (const PT &a) const { return PT(x - a.x,
    y - a.y); }
    PT operator * (const double a) const { return PT(x * a,
    y * a); }
    friend PT operator * (const double &a, const PT &b) {
    return PT(a * b.x, a * b.y); }
    PT operator / (const double a) const { return PT(x / a,
    y / a); }
    bool operator == (PT a) const { return sign(a.x - x) ==
    0 && sign(a.y - y) == 0; }
    bool operator != (PT a) const { return !(*this == a); }
    bool operator < (PT a) const { return sign(a.x - x) ==
    0 ? y < a.y : x < a.x; }
    bool operator > (PT a) const { return sign(a.x - x) ==
    0 ? y > a.y : x > a.x; }
    double norm() { return sqrt(x * x + y * y); }
    double norm2() { return x * x + y * y; }
    PT perp() { return PT(-y, x); }
    double arg() { return atan2(y, x); }
    PT truncate(double r) { // returns a vector with norm r
    and having same direction
        double k = norm();
        if (!sign(k)) return *this;
        r /= k;
        return PT(x * r, y * r);
    }
};
istream &operator >> (istream &in, PT &p) { return in >>
p.x >> p.y; }
ostream &operator << (ostream &out, PT &p) { return out <<
"(" << p.x << "," << p.y << ")"; }
inline double dot(PT a, PT b) { return a.x * b.x + a.y *
b.y; }
inline double dist2(PT a, PT b) { return dot(a - b, a - b);
}
inline double dist(PT a, PT b) { return sqrt(dot(a - b, a -
b)); }
inline double cross(PT a, PT b) { return a.x * b.y - a.y *
b.x; }
inline double cross2(PT a, PT b, PT c) { return cross(b -
a, c - a); }
inline int orientation(PT a, PT b, PT c) { return
sign(cross(b - a, c - a)); }
PT perp(PT a) { return PT(-a.y, a.x); }
PT rotateccw90(PT a) { return PT(-a.y, a.x); }
PT rotatecw90(PT a) { return PT(a.y, -a.x); }
PT rotateccw(PT a, double t) { return PT(a.x * cos(t) - a.y
* sin(t), a.x * sin(t) + a.y * cos(t)); }
PT rotatecw(PT a, double t) { return PT(a.x * cos(t) + a.y
* sin(t), -a.x * sin(t) + a.y * cos(t)); }
double SQ(double x) { return x * x; }
double rad_to_deg(double r) { return (r * 180.0 / PI); }
double deg_to_rad(double d) { return (d * PI / 180.0); }
double get_angle(PT a, PT b) {
    double costheta = dot(a, b) / a.norm() / b.norm();
    return acos(max((double)-1.0, min((double)1.0,
    costheta)));
}
bool is_point_in_angle(PT b, PT a, PT c, PT p) { // does
point p lie in angle <bac
    assert(orientation(a, b, c) != 0);
    if (orientation(a, c, b) < 0) swap(b, c);
    return orientation(a, c, p) >= 0 && orientation(a, b,
    p) <= 0;
}
bool half(PT p) {
    return p.y > 0.0 || (p.y == 0.0 && p.x < 0.0);
}
void polar_sort(vector<PT> &v) { // sort points in
counterclockwise
    sort(v.begin(), v.end(), [](PT a,PT b) {
        return make_tuple(half(a), 0.0, a.norm2()) <
        make_tuple(half(b), cross(a, b), b.norm2());
    });
}
void polar_sort(vector<PT> &v, PT o) { // sort points in
counterclockwise with respect to point o
    sort(v.begin(), v.end(), [&](PT a,PT b) {
        return make_tuple(half(a - o), 0.0, (a -
        o).norm2()) < make_tuple(half(b - o), cross(a - o,
        b - o), (b - o).norm2());
    });
}
struct line {
    PT a, b; // goes through points a and b
    PT v; double c;  //line form: direction vec [cross] (x,
    y) = c
    line() {}
    //direction vector v and offset c
 line(PT v, double c) : v(v), c(c) {
        auto p = get_points();
        a = p.first; b = p.second;
 }
 // equation ax + by + c = 0
 line(double _a, double _b, double _c) : v({_b, -_a}),
 c(-_c) {
  auto p = get_points();
        a = p.first; b = p.second;
 }
 // goes through points p and q
 line(PT p, PT q) : v(q - p), c(cross(v, p)), a(p), b(q) {}
     pair<PT, PT> get_points() { //extract any two points
     from this line
  PT p, q; double a = -v.y, b = v.x; // ax + by = c
  if (sign(a) == 0) {
      p = PT(0, c / b);
      q = PT(1, c / b);
  }
  else if (sign(b) == 0) {
      p = PT(c / a, 0);
      q = PT(c / a, 1);
  }
  else {
      p = PT(0, c / b);
      q = PT(1, (c - a) / b);
  }
  return {p, q};
    }
    // ax + by + c = 0
    array<double, 3> get_abc() {
        double a = -v.y, b = v.x;
        return {a, b, -c};
    }
    // 1 if on the left, -1 if on the right, 0 if on the
    line
    int side(PT p) { return sign(cross(v, p) - c); }
    // line that is perpendicular to this and goes through
    point p
    line perpendicular_through(PT p) { return {p, p +
    perp(v)}; }
    // translate the line by vector t i.e. shifting it by
    vector t
    line translate(PT t) { return {v, c + cross(v, t)}; }
    // compare two points by their orthogonal projection on
    this line
    // a projection point comes before another if it comes
    first according to vector v
    bool cmp_by_projection(PT p, PT q) { return dot(v, p) <
    dot(v, q); }
 line shift_left(double d) {
  PT z = v.perp().truncate(d);
  return line(a + z, b + z);
 }
};
// find a point from a through b with distance d
PT point_along_line(PT a, PT b, double d) {
    assert(a != b);
    return a + (((b - a) / (b - a).norm()) * d);
}
// projection point c onto line through a and b  assuming a
!= b
PT project_from_point_to_line(PT a, PT b, PT c) {
    return a + (b - a) * dot(c - a, b - a) / (b -
    a).norm2();
}
// reflection point c onto line through a and b  assuming a
!= b
PT reflection_from_point_to_line(PT a, PT b, PT c) {
    PT p = project_from_point_to_line(a,b,c);
    return p + p - c;
}
// minimum distance from point c to line through a and b
double dist_from_point_to_line(PT a, PT b, PT c) {
    return fabs(cross(b - a, c - a) / (b - a).norm());
}
// returns true if  point p is on line segment ab
bool is_point_on_seg(PT a, PT b, PT p) {
    if (fabs(cross(p - b, a - b)) < eps) {
```

```
        if (p.x < min(a.x, b.x) - eps || p.x > max(a.x,
        b.x) + eps) return false;
        if (p.y < min(a.y, b.y) - eps || p.y > max(a.y,
        b.y) + eps) return false;
        return true;
    }
    return false;
}
// minimum distance point from point c to segment ab that
lies on segment ab
PT project_from_point_to_seg(PT a, PT b, PT c) {
    double r = dist2(a, b);
    if (sign(r) == 0) return a;
    r = dot(c - a, b - a) / r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b - a) * r;
}
// minimum distance from point c to segment ab
double dist_from_point_to_seg(PT a, PT b, PT c) {
    return dist(c, project_from_point_to_seg(a, b, c));
}
// 0 if not parallel, 1 if parallel, 2 if collinear
int is_parallel(PT a, PT b, PT c, PT d) {
    double k = fabs(cross(b - a, d - c));
    if (k < eps){
        if (fabs(cross(a - b, a - c)) < eps && fabs(cross(c
        - d, c - a)) < eps) return 2;
        else return 1;
    }
    else return 0;
}
// check if two lines are same
bool are_lines_same(PT a, PT b, PT c, PT d) {
    if (fabs(cross(a - c, c - d)) < eps && fabs(cross(b -
    c, c - d)) < eps) return true;
    return false;
}
// bisector vector of <abc
PT angle_bisector(PT &a, PT &b, PT &c){
    PT p = a - b, q = c - b;
    return p + q * sqrt(dot(p, p) / dot(q, q));
}
// 1 if point is ccw to the line, 2 if point is cw to the
line, 3 if point is on the line
int point_line_relation(PT a, PT b, PT p) {
    int c = sign(cross(p - a, b - a));
    if (c < 0) return 1;
    if (c > 0) return 2;
    return 3;
}
// intersection point between ab and cd assuming unique
intersection exists
bool line_line_intersection(PT a, PT b, PT c, PT d, PT
&ans) {
    double a1 = a.y - b.y, b1 = b.x - a.x, c1 = cross(a,
    b);
    double a2 = c.y - d.y, b2 = d.x - c.x, c2 = cross(c,
    d);
    double det = a1 * b2 - a2 * b1;
    if (det == 0) return 0;
    ans = PT((b1 * c2 - b2 * c1) / det, (c1 * a2 - a1 * c2)
    / det);
    return 1;
}
// intersection point between segment ab and segment cd
assuming unique intersection exists
bool seg_seg_intersection(PT a, PT b, PT c, PT d, PT &ans)
{
    double oa = cross2(c, d, a), ob = cross2(c, d, b);
    double oc = cross2(a, b, c), od = cross2(a, b, d);
    if (oa * ob < 0 && oc * od < 0){
        ans = (a * ob - b * oa) / (ob - oa);
        return 1;
    }
    else return 0;
}
// intersection point between segment ab and segment cd
assuming unique intersection may not exists
// se.size()==0 means no intersection
// se.size()==1 means one intersection
// se.size()==2 means range intersection
set<PT> seg_seg_intersection_inside(PT a,  PT b,  PT c,  PT
d) {
    PT ans;
    if (seg_seg_intersection(a, b, c, d, ans)) return
    {ans};
    set<PT> se;
    if (is_point_on_seg(c, d, a)) se.insert(a);
    if (is_point_on_seg(c, d, b)) se.insert(b);
    if (is_point_on_seg(a, b, c)) se.insert(c);
    if (is_point_on_seg(a, b, d)) se.insert(d);
    return se;
}
// intersection  between segment ab and line cd
// 0 if do not intersect, 1 if proper intersect, 2 if
segment intersect
int seg_line_relation(PT a, PT b, PT c, PT d) {
    double p = cross2(c, d, a);
    double q = cross2(c, d, b);
    if (sign(p) == 0 && sign(q) == 0) return 2;
    else if (p * q < 0) return 1;
    else return 0;
}
// intersection between segament ab and line cd assuming
unique intersection exists
bool seg_line_intersection(PT a, PT b, PT c, PT d, PT &ans)
{
    bool k = seg_line_relation(a, b, c, d);
    assert(k != 2);
    if (k) line_line_intersection(a, b, c, d, ans);
    return k;
}
// minimum distance from segment ab to segment cd
double dist_from_seg_to_seg(PT a, PT b, PT c, PT d) {
    PT dummy;
    if (seg_seg_intersection(a, b, c, d, dummy)) return
    0.0;
    else return min({dist_from_point_to_seg(a, b, c),
    dist_from_point_to_seg(a, b, d),
        dist_from_point_to_seg(c, d, a),
        dist_from_point_to_seg(c, d, b)});
}
// minimum distance from point c to ray (starting point a
and direction vector b)
double dist_from_point_to_ray(PT a, PT b, PT c) {
    b = a + b;
    double r = dot(c - a, b - a);
    if (r < 0.0) return dist(c, a);
    return dist_from_point_to_line(a, b, c);
}
// starting point as and direction vector ad
bool ray_ray_intersection(PT as, PT ad, PT bs, PT bd) {
    double dx = bs.x - as.x, dy = bs.y - as.y;
    double det = bd.x * ad.y - bd.y * ad.x;
    if (fabs(det) < eps) return 0;
    double u = (dy * bd.x - dx * bd.y) / det;
    double v = (dy * ad.x - dx * ad.y) / det;
    if (sign(u) >= 0 && sign(v) >= 0) return 1;
    else return 0;
}
double ray_ray_distance(PT as, PT ad, PT bs, PT bd) {
    if (ray_ray_intersection(as, ad, bs, bd)) return 0.0;
    double ans = dist_from_point_to_ray(as, ad, bs);
    ans = min(ans, dist_from_point_to_ray(bs, bd, as));
    return ans;
}
struct circle {
    PT p; double r;
    circle() {}
    circle(PT _p, double _r): p(_p), r(_r) {};
    // center (x, y) and radius r
    circle(double x, double y, double _r): p(PT(x, y)),
    r(_r) {};
    // circumcircle of a triangle
    // the three points must be unique
    circle(PT a, PT b, PT c) {
        b = (a + b) * 0.5;
        c = (a + c) * 0.5;
        line_line_intersection(b, b + rotatecw90(a - b), c,
        c + rotatecw90(a - c), p);
        r = dist(a, p);
    }
    // inscribed circle of a triangle
    // pass a bool just to differentiate from circumcircle
    circle(PT a, PT b, PT c, bool t) {
        line u, v;
        double m = atan2(b.y - a.y, b.x - a.x), n =
        atan2(c.y - a.y, c.x - a.x);
        u.a = a;
        u.b = u.a + (PT(cos((n + m)/2.0), sin((n +
        m)/2.0)));
        v.a = b;
        m = atan2(a.y - b.y, a.x - b.x), n = atan2(c.y -
        b.y, c.x - b.x);
        v.b = v.a + (PT(cos((n + m)/2.0), sin((n +
        m)/2.0)));
        line_line_intersection(u.a, u.b, v.a, v.b, p);
        r = dist_from_point_to_seg(a, b, p);
    }
    bool operator == (circle v) { return p == v.p && sign(r
    - v.r) == 0; }
    double area() { return PI * r * r; }
    double circumference() { return 2.0 * PI * r; }
};
//0 if outside, 1 if on circumference, 2 if inside circle
int circle_point_relation(PT p, double r, PT b) {
    double d = dist(p, b);
    if (sign(d - r) < 0) return 2;
    if (sign(d - r) == 0) return 1;
    return 0;
}
// 0 if outside, 1 if on circumference, 2 if inside circle
int circle_line_relation(PT p, double r, PT a, PT b) {
    double d = dist_from_point_to_line(a, b, p);
    if (sign(d - r) < 0) return 2;
    if (sign(d - r) == 0) return 1;
    return 0;
}
//compute intersection of line through points a and b with
//circle centered at c with radius r > 0
vector<PT> circle_line_intersection(PT c, double r, PT a,
PT b) {
    vector<PT> ret;
    b = b - a; a = a - c;
    double A = dot(b, b), B = dot(a, b);
    double C = dot(a, a) - r * r, D = B * B - A * C;
    if (D < -eps) return ret;
    ret.push_back(c + a + b * (-B + sqrt(D + eps)) / A);
    if (D > eps) ret.push_back(c + a + b * (-B - sqrt(D)) /
    A);
    return ret;
}
//5 - outside and do not intersect
//4 - intersect outside in one point
//3 - intersect in 2 points
//2 - intersect inside in one point
//1 - inside and do not intersect
int circle_circle_relation(PT a, double r, PT b, double R)
{
    double d = dist(a, b);
    if (sign(d - r - R) > 0)  return 5;
    if (sign(d - r - R) == 0) return 4;
    double l = fabs(r - R);
    if (sign(d - r - R) < 0 && sign(d - l) > 0) return 3;
    if (sign(d - l) == 0) return 2;
    if (sign(d - l) < 0) return 1;
    assert(0); return -1;
}
vector<PT> circle_circle_intersection(PT a, double r, PT b,
double R) {
    if (a == b && sign(r - R) == 0) return {PT(1e18,
    1e18)};
    vector<PT> ret;
    double d = sqrt(dist2(a,  b));
    if (d > r + R || d + min(r,  R) < max(r,  R)) return
    ret;
    double x = (d * d - R * R + r * r) / (2 * d);
    double y = sqrt(r * r - x * x);
    PT v = (b - a) / d;
    ret.push_back(a + v * x  +  rotateccw90(v) * y);
    if (y > 0) ret.push_back(a + v * x - rotateccw90(v) *
    y);
    return ret;
}
// returns two circle c1, c2 through points a, b and of
radius r
// 0 if there is no such circle, 1 if one circle, 2 if two
circle
int get_circle(PT a, PT b, double r, circle &c1, circle
&c2) {
    vector<PT> v = circle_circle_intersection(a, r, b, r);
    int t = v.size();
    if (!t) return 0;
    c1.p = v[0], c1.r = r;
    if (t == 2) c2.p = v[1], c2.r = r;
    return t;
}
// returns two circle c1, c2 which is tangent to line u,
goes through
// point q and has radius r1; 0 for no circle, 1 if c1 = c2
, 2 if c1 != c2
int get_circle(line u, PT q, double r1, circle &c1, circle
&c2) {
    double d = dist_from_point_to_line(u.a, u.b, q);
    if (sign(d - r1 * 2.0) > 0) return 0;
    if (sign(d) == 0) {
     cout << u.v.x << ' ' << u.v.y << '\n';
        c1.p = q + rotateccw90(u.v).truncate(r1);
        c2.p = q + rotatecw90(u.v).truncate(r1);
        c1.r = c2.r = r1;
        return 2;
    }
    line u1 = line(u.a + rotateccw90(u.v).truncate(r1), u.b
    + rotateccw90(u.v).truncate(r1));
    line u2 = line(u.a + rotatecw90(u.v).truncate(r1), u.b
    + rotatecw90(u.v).truncate(r1));
    circle cc = circle(q, r1);
    PT p1, p2; vector<PT> v;
```

```
    v = circle_line_intersection(q, r1, u1.a, u1.b);
    if (!v.size()) v = circle_line_intersection(q, r1,
    u2.a, u2.b);
    v.push_back(v[0]);
    p1 = v[0], p2 = v[1];
    c1 = circle(p1, r1);
    if (p1 == p2) {
        c2 = c1;
        return 1;
    }
    c2 = circle(p2, r1);
    return 2;
}
// returns the circle such that for all points w on the
circumference of the circle
// dist(w, a) : dist(w, b) = rp : rq
// rp != rq
// https://en.wikipedia.org/wiki/Circles_of_Apollonius
circle get_apollonius_circle(PT p, PT q, double rp, double
rq ){
    rq *= rq ;
    rp *= rp ;
    double a = rq - rp ;
    assert(sign(a));
    double g = rq * p.x - rp * q.x ; g /= a ;
    double h = rq * p.y - rp * q.y ; h /= a ;
    double c = rq * p.x * p.x - rp * q.x * q.x + rq * p.y *
    p.y - rp * q.y * q.y ;
    c /= a ;
    PT o(g, h);
    double r = g * g + h * h - c ;
    r = sqrt(r);
    return circle(o,r);
}
// returns area of intersection between two circles
double circle_circle_area(PT a, double r1, PT b, double r2)
{
    double d = (a - b).norm();
    if(r1 + r2 < d + eps) return 0;
    if(r1 + d < r2 + eps) return PI * r1 * r1;
    if(r2 + d < r1 + eps) return PI * r2 * r2;
    double theta_1 = acos((r1 * r1 + d * d - r2 * r2) / (2
    * r1 * d)),
     theta_2 = acos((r2 * r2 + d * d - r1 * r1)/(2 * r2 *
     d));
    return r1 * r1 * (theta_1 - sin(2 * theta_1)/2.) + r2 *
    r2 * (theta_2 - sin(2 * theta_2)/2.);
}
// tangent lines from point q to the circle
int tangent_lines_from_point(PT p, double r, PT q, line &u,
line &v) {
    int x = sign(dist2(p, q) - r * r);
    if (x < 0) return 0; // point in cricle
    if (x == 0) { // point on circle
        u = line(q, q + rotateccw90(q - p));
        v = u;
        return 1;
    }
    double d = dist(p, q);
    double l = r * r / d;
    double h = sqrt(r * r - l * l);
    u = line(q, p + ((q - p).truncate(l) + (rotateccw90(q -
    p).truncate(h))));
    v = line(q, p + ((q - p).truncate(l) + (rotatecw90(q -
    p).truncate(h))));
    return 2;
}
// returns outer tangents line of two circles
// if inner == 1 it returns inner tangent lines
int tangents_lines_from_circle(PT c1, double r1, PT c2,
double r2, bool inner, line &u, line &v) {
    if (inner) r2 = -r2;
    PT d = c2 - c1;
    double dr = r1 - r2, d2 = d.norm2(), h2 = d2 - dr * dr;
    if (d2 == 0 || h2 < 0) {
        assert(h2 != 0);
        return 0;
    }
    vector<pair<PT, PT>>out;
    for (int tmp: {-1, 1}) {
        PT v = (d * dr + rotateccw90(d) * sqrt(h2) * tmp) /
        d2;
        out.push_back({c1 + v * r1, c2 + v * r2});
    }
    u = line(out[0].first, out[0].second);
    if (out.size() == 2) v = line(out[1].first,
    out[1].second);
    return 1 + (h2 > 0);
}
// O(n^2 log n)
// https://vjudge.net/problem/UVA-12056
struct CircleUnion {
    int n;
    double x[2020], y[2020], r[2020];
    int covered[2020];
    vector<pair<double, double> > seg, cover;
    double arc, pol;
    inline int sign(double x) {return x < -eps ? -1 : x >
    eps;}
    inline int sign(double x, double y) {return sign(x -
    y);}
    inline double SQ(const double x) {return x * x;}
    inline double dist(double x1, double y1, double x2,
    double y2) {return sqrt(SQ(x1 - x2) + SQ(y1 - y2));}
    inline double angle(double A, double B, double C) {
        double val = (SQ(A) + SQ(B) - SQ(C)) / (2 * A * B);
        if (val < -1) val = -1;
        if (val > +1) val = +1;
        return acos(val);
    }
    CircleUnion() {
        n = 0;
        seg.clear(), cover.clear();
        arc = pol = 0;
    }
    void init() {
        n = 0;
        seg.clear(), cover.clear();
        arc = pol = 0;
    }
    void add(double xx, double yy, double rr) {
        x[n] = xx, y[n] = yy, r[n] = rr, covered[n] = 0,
        n++;
    }
    void getarea(int i, double lef, double rig) {
        arc += 0.5 * r[i] * r[i] * (rig - lef - sin(rig -
        lef));
        double x1 = x[i] + r[i] * cos(lef), y1 = y[i] +
        r[i] * sin(lef);
        double x2 = x[i] + r[i] * cos(rig), y2 = y[i] +
        r[i] * sin(rig);
        pol += x1 * y2 - x2 * y1;
    }
    double solve() {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < i; j++) {
                if (!sign(x[i] - x[j]) && !sign(y[i] -
                y[j]) && !sign(r[i] - r[j])) {
                    r[i] = 0.0;
                    break;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i != j && sign(r[j] - r[i]) >= 0 &&
                sign(dist(x[i], y[i], x[j], y[j]) - (r[j] -
                r[i])) <= 0) {
                    covered[i] = 1;
                    break;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            if (sign(r[i]) && !covered[i]) {
                seg.clear();
                for (int j = 0; j < n; j++) {
                    if (i != j) {
                        double d = dist(x[i], y[i], x[j],
                        y[j]);
                        if (sign(d - (r[j] + r[i])) >= 0 ||
                        sign(d - abs(r[j] - r[i])) <= 0) {
                            continue;
                        }
                        double alpha = atan2(y[j] - y[i],
                        x[j] - x[i]);
                        double beta = angle(r[i], d, r[j]);
                        pair<double, double> tmp(alpha -
                        beta, alpha + beta);
                        if (sign(tmp.first) <= 0 &&
                        sign(tmp.second) <= 0) {
                            seg.push_back(pair<double,
                            double>(2 * PI + tmp.first, 2 *
                            PI + tmp.second));
                        }
                        else if (sign(tmp.first) < 0) {
                            seg.push_back(pair<double,
                            double>(2 * PI + tmp.first, 2 *
                            PI));
                            seg.push_back(pair<double,
                            double>(0, tmp.second));
                        }
                        else {
                            seg.push_back(tmp);
                        }
                    }
                }
                sort(seg.begin(), seg.end());
                double rig = 0;
                for (vector<pair<double, double>
                >::iterator iter = seg.begin(); iter !=
                seg.end(); iter++) {
                    if (sign(rig - iter->first) >= 0) {
                        rig = max(rig, iter->second);
                    }
                    else {
                        getarea(i, rig, iter->first);
                        rig = iter->second;
                    }
                }
                if (!sign(rig)) {
                    arc += r[i] * r[i] * PI;
                }
                else {
                    getarea(i, rig, 2 * PI);
                }
            }
        }
        return pol / 2.0 + arc;
    }
} CU;
double area_of_triangle(PT a, PT b, PT c) {
    return fabs(cross(b - a, c - a) * 0.5);
}
// -1 if strictly inside, 0 if on the polygon, 1 if
strictly outside
int is_point_in_triangle(PT a, PT b, PT c, PT p) {
    if (sign(cross(b - a,c - a)) < 0) swap(b, c);
    int c1 = sign(cross(b - a,p - a));
    int c2 = sign(cross(c - b,p - b));
    int c3 = sign(cross(a - c,p - c));
    if (c1<0 || c2<0 || c3 < 0) return 1;
    if (c1 + c2 + c3 != 3) return 0;
    return -1;
}
double perimeter(vector<PT> &p) {
    double ans=0; int n = p.size();
    for (int i = 0; i < n; i++) ans += dist(p[i], p[(i + 1)
    % n]);
    return ans;
}
double area(vector<PT> &p) {
    double ans = 0; int n = p.size();
    for (int i = 0; i < n; i++) ans += cross(p[i], p[(i +
    1) % n]);
    return fabs(ans) * 0.5;
}
// centroid of a (possibly non-convex) polygon,
// assuming that the coordinates are listed in a clockwise
or
// counterclockwise fashion.  Note that the centroid is
often known as
// the "center of gravity" or "center of mass".
PT centroid(vector<PT> &p) {
    int n = p.size(); PT c(0, 0);
    double sum = 0;
    for (int i = 0; i < n; i++) sum += cross(p[i], p[(i +
    1) % n]);
    double scale = 3.0 * sum;
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        c = c + (p[i] + p[j]) * cross(p[i], p[j]);
    }
    return c / scale;
}
// 0 if cw, 1 if ccw
bool get_direction(vector<PT> &p) {
    double ans = 0; int n = p.size();
    for (int i = 0; i < n; i++) ans += cross(p[i], p[(i +
    1) % n]);
    if (sign(ans) > 0) return 1;
    return 0;
}
// it returns a point such that the sum of distances
// from that point to all points in p  is minimum
// O(n log^2 MX)
PT geometric_median(vector<PT> p) {
 auto tot_dist = [&](PT z) {
     double res = 0;
     for (int i = 0; i < p.size(); i++) res += dist(p[i],
     z);
     return res;
 };
 auto findY = [&](double x) {
     double yl = -1e5, yr = 1e5;
     for (int i = 0; i < 60; i++) {
         double ym1 = yl + (yr - yl) / 3;
         double ym2 = yr - (yr - yl) / 3;
         double d1 = tot_dist(PT(x, ym1));
```

```
            double d2 = tot_dist(PT(x, ym2));
            if (d1 < d2) yr = ym2;
            else yl = ym1;
        }
        return pair<double, double> (yl, tot_dist(PT(x, yl)));
    };
    double xl = -1e5, xr = 1e5;
    for (int i = 0; i < 60; i++) {
        double xm1 = xl + (xr - xl) / 3;
        double xm2 = xr - (xr - xl) / 3;
        double y1, d1, y2, d2;
        auto z = findY(xm1); y1 = z.first; d1 = z.second;
        z = findY(xm2); y2 = z.first; d2 = z.second;
        if (d1 < d2) xr = xm2;
        else xl = xm1;
    }
    return {xl, findY(xl).first };
}
vector<PT> convex_hull(vector<PT> &p) {
 if (p.size() <= 1) return p;
 vector<PT> v = p;
    sort(v.begin(), v.end());
    vector<PT> up, dn;
    for (auto& p : v) {
        while (up.size() > 1 && orientation(up[up.size() -
        2], up.back(), p) >= 0) {
            up.pop_back();
        }
        while (dn.size() > 1 && orientation(dn[dn.size() -
        2], dn.back(), p) <= 0) {
            dn.pop_back();
        }
        up.push_back(p);
        dn.push_back(p);
    }
    v = dn;
    if (v.size() > 1) v.pop_back();
    reverse(up.begin(), up.end());
    up.pop_back();
    for (auto& p : up) {
        v.push_back(p);
    }
    if (v.size() == 2 && v[0] == v[1]) v.pop_back();
    return v;
}
 //checks if convex or not
bool is_convex(vector<PT> &p) {
    bool s[3]; s[0] = s[1] = s[2] = 0;
    int n = p.size();
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        int k = (j + 1) % n;
        s[sign(cross(p[j] - p[i], p[k] - p[i])) + 1] = 1;
        if (s[0] && s[2]) return 0;
    }
    return 1;
}
// -1 if strictly inside, 0 if on the polygon, 1 if
strictly outside
// it must be strictly convex, otherwise make it strictly
convex first
int is_point_in_convex(vector<PT> &p, const PT& x) { //
O(log n)
    int n = p.size(); assert(n >= 3);
    int a = orientation(p[0], p[1], x), b =
    orientation(p[0], p[n - 1], x);
    if (a < 0 || b > 0) return 1;
    int l = 1, r = n - 1;
    while (l + 1 < r) {
        int mid = l + r >> 1;
        if (orientation(p[0], p[mid], x) >= 0) l = mid;
        else r = mid;
    }
    int k = orientation(p[l], p[r], x);
    if (k <= 0) return -k;
    if (l == 1 && a == 0) return 0;
    if (r == n - 1 && b == 0) return 0;
    return -1;
}
bool is_point_on_polygon(vector<PT> &p, const PT& z) {
    int n = p.size();
    for (int i = 0; i < n; i++) {
     if (is_point_on_seg(p[i], p[(i + 1) % n], z)) return
     1;
    }
    return 0;
}
// returns 1e9 if the point is on the polygon
int winding_number(vector<PT> &p, const PT& z) { // O(n)
    if (is_point_on_polygon(p, z)) return 1e9;
    int n = p.size(), ans = 0;
    for (int i = 0; i < n; ++i) {
        int j = (i + 1) % n;
        bool below = p[i].y < z.y;
        if (below != (p[j].y < z.y)) {
            auto orient = orientation(z, p[j], p[i]);
            if (orient == 0) return 0;
            if (below == (orient > 0)) ans += below ? 1 :
            -1;
        }
    }
    return ans;
}
// -1 if strictly inside, 0 if on the polygon, 1 if
strictly outside
int is_point_in_polygon(vector<PT> &p, const PT& z) { //
O(n)
    int k = winding_number(p, z);
    return k == 1e9 ? 0 : k == 0 ? 1 : -1;
}
// id of the vertex having maximum dot product with z
// polygon must need to be convex
// top = upper right vertex
// for minimum dot product negate z and return -dot(z,
p[id])
int extreme_vertex(vector<PT> &p, const PT &z, const int
top) { // O(log n)
    int n = p.size();
    if (n == 1) return 0;
 double ans = dot(p[0], z); int id = 0;
    if (dot(p[top], z) > ans) ans = dot(p[top], z), id =
    top;
    int l = 1, r = top - 1;
    while (l < r) {
        int mid = l + r >> 1;
        if (dot(p[mid + 1], z) >= dot(p[mid], z)) l = mid +
        1;
        else r = mid;
    }
    if (dot(p[l], z) > ans) ans = dot(p[l], z), id = l;
    l = top + 1, r = n - 1;
    while (l < r) {
        int mid = l + r >> 1;
        if (dot(p[(mid + 1) % n], z) >= dot(p[mid], z)) l =
        mid + 1;
        else r = mid;
    }
    l %= n;
    if (dot(p[l], z) > ans) ans = dot(p[l], z), id = l;
    return id;
}
// maximum distance from any point on the perimeter to
another point on the perimeter
double diameter(vector<PT> &p) {
    int n = (int)p.size();
    if (n == 1) return 0;
    if (n == 2) return dist(p[0], p[1]);
    double ans = 0;
    int i = 0, j = 1;
    while (i < n) {
        while (cross(p[(i + 1) % n] - p[i], p[(j + 1) % n]
        - p[j]) >= 0) {
         ans = max(ans, dist2(p[i], p[j]));
         j = (j + 1) % n;
        }
        ans = max(ans, dist2(p[i], p[j]));
        i++;
    }
    return sqrt(ans);
}
// minimum distance between two parallel lines (non
necessarily axis parallel)
// such that the polygon can be put between the lines
double width(vector<PT> &p) {
    int n = (int)p.size();
    if (n <= 2) return 0;
    double ans = inf;
    int i = 0, j = 1;
    while (i < n) {
        while (cross(p[(i + 1) % n] - p[i], p[(j + 1) % n]
        - p[j]) >= 0) j = (j + 1) % n;
        ans = min(ans, dist_from_point_to_line(p[i], p[(i +
        1) % n], p[j]));
        i++;
    }
    return ans;
}
// minimum perimeter
double minimum_enclosing_rectangle(vector<PT> &p) {
 int n = p.size();
 if (n <= 2) return perimeter(p);
 int mndot = 0; double tmp = dot(p[1] - p[0], p[0]);
 for (int i = 1; i < n; i++) {
  if (dot(p[1] - p[0], p[i]) <= tmp) {
   tmp = dot(p[1] - p[0], p[i]);
   mndot = i;
  }
 }
 double ans = inf;
 int i = 0, j = 1, mxdot = 1;
 while (i < n) {
  PT cur = p[(i + 1) % n] - p[i];
        while (cross(cur, p[(j + 1) % n] - p[j]) >= 0) j =
        (j + 1) % n;
        while (dot(p[(mxdot + 1) % n], cur) >=
        dot(p[mxdot], cur)) mxdot = (mxdot + 1) % n;
        while (dot(p[(mndot + 1) % n], cur) <=
        dot(p[mndot], cur)) mndot = (mndot + 1) % n;
        ans = min(ans, 2.0 * ((dot(p[mxdot], cur) /
        cur.norm() - dot(p[mndot], cur) / cur.norm()) +
        dist_from_point_to_line(p[i], p[(i + 1) % n],
        p[j])));
        i++;
    }
    return ans;
}
// given n points, find the minimum enclosing circle of the
points
// call convex_hull() before this for faster solution
// expected O(n)
circle minimum_enclosing_circle(vector<PT> &p) {
    random_shuffle(p.begin(), p.end());
    int n = p.size();
    circle c(p[0], 0);
    for (int i = 1; i < n; i++) {
        if (sign(dist(c.p, p[i]) - c.r) > 0) {
            c = circle(p[i], 0);
            for (int j = 0; j < i; j++) {
                if (sign(dist(c.p, p[j]) - c.r) > 0) {
                    c = circle((p[i] + p[j]) / 2,
                    dist(p[i], p[j]) / 2);
                    for (int k = 0; k < j; k++) {
                        if (sign(dist(c.p, p[k]) - c.r) >
                        0) {
                            c = circle(p[i], p[j], p[k]);
                        }
                    }
                }
            }
        }
    }
    return c;
}
// returns a vector with the vertices of a polygon with
everything
// to the left of the line going from a to b cut away.
vector<PT> cut(vector<PT> &p, PT a, PT b) {
    vector<PT> ans;
    int n = (int)p.size();
    for (int i = 0; i < n; i++) {
        double c1 = cross(b - a, p[i] - a);
        double c2 = cross(b - a, p[(i + 1) % n] - a);
        if (sign(c1) >= 0) ans.push_back(p[i]);
        if (sign(c1 * c2) < 0) {
            if (!is_parallel(p[i], p[(i + 1) % n], a, b)) {
             PT tmp; line_line_intersection(p[i], p[(i + 1)
             % n], a, b, tmp);
                ans.push_back(tmp);
            }
        }
    }
    return ans;
}
// not necessarily convex, boundary is included in the
intersection
// returns total intersected length
// it returns the sum of the lengths of the portions of the
line that are inside the polygon
double polygon_line_intersection(vector<PT> p, PT a, PT b)
{
    int n = p.size();
    p.push_back(p[0]);
    line l = line(a, b);
    double ans = 0.0;
    vector< pair<double, int> > vec;
    for (int i = 0; i < n; i++) {
        int s1 = orientation(a, b, p[i]);
        int s2 = orientation(a, b, p[i + 1]);
        if (s1 == s2) continue;
        line t = line(p[i], p[i + 1]);
        PT inter = (t.v * l.c - l.v * t.c) / cross(l.v,
        t.v);
        double tmp = dot(inter, l.v);
        int f;
        if (s1 > s2) f = s1 && s2 ? 2 : 1;
        else f = s1 && s2 ? -2 : -1;
        vec.push_back(make_pair((f > 0 ? tmp - eps : tmp +
        eps), f)); // keep eps very small like 1e-12
    }
```

```
    sort(vec.begin(), vec.end());
    for (int i = 0, j = 0; i + 1 < (int)vec.size(); i++){
        j += vec[i].second;
        if (j) ans += vec[i + 1].first - vec[i].first; //
        if this portion is inside the polygon
        // else ans = 0; // if we want the maximum
        intersected length which is totally inside the
        polygon, uncomment this and take the maximum of ans
    }
    ans = ans / sqrt(dot(l.v, l.v));
    p.pop_back();
    return ans;
}
// given a convex polygon p, and a line ab and the top
vertex of the polygon
// returns the intersection of the line with the polygon
// it returns the indices of the edges of the polygon that
are intersected by the line
// so if it returns i, then the line intersects the edge
(p[i], p[(i + 1) % n])
array<int, 2> convex_line_intersection(vector<PT> &p, PT a,
PT b, int top) {
 int end_a = extreme_vertex(p, (a - b).perp(), top);
    int end_b = extreme_vertex(p, (b - a).perp(), top);
    auto cmp_l = [&](int i) { return orientation(a, p[i],
    b); };
    if (cmp_l(end_a) < 0 || cmp_l(end_b) > 0)
        return {-1, -1}; // no intersection
    array<int, 2> res;
    for (int i = 0; i < 2; i++) {
        int lo = end_b, hi = end_a, n = p.size();
        while ((lo + 1) % n != hi) {
            int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) %
            n;
            (cmp_l(m) == cmp_l(end_b) ? lo : hi) = m;
        }
        res[i] = (lo + !cmp_l(hi)) % n;
        swap(end_a, end_b);
    }
    if (res[0] == res[1]) return {res[0], -1}; // touches
    the vertex res[0]
    if (!cmp_l(res[0]) && !cmp_l(res[1]))
        switch ((res[0] - res[1] + (int)p.size() + 1) %
        p.size()) {
            case 0: return {res[0], res[0]}; // touches the
            edge (res[0], res[0] + 1)
            case 2: return {res[1], res[1]}; // touches the
            edge (res[1], res[1] + 1)
        }
    return res; // intersects the edges (res[0], res[0] +
    1) and (res[1], res[1] + 1)
}

pair<PT, int> point_poly_tangent(vector<PT> &p, PT Q, int
dir, int l, int r) {
    while (r - l > 1) {
        int mid = (l + r) >> 1;
        bool pvs = orientation(Q, p[mid], p[mid - 1]) !=
        -dir;
        bool nxt = orientation(Q, p[mid], p[mid + 1]) !=
        -dir;
        if (pvs && nxt) return {p[mid], mid};
        if (!(pvs || nxt)) {
            auto p1 = point_poly_tangent(p, Q, dir, mid +
            1, r);
            auto p2 = point_poly_tangent(p, Q, dir, l, mid
            - 1);
            return orientation(Q, p1.first, p2.first) ==
            dir ? p1 : p2;
        }
        if (!pvs) {
            if (orientation(Q, p[mid], p[l]) == dir)  r =
            mid - 1;
            else if (orientation(Q, p[l], p[r]) == dir) r =
            mid - 1;
            else l = mid + 1;
        }
        if (!nxt) {
            if (orientation(Q, p[mid], p[l]) == dir)  l =
            mid + 1;
            else if (orientation(Q, p[l], p[r]) == dir) r =
            mid - 1;
            else l = mid + 1;
        }
    }
    pair<PT, int> ret = {p[l], l};
    for (int i = l + 1; i <= r; i++) ret = orientation(Q,
    ret.first, p[i]) != dir ? make_pair(p[i], i) : ret;
    return ret;
}
// (ccw, cw) tangents from a point that is outside this
convex polygon
// returns indexes of the points
// ccw means the tangent from Q to that point is in the
same direction as the polygon ccw direction
pair<int, int> tangents_from_point_to_polygon(vector<PT>
&p, PT Q){
    int ccw = point_poly_tangent(p, Q, 1, 0, (int)p.size()
    - 1).second;
    int cw = point_poly_tangent(p, Q, -1, 0, (int)p.size()
    - 1).second;
    return make_pair(ccw, cw);
}
// minimum distance from a point to a convex polygon
// it assumes point lie strictly outside the polygon
double dist_from_point_to_polygon(vector<PT> &p, PT z) {
    double ans = inf;
    int n = p.size();
    if (n <= 3) {
        for(int i = 0; i < n; i++) ans = min(ans,
        dist_from_point_to_seg(p[i], p[(i + 1) % n], z));
        return ans;
    }
    auto [r, l] = tangents_from_point_to_polygon(p, z);
    if(l > r) r += n;
    while (l < r) {
        int mid = (l + r) >> 1;
        double left = dist2(p[mid % n], z), right=
        dist2(p[(mid + 1) % n], z);
        ans = min({ans, left, right});
        if(left < right) r = mid;
        else l = mid + 1;
    }
    ans = sqrt(ans);
    ans = min(ans, dist_from_point_to_seg(p[l % n], p[(l +
    1) % n], z));
    ans = min(ans, dist_from_point_to_seg(p[l % n], p[(l -
    1 + n) % n], z));
    return ans;
}
// minimum distance from convex polygon p to line ab
// returns 0 is it intersects with the polygon
// top - upper right vertex
double dist_from_polygon_to_line(vector<PT> &p, PT a, PT b,
int top) { //O(log n)
 PT orth = (b - a).perp();
 if (orientation(a, b, p[0]) > 0) orth = (a - b).perp();
 int id = extreme_vertex(p, orth, top);
 if (dot(p[id] - a, orth) > 0) return 0.0; //if orth and a
 are in the same half of the line, then poly and line
 intersects
 return dist_from_point_to_line(a, b, p[id]); //does not
 intersect
}
// minimum distance from a convex polygon to another convex
polygon
// the polygon doesnot overlap or touch
// tested in https://toph.co/p/the-wall
double dist_from_polygon_to_polygon(vector<PT> &p1,
vector<PT> &p2) { // O(n log n)
    double ans = inf;
    for (int i = 0; i < p1.size(); i++) {
        ans = min(ans, dist_from_point_to_polygon(p2,
        p1[i]));
    }
    for (int i = 0; i < p2.size(); i++) {
        ans = min(ans, dist_from_point_to_polygon(p1,
        p2[i]));
    }
    return ans;
}
// maximum distance from a convex polygon to another convex
polygon
double maximum_dist_from_polygon_to_polygon(vector<PT> &u,
vector<PT> &v){ //O(n)
    int n = (int)u.size(), m = (int)v.size();
    double ans = 0;
    if (n < 3 || m < 3) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) ans = max(ans,
            dist2(u[i], v[j]));
        }
        return sqrt(ans);
    }
    if (u[0].x > v[0].x) swap(n, m), swap(u, v);
    int i = 0, j = 0, step = n + m + 10;
    while (j + 1 < m && v[j].x < v[j + 1].x) j++ ;
    while (step--) {
        if (cross(u[(i + 1)%n] - u[i], v[(j + 1)%m] - v[j])
        >= 0) j = (j + 1) % m;
        else i = (i + 1) % n;
        ans = max(ans, dist2(u[i], v[j]));
    }
    return sqrt(ans);
}
// calculates the area of the union of n polygons (not
necessarily convex).
// the points within each polygon must be given in CCW
order.
// complexity: O(N^2), where N is the total number of
points
double rat(PT a, PT b, PT p) {
        return !sign(a.x - b.x) ? (p.y - a.y) / (b.y - a.y)
        : (p.x - a.x) / (b.x - a.x);
 };
double polygon_union(vector<vector<PT>> &p) {
 int n = p.size();
    double ans=0;
    for(int i = 0; i < n; ++i) {
        for (int v = 0; v < (int)p[i].size(); ++v) {
            PT a = p[i][v], b = p[i][(v + 1) %
            p[i].size()];
            vector<pair<double, int>> segs;
            segs.emplace_back(0,  0), segs.emplace_back(1,
            0);
            for(int j = 0; j < n; ++j) {
             if(i != j) {
                    for(size_t u = 0; u < p[j].size(); ++u)
                    {
                        PT c = p[j][u], d = p[j][(u + 1) %
                        p[j].size()];
                        int sc = sign(cross(b - a, c - a)),
                        sd = sign(cross(b - a, d - a));
                        if(!sc && !sd) {
                            if(sign(dot(b - a, d - c)) > 0
                            && i > j) {
                                segs.emplace_back(rat(a, b,
                                c), 1),
                                segs.emplace_back(rat(a, b,
                                d),  -1);
                            }
                        }
                        else {
                            double sa = cross(d - c, a -
                            c), sb = cross(d - c, b - c);
                            if(sc >= 0 && sd < 0)
                            segs.emplace_back(sa / (sa -
                            sb), 1);
                            else if(sc < 0 && sd >= 0)
                            segs.emplace_back(sa / (sa -
                            sb),  -1);
                        }
                    }
                }
            }
            sort(segs.begin(),  segs.end());
            double pre = min(max(segs[0].first, 0.0), 1.0),
            now, sum = 0;
            int cnt = segs[0].second;
            for(int j = 1; j < segs.size(); ++j) {
                now = min(max(segs[j].first, 0.0), 1.0);
                if (!cnt) sum += now - pre;
                cnt += segs[j].second;
                pre = now;
            }
            ans += cross(a, b) * sum;
        }
    }
    return ans * 0.5;
}
// contains all points p such that: cross(b - a, p - a) >=
0
struct HP {
    PT a, b;
    HP() {}
    HP(PT a, PT b) : a(a), b(b) {}
    HP(const HP& rhs) : a(rhs.a), b(rhs.b) {}
    int operator < (const HP& rhs) const {
        PT p = b - a;
        PT q = rhs.b - rhs.a;
        int fp = (p.y < 0 || (p.y == 0 && p.x < 0));
        int fq = (q.y < 0 || (q.y == 0 && q.x < 0));
        if (fp != fq) return fp == 0;
        if (cross(p, q)) return cross(p, q) > 0;
        return cross(p, rhs.b - a) < 0;
    }
    PT line_line_intersection(PT a, PT b, PT c, PT d) {
        b = b - a; d = c - d; c = c - a;
        return a + b * cross(c, d) / cross(b, d);
    }
    PT intersection(const HP &v) {
        return line_line_intersection(a, b, v.a, v.b);
```

```
    }
};
int check(HP a, HP b, HP c) {
    return cross(a.b - a.a, b.intersection(c) - a.a) >
    -eps; //-eps to include polygons of zero area (straight
    lines, points)
}
// consider half-plane of counter-clockwise side of each
line
// if lines are not bounded add infinity rectangle
// returns a convex polygon, a point can occur multiple
times though
// complexity: O(n log(n))
vector<PT> half_plane_intersection(vector<HP> h) {
    sort(h.begin(), h.end());
    vector<HP> tmp;
    for (int i = 0; i < h.size(); i++) {
        if (!i || cross(h[i].b - h[i].a, h[i - 1].b - h[i -
        1].a)) {
            tmp.push_back(h[i]);
        }
    }
    h = tmp;
    vector<HP> q(h.size() + 10);
    int qh = 0, qe = 0;
    for (int i = 0; i < h.size(); i++) {
        while (qe - qh > 1 && !check(h[i], q[qe - 2], q[qe
        - 1])) qe--;
        while (qe - qh > 1 && !check(h[i], q[qh], q[qh +
        1])) qh++;
        q[qe++] = h[i];
    }
    while (qe - qh > 2 && !check(q[qh], q[qe - 2], q[qe -
    1])) qe--;
    while (qe - qh > 2 && !check(q[qe - 1], q[qh], q[qh +
    1])) qh++;
    vector<HP> res;
    for (int i = qh; i < qe; i++) res.push_back(q[i]);
    vector<PT> hull;
    if (res.size() > 2) {
        for (int i = 0; i < res.size(); i++) {
            hull.push_back(res[i].intersection(res[(i + 1)
            % ((int)res.size())]));
        }
    }
    return hull;
}
// rotate the polygon such that the (bottom, left)-most
point is at the first position
void reorder_polygon(vector<PT> &p) {
  int pos = 0;
  for (int i = 1; i < p.size(); i++) {
    if (p[i].y < p[pos].y || (sign(p[i].y - p[pos].y) == 0
    && p[i].x < p[pos].x)) pos = i;
  }
  rotate(p.begin(), p.begin() + pos, p.end());
}
// a and b are convex polygons
// returns a convex hull of their minkowski sum
// min(a.size(), b.size()) >= 2
// https://cp-algorithms.com/geometry/minkowski.html
vector<PT> minkowski_sum(vector<PT> a, vector<PT> b) {
  reorder_polygon(a); reorder_polygon(b);
  int n = a.size(), m = b.size();
  int i = 0, j = 0;
  a.push_back(a[0]); a.push_back(a[1]);
  b.push_back(b[0]); b.push_back(b[1]);
  vector<PT> c;
  while (i < n || j < m) {
    c.push_back(a[i] + b[j]);
    double p = cross(a[i + 1] - a[i], b[j + 1] - b[j]);
    if (sign(p) >= 0) ++i;
    if (sign(p) <= 0) ++j;
  }
  return c;
}
// returns the area of the intersection of the circle with
center c and radius r
// and the triangle formed by the points c, a, b
double _triangle_circle_intersection(PT c, double r, PT a,
PT b) {
    double sd1 = dist2(c, a), sd2 = dist2(c, b);
    if(sd1 > sd2) swap(a, b), swap(sd1, sd2);
    double sd = dist2(a, b);
    double d1 = sqrtl(sd1), d2 = sqrtl(sd2), d = sqrt(sd);
    double x = abs(sd2 - sd - sd1) / (2 * d);
    double h = sqrtl(sd1 - x * x);
    if(r >= d2) return h * d / 2;
    double area = 0;
    if(sd + sd1 < sd2) {
        if(r < d1) area = r * r * (acos(h / d2) - acos(h /
        d1)) / 2;
        else {
            area = r * r * ( acos(h / d2) - acos(h / r)) /
            2;
            double y = sqrtl(r * r - h * h);
            area += h * (y - x) / 2;
        }
    }
    else {
        if(r < h) area = r * r * (acos(h / d2) + acos(h /
        d1)) / 2;
        else {
            area += r * r * (acos(h / d2) - acos(h / r)) /
            2;
            double y = sqrtl(r * r - h * h);
            area += h * y / 2;
            if(r < d1) {
                area += r * r * (acos(h / d1) - acos(h /
                r)) / 2;
                area += h * y / 2;
            }
            else area += h * x / 2;
        }
    }
    return area;
}
// intersection between a simple polygon and a circle
double polygon_circle_intersection(vector<PT> &v, PT p,
double r) {
    int n = v.size();
    double ans = 0.00;
    PT org = {0, 0};
    for(int i = 0; i < n; i++) {
        int x = orientation(p, v[i], v[(i + 1) % n]);
        if(x == 0) continue;
        double area = _triangle_circle_intersection(org, r,
        v[i] - p, v[(i + 1) % n] - p);
        if (x < 0) ans -= area;
        else ans += area;
    }
    return abs(ans);
}
// find a circle of radius r that contains as many points
as possible
// O(n^2 log n);
double maximum_circle_cover(vector<PT> p, double r, circle
&c) {
    int n = p.size();
    int ans = 0;
    int id = 0; double th = 0;
    for (int i = 0; i < n; ++i) {
        // maximum circle cover when the circle goes
        through this point
        vector<pair<double, int>> events = {{-PI, +1}, {PI,
        -1}};
        for (int j = 0; j < n; ++j) {
            if (j == i) continue;
            double d = dist(p[i], p[j]);
            if (d > r * 2) continue;
            double dir = (p[j] - p[i]).arg();
            double ang = acos(d / 2 / r);
            double st = dir - ang, ed = dir + ang;
            if (st > PI) st -= PI * 2;
            if (st <= -PI) st += PI * 2;
            if (ed > PI) ed -= PI * 2;
            if (ed <= -PI) ed += PI * 2;
            events.push_back({st - eps, +1}); // take care
            of precisions!
            events.push_back({ed, -1});
            if (st > ed) {
                events.push_back({-PI, +1});
                events.push_back({+PI, -1});
            }
        }
        sort(events.begin(), events.end());
        int cnt = 0;
        for (auto &&e: events) {
            cnt += e.second;
            if (cnt > ans) {
             ans = cnt;
             id = i; th = e.first;
            }
        }
    }
    PT w = PT(p[id].x + r * cos(th), p[id].y + r *
    sin(th));
    c = circle(w, r); //best_circle
    return ans;
}
// radius of the maximum inscribed circle in a convex
polygon
double maximum_inscribed_circle(vector<PT> p) {
 int n = p.size();
 if (n <= 2) return 0;
 double l = 0, r = 20000;
 while (r - l > eps) {
  double mid = (l + r) * 0.5;
  vector<HP> h;
  const int L = 1e9;
  h.push_back(HP(PT(-L, -L), PT(L, -L)));
  h.push_back(HP(PT(L, -L), PT(L, L)));
  h.push_back(HP(PT(L, L), PT(-L, L)));
  h.push_back(HP(PT(-L, L), PT(-L, -L)));
  for (int i = 0; i < n; i++) {
   PT z = (p[(i + 1) % n] - p[i]).perp();
   z = z.truncate(mid);
   PT y = p[i] + z, q = p[(i + 1) % n] + z;
   h.push_back(HP(p[i] + z, p[(i + 1) % n] + z));
  }
  vector<PT> nw = half_plane_intersection(h);
  if (!nw.empty()) l = mid;
  else r = mid;
 }
 return l;
}
// ear decomposition, O(n^3) but faster
vector<vector<PT>> triangulate(vector<PT> p) {
  vector<vector<PT>> v;
  while (p.size() >= 3) {
    for (int i = 0, n = p.size(); i < n; i++) {
      int pre = i == 0 ? n - 1 : i - 1;;
      int nxt = i == n - 1 ? 0 : i + 1;;
      int ori = orientation(p[i], p[pre], p[nxt]);
      if (ori < 0) {
        int ok = 1;
        for (int j = 0; j < n; j++) {
          if (j == i || j == pre || j == nxt)continue;
          if (is_point_in_triangle(p[i], p[pre], p[nxt] ,
          p[j]) < 1) {
            ok = 0;
            break;
          }
        }
        if (ok) {
          v.push_back({p[pre], p[i], p[nxt]});
          p.erase(p.begin() + i);
          break;
        }
      }
    }
  }
  return v;
}
struct star {
 int n;    // number of sides of the star
 double r; // radius of the circumcircle
 star(int _n, double _r) {
  n = _n;
  r = _r;
 }
 double area() {
  double theta = PI / n;
  double s = 2 * r * sin(theta);
  double R = 0.5 * s / tan(theta);
  double a = 0.5 * n * s * R;
  double a2 = 0.25 * s * s / tan(1.5 * theta);
  return a - n * a2;
 }
};
// given a list of lengths of the sides of a polygon in
counterclockwise order
// returns the maximum area of a non-degenerate polygon
that can be formed using those lengths
double
get_maximum_polygon_area_for_given_lengths(vector<double>
v) {
  if (v.size() < 3) {
    return 0;
  }
  int m = 0;
  double sum = 0;
  for (int i = 0; i < v.size(); i++) {
    if (v[i] > v[m]) {
      m = i;
    }
    sum += v[i];
  }
  if (sign(v[m] - (sum - v[m])) >= 0) {
    return 0; // no non-degenerate polygon is possible
  }
  // the polygon should be a circular polygon
  // that is all points are on the circumference of a
  circle
  double l = v[m] / 2, r = 1e6; // fix it correctly
  int it = 60;
  auto ang = [](double x, double r) { // x = length of the
  chord, r = radius of the circle
    return 2 * asin((x / 2) / r);
```

```
  };
  auto calc = [=](double r) {
    double sum = 0;
    for (auto x: v) {
      sum += ang(x, r);
    }
    return sum;
  };
  // compute the radius of the circle
  while (it--) {
    double mid = (l + r) / 2;
    if (calc(mid) <= 2 * PI) {
      r = mid;
    }
    else {
      l = mid;
    }
  }
  if (calc(r) <= 2 * PI - eps) { // the center of the
  circle is outside the polygon
    auto calc2 = [&](double r) {
      double sum = 0;
      for (int i = 0; i < v.size(); i++) {
        double x = v[i];
        double th = ang(x, r);
        if (i != m) {
          sum += th;
        }
        else {
          sum += 2 * PI - th;
        }
      }
      return sum;
    };
    l = v[m] / 2; r = 1e6;
    it = 60;
    while (it--) {
      double mid = (l + r) / 2;
      if (calc2(mid) > 2 * PI) {
        r = mid;
      }
      else {
        l = mid;
      }
    }
    auto get_area = [=](double r) {
      double ans = 0;
      for (int i = 0; i < v.size(); i++) {
        double x = v[i];
        double area = r * r * sin(ang(x, r)) / 2;
        if (i != m) {
          ans += area;
        }
        else {
          ans -= area;
        }
      }
      return ans;
    };
    return get_area(r);
  }
  else { // the center of the circle is inside the polygon
    auto get_area = [=](double r) {
      double ans = 0;
      for (auto x: v) {
        ans += r * r * sin(ang(x, r)) / 2;
      }
      return ans;
    };
    return get_area(r);
  }
}
```

7.2 Closest Pair of Points

```
pair<int, int> closest_pair(vector<pair<int, int>> a) {int
n = a.size();assert(n >= 2);vector<pair<pair<int, int>,
int>> p(n);for (int i = 0; i < n; i++) p[i] = {a[i],
i};sort(p.begin(), p.end());int l = 0, r = 2;long long ans
= dist2(p[0].x, p[1].x);pair<int, int> ret = {p[0].y,
p[1].y};while (r < n) {while (l < r && 1LL * (p[r].x.x -
p[l].x.x) * (p[r].x.x - p[l].x.x) >= ans) l++;for (int i =
l; i < r; i++) {long long nw = dist2(p[i].x, p[r].x);if (nw
< ans) {ans = nw;ret = {p[i].y, p[r].y};}}r++;}return
ret;}pair<int, int> z = closest_pair(p);
```

8 Misc

8.1 Submask Enumeration

Time Complexity: $\mathcal{O}(3^N)$

```
for (int m=0; m<(1<<n); ++m)
for (int s=m; s; s=(s-1)&m)
{
// do something m=mask, s=submask
}
```

8.2 int128 template

```
__int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}
bool cmp(__int128 x, __int128 y) { return x > y; }
```

8.3 Bash File

```
// file name as "s.sh'
// run: bash s.sh
#set -e
#g++ -std=c++17 -O2 -Wshadow -Wall -Wextra
-Wshift-overflow=2 -fno-sanitize-recover -fstack-protector
-o "%e" -g -D_GLIBCXX_DEBUG a.cpp -o a
#g++ -std=c++17 test.cpp -o test
#g++ -std=c++17 brute.cpp -o brute
ok=1
for((i = 1; i < 100; ++i)); do
    ./gen $i > input_file
    ./a < input_file > myAnswer
    ./brute < input_file > correctAnswer
    if ! diff -Z myAnswer correctAnswer > /dev/null; then
        ok=0
        break
    fi
    echo "Passed test: "  $i
done
if [ $ok -eq 0 ]; then
 echo "WA on the following test:"
 cat input_file
 echo "Your answer is:"
 cat myAnswer
 echo "Correct answer is:"
 cat correctAnswer
fi
echo "finished"
```

```
Checker.cpp must include
//ifstream fin("input_file", ifstream::in);
//ifstream ans("myAnswer", ifstream::in);
// ifstream cor("correctAnswer", ifstream::in);
```

8.4 Test Generator

```
mt19937 rng(chrono::steady_clock::now().time
_since_epoch().count());
int rand(int l, int r){
 uniform_int_distribution<int> uid(l, r);
 return uid(rng);}
// Random n numbers between l and r
void num(int l, int r, int n) {
 for (int i = 0; i < n; ++i) cout << rand(l,r) << " ";}
//Random n real numbers between l and r with dig decimal
places
void real(int l, int r, int dig, int n) {
 for (int i = 0; i < n; ++i) cout << rand(l,r)
 <<"."<<rand(0,pow(10,dig)-1)<< " ";
}
// Random n strings of length l
void str(int l, int n) {
 for (int i = 0; i < n; ++i){
      for(int j = 0; j < l; ++j) {
          int v = rand(1,150);
          if(v%3==0) cout<<(char)rand('a','z');
          else if(v%3==1) cout<<(char)rand('A','Z');
          else cout<<rand(0,9);
      }
      cout<<" ";}}
// Random n strings of max length l
void strmx(int mxlen, int n) {
 for (int i = 0; i < n; ++i){
      int l = rand(1,mxlen);
      for(int j = 0; j < l; ++j) {
          int v = rand(1,150);
          if(3%3==0) cout<<(char)rand('a','z');
          else if(v%3==1) cout<<(char)rand('A','Z');
          else cout<<rand(0,9);
      }
      cout<<" ";}}
// Random tree of n nodes
void tree(int n) {
 int prufer[n-2];
 for ( int i = 0; i < n; i++ ){
      prufer[i] = rand(1,n);}
 int m = n-2;
 int vertices = m + 2;
 int vertex_set[vertices];
 for (int i = 0; i < vertices; i++) vertex_set[i] = 0;
 for (int i = 0; i < vertices - 2; i++)
 vertex_set[prufer[i] - 1] += 1;
 int j = 0;
 for (int i = 0; i < vertices - 2; i++) {
      for (j = 0; j < vertices; j++) {
          if (vertex_set[j] == 0) {
              vertex_set[j] = -1;
              cout << (j + 1) << " "
                   << prufer[i] << '\n';
              vertex_set[prufer[i] - 1]--;
              Break;
          }}}
 j = 0;
 for (int i = 0; i < vertices; i++) {
      if (vertex_set[i] == 0 && j == 0) {
          cout << (i + 1) << " ";
          j++;}
      else if (vertex_set[i] == 0 && j == 1)
          cout << (i + 1) << "\n";}
}
// from errichto
Void tree2(int argc, char* argv[]) {
 srand(atoi(argv[1]));
 int n = rand(2, 20);
 printf("%d\n", n);
 vector<pair<int,int>> edges;
 for(int i = 2; i <= n; ++i) {
      edges.emplace_back(rand(1, i - 1), i); }
 vector<int> perm(n + 1); // re-naming vertices
 for(int i = 1; i <= n; ++i) { perm[i] = i; }
 random_shuffle(perm.begin() + 1, perm.end());
 random_shuffle(edges.begin(), edges.end()); // random
 order of edges
 for(pair<int, int> edge : edges) {
      int a = edge.first, b = edge.second;
      if(rand() % 2) {
          swap(a, b); // random order of two vertices
      }
      printf("%d %d\n", perm[a], perm[b]);}}
```

8.5 Team Main Template

```
typedef long long ll;typedef long double ld;
#define endl "\n"#define all(a) a.begin(), a.end()
#define pb push_back#define mp make_pair
ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0);
```

8.6 Pragma Optimization

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
// #pragma GCC optimize("Ofast,unroll-loops")
// #pragma GCC target("avx2,tune=native")
```

8.7 Ordered Multiset

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> indexed_set;
//  .. operations ..
//  same ones as set..
///  extra: 1. s.order_of_key(x) // returns order of x;
///  extra: 2. s.find_by_order(K) // returns K-th element;
```