TEAM REFERENCE DOCUMENT

METROPOLITAN UNIVERSITY

এমইউ_অনির্বাণ

Amanur Rahman

Md. Mahbub Abedin Talukder

Mushfiqur Rahman

Coach : Ranit Debnath Akash

*** Data Structure ***

Binary Indexed Tree

```
int tree[MAX];
void update(int idx,int x,int n) {
    while(idx <= n){
        tree[idx] += x;
        idx += idx&(-idx);  }
}
int query(int idx) {
    int sum = 0;
    while(idx > 0) {
        sum += tree[idx];
        idx -= idx&(-idx);  }
    return sum;  }
```

Merge Sort Tree

```
ll cum[MAX];
vector<ll>tree[4*MAX];
void build(int l,int r,int node) {
    if(l == r){
        tree[node].push_back(cum[l]);
        return;  }
    int mid = (l+r)/2;
    int left = 2*node, right =
2*node+1;
    build(l,mid,left);
    build(mid+1,r,right);
merge(tree[left].begin(),tree[left].end
(),tree[right].begin(),tree[right].end(
),back_inserter(tree[node]));    }
int query(int L,int R,int l,int r,int
node,ll t){
    if(l > R || r < L)
        return 0;
    else if(l>=L && r<=R)
        return
lower_bound(tree[node].begin(),tree[nod
e].end(),t)-tree[node].begin();
    int mid = (l+r)/2;
    return query(L,R,l,mid,2*node,t) +
query(L,R,mid+1,r,2*node+1,t);
}
```

Maximum Histogram

```
ll maxHistogram(vector<ll> &hist,int n)
{
    stack<int>st;
    ll mx = -1;
    int i = 0;
    while(i <= n) {
        ll h = (i == n) ? 0 : hist[i];
        if(st.empty() || hist[i] >=
hist[st.top()])
            st.push(i++);
        else {
            int top = st.top();
            st.pop();
            mx = max(mx, hist[top] *
(st.empty() ? i : i-1-st.top()));   }
    }
    return mx;   }
```

Policy Based Data Structure

```
#include <bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
template <typename T> using Set =
tree<T,
```

```
null_type,less<T>,rb_tree_tag,tree_orde
r_statistics_node_update>;

Set <int> st;

int main() {

    st.insert(5); //Insert

    st.erase(5); //Delete

    st.insert(1);

    st.insert(2);

    st.insert(9);

    cout << *st.find_by_order(0) <<
endl; //Find value by rank

    cout << st.order_of_key(9) << endl;
//Find value's rank

    /* For multiple same element,

    use pair, store index in second

    of pair */  }
```

Persistent Segment Tree

```
struct Node {

    int left, right, val;

} tree[MAX*20];

int a[MAX], root[MAX], id;

void build(int pos,int l,int r) {

    if(l == r) {

        tree[pos].val = a[l];

        return; }

    int mid = (l+r)>>1;

    tree[pos].left = ++id,
tree[pos].right = ++id;

    build(tree[pos].left,l,mid);

    build(tree[pos].right,mid+1,r);

    tree[pos].val =
tree[tree[pos].left].val +
tree[tree[pos].right].val;

}

int update(int pos,int l,int r,int
idx,int v) {

    if(idx > r || idx < l)

        return pos;

    else if(l == r) {

        tree[++id] = tree[pos];

        tree[id].val += v;

        return id;  }

    int mid = (l+r)>>1;

    tree[++id] = tree[pos], pos = id;

    tree[pos].left =
update(tree[pos].left,l,mid,idx,v);

    tree[pos].right =
update(tree[pos].right,mid+1,r,idx,v);

    tree[pos].val =
tree[tree[pos].left].val +
tree[tree[pos].right].val;

    return pos;   }

int query(int pos,int l,int r,int L,int
R) {

    if(l > R || r < L)

        return 0;

    else if(l >= L && r <= R)

        return tree[pos].val;

    int mid = (l+r)/2;

    int x =
query(tree[pos].left,l,mid,L,R);

    int y =
query(tree[pos].right,mid+1,r,L,R);

    return x+y;   }

int kthElement(int a,int b,int l,int
r,int k) {

    if(l == r)

        return l;
```

```
    int mid = (l+r)>>1;
    int cnt = tree[tree[a].left].val - tree[tree[b].left].val;
    if(cnt >= k)
        return kthElement(tree[a].left,tree[b].left,l,mid,k);
    else
        return kthElement(tree[a].right,tree[b].right,mid+1,r,k-cnt);    }
int lessCnt(int a,int b,int l,int r,int idx) {
    if(r <= idx)
        return tree[a].val - tree[b].val;
    int mid = (l+r)>>1;
    if(idx <= mid)
        return lessCnt(tree[a].left,tree[b].left,l,mid,idx);
    else
        return lessCnt(tree[a].left,tree[b].left,l,mid,idx) + lessCnt(tree[a].right,tree[b].right,mid+1,r,idx);    }
void init(int n,int m) {
    root[0] = tree[0].left = tree[0].right = tree[0].val = 0;
    for(int i=1;i<=n;i++)
        root[i] = update(root[i-1] , 1 , m , mp[a[i]]);    }
```

Kth Element in a range

```
int findKth(int pos,int l,int r,int k) {
    if(l == r)
        return l;
    int mid = (l+r) >> 1;
    if(tree[pos*2] >= k)
        return findKth(pos*2,l,mid,k);
    else
        return findKth(pos*2+1,mid+1,r,k-tree[pos*2]);
}
```

Sliding RMQ

```
vector<int> slidingRMQ(int a[],int n,int k) {
    deque<int>d;
    vector<int>res;
    for(int i=0;i<n;i++) {
        while(!d.empty() && d.front() >= a[i])
            d.pop_front();
        d.push_front(a[i]);
        if(i>=k && a[i-k] == d.back())
            d.pop_back();
        if(i >= k-1)
            res.push_back(d.back());
    }
    return res;   }
```

Trie

```
bool Check(int n,int pos) {
    return (n & (1<<pos));   }
void Set(int &n,int pos) {
    n = n | (1<<pos);   }
void Clear(int &n,int pos) {
    n = n & ~(1<<pos);
}
```

```
struct Trie {
    Trie *zero, *one;
    Trie() {
        zero = NULL;
        one = NULL; }   };
Trie *root;
void Insert(int n) {
    Trie *cur = root;
    for(int i=30; i>=0; i--) {
        if(Check(n,i)) {
            if(cur->one == NULL)
                cur->one = new Trie();
            cur = cur->one;    }
        else {
            if(cur->zero == NULL)
                cur->zero = new Trie();
            cur = cur->zero;    }
    }    }
int getMax(int n) {
    Trie *cur = root;
    for(int i=30; i>=0; i--) {
        if(Check(n,i)) {
            if(cur->zero)
                cur = cur->zero;
            else
                cur = cur->one,
Clear(n,i);     }
        else {
            if(cur->one)
                cur = cur->one,
Set(n,i);
            else
                cur = cur->zero;   }
    }
    return n;    }
int getMin(int n) {
    Trie *cur = root;
    for(int i=30; i>=0; i--) {
        if(Check(n,i)) {
            if(cur->one)
                cur = cur->one,
Clear(n,i);
            else
                cur = cur->zero;   }
        else {
            if(cur->zero)
                cur = cur->zero;
            else
                cur = cur->one,
Set(n,i);     }
    }
    return n;     }
void Delete(Trie *cur) {
    if(cur->one)
        Delete(cur->one);
    if(cur->zero)
        Delete(cur->zero);
    delete(cur);   }
```

Centroid Decomposition

```
struct CentroidDecomposition {
    int path[MAX] , sub[MAX];
    bool vis[MAX];
    CentroidDecomposition() {
        memset(vis,0,sizeof vis);
```

```
        memset(path,0,sizeof path);  }
    void subDFS(int src,int par) {
        sub[src] = 1;
        for(auto i : adj[src]) {
            if(i == par || vis[i])
                continue;
            subDFS(i,src);
            sub[src] += sub[i];  }
    }
    int centroid(int src,int par,int
sz) {
        for(auto i : adj[src]) {
            if(i == par || vis[i])
                continue;
            else if(sub[i] > sz)
                return
centroid(i,src,sz);    }
        return src;    }
    void decompose(int src,int par) {
        subDFS(src,-1);
        int c = centroid(src,-
1,sub[src]/2);
        vis[c] = 1;
        path[c] = par;
        for(auto i : adj[c]) {
            if(!vis[i])
                decompose(i,c);  }
    }
} tree;
bool color[MAX];
multiset<int>data[MAX];
struct QueryHandler {
    void update(int u) {
        color[u] ^= 1;
        int cur = u;
        while(cur != -1) {
            if(color[u])

data[cur].insert(dist(u,cur));
            else

data[cur].erase(data[cur].find(dist(u,c
ur)));
            cur = tree.path[cur];   }
    }
    int query(int u) {
        int cur = u , ret = 1e9;
        while(cur != -1) {
            if(data[cur].size())
                ret = min(ret ,
*data[cur].begin() + dist(u,cur) );
            cur = tree.path[cur];   }
        if(ret == 1e9)
            ret = -1;
        return ret;   }
} ds;
->DFS(1,1,0);
->initLCA();
->tree.decompose(1,-1);
->ds.update(u);
->ds.query(u)
```

Lowest Common Ancestor

```
int dep[MAX] , T[MAX] , P[MAX][30];
void DFS(int src,int par,int lev) {
```

```
    dep[src] = lev;
    T[src] = par;
    for(int i=0;i<adj[src].size();i++){
        int x = adj[src][i];
        if(x == par)
            continue;
        DFS(x,src,lev+1);  }
}
void initLCA(int n) {
    memset(P,-1,sizeof P);
    for(int i=1;i<=n;i++)
        P[i][0] = T[i];
    for(int j=1; 1<<j <n;j++) {
        for(int i=1;i<=n;i++) {
            if(P[i][j-1] != -1)
                P[i][j] = P[P[i][j-
1]][j-1];   }
    }
}
int query(int n,int u,int v) {
    if(dep[u] < dep[v])
        swap(u,v);
    int log = 1;
    while(1) {
        int next = log+1;
        if((1<<next) > dep[u])
            break;
        log++;  }
    for(int i=log;i>=0;i--) {
        if(dep[u]-(1<<i) >= dep[v])
            u = P[u][i];   }
    if(u == v)
        return u;
    for(int i=log;i>=0;i--) {
        if(P[u][i] != -1 && P[u][i] !=
P[v][i]) {
            u = P[u][i];
            v = P[v][i];   }
    }
    return T[u];   }
-> DFS(1,1,0);
-> initLCA(n);
```

Heavy-Light Decomposition

```
vector<int>adj[MAX];
int a[MAX];
int chainNo, ptr, chainHead[MAX],
chainPos[MAX], chainIdx[MAX] , sub[MAX]
, maxSub[MAX];
int arr[MAX], tree[4*MAX];
int dep[MAX], T[MAX], P[MAX][20];
void HLD(int cur,int par) {
    if(chainHead[chainNo] == -1)
        chainHead[chainNo] = cur;
    chainIdx[cur] = chainNo;
    chainPos[cur] = ++ptr;
    arr[ptr] = a[cur];
    if(maxSub[cur] != -1)
        HLD(maxSub[cur],cur);
    for(int j=0; j<adj[cur].size();
j++) {
        int i = adj[cur][j];
        if(i != par && i !=
maxSub[cur])
            chainNo++, HLD(i,cur);
```

```
    }
}
int query_up(int u,int v,const int& n){
    int uchain , vchain = chainIdx[v] , ans = 0;
    while(chainIdx[u] != vchain) {
        uchain = chainIdx[u];
        ans += BIT_Query(chainPos[chainHead[uchain]],chainPos[u],n);
        u = chainHead[uchain];
        u = P[u][0];   }
    ans += BIT_Query(chainPos[v],chainPos[u],n);
    return ans;    }
void ansUpdate(int i,int v,const int& n) {
    BIT_Update(chainPos[i],-a[i],n);
    BIT_Update(chainPos[i],v,n);
    a[i] = v;    }
int ansQuery(const int& n,int u,int v){
    int lca = LCA_query(n,u,v);
    int q1 = query_up(u,lca,n);
    int q2 = query_up(v,lca,n);
    return q1+q2-a[lca];    }
-> DFS(1,-1,0); -> BIT_UPDATE()
-> initLCA(n); -> ansUpdate();
-> HLD(1,-1) -> ansQuery();
```

MO's Algo

```
struct data {
int l,r,idx,k;
    bool operator<(const data &b) const {
        int x = l/BLOCK_SIZE, y = b.l/BLOCK_SIZE;
        if(x != y)
            return x < y;
        return r < b.r;   }      };
int BLOCK_SIZE;
ll cnt, ans[MAX];
ll n, q, a[MAX], freq[MAX];
data Q[MAX];
void add(ll x) {
    freq[x]++;
    if(freq[x] == 1)
        cnt++;    }
void del(ll x) {
    freq[x]--;
    if(freq[x] == 0)
        cnt--;    }
void MO() {
    BLOCK_SIZE = sqrt(n);
    sort(Q,Q+q,cmp);
    int st = 1, en = 0;
    for(int i=0; i<q; i++) {
        int l = Q[i].l, r = Q[i].r , idx = Q[i].idx;
        while(en < r) { en++; add(a[en]); }
        while(en > r) { del(a[en]); en--; }
        while(st > l) { st--; add(a[st]); }
        while(st < l) { del(a[st]); st++; }
        ans[idx] = cnt;     }     }
```

Next Greater Element

```
vector<int>
nextGreaterElement(vector<int> &v) {
    int n = v.size();
    vector<int> ret(n+1,n);
    stack<int>s;
    for(int i=n-1;i>=0;i--) {
        while(!s.empty() && v[i] >=
v[s.top()])
            s.pop();
        if(!s.empty())
            ret[i] = s.top();
        s.push(i);   }
    return ret;    }
```

Suffix Array

```
struct Info {
    int prev , now , pos;    };
int sa[MAX] , P[LOGN][MAX] , lcp[MAX] ,
logn , n;
Info L[MAX];
string s;
bool cmp(Info a,Info b) {
    if(a.prev == b.prev)
        return a.now < b.now;
    return a.prev < b.prev;    }
bool cmp2(int i,int j) {
    return P[logn][i] < P[logn][j];
}
void buildSuffixArray() {
    for(int i=0;i<n;i++)
        P[0][i] = s[i]-'a' , sa[i] = i;
    int gap = 1 , step = 1;
    while(gap < n) {
        for(int i=0;i<n;i++) {
            L[i].prev = P[step-1][i];
            L[i].now = (i+gap < n) ?
P[step-1][i+gap] : -1;
            L[i].pos = i;    }
        sort(L,L+n,cmp);
        for(int i=0;i<n;i++) {
            if(i && L[i].prev == L[i-
1].prev && L[i].now == L[i-1].now)
                P[step][L[i].pos] =
P[step][L[i-1].pos];
            else
                P[step][L[i].pos] = i;
        }
        step++ , gap *= 2;    }
    logn = step-1;
    sort(sa,sa+n,cmp2);     }
void buildLCP() {
    lcp[0] = 0;
    for(int i=1;i<n;i++) {
        int x = sa[i] , y = sa[i-1];
        lcp[i] = 0;
        for(int j=logn;j>=0 && x<n &&
y<n;j--) {
            if(P[j][x] == P[j][y]) {
                lcp[i] += (1<<j);
                x += (1<<j);
                y += (1<<j);
            }
        }
    }
}
```

```
-> n = s.size();
-> buildSuffixArray();
-> buildLCP();
```

Sparse Table

```
int P[MAX][20];
void buildSparseTable(int arr[], int n)
{
    for(int i = 1; i <= n; i++)
        P[i][0] = arr[i];
    for(int j = 1; (1 << j) <= n; j++)
{
        for(int i = 1; (i + (1 << j) -
1) <= n; i++)
            P[i][j] = max(P[i][j - 1] ,
P[i + (1 << (j - 1))][j - 1]);
    }
}
int query(int L, int R) {
    int log = log2(R - L + 1);
    return max(P[L][log] , P[R - (1 <<
log) + 1][log]);
    ///return min(P[L][log] , P[R - (1
<< log) + 1][log]);
}
```

*** String ***

KMP

```
vector<int> createLPSArray(string
pattern) {
    vector<int> lps(pattern.size());
    int index = 0;
    for(int i=1;i<pattern.size();) {
        if(pattern[index] ==
pattern[i]) {
            lps[i] = index+1;
            index++ , i++;   }
        else {
            if(index)
                index = lps[index-1];
            else
                lps[i] = index , i++; }
    }
    return lps;  }
bool KMP(string text,string pattern) {
    vector<int> lps =
createLPSArray(pattern);
    int i = 0 , j = 0;
    while(i < text.size()) {
        if(text[i] == pattern[j])
            i++ , j++;
        else {
            if(j)
                j = lps[j-1];
            else
                i++;   }
        if(j == pattern.size())
            return true;   }
    return false;   }
int countKMP(string text,string
pattern) {
    vector<int> lps =
createLPSArray(pattern);
    int i = 0 , j = 0 , cnt = 0;
    while(i < text.size()) {
        if(text[i] == pattern[j])
            i++ , j++;
        else {
```

```
            if(j)
                j = lps[j-1];
            else
                i++;     }
        if(j == pattern.size()) {
            cnt++;
            j = lps[j-1];   }
    }
    return cnt;   }
```

Z-Algorithm

```
string S;
int z[MAX];
void zFunction() {
    int left, right;
    left = right = z[0] = 0;
    for(int i=1; i<S.size(); i++) {
        if(i <= right)
            z[i] = min(z[i-left],right-
i+1);
        while(i+z[i] < S.size() &&
S[i+z[i]] == S[z[i]])
            z[i]++;
        if(i+z[i]-1 > right)
            left = i, right = i+z[i]-1;
    }
}
bool isSubstr(string t,string p) {
    S = p + "#" + t;
    zFunction();
    for(int i=p.size()+1; i<S.size();
i++) {
        if(z[i] == p.size())
            return true;   }
    return false;   }
int countSubstr(string t,string p) {
    S = p + "#" + t;
    memset(z,0,sizeof z);
    zFunction();
    int cnt = 0;
    for(int i=p.size()+1; i<S.size();
i++) {
        if(z[i] == p.size())
            cnt++;    }
    return cnt;    }
int countNonOverlappingSubstr(string
t,string p) {
    S = p + "#" + t;
    memset(z,0,sizeof z);
    zFunction();
    int cnt = 0;
    for(int i=p.size()+1; i<S.size();
i++) {
        if(z[i] == p.size())
            cnt++ , i = i+z[i]-1;   }
    return cnt;   }
```

Hashing

```
ll base = 1331 , pw[MAX];
void preCalc() {
    pw[0] = 1;
    for(int i=1;i<MAX;i++)
        pw[i] = pw[i-1]*base;   }
ll H[MAX];
void setHash(string s) { ///s="#"+s
    H[0] = 0;
```

```
    for(int i=1;i<s.size();i++)
        H[i] = H[i-1]*base+s[i];   }
ll getHash(int l,int r) {
    return H[r]-(H[l-1]*pw[r-l+1]);  }
ll Hasher(string s) {
    ll hashValue = 0;
    for(int i=0;i<s.size();i++)
        hashValue =
hashValue*base+s[i];
    return hashValue;   }
```

*** Graph ***

MaxFlow

```
struct Edge {
    int u , v , cap;
    Edge(){}
    Edge(int _u,int _v,int _cap){ u =
_u , v = _v , cap = _cap; }    };
vector<Edge>edges;
vector<int>adj[MAX];
bool vis[MAX];
int s,t;
void init(int _s,int _t) {
    s = _s , t = _t;
    for(int i=0;i<MAX;i++)
        adj[i].clear();
    edges.clear();    }
void addEdge(int u,int v,int w) {
    edges.push_back(Edge(u,v,w));
    edges.push_back(Edge(v,u,0));
    adj[u].push_back(edges.size()-2);
    adj[v].push_back(edges.size()-1); }
int pushFlow(int u,int flow = 1e9) {
    vis[u] = 1;
    if(u == t)
        return flow;
    int ret = 0;
    for(int i=0;i<adj[u].size();i++) {
        int idx = adj[u][i];
        Edge &e = edges[idx];
        if(!e.cap || vis[e.v])
            continue;
        ret =
pushFlow(e.v,min(flow,e.cap));
        if(ret) {
            Edge &rev = edges[idx^1];
            e.cap -= ret;
            rev.cap += ret;
            return ret;    }
    }
    return ret;    }
int maxFlow() {
    int ans = 0;
    while(1) {
        memset(vis,0,sizeof vis);
        int flow = pushFlow(s);
        if(flow == 0)
            break;
        ans += flow;    }
    return ans;     }
```

Articular Bridges

```
vector<int>adj[MAX];
set<pii>bridges;
bool vis[MAX];
```

```
int dist[MAX] , lowTime[MAX] , timo;
void init() {
    timo = 0;
    memset(vis,0,sizeof vis);
    memset(dist,0,sizeof dist);
    memset(lowTime,0,sizeof lowTime);
    for(int i=0;i<MAX;i++)
        adj[i].clear();   }
void addBridge(int u,int v) {
    if(u > v)
        swap(u,v);
    bridges.insert({u,v});   }
void DFS(int src,int par) {
    vis[src] = 1;
    dist[src] = lowTime[src] = timo++;
    for(auto i : adj[src]) {
        if(i == par)
            continue;
        if(!vis[i]) {
            DFS(i,src);
            lowTime[src] =
min(lowTime[src],lowTime[i]);
            if(dist[src] < lowTime[i])
                addBridge(i,src);
        }
        else
            lowTime[src] =
min(lowTime[src],dist[i]);
    }     }
void findBridges() {
    DFS(1,-1);
    for(auto i : bridges)
        cout << i.first << "->" <<
i.second << endl;   }
```

Articulation Points

```
vector<int>adj[MAX];
bool vis[MAX], isArt[MAX];
int dist[MAX], lowTime[MAX], timo;
void init() {
    timo = 1;
    memset(dist,0,sizeof dist);
    memset(lowTime,0,sizeof lowTime);
    memset(vis,0,sizeof vis);
    memset(isArt,0,sizeof isArt);
    for(int i=0; i<MAX; i++)
        adj[i].clear();    }
void DFS(int src,int par) {
    vis[src] = 1;
    dist[src] = lowTime[src] = timo++;
    int child = 0;
    for(auto i : adj[src]) {
        if(i == par)
            continue;
        if(!vis[i]) {
            DFS(i,src);
            lowTime[src] =
min(lowTime[src],lowTime[i]);
            if(dist[src] <= lowTime[i]
&& par != -1)
                isArt[src] = 1;
            child++;     }
        else
            lowTime[src] =
min(lowTime[src],dist[i]);
    }
```

```
    if(child > 1 && par == -1)
        isArt[src] = 1;    }
void findArticulationPoints(int n) {
    DFS(1,-1);
    for(int i=1; i<=n; i++) {
        if(isArt[i])
            cout << i << " -> ";   }
    cout << endl;   }
```

Floyd Warshall

```
void floydWarsh(){
    for(int via=1; via<=n; via++) {
        for(int from=1; from<=n;
from++) {
            for(int to=1; to<=n; to++)
{

if(adj[from][via]+adj[via][to] <
adj[from][to]) {
                    adj[from][to] =
adj[from][via]+adj[via][to];
                    path[from][to] =
path[via][to];
                }
            }
        }
    }
}
```

Strongly Connected Components

```
vector < int > graph[MAX] ,
reverseGraph[MAX] , components[MAX];
bool vis[MAX];
int compCount;
stack<int>nodes;
void DFS(int src) {
    vis[src] = 1;
    for(auto i : graph[src]) {
        if(!vis[i])
            DFS(i);   }
    nodes.push(src);   }
void DFS2(int src) {
    vis[src] = 1;
    for(auto i : reverseGraph[src]) {
        if(!vis[i])
            DFS2(i);    }
  components[compCount].push_back(src);
}
void init() {
    compCount = 1;
    for(int i=1;i<MAX;i++)
        graph[i].clear() ,
reverseGraph[i].clear() ,
components[i].clear();    }
void addEdge(int u,int v) {
    graph[u].push_back(v);
    reverseGraph[v].push_back(u);  }
void kosaraju_SCC(int n) {
    memset(vis,0,sizeof vis);
    for(int i=1;i<=n;i++) {
        if(!vis[i])
            DFS(i);    }
    memset(vis,0,sizeof vis);
    while(nodes.size()) {
        int top = nodes.top();
        nodes.pop();
        if(!vis[top]) {
            DFS2(top);
```

```
            compCount++;    }
    }
}
```

Bipartite Matching

```
vector < int > adj[MAX];
bool vis[MAX];
int match[MAX];
bool DFS(int src) {
    for(int i=0;i<adj[src].size();i++)
{
        int x = adj[src][i];
        if(vis[x])
            continue;
        vis[x] = 1;
        if(match[x] == -1 ||
DFS(match[x])) {
            match[x] = src;
            match[src] = x;
            return 1;    }
    }
    return 0;     }
int BPM(int n) {
    int ans = 0;
    for(int i=0;i<n;i++) {
        memset(vis,0,sizeof vis);
        if(DFS(i))
            ans++;   }
    return ans;    }
void init() {
    for(int i=0;i<MAX;i++)
        adj[i].clear();
    memset(match,-1,sizeof match);    }
```

Kruskal's MST

```
struct edge {
    int u,v,w;   };
vector < edge > adj;
int N,E,par[mx];
bool cmp(edge a,edge b) {
    return a.w < b.w;   }
int Find(int n) {
    if(par[n] == n)
        return n;
    return par[n] = Find(par[n]);   }
int kruskal()  {
    sort(adj.begin(),adj.end(),cmp);
    int ans = 0 , cnt = 0 , uu , vv;
    for(int i=1;i<=N;i++)
        par[i] = i;
    for(int i=0; i<E; i++) {
        uu = Find(adj[i].u) , vv =
Find(adj[i].v);
        if(uu != vv) {
            par[uu] = vv;
            cnt++;
            ans += adj[i].w;
            if(cnt == N-1)
                break;    }
    }
    return ans;   }
```

*** Matrix ***

Matrix Exponentiation

```
struct Matrix {
    int n,m;
```

```
    vector< vector<int> > mat;
    Matrix() {}
    Matrix(int _n,int _m) {
        n = _n, m = _m;
        mat = vector< vector<int> >
(n,vector<int>(m));   }   };
Matrix multiply(Matrix a,Matrix b,int
MOD) {
    Matrix c = Matrix(a.n,b.m);
    for(int i=0; i<a.n; i++) {
        for(int j=0; j<b.m; j++) {
            c.mat[i][j] = 0;
            for(int k=0; k<a.m; k++) {
                c.mat[i][j] += (1LL *
a.mat[i][k] * b.mat[k][j])%MOD;
                if(c.mat[i][j] >= MOD)
                    c.mat[i][j] -= MOD;
            }
        }
    }
    return c;   }
Matrix pow(Matrix a,ll p,int MOD) {
    if(p == 1)
        return a;
    Matrix x = pow(a,p/2,MOD);
    x = multiply(x,x,MOD);
    if(p&1)
        x = multiply(x,a,MOD);
    return x;   }
Matrix createA() {
    Matrix a = Matrix(2,2);
    .........
    return a;   }
Matrix createB(int a,int b) {
    Matrix bb = Matrix(2,1);
    .........
    return bb;   }
-> Matrix M = pow(createA(),n-2,MOD);
-> ans =
multiply(M,createB(),MOD).mat[0][0];
```

*** Number Theory ***

ModInverse using ExtendedGCD

```
ll extendedGCD(ll a,ll b,ll *x,ll *y) {
    if(a == 0) {
        *x = 0 , *y = 1;
        return b;  }
    ll x1 , y1;
    ll gcd =
extendedGCD(b%a,a,&x1,&y1);
    *x = y1 - (b/a)*x1;
    *y = x1;
    return gcd;    }
ll modInverse(ll a,ll M) {
    if(__gcd(a,M) > 1)
        return -1;
    ll x , y;
    ll gcd = extendedGCD(a,M,&x,&y);
    return (x+M)%M;   }
int main() {
    cout << modInverse(3,11) << endl;
    /// ans = 4 , because (4*3)%11 =
12%11 = 1    }
```

Chinese Remainder Theorem

```
int n;
ll num[MAX] , rem[MAX];
ll chineseRemainderTheorem() {
    ll res = 0 , prod = 1;
    for(int i=0;i<n;i++)
        prod *= num[i];
    for(int i=0;i<n;i++) {
        ll pp = prod/num[i];
        res += rem[i] * pp *
modInverse(pp,num[i]);
        res %= prod;    }
    return res;    }
```

Divisors of N!

```
ll factorialDivisors(ll n) {
    ll res = 1;
    for(int i=0;primes[i]<=n;i++) {
        ll exp = 0;
        ll p = primes[i];
        while(p <= n) {
            exp += (n/p);
            p *= primes[i];
        }
        res *= (exp+1);
    }
    return res;   }
```

Trailing Zeroes of N!

```
int trailingZeroes(int n) {
    int cnt = 0 , f = 5;
    while(f <= n) {
        cnt += n/f;
        f *= 5;   }
    return cnt;   }
```

Euler Phi

```
int phi[MAX],mark[MAX];
void initPhi() {
    for(int i=1;i<MAX;i++)
        phi[i] = i;
    mark[1] = 1;
    for(int i=2;i<MAX;i++) {
        if(!mark[i]) {
            for(int j=i;j<MAX;j+=i) {
                mark[j] = 1;
                phi[j] = phi[j]/i*(i-
1);
            }
        }
    }
}
```

Phi for a large number

```
int phi(int x) {
    int res = x;
    for(int i = 2; i * i <= x; i++) {
        if(x % i == 0) {
            while(x % i == 0)
                x /= i;
            res -= res/i;
        }
    }
    if(x > 1)
        res -= res/x;
    return res;   }
```

Digits of N!

```
int findDigits(int n) {
    if(n<=1)
        return n;
    double digits = 0;
    for(int i=2;i<=n;i++)
        digits += log10(i);
    return floor(digits)+1;   }
```

N! under modulo P

```
int largestPower(int n,int p) {
    int cnt = 0;
    while(n) {
        n /= p;
        cnt += n;   }
    return cnt;   }
int fact(int n,int p) {
    int res = 1;
    for(int i=0;primes[i]<=n;i++) {
        int k =
largestPower(n,primes[i]);
        res =
(res*bigMod(primes[i],k,p))%p;
    }
    return res;   }
```

PrimeFact of N!

```
int freq[MAX];
void primeFact(int n) {
    for(int i=0;i<primes.size() &&
primes[i]<=n;i++) {
        if(n%primes[i] == 0) {
            while(n%primes[i] == 0)
                freq[primes[i]]++ , n
/= primes[i];    }
    }
    if(n > 1)
        freq[n]++;    }
void factorialPrimeFact(int n) {
    memset(freq,0,sizeof freq);
    for(int i=2;i<=n;i++)
        primeFact(i);
    bool flag = 0;
    cout << n << " = ";
    for(int i=0;i<primes.size();i++) {
        int x = primes[i];
        if(freq[x]) {
            if(flag)
                cout << " * ";
            cout << x << " (" <<
freq[x] << ")";
            flag = 1;   }
    }
    cout << endl;  }
```

Segmented Sieve

```
void segmented_sieve(ll l,ll r) {
    sieve();
    memset(mark,true,sizeof mark);
    if(l == 1)
        mark[0] = false;
    for(int
i=0;primes[i]*primes[i]<=r;i++) {
        ll base = primes[i]*primes[i];
        if(base < l)
            base = (l+primes[i]-
1)/primes[i] * primes[i];
```

```
        for(int
j=base;j<=r;j+=primes[i])
            mark[j-l] = false;
    }
}
```

Sum of divisors of n^m

```
ll primeFact(ll n,int m) {
    ll sum = 1;
    for(int i=0; i<primes.size() &&
primes[i]<=n; i++) {
        ll cnt = 0, p = primes[i];
        if(n%p == 0) {
            while(n%p == 0)
                cnt++ , n /= p;
            cnt = cnt*m+1;
            ll calc =
(bigMod(p,cnt,MOD)+MOD-1)%MOD;
            calc *= bigMod(p-1,MOD-
2,MOD);
            calc %= MOD;
            sum = (sum*calc)%MOD;
        }
    }
    if(n > 1) {
        ll calc =
(bigMod(n,1+m,MOD)+MOD-1)%MOD;
        calc *= bigMod(n-1,MOD-2,MOD);
        calc %= MOD;
        sum = (sum*calc)%MOD;   }
    return sum;   }
```

Sum of SOD of all numbers in range 1-N

```
ll solve(int n) {
    int sq = sqrt(n);
    ll sum = 0;
    for(int i=2;i<=sq;i++) {
        int j = n/i;
        sum += (j+i)*1LL*(j-i+1)/2;
        sum += (j-i)*1LL*i;    }
    return sum;    }
```

nCr using MOD

```
const int MAX = 2e5+10;
const int MOD = 1e9+7;
ll f[MAX];
void calcFact() {
    f[0] = 1;
    for(int i=1;i<MAX;i++)
        f[i] = (f[i-1]*i)%MOD;    }
ll bigMod(ll a,ll b) {
    if(b == 0)
        return 1;
    ll x = bigMod(a,b/2);
    x = (x*x)%MOD;
    if(b&1)
        x = (x*a)%MOD;
    return x;      }
ll nCr(ll n,ll r) {
    return (f[n]*bigMod((f[r]*f[n-
r])%MOD,MOD-2))%MOD;
}
```

nCr using DP

```
int dp[700][700];
int nCr(int n,int r) {
    if(r==n)
        return 1;
```

```
    else if(r==1)
        return n;
    if(dp[n][r] != -1)
        return dp[n][r];
    else
        return dp[n][r] = nCr(n-1,r) +
nCr(n-1,r-1);    }
```

*** **Geometry Template** ***

```
#define PI acos(-1)
const double INF=1e4;
const double EPS=1e-10;
struct Point {
    double x,y;
    Point() {}
    Point(double x, double y):x(x),y(y)
{}
    Point(const Point &p):
x(p.x),y(p.y) {}
    void input() {
        scanf("%lf%lf",&x,&y);
    }
    Point operator + (const Point &p)
const {
        return Point(x+p.x, y+p.y);
    }
    Point operator - (const Point &p)
const {
        return Point(x-p.x, y-p.y);
    }
    Point operator * (double c) const {
        return Point(x*c, y*c);
    }
    Point operator / (double c) const {
        return Point(x/c, y/c);
    }
};
vector<Point>polygon;
double getClockwiseAngle(Point p) {
    return -1 * atan2(p.x, -1 * p.y);
}
//compare function to compare clockwise
bool comparePoints(Point p1, Point p2)
{
    return getClockwiseAngle(p1) <
getClockwiseAngle(p2);
}
// rotate 90 degree counter clockwise
Point RotateCCW90(Point p) {
    return Point(-p.y,p.x);
}
// rotate 90 degree clockwise
Point RotateCW90(Point p) {
    return Point(p.y,-p.x);
}
Point RotateCCW(Point p, double t) {
    return Point(p.x*cos(t)-
p.y*sin(t),p.x*sin(t)+p.y*cos(t));
}
Point RotateCW(Point p, double t) {
    return
Point(p.x*cos(t)+p.y*sin(t),-
p.x*sin(t)+p.y*cos(t));
}
double dot(Point A, Point B) {
    return A.x*B.x+A.y*B.y;
}
```

```
double cross(Point A, Point B) {
    return A.x*B.y-A.y*B.x;
}
double dist2(Point A, Point B) {
    return dot(A-B,A-B);
}
// returns distance between two point
double dist(Point A, Point B) {
    return sqrt(dot(A-B,A-B));
}
// Distance between point A and B
double distBetweenPoint(Point A, Point B) {
    return sqrt(dot(A-B,A-B));
}
// project point c onto line AB (A!=B)
Point ProjectPointLine(Point A, Point B, Point C) {
    return A+(B-A)*dot(C-A,B-A)/dot(B-A,B-A);
}
// Determine if Line AB and CD are parallel or collinear
bool LinesParallel(Point A, Point B, Point C, Point D) {
    return fabs(cross(B-A,C-D))<EPS;
}
// Determine if Line AB and CD are collinear
bool LinesCollinear(Point A, Point B, Point C, Point D) {
    return LinesParallel(A,B,C,D) && fabs(cross(A-B,A-C))<EPS && fabs(cross(C-D,C-A))<EPS;
}
//checks if AB intersect with CD
bool SegmentIntersect(Point A, Point B, Point C, Point D) {
    if(LinesCollinear(A,B,C,D)) {
        if(dist2(A,C)<EPS || dist2(A,D)<EPS || dist2(B,C)<EPS || dist2(B,D)<EPS)
            return true;
        if(dot(C-A,C-B) > 0 && dot(D-A,D-B) > 0 && dot(C-B,D-B) > 0)
            return false;
        return true;
    }
    if(cross(D-A,B-A) * cross(C-A,B-A) > 0)
        return false;
    if(cross(A-C,D-C) * cross(B-C,D-C) > 0)
        return false;
    return true;
}
// Compute the coordinates where AB and CD intersect
Point ComputeLineIntersection(Point A, Point B, Point C, Point D) {
    double a1,b1,c1,a2,b2,c2;
    a1=A.y-B.y;
    b1=B.x-A.x;
    c1=cross(A,B);
    a2=C.y-D.y;
    b2=D.x-C.x;
    c2=cross(C,D);
    double Dist=a1*b2-a2*b1;
    return Point((b1*c2-b2*c1)/Dist,(c1*a2-c2*a1)/Dist);
```

```
}

//Project point C onto line segment AB -- return the Point from AB which is the closest to C --

Point ProjectPointSegment(Point A, Point B, Point C) {

    double r=dot(B-A,B-A);

    if(fabs(r)<EPS)

        return A;

    r=dot(C-A,B-A)/r;

    if(r<0)

        return A;

    if(r>1)

        return B;

    return A+(B-A)*r;

}

// return the minimum distance from a point C to a line AB

double DistancePointSegment(Point A, Point B, Point C) {

    return distBetweenPoint(C,ProjectPointSegment(A,B,C));

}

// return distance between P and a point where p is perpendicular on AB. AB er upore p jei point e lombo shei point theke p er distance

double distToLine(Point p, Point a, Point b) {

    pair<double,double>c;

    double scale=(double)(dot(p-a,b-a))/(dot(b-a,b-a));

    c.first=a.x+scale*(b.x-a.x);

    c.second=a.y+scale*(b.y-a.y);

    double dx=(double)p.x-c.first,dy=(double)p.y-c.second;

    return sqrt(dx*dx+dy*dy);

}

long long orientation(Point p, Point q, Point r) {

    long long val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);

    if (val > 0)

        return 1;

    if (val < 0)

        return 2;

    else

        return val;


}

// Given three colinear points p, q, r, the function checks if

// point q lies on line segment 'pr'

bool onSegment(Point p, Point q, Point r) {

    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) && q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))

        return true;

    return false;

}

//checks if Point P is inside of polygon or not

bool isInside(int n, Point p) {

    if (n < 3)

        return false;

    Point extreme = Point(INF, p.y); // here INF=1e4

    int count = 0, i = 0;
```

```
    do{
        int next = (i+1)%n;
        if
(SegmentIntersect(polygon[i],
polygon[next], p, extreme)) {
            if (orientation(polygon[i],
p, polygon[next]) == 0)
                return
onSegment(polygon[i], p,
polygon[next]);
            count++;
        }
        i = next;
    }
    while (i != 0);
    return count&1;
}
// returns the perimeter of a polygon
double polygonPerimeter(int n) {
    double perimeter = 0.0;
    for (int i = 0; i < n - 1; i++)
//polygon vector holds the corner
points of the given polygon
        perimeter += dist(polygon[i],
polygon[i + 1]);
    perimeter += dist(polygon[0],
polygon[n - 1]);
    return perimeter;
}
//returns the area of a polygon
double polygonArea(int n) {
    double area = 0.0;
    int j = n - 1;
    for (int i = 0; i < n; i++) {
        area += (polygon[j].x +
polygon[i].x) * (polygon[j].y -
polygon[i].y);
        j = i;
    }
    return fabs(area)*0.5;
}
double getTriangleArea(Point a, Point
b, Point c) {
    return fabs(cross(b - a, c - a));
}
bool compareConvex(Point X , Point Y) {
    long long ret =
orientation(points[0],X,Y);
    if(ret==0) {
        long long dist11 =
dist2(points[0],X);
        long long dist22 =
dist2(points[0],Y);
        return dist11 < dist22 ;
    }
    else if(ret==2) return true ;
    else return false ;
}
Point nextToTop(stack<Point> &S) {
    Point p = S.top();
    S.pop();
    Point res = S.top();
    S.push(p);
    return res;
}
// make a minimum area polygon
stack<Point> convexHull(int N) {
```

```
    int ymin = points[0].y , index = 0
;
    for(int i=1;i<N;i++) {

if(points[i].y<ymin||(points[i].y==ymin
&&points[i].x<points[index].x)) {
            ymin = points[i].y ;
            index = i ;
        }
    }
    stack<Point>S;
    swap(points[0],points[index]);

sort(&points[1],&points[N],compareConve
x);
    S.push(points[0]);
    for(int i=1;i<N;i++) {

while(S.size()>1&&orientation(nextToTop
(S),S.top(),points[i])!=2){
            S.pop();
        }
        S.push(points[i]);
    }
    return S;
}
// Angle between Line AB and AC in
degree
double angle(Point B, Point A, Point C)
{
    double c=dist(A,B);
    double a=dist(B,C);
    double b=dist(A,C);
    double ans=acos((b*b+c*c-
a*a)/(2*b*c));
    return (ans*180)/acos(-1);


}
// returns number of vertices on
boundary of a polygon
long long
picks_theorem_boundary_count() {
int sz=polygon.size(), i ;
    long long res=__gcd((long
long)abs(polygon[0].x-polygon[sz-
1].x),(long long)abs(polygon[0].y-
polygon[sz-1].y));
    for ( i = 0; i < sz-1 ; i++ )
        res += __gcd((long
long)abs(polygon[i].x-
polygon[i+1].x),(long
long)abs(polygon[i].y-polygon[i+1].y));
    return res;  }
// picks theorem
// Polygon area= inside points +
boundary points/2 -1
// return inside points counts
long long
lattice_points_inside_polygon() {
    long long ar=polygonArea(n);
    long long
b=picks_theorem_boundary_count();
    long long tot=ar+1-b/2;
    return tot;
}
```

*** Others ***

LIS length in n logn

```
int main() {
    int n, t;
    vector<int> v;
```

```
    vector<int> LIS;
    scanf("%d", &n);
    for(int i = 1; i <= n; i++) {
        scanf("%d", &t);
        v.push_back(t);
        vector<int> ::iterator it;
        it = lower_bound(LIS.begin(), LIS.end(), t);
        if(it != LIS.end())
            *it = t;
        else
            LIS.push_back(t);   }
    cout << LIS.size() << endl;   }
```

Edit Distance

```
int main() {
    int t;
    cin >> t;
    while(t--) {
        memset(dp,0,sizeof(dp));
        cin >> s >> s1;
        int len1 = s.length() , len2 = s1.length();
        for(int i=0;i<=len1;i++)
            dp[i][0] = i;
        for(int i=0;i<=len1;i++)
            dp[0][i] = i;
        for(int i=1;i<=len1;i++) {
            for(int j=1;j<=len2;j++) {
                if(s[i-1]==s1[j-1])
                    dp[i][j] = dp[i-1][j-1];
                else
                    dp[i][j] = min(dp[i-1][j-1]+1,min(dp[i-1][j]+1,dp[i][j-1]+1));
            }
        }
        cout << dp[len1][len2] << endl;
    }
}
```

*** Formulas ***

Sums:

-> c^a + c^(a+1) + … + c^b = (c^(b+1) – c^a) / (c-1) , c != 1

-> 1+2+..+n = (n*(n+1))/2

-> 1^2 + 2^2 + .. = (n(2n+1)(n+1))/6

-> 1^3 + 2^3 + .. = ((n^2)+(n+1)^2)/4

-> 1^4 + 2^4 + .. = (n(n+1)(2n+1)(3n^2+3n-1))/30

Combinatorics:

C(n,r): n! / (r! * (n-r)!)

C(n,r): (n*(n-1)*..*(n-r+1)) / r!

P(n,k): n! / (n-k)!

-> nCk = nCn-k

-> Ways to go from (0,0) to (r,c):

(r+c)Cr or (r+c)Cc

-> Ways to go from (0,0,0) to

(x,y,z): (x+y+z)Cx * (y+z)Cy

-> a1+a2+.+an = k , ai >= 0:

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

-> Catalan Numbers:

C(n) = (2n)! / ((n+1)! * r!)

Triangle:

*__To form__ a+b>c,b+c>a,a+c>b

*__Check if 3 points form triangle:__

|(x2-x1)(y3-y1)-(y2-y1)(x3-x1)| > 0

Perimeter: p = a+b+c

Area(A): (½) ab

A: abSinC/2

A: (Ax(By-Cy)+Bx(Cy-Ay)+Cx(Ay-By)/2

s = p/2

A: sqrt(s*(s-a)*(s-b)*(s-c))

SineRule: a/SinA=b/Sinb=c/SinC

CosineRule: a^2=b^2+c^2-2bcCosA

Angle: cosA = (b^2+c^2-a^2)/2bc

x=(x1+x2+x3)/3,y=(y1+y2+y3)/3

Median: AD=sq((2b^2+2c^2-a^2)/4)

Centroid: AG=sq(2b^2-2c^2-a^2)/3

Center: x=(x1+x2+x3)/3,y=(y1+y2+y3)/3

Circle:

Distance: sqrt((x2-x1)^2 +

(y2-y1)^2)

Check if 3 points are in same line:

x1*(y2-y3)-x2(y1-y3)+x3(y1-y2) = 0

Find a circle that covers 2 given:

x3 = (x1+x2)/2 , y3 = (y1+y2)/2

r = dist(x1,y1,x2,y2)

Lattice Points:

1 + gcd(|x1-x2|,|y1-y2|)

Slope formed by 2 points:

(y2-y1) / (x2-x1)

Area of sector of circle: ½ r^2* θ

Arc Length: r*θ

Parallelogram:

Given 3 points find 4th point:

Dx = Ax + (Cx-Bx)

Dy = Ay + (Cy-By)

Area: |½((Ax*By+Bx*Cy+Cx*Dy+Dx*Ay)

-(Ay*Bx + By*Cx + Cx*Dx + Dy*Ax))|

Trapezium:

Area: (a+b)/(a-b) * sqrt((s-a)(s-b)

(s-b-c)(s-b-d))

-> s = (a+b+c+d)/2

-> a = long parallel side

-> b = short parallel side

-> c,d = non-parallel side