

Contents

1	Setting	1
1.1	Header	1
1.2	HeaderDebug	1
1.3	vimrc	2
1.4	Sublime text	2
2	String	2
2.1	KMP	2
2.2	Aho Chorasick	3
2.3	Suffix array	3
2.4	Manacher’s algorithm	4
2.5	Z algorithm	4
3	Graph & Flow	5
3.1	BCC	5
3.2	Dinic	5
3.3	Bipartite matching (simple)	6
3.4	MCMF	6
3.5	2SAT & answer recover	7
3.6	Stoer Wagner	8
3.7	LR-flow	8
4	Query	8
4.1	HLD	8
4.2	Centroid decomposition	10
4.3	Mo’s algorithm	10
4.4	Mo’s algorithm on tree	11
4.5	Parallel binary search	12
4.6	Lazy Propagation 1	13
5	Geometry	14
5.1	Closest pair	14
5.2	Convex hull	14
5.3	Rotating Calipers	15
6	Math	15
6.1	FFT	15
6.2	Extended Euclidean	15
6.3	Gaussian Elimination	15
6.4	Prime Algorithms	16
7	Miscellaneous	17
7.1	Grundy number	17
7.2	Hungarian	17

7.3	Hungarian2	18
7.4	LiChao Tree	20
7.5	Persistence Segment Tree	21
7.6	XOR FFT	21
7.7	NTT	22
7.8	2D FFT	22
7.9	Divide and Conquer DP optimization	24
7.10	Order Statistic Tree	24
7.11	BITSET	24

1 Setting

1.1 Header

```
#include<bits/stdc++.h>

using namespace std;
typedef long long ll;
typedef unsigned long long ull;
typedef pair<int, int> Pi;
typedef pair<ll,ll> Pll;

#define Fi first
#define Se second
#define pb(x) push_back(x)
#define sz(x) (int)x.size()
#define rep(i, n) for(int i=0;i<n;i++)
#define repp(i, n) for(int i=1;i<=n;i++)
#define all(x) x.begin(), x.end()

#define INF 987654321
#define IINF 654321987654321LL
```

1.2 HeaderDebug

```
#include<bits/stdc++.h>

using namespace std;
typedef long long ll;
typedef unsigned long long ull;
typedef pair<int, int> Pi;
typedef pair<ll,ll> Pll;

#define Fi first
#define Se second
#define pb(x) push_back(x)
#define sz(x) (int)x.size()
#define rep(i, n) for(int i=0;i<n;i++)
#define repp(i, n) for(int i=1;i<=n;i++)
#define all(x) x.begin(), x.end()
```

```

#define VA_NUM_ARGS(...) VA_NUM_ARGS_IMPL((0, __VA_ARGS__, 6,5,4,3,2,1))
#define VA_NUM_ARGS_IMPL_(tuple) VA_NUM_ARGS_IMPL tuple
#define VA_NUM_ARGS_IMPL(_0,_1,_2,_3,_4,_5,_6,N,...) N
#define macro_dispatcher(macro, ...) macro_dispatcher_(macro, VA_NUM_ARGS(
__VA_ARGS__))
#define macro_dispatcher_(macro, nargs) macro_dispatcher__(macro, nargs)
#define macro_dispatcher__(macro, nargs) macro_dispatcher___(macro, nargs)
#define macro_dispatcher___(macro, nargs) macro ## nargs
#define Debug1(a) cout<<#a<<""<<(a)<<"\n"
#define Debug2(a,b) cout<<#a<<""<<(a)<<", "<<#b<<""<<(b)<<"\n"
#define Debug3(a,b,c) cout<<#a<<""<<(a)<<", "<<#b<<""<<(b)<<", "<<#c
<<""<<(c)<<"\n"
#define Debug4(a,b,c,d) cout<<#a<<""<<(a)<<", "<<#b<<""<<(b)<<", "<<#c
<<""<<(c)<<", "<<#d<<""<<(d)<<"\n"
#define Debug5(a,b,c,d,e) cout<<#a<<""<<(a)<<", "<<#b<<""<<(b)<<", "<<#c
<<""<<(c)<<", "<<#d<<""<<(d)<<", "<<#e<<""<<(e)<<"\n"
#define Debug6(a,b,c,d,e,f) cout<<#a<<""<<(a)<<", "<<#b<<""<<(b)<<", "<<#c
<<""<<(c)<<", "<<#d<<""<<(d)<<", "<<#e<<""<<(e)<<", "<<#f<<""<<(f)<<"\n"
#define Debug(...) macro_dispatcher(Debug, __VA_ARGS__)(__VA_ARGS__)
#define DA(a,n) cout<<#a<<""<<["; printarray(a,n); cout<<"]\n"
#define DAR(a,n,s) cout<<#a<<""<<["<<s<<""<<n-1<<"]<<["; printarray(a,n,s); cout
<<"]\n"

#define TT1 template<class T>
#define TT1T2 template<class T1, class T2>
#define TT1T2T3 template<class T1, class T2, class T3>
template<class T, size_t N> ostream& operator << (ostream& os, const array<T, N
>& v);
TT1T2 ostream& operator << (ostream& os, const pair<T1, T2>& p){ return os
<< "("<<p.first<<", "<< p.second<<")"; }
TT1 ostream& operator << (ostream& os, const vector<T>& v){ bool f=1;os
<< "[";for(auto& i : v) { if (!f)os << ", ";os<<i;f=0;}return os << "]"; }
template<class T, size_t N> ostream& operator << (ostream& os, const array<T, N
>& v) { bool f=1;os<< "[";for(auto& i : v) { if (!f)os << ", ";os<<i;f=0;}
return os << "]"; }
TT1T2 ostream& operator << (ostream& os, const set<T1, T2>&v){ bool f=1;os
<< "[";for(auto& i : v) { if (!f)os << ", ";os<<i;f=0;}return os << "]"; }
TT1T2 ostream& operator << (ostream& os, const multiset<T1,T2>&v){bool f=1;os
<< "[";for(auto& i : v) { if (!f)os << ", ";os<<i;f=0;}return os << "]"; }
TT1T2T3 ostream& operator << (ostream& os, const map<T1,T2,T3>& v){ bool f = 1;
os << "["; for (auto& ii : v) { if (!f)os << ", "; os << "(" << ii.first << "
-> " << ii.second << " ) "; f = 0; }return os << "]"; }
TT1T2 ostream& operator << (ostream& os, const multimap<T1, T2>& v){ bool f = 1;
os << "["; for (auto& ii : v) { if (!f)os << ", "; os << "(" << ii.first << "
-> " << ii.second << " ) "; f = 0; }return os << "]"; }
TT1T2 ostream& operator << (ostream& os, priority_queue<T1, T2> v) { bool f = 1;
os << "["; while (!v.empty()) { auto x = v.top(); v.pop(); if (!f) os << ",
"; f = 0; os << x; } return os << "]"; }
TT1T2 void printarray(const T1& a, T2 sz, T2 beg = 0){ for (T2 i = beg; i<sz; i
++) cout << a[i] << " "; cout << endl; }

```

1.3 vimrc

syntax on

```

set nu ai ci si nobk et ar ru nocp hls
set bs=2 ts=4 sw=4 sts=4
set cb=unnamed
set mouse=an
command PS vsp %:r.in|sp %:r.out|vert res 30|wa
command RIO wall|!g++ -O2 -std=c++14 -Wall -lm %:r.cpp && ./a.out < %:r.in
.out
command RI wall|!g++ -O2 -std=c++14 -Wall -lm %:r.cpp && ./a.out < %:r.in

```

1.4 Sublime text

```

{
    "shell_cmd": "g++ -O2 -std=c++11 \"${file}\" -o \"${file_path}/${
file_base_name}\" && \"${file_path}/${file_base_name}\" < input.txt",
    "working_dir": "${file_path}",
    "selector": "source.c++",
}

```

2 String

2.1 KMP

```

vector<int> preprocess(string p){
    int m = p.size();
    vector<int> fail(m);
    fail[0] = 0; int j = 0;
    for(int i=1;i<m;i++){
        while(j>0&&p[i]!=p[j]) j = fail[j-1];
        if( p[i] == p[j] ){
            fail[i] = j+1; j++;
        }else{
            fail[i] = 0;
        }
    }
    return fail;
}

vector<int> kmp(string s, string p){
    auto fail = preprocess(p);
    vector<int> ans; int n = s.size(), m = p.size();
    int j = 0;
    for(int i=0;i<n;i++){
        while(j>0 && s[i]!=p[j]) j = fail[j-1];
        if( s[i] == p[j] ){
            if( j == m-1 ){
                ans.pb(i-m+1); j = fail[j];
            }else{
                j++;
            }
        }
    }
    return ans;
}

```

2.2 Aho Chorasick

```

struct AhoCorasick{
    struct Node{
        int fail;
        vector<int> output;
        int children[26];

        Node(){
            for(int i=0;i<26;i++) children[i] = -1;
            fail = -1;
        }
    };

    vector<Node> trie;
    int new_node(){
        Node x;
        trie.push_back(x);
        return (int)trie.size()-1;
    }

    void add(int node, string &s, int idx, int string_num){
        //cout << node << " " << idx << endl;
        if( idx == s.size() ){
            trie[node].output.push_back(string_num);
            return;
        }
        int c = s[idx] - 'a';
        if( trie[node].children[c] == -1 ){
            int next = new_node();
            trie[node].children[c] = next;
        }

        add(trie[node].children[c], s, idx+1, string_num);
    }

    void build(vector<string> v){
        int root = new_node();
        for(int i=0;i<v.size();i++){
            add(root,v[i],0,i);
        }

        queue<int> q;
        q.push(root); trie[root].fail = root;
        while( !q.empty() ){
            int cur = q.front(); q.pop();
            for(int i=0;i<26;i++){
                int next = trie[cur].children[i];
                if( next == -1 ) continue;

                // build fail
                if( cur == root ){
                    trie[next].fail = root;
                }
                else{
                    int x = trie[cur].fail;

```

```

                    while( x != root && trie[x].children[i] == -1 ) x = trie[x].fail;
                    if( trie[x].children[i] != -1 ) x = trie[x].children[i];
                    trie[next].fail = x;
                }
            }
            // build output
            int f = trie[next].fail;
            for(auto e : trie[f].output) trie[next].output.push_back(e);
            q.push(next);
        }
    }

    vector<Pi> find(string s){
        int n = (int) s.size();
        int cur = 0, root = 0;
        vector<Pi> ans;
        for(int i=0;i<n;i++){
            int c = s[i] - 'a';
            while( cur != root && trie[cur].children[c] == -1 ) cur = trie[cur].fail;
            if( trie[cur].children[c] != -1 ) cur = trie[cur].children[c];

            for(auto e : trie[cur].output){
                ans.push_back({e,i});
            }
        }
        return ans;
    }
};

```

2.3 Suffix array

```

// Make sure to add !, #, $, %, & at the end of input string
class SuffixArray{
public:
    int n;
    string s;
    vector<int> rank, temprank, sa, tempa, c;
    vector<int> lcp;
    SuffixArray(string _s){
        n = _s.size(); s = _s;
        rank.resize(n); temprank.resize(n); sa.resize(n); tempa.resize(n);
        lcp.resize(n);
        constructSA();
        constructLCP();
    }

    void countingSort(int k){
        int sum = 0, maxi = max(270, n); //ASCII 256
        c.clear(); c.resize(maxi+10);
        for(auto& e : c) e = 0;
        for(int i=0; i<n; i++) c[ i+k<n ? rank[i+k] : 0 ] ++;
        for(int i=0; i<maxi; i++){

```

```

        int t = c[i]; c[i] = sum; sum += t;
    }
    for(int i=0; i<n; i++) tempa[ c[ sa[i]+k < n ? rank[sa[i]+k] : 0 ] ++ ]
        = sa[i];
    for(int i=0; i<n; i++) sa[i] = tempa[i];
}

void constructSA(){
    for(int i=0; i<n; i++) rank[i] = s[i];
    for(int i=0; i<n; i++) sa[i] = i;
    for(int k=1; k<n; k<=1){
        countingSort(k);
        countingSort(0);
        int r = 0;
        temprank[sa[0]] = 0;
        for(int i=1; i<n; i++){
            temprank[sa[i]] = (rank[sa[i]] == rank[sa[i-1]] && rank[sa[i]+k]
                == rank[sa[i-1]+k] ) ? r : ++r;
        }
        for(int i=0; i<n; i++) rank[i] = temprank[i];
        if( rank[sa[n-1]] == n-1 ) break;
    }
}

// lcp Implementation from
// http://m.blog.naver.com/dark__nebula/220419358547
void constructLCP(){
    int h = 0;
    for(int i=0; i<n; i++){
        if( rank[i] ){
            int j = sa[rank[i]-1];
            while( s[i+h] == s[j+h] ) h++;
            lcp[rank[i]] = h;
        }
        if( h > 0 ) h--;
    }
}

};

```

2.4 Manacher's algorithm

```

// finds radius of longest palindrome centered at s[i]
// If you also want to find even-length palindromes, use dummy characters
// baab -> #b#a#a#b#
vector<int> ManacherAlgorithm(string s){
    int n = (int) s.size();
    int p = -1, r = -1;
    vector<int> A(n);
    for(int i=0; i<n; i++){
        if( r < i ){
            A[i] = 0;

```

```

            int j = 0;
            while( i + A[i] < n && i - A[i] >= 0 && s[ i+A[i] ] == s[ i-A[i] ] )
                A[i]++;
            A[i]--;
        }
    }
    else{
        A[i] = min( A[2*p - i] , r-i );
        while( i + A[i] < n && i - A[i] >= 0 && s[ i+A[i] ] == s[ i-A[i] ] )
            A[i]++;
        A[i]--;
    }
}

// update r
if( r < i + A[i] ){
    r = i + A[i];
    p = i;
}
}
return A;
}

```

2.5 Z algorithm

```

// Calculates LCP[i] for all 0 <= i < n
vector<int> Zalgorithm(string s){
    int l=0, r=0;
    int n = (int) s.size();
    vector<int> Z(n);
    Z[0] = n;
    for(int i=1; i<n; i++){
        // reset and calculate again
        if( i > r ){
            l = r = i;
            while( r<n && s[r] == s[r-l] ) r++;
            r--;
            Z[i] = r-l+1;
        }

        // extend [l,r]
        else{
            int k = i-l;
            // not enough matching at position k
            if( Z[k] < r-i+1 ) Z[i] = Z[k];
            // enough matching. extend [l,r]
            else{
                l = i;
                while( r<n && s[r] == s[r-l] ) r++;
                r--;
                Z[i] = r-l+1;
            }
        }
    }
    return Z;
}
};

```

3 Graph & Flow

3.1 BCC

```
int N,M;
int timer = 0;
vector<int> E[300500];
int vis[300500], low[300500];

// dfs1 is to fill vis(discover time) and low array
int dfs1(int x, int pa){
    vis[x] = ++timer;
    low[x] = vis[x];
    for(auto e : E[x])if(e!=pa){
        if( vis[e] ){
            low[x] = min(low[x], vis[e]);
        }
        else{
            dfs1(e,x);
            low[x] = min(low[x], low[e]);
        }
    }
    return low[x] ;
}

int color = 0;
vector<int> colors[300500], E2[300500];
int vis2[300500];

// dfs2 is to color every nodes
// Store node's colors into colors array
// Store new edges into E2
void dfs2(int x, int pa, int c){
    colors[x].pb(c);
    vis2[x] = 1;
    for(auto e : E[x])if(!vis2[e]){
        // x-e is an articulation edge
        if( low[e] > vis[x] ){
            ++color;
            colors[x].pb(color);
            E2[c].pb(color); E2[color].pb(c);
            dfs2(e,x,color);
        }
        // x-e is not an articulation edge
        else dfs2(e,x,c);
    }
}

int main(){
    geti(N,M);
    repp(i,M){
        int a, b; geti(a,b);
        E[a].pb(b); E[b].pb(a);
    }
}
```

```
// fill vis & low
dfs1(1,-1);
// find out articulation edge and color of nodes
color = 1;
dfs2(1,-1,color);
}
```

3.2 Dinic

```
struct MaxFlowDinic{
    struct Edge{
        // next, inv, residual
        int to, inv; ll res;
    };

    int n;
    vector<vector<Edge>> graph;

    vector<int> lev,work;

    void init(int x){
        n = x+10;
        graph.resize(x+10);
        lev.resize(n); work.resize(n);
    }

    void make_edge(int s, int e, ll cap, ll caprev = 0){
        Edge forward = {e, (int)graph[e].size(), cap};
        Edge backward = {s, (int)graph[s].size(), caprev};
        graph[s].push_back(forward);
        graph[e].push_back(backward);
    }

    bool bfs(int source, int sink){
        queue<int> q;
        for(auto& e : lev) e = -1;
        lev[source] = 0; q.push(source);
        while(!q.empty()){
            int cur = q.front(); q.pop();
            for(auto e : graph[cur]){
                if(lev[e.to]==-1 && e.res > 0){
                    lev[e.to] = lev[cur]+1;
                    q.push(e.to);
                }
            }
        }
        return lev[sink] != -1;
    }

    ll dfs(int cur, int sink, ll flow){
        if( cur == sink ) return flow;
        for(int &i = work[cur]; i < (int)graph[cur].size(); i++){
            Edge &e = graph[cur][i];
            if( e.res == 0 || lev[e.to] != lev[cur]+1 ) continue;
            ll df = dfs(e.to, sink, min(flow, e.res) );
        }
    }
}
```

```

        if( df > 0 ){
            e.res -= df;
            graph[e.to][e.inv].res += df;
            return df;
        }
    }
    return 0;
}

ll solve( int source, int sink ){
    ll ans = 0;
    while( bfs(source, sink) ){
        for(auto& e : work) e = 0;
        while( true ){
            ll flow = dfs(source,sink,54321987654321LL);
            if( flow == 0 ) break;
            ans += flow;
        }
    }
    return ans;
}
};

```

3.3 Bipartite matching (simple)

```

int yx[5000], xy[5000];
bool vis[5000];
vector<int> E[5000];
int dfs(int x){
    vis[x] = true;
    for(auto e : E[x]){
        if( yx[e] == -1 || (vis[yx[e]] == false && dfs(yx[e])) ){
            yx[e] = x;
            xy[e] = e;
            return 1;
        }
    }
    return 0;
}

int main(){
    memset(yx,-1,sizeof yx);
    int ans = 0;
    rep(i,N){
        memset(vis,0,sizeof vis);
        ans += dfs(i);
    }
    cout << ans;
}

```

3.4 MCMF

```

struct MCMF{
    struct edge{
        int to, inv, cap, flow, cost;
        int res(){
            return cap - flow;
        }
    };

    vector<vector<edge>> graph;
    vector<int> pv, pe;
    vector<int> dist, inq;

    void init(int x){
        graph.resize(x+10);
        for(auto& e : graph) e.resize(x+10);
        pv.resize(x+10); pe.resize(x+10);
        dist.resize(x+10);
        inq.resize(x+10);
    }

    void make_edge(int from, int to, int cap, int cost){
        //printf("%d -> %d | cost = %d\n",from,to,cost);
        edge forward = {to, (int)graph[to].size(), cap, 0, cost};
        edge backward = {from, (int)graph[from].size(), 0, 0, -cost};
        graph[from].push_back(forward);
        graph[to].push_back(backward);
    }

    int solve(int source, int sink){
        int ans = 0;
        int totalflow = 0;
        while(true){
            for(auto& e : dist) e = INF;
            for(auto& e : inq) e = 0;
            queue<int> q;
            q.push(source); inq[source] = 1;
            dist[source] = 0;

            while(!q.empty()){
                int cur = q.front(); q.pop();
                inq[cur] = 0;
                for(int i=0;i<(int)graph[cur].size();i++){
                    auto& e = graph[cur][i];
                    if( e.res() > 0 && dist[e.to] > dist[cur] + e.cost ){
                        dist[e.to] = dist[cur] + e.cost;
                        pv[e.to] = cur; pe[e.to] = i;
                        if( inq[e.to] == 0 ){
                            q.push(e.to); inq[e.to] = 1;
                        }
                    }
                }
            }

            if( dist[sink] == INF ) break;

            // add this limit when we don't require maxflow

```

```

        //if( dist[sink] > 0 ) break;

        int mnflow = INF;
        for( int v = sink; v != source; v = pv[v] ){
            mnflow = min( mnflow, graph[pv[v]][pe[v]].res() );
        }

        for( int v = sink; v != source; v = pv[v] ){
            int tmp = graph[pv[v]][pe[v]].inv;
            graph[pv[v]][pe[v]].flow += mnflow;
            graph[v][tmp].flow -= mnflow;
        }
        totalflow += mnflow;
        ans += dist[sink] * mnflow;
    }
    return ans;
}
};

```

3.5 2SAT & answer recover

```

#define MAX_V 20010
int V,M;

vector<int> Edge[MAX_V];
vector<int> rEdge[MAX_V];
vector<int> vs;

bool vis[MAX_V];
int cmp[MAX_V];
set<int> printSet[MAX_V];

void addEdge(int from, int to){
    Edge[from].push_back(to);
    rEdge[to].push_back(from);
}

void dfs(int v){
    vis[v] = true;
    for (int i = 0; i < Edge[v].size(); i++){
        if (!vis[Edge[v][i]]) dfs(Edge[v][i]);
    }
    vs.push_back(v);
}

void rdfs(int v, int k){
    vis[v] = true;
    cmp[v] = k;
    printSet[k].insert(v);
    for (int i = 0; i < rEdge[v].size(); i++){
        if (!vis[rEdge[v][i]]) rdfs(rEdge[v][i], k);
    }
}

```

```

bool cmp1(set<int>& a, set<int>& b) {
    return *a.begin() < *b.begin();
}

int main()
{
    geti(V); geti(M);
    int cnt = 0;
    while (M--){
        int a, b;
        scanf("%d%d", &a, &b);
        if (a > 0 && b > 0 ){
            addEdge(a + V, b);
            addEdge(b + V, a);
        }
        else if (a > 0 && b < 0){
            b = -b;
            addEdge(a + V, b + V);
            addEdge(b , a);
        }
        else if (a < 0 && b > 0){
            a = -a;
            addEdge(a, b);
            addEdge(b + V, a + V);
        }
    }
    else{
        a = -a; b = -b;
        addEdge(a, b + V);
        addEdge(b, a + V);
    }
}

memset(vis, false, sizeof(vis));
for (int i = 1; i <= 2*V; i++){
    if (!vis[i]) dfs(i);
}

memset(vis, false, sizeof(vis));
int k = 0;
for (int i = vs.size()-1; i >= 0 ; i--){
    if (!vis[vs[i]]) rdfs(vs[i],k++);
}

for (int i = 1; i <= V; i++){
    if (cmp[i] == cmp[V + i]){
        printf("0\n");
        return 0;
    }
}
printf("1\n");

for (int i = 1; i <= V; i++){

```

```

        if (cmp[i] > cmp[V + i]){
            printf("1 ");
        }
        else printf("0 ");
    }
}

```

3.6 Stoer Wagner

```

// Stoer-Wagner algorithm
struct mincut {
    int n;
    vector<vector<int>> graph;

    void init(int nn) {
        n = nn;
        graph.resize(n, vector<int>(n, 0));
    }

    void addEdge(int u, int v, int w) {
        graph[u][v] += w;
        graph[v][u] += w;
    }

    pair<int, vector<int>> findMincut() {
        vector<vector<int>> weight = graph;
        vector<bool> used(n, 0);
        vector<int> best_cut;
        int best_weight = -1;

        vector<vector<int>> group(n);
        for(int i = 0; i < n; i++)
            group[i].push_back(i);

        for(int phase = n-1; phase >= 0; phase--) {
            int start = 0;
            vector<int> w = weight[start];
            vector<bool> inSet = used;
            inSet[start] = true;
            int prev, last = start;

            for(int i = 0; i < phase; i++) {
                prev = last;
                last = -1;
                for(int j = 0; j < n; j++)
                    if(!inSet[j] && (last == -1 || w[j] > w[last])) last = j;

                if(i < phase-1) {
                    inSet[last] = true;
                    for(int j = 0; j < n; j++)
                        w[j] += weight[last][j];
                } else { // last step - merge two nodes: prev & last
                    for(int j = 0; j < n; j++) {
                        weight[prev][j] += weight[last][j];

```

```

                        weight[j][prev] = weight[prev][j];
                    }
                }
                used[last] = true;
                group[prev].insert(group[prev].end(), group[last].begin(),
                    group[last].end());
                if(best_weight == -1 || w[last] < best_weight) {
                    best_weight = w[last];
                    best_cut = group[last];
                }
            }
        }
        return make_pair(best_weight, best_cut);
    }
};

```

3.7 LR-flow

G has a feasible (s,t)-flow iff G' has a saturating (s',t')-flow
 in G' total capacity out of s' and into t' are both D (sum of demands)
 saturating flow : flow with value exactly D.

1. Make new source, new sink (s', t')
2. for every v:
 $c'(s' \rightarrow v) = \sum \{ d(u \rightarrow v) \}$ (give demands into v)
 $c'(v \rightarrow t') = \sum \{ d(v \rightarrow w) \}$ (take demands out of v)
3. for every $u \rightarrow v$:
 $c'(u \rightarrow v) = c(u \rightarrow v) - d(u \rightarrow v)$ (difference of cap, demand)
3. make $t \rightarrow s$ cap: INF

4 Query

4.1 HLD

```

int N,K,M,tc,T;

struct segTree{ //range max query
    vector<int> v; int n;
    void init(int _n){
        _n+=3; v.resize(_n*2+10); n = _n;
    }
    void update(int x, int val){
        for(v[x+=n]=val;x>1;x>>=1) v[x/2] = max(v[x],v[x^1]);
    }
    int query(int l, int r){ // [l,r]
        r++; //to make range as [l,r+1]
        int res = 0;
        for(l+=n,r+=n;l<r;l>>=1,r>>=1){
            if( l&1 ) res = max(res,v[l++]);
            if( r&1 ) res = max(res,v[--r]);

```



```

    }
    return res;
}
};

#define MAXV 100500
#define LOGV 18
// cNo: node# -> hld# mapping
int cNo[MAXV];
// other arrays are accesed using hld#
int cPos[MAXV], cSize[MAXV], cHead[MAXV], cN; int cLeaf[MAXV];
vector<Pi> E[MAXV]; int pa[LOGV][MAXV]; int sz[MAXV]; int val[MAXV]; int level[
MAXV];
bool vis[MAXV]; vector<segTree> tree; vector<Pi> edges;
int dfs_build(int x, int p, int v, int lev){
    pa[0][x] = p; sz[x] = 1; val[x] = v; level[x] = lev;
    for(auto e : E[x])if(e.Fi!=p){
        sz[x] += dfs_build(e.Fi,x,e.Se,lev+1);
    }
    return sz[x];
}
void lca_build(){
    for(int k=1;k<LOGV;k++){
        repp(i,N){
            if( pa[k-1][i] != -1 )pa[k][i] = pa[k-1][pa[k-1][i]];
            else pa[k][i] = -1;
        }
    }
}
int lca(int x, int y){
    if( level[x] < level[y] ) swap(x,y);
    int diff = level[x] - level[y];
    for(int k=0;k<LOGV;k++){
        if( diff & (1<<k) ) x = pa[k][x];

    if( x == y ) return x;
    for(int k=LOGV-1;k>=0;k--){
        if( pa[k][x] != pa[k][y] ){
            x = pa[k][x]; y = pa[k][y];
        }
    }
    return pa[0][x];
}

void hld(int cur){
    vis[cur] = true;
    if( cHead[cN] == 0 ) cHead[cN] = cur;
    cLeaf[cN] = cur;
    cNo[cur] = cN;
    cPos[cur] = cSize[cN]; cSize[cN]++;
    int nxt = -1; int mx = -1;
    // get max subtree (special child)
    for(auto e : E[cur])if(!vis[e.Fi]){
        if( sz[e.Fi] > mx ){
            nxt = e.Fi; mx = sz[e.Fi];
        }
    }
}

```

```

    if( mx >= 0 ) hld(nxt);
    for(auto e : E[cur])if(!vis[e.Fi]){
        cN++; hld(e.Fi);
    }
}

void build_hld_segTree(){
    for(int n=1;n<=cN;n++){
        int cur = cLeaf[n];
        tree[n].init(cSize[n]+5);
        while( cur!=-1 && cNo[cur]==n ){
            tree[n].update(cPos[cur],val[cur]);
            cur = pa[0][cur];
        }
    }
}
void update_query(int x, int val){
    tree[cNo[x]].update(cPos[x],val);
}

int query_up(int u, int v){
    int uc = cNo[u], vc = cNo[v]; int ret = 0;
    while(true){
        if( uc == vc ){
            ret = max(ret, tree[uc].query(cPos[v]+1,cPos[u]) );
            break;
        }
        ret = max(ret, tree[uc].query( cPos[cHead[uc]], cPos[u]) );
        u = cHead[uc]; u = pa[0][u]; uc = cNo[u];
    }
    return ret;
}
int query(int u, int v){
    int l = lca(u,v);
    return max(query_up(u,l), query_up(v,l));
}

int main(){
    geti(N);
    rep(i,N-1){
        int a,b,c; geti(a,b,c);
        E[a].push_back({b,c}); E[b].push_back({a,c});
        edges.push_back({a,b});
    }

    dfs_build(1,-1,0,0); lca_build();
    cN = 1;
    hld(1);
    tree.resize(cN+3);
    build_hld_segTree();
    geti(K);
    rep(i,K){
        int a,b,c; geti(a,b,c);
        if( a == 1 ){
            b--; int u = edges[b].Fi; int v = edges[b].Se;

```

```

        if( level[u] > level[v] ) swap(u,v);
        update_query(v,c);
    }else{
        printf("%d\n",query(b,c));
    }
}
}

```

4.2 Centroid decomposition

```

int n;
set<int> adj[MAXN];
int sub[MAXN], dep[MAXN];

void dfsSubtree(int node, int pnode) {
    sub[node] = 1;
    for(auto cnode : adj[node]) {
        if(cnode != pnode) {
            dfsSubtree(cnode, node);
            sub[node] += sub[cnode];
        }
    }
}

int findCentroid(int node, int pnode, int size) {
    for(auto cnode : adj[node]) {
        if(cnode != pnode && sub[cnode] > size / 2)
            return findCentroid(cnode, node, size);
    }
    return node;
}

bool decompose(int node, int depth) {
    bool result = true;
    if(depth >= 26) {
        return false;
    }
    dfsSubtree(node, -1);
    int ctr = findCentroid(node, -1, sub[node]);
    dep[ctr] = depth;
    for(auto cnode : adj[ctr]) {
        adj[cnode].erase(ctr);
        result &= decompose(cnode, depth + 1);
    }
    adj[ctr].clear();
    return result;
}

int main() {
    geti(n);
    rep(i, n-1) {
        int u, v;
        geti(u, v);
        adj[u].insert(v);
        adj[v].insert(u);
    }
}

```

```

    }
    if(decompose(1, 0)) {
        repp(i, n) printf("%c ", dep[i] + 'A');
    } else {
        cout << "Impossible!";
    }
}
}

```

4.3 Mo's algorithm

```

int N,M,K,tc;
ll c[1000005];
ll p[1000005]; int Bsize;
typedef struct query{
    int l,r,n; ll ans;
} query;
bool cmp(query& a, query& b){
    if( a.l/Bsize == b.l/Bsize ) return a.r < b.r;
    else return a.l/Bsize < b.l/Bsize;
}
bool cmp2(query&a, query& b ){ return a.n < b.n; }
int main(void)
{
    geti(N,M); rep(i,N) scanf("%lld",p+i);
    Bsize = (int) sqrt(1.0*N);
    vector<query> q;
    rep(i,M){
        int a,b; geti(a,b); a--;b--;
        q.push_back({a,b,i});
    }

    sort(all(q),cmp);
    int l=0, r=-1; ll sum = 0;

    for(int i=0;i<q.size();i++){
        query& e = q[i];
        int ql = e.l, qr = e.r;
        while( r < qr ){
            r++;
            sum += p[r]*(2*c[p[r]]+1); c[p[r]]++;
        }
        while( r > qr ){
            sum -= p[r]*(1-2*c[p[r]]); c[p[r]]--;
            r--;
        }
        while( l < ql ){
            sum += p[l]*(1-2*c[p[l]]); c[p[l]]--;
            l++;
        }
        while( l > ql ){
            l--;
            sum += p[l]*(2*c[p[l]]+1); c[p[l]]++;
        }
        e.ans = sum;
    }
}

```

```

    sort(all(q),cmp2);
    for(auto e : q ){
        printf("%lld\n",e.ans);
    }
}

```

4.4 Mo's algorithm on tree

```

int N;
int g[MAXN];
int f[MAXN];
int pa[LOGV][MAXV]; int level[MAXN];
int ST[MAXN], EN[MAXN], arr[MAXN*3];
int tt = 0;
vector<int> E[MAXN];

void dfs_build(int x, int p, int lev){
    pa[0][x] = p;
    level[x] = lev;
    ST[x] = ++tt; arr[tt] = x;
    for(auto e : E[x])if(e!=p){
        dfs_build(e,x,lev+1);
    }
    EN[x] = ++tt; arr[tt] = x;
}

void lca_build(){
    for(int k=1;k<LOGV;k++){
        repp(i,N){
            if( pa[k-1][i] != -1 )pa[k][i] = pa[k-1][pa[k-1][i]];
            else pa[k][i] = -1;
        }
    }
}

int lca(int x, int y){
    if( level[x] < level[y] ) swap(x,y);
    int diff = level[x] - level[y];
    for(int k=0;k<LOGV;k++){
        if( diff & (1<<k) ) x = pa[k][x];
    }

    if( x == y ) return x;
    for(int k=LOGV-1;k>=0;k--){
        if( pa[k][x] != pa[k][y] ){
            x = pa[k][x]; y = pa[k][y];
        }
    }
    return pa[0][x];
}

int Bsize;
struct query{
    int l,r,n;
};
bool cmp1(query& a, query& b){

```

```

    if( a.l/Bsize == b.l/Bsize ) return a.r < b.r;
    else return a.l/Bsize < b.l/Bsize;
};
bool cmp2(query&a, query& b ){ return a.n < b.n; }

ll ans[100500];
ll cnt[2][200500];
int vis[100500];
ll sum = 0;

void update(int x, int type){
    // add node to range
    if( type == 1 ){
        sum += cnt[g[x]^1][f[x]];
        cnt[g[x]][f[x]]++;
    }
    // remove node from range
    if( type == 0 ){
        sum -= cnt[g[x]^1][f[x]];
        cnt[g[x]][f[x]]--;
    }
}

int main(void){
    geti(N);
    repp(i,N) geti(g[i]);
    repp(i,N) geti(f[i]);

    set<int> flist;
    map<int,int> fmp;
    repp(i,N) flist.insert(f[i]);
    int tmp = 1;
    for(auto e: flist) fmp[e] = tmp++;
    repp(i,N) f[i] = fmp[f[i]];

    repp(i,N-1){
        int a,b; geti(a,b);
        E[a].pb(b); E[b].pb(a);
    }
    tt = 0;
    dfs_build(1,-1,0);
    lca_build();
    Bsize = (int) sqrt(1.0*tt);

    int Q; geti(Q);
    vector<query> v;
    repp(q,Q){
        int a,b; geti(a,b);
        if( ST[a] > ST[b] ) swap(a,b);
        int l = lca(a,b);
        if( a == l || b == l){
            v.push_back({ST[a],ST[b],q});
        }
        else{
            v.push_back({EN[a],ST[b],q});
        }
    }
}

```

```

}

sort(all(v),cmp1);
int l=1, r=0;
for(int i=0;i<v.size();i++){
    query& e = v[i];
    int ql = e.l, qr = e.r;
    while( r < qr ){
        r++;
        int node = arr[r];
        vis[node]++;
        if( vis[node] == 1 ) update(node,1);
        if( vis[node] == 2 ) update(node,0);
    }
    while( r > qr ){
        int node = arr[r];
        vis[node]--;
        if( vis[node] == 0 ) update(node,0);
        if( vis[node] == 1 ) update(node,1);
        r--;
    }
    while( l < ql ){
        int node = arr[l];
        vis[node]--;
        if( vis[node] == 0 ) update(node,0);
        if( vis[node] == 1 ) update(node,1);
        l++;
    }
    while( l > ql ){
        l--;
        int node = arr[l];
        vis[node]++;
        if( vis[node] == 1 ) update(node,1);
        if( vis[node] == 2 ) update(node,0);
    }

    int u = arr[ql]; int v = arr[qr];
    int l = lca(u,v);

    if( u != l && v != l ){
        int node = l;
        vis[node]++;
        if( vis[node] == 1 ) update(node,1);
        if( vis[node] == 2 ) update(node,0);
    }

    ans[e.n] += sum;

    if( u != l && v != l ){
        int node = l;
        vis[node]--;
        if( vis[node] == 0 ) update(node,0);
        if( vis[node] == 1 ) update(node,1);
    }
}
}

```

```

    repp(i,Q) printf("%lld\n",ans[i]);
}

```

4.5 Parallel binary search

```

int N,M,K,Q;

vector<Pi> edge[1000500];
int pa[MAXN]; int sz[MAXN];

// each query's answer
Pi ans[MAXN];
// each query's possible answer range for binary search
int low[MAXN], high[MAXN];
// focus[x] : list of query # where it's mid value is x
vector<int> focus[1000500];

int find(int x){
    if( x == pa[x] ) return x;
    return pa[x] = find(pa[x]);
}

int x[MAXN], y[MAXN];

void uni(int a, int b){
    a = find(a); b = find(b);
    if( a == b ) return;
    pa[a] = b;
    sz[b] += sz[a];
}

int main(void){
    //ios::sync_with_stdio(false);
    geti(N,M);
    int C = -1;
    repp(i,M){
        int a,b,c; geti(a,b,c);
        edge[c].push_back({a,b});
        C = max(C, c);
    }

    geti(Q);
    repp(i,Q){
        int a,b;
        geti(a,b); x[i] = a; y[i] = b;
        ans[i] = {INF,-1};
        // Initially, every query has answer in [0,C] range
        low[i] = 0; high[i] = C;
    }

    bool changed = true;
    while( changed ){
        changed = false;

        // Clear variables
        rep(i,C+1) focus[i].clear();
    }
}

```

```

    repp(i,N) pa[i] = i, sz[i] = 1;

    // Put each query into corresponding focus group
    repp(i,Q){
        if( low[i] > high[i] ) continue;
        focus[ (low[i] + high[i])/2 ].push_back(i);
    }

    // for every time 0~C
    for(int k=0;k<=C;k++){
        // perform action of that time
        for(auto e : edge[k]) uni(e.Fi,e.Se);

        // for each focus group
        // determine it's answer & next position
        for(auto e : focus[k]){
            changed = true;
            int a = x[e]; int b = y[e];
            if( find(a) == find(b) ){
                ans[e].Fi = min(ans[e].Fi, k);
                ans[e].Se = sz[find(a)];
                high[e] = k-1;
            }
            else{
                low[e] = k+1;
            }
        }
    }

    repp(i,Q){
        if( ans[i].Fi == INF ) printf("%d\n",-1);
        else printf("%d %d\n",ans[i].Fi, ans[i].Se);
    }
}

```

4.6 Lazy Propagation 1

```

int N,M,K;

struct segTree{
    struct Node{
        ll d, lazy;
    };
    vector<Node> data;
    int n;
    void init(int x){
        n = 1; while( n < x ) n *= 2;
        data.resize(n*2+10);
    }
    void propagate(int node, int nodeL, int nodeR){
        if( data[node].lazy == 0 ) return;
        ll len = nodeR - nodeL + 1;
        data[node].d += len*data[node].lazy;

```

```

        if( len > 1 ){
            data[node*2].lazy += data[node].lazy;
            data[node*2+1].lazy += data[node].lazy;
        }
        data[node].lazy = 0;
    }

    void update(int l, int r, ll val, int node, int nodeL, int nodeR){
        propagate(node, nodeL, nodeR);
        if( l > nodeR || r < nodeL ) return;
        if( l <= nodeL && nodeR <= r ){
            data[node].lazy += val;
            propagate(node,nodeL,nodeR);
            return;
        }
        update(l,r,val,node*2,nodeL,(nodeL+nodeR)/2);
        update(l,r,val,node*2+1,(nodeL+nodeR)/2+1,nodeR);
        data[node].d = data[node*2].d + data[node*2+1].d;
    }

    ll query(int l, int r, int node, int nodeL, int nodeR){
        propagate(node, nodeL, nodeR);
        if( l > nodeR || r < nodeL ) return 0;
        if( l <= nodeL && nodeR <= r ){
            return data[node].d;
        }
        ll sum = 0;
        sum += query(l,r,node*2,nodeL,(nodeL+nodeR)/2);
        sum += query(l,r,node*2+1,(nodeL+nodeR)/2+1,nodeR);
        return sum;
    }

};

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    segTree tree;
    cin >> N >> M >> K;
    tree.init(N);
    repp(i,N){
        ll x; cin >> x;
        tree.update(i,i,x,1,1,tree.n);
    }
    repp(i,M+K){
        int a; cin >> a;
        if( a == 1 ){
            int b,c; ll d;
            cin >> b >> c >> d;
            tree.update(b,c,d,1,1,tree.n);
        }
        else{
            int b,c; cin >> b >> c;
            printf("%lld\n",tree.query(b,c,1,1,tree.n));
        }
    }
}

```

}

5 Geometry

5.1 Closest pair

```

int N,M,T,K,V;

typedef struct Point{
    int x,y;
    bool operator<(const Point& l) const{
        if( y == l.y ) return x < l.x;
        return y < l.y;
    }
    bool operator==(const Point& l) const{
        return (x==l.x)&&(y==l.y);
    }
} Point;

bool cmp(const Point& l, const Point& r){
    if(l.x == r.x ) return l.y < r.y;
    return l.x < r.x;
}

int dist(Point& l, Point& r ){
    return (l.x-r.x)*(l.x-r.x) + (l.y-r.y)*(l.y-r.y);
}

int main(void)
{
    geti(N); vector<Point> v(N);
    for(int i=0;i<N;i++){
        int x ,y; geti(x,y); v[i].x = x; v[i].y = y;
    }
    sort(all(v),cmp);
    int ans = dist(v[0],v[1]); int left = 0;
    set<Point> possible; possible.insert(v[0]); possible.insert(v[1]);

    for(int i=2;i<N;i++){
        while( (v[i].x - v[left].x)*(v[i].x - v[left].x ) > ans ){
            possible.erase(v[left]);
            left++;
        }
        int d = (int) sqrt(ans) + 1;
        auto bottom = possible.lower_bound({-100000,v[i].y-d});
        auto top = possible.upper_bound({100000,v[i].y+d});
        for(auto it = bottom; it != top; it++){
            Point cur = *it;
            if( dist(v[i],cur) < ans ) ans = dist(v[i],cur);
        }
        possible.insert(v[i]);
    }
    cout << ans;
}

```

}

5.2 Convex hull

```

typedef struct Point{
    ll x,y,n;
} Point;

ll ccw(Point a, Point b, Point c){
    b.x -= a.x, b.y -= a.y;
    c.x -= a.x, c.y -= a.y;
    return b.x*c.y - c.x*b.y;
}

vector<Point> convex_hull(vector<Point> ps){
    if (ps.size() < 3)return ps;
    vector<Point> upper, lower;
    sort(ps.begin(), ps.end(), [](const Point &a, const Point &b) {
        if (a.x == b.x) return a.y < b.y; return a.x < b.x;
    });
    for(const auto &p : ps){ // ccw without `=` when include every point in
        convex hull
        while(upper.size() >= 2 && ccw(++upper.rbegin(), *upper.rbegin(), p) >=
            0)upper.pop_back();
        while(lower.size() >= 2 && ccw(++lower.rbegin(), *lower.rbegin(), p) <=
            0)lower.pop_back();
        upper.emplace_back(p);
        lower.emplace_back(p);
    }
    lower.insert(lower.end(), ++upper.rbegin(), --upper.rend());
    return lower;
}

vector<Point> convex_hull2(vector<Point> ps){ // sorting angle
    if (ps.size() < 3)return ps;
    vector<Point> convex;
    sort(ps.begin(), ps.end(), [](Point &a, Point &b){
        if(a.x == b.x)return a.y < b.y; return a.x<b.x;
    });
    Point d = ps[0];
    for(auto &p : ps){
        p.x -= d.x;p.y -= d.y;
    }
    sort(ps.begin(), ps.end(), [](Point &a, Point &b){
        if (ccw({0,0},a,b) == 0) return a.x*a.x + a.y*a.y < b.x*b.x + b.y*b.y;
        return ccw({0,0},a,b) > 0;
    });
    for(auto &p : ps){
        while(convex.size() >= 2 && ccw(++convex.rbegin(), *convex.rbegin(), p)
            <= 0)convex.pop_back();
        convex.emplace_back(p);
    }
    for(auto &p : convex){
        p.x += d.x;p.y += d.y;
    }
}

```

```

    return convex;
}

```

5.3 Rotating Calipers

```

int main(){
    vector<Point> convex;
    int ans = 0;

    int mid = 0;
    // if you want iterate `only` antipodal pairs
    // while(ccw(convex.back(), convex[0], convex[mid], convex[mid+1]) > 0) mid
    ++;

    for(int i=0,j=mid; i < convex.size();){
        // do something with pair of i, j

        int nextj = (j+1) % convex.size();
        int nexti = (i+1) % convex.size();
        if (ccw(convex[i], convex[nexti], convex[j], convex[nextj]) > 0)j =
            nextj;
        else i++;
    }
}

```

6 Math

6.1 FFT

```

#include <cmath>
#include <complex>
using namespace std;
typedef pair<int,int> pii;
typedef complex<double> base;

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

```

```

    }
}

if (invert) {
    for(int i=0;i<n;i++) a[i] /= n;
}

void multiply(const vector<int> &a, const vector<int> &b, vector<int> &res){
    vector<base> fa(a.begin(), a.end()), fb(b.begin(),b.end());
    int n = 1;
    while(n < max(a.size(), b.size())) n <= 1;
    n <= 1;
    fa.resize(n); fb.resize(n);
    fft(fa,false);fft(fb,false);
    for(int i=0;i<n;i++) fa[i] *= fb[i];
    fft(fa,true);
    res.resize(n);
    for(int i=0;i<n;i++) res[i] = int(fa[i].real() + (fa[i].real() > 0 ? 0.5 :
        -0.5));
}

```

6.2 Extended Euclidean

```

pair<int,int> ext_gcd(int a,int b){
    if(b){
        auto tmp = ext_gcd(b, a%b);
        return {tmp.second, tmp.first - (a/b) * tmp.second};
    } else return {1, 0};
}

int mod_inv(int a, int M){
    return (ext_gcd(a, M).first + M) % M;
}

```

6.3 Gaussian Elimination

```

#define MAX_N 300 // adjust this value as needed
struct AugmentedMatrix { double mat[MAX_N][MAX_N + MAX_N + 10]; };
struct ColumnVector { double vec[MAX_N]; };

// 0 indexed row and column
AugmentedMatrix GaussianElimination(int N, AugmentedMatrix Aug) {
    // input: N X 2N matrix [A I], output: [I invA]

    // forward eliminataion phase
    for(int i=0;i<N;i++){
        int l = i;
        // which row has largest column value
        for(int j=i+1;j<N;j++){
            if( fabs(Aug.mat[j][i]) > fabs(Aug.mat[l][i]) )
                l = j;
        }
        // swap this pivot row to minimize error
        for(int k=i;k<2*N;k++)

```

```

        swap(Aug.mat[i][k], Aug.mat[l][k]);
    // calculate forward elimination
    for(int j=i+1; j<N; j++)
        for(int k=2*N-1; k>=i; k--)
            Aug.mat[j][k] -= Aug.mat[i][k] * Aug.mat[j][i] / Aug.mat[i][i];
}

// normalize pivots
for(int i=0; i<N; i++)
    for(int j=2*N-1; j>=i; j--)
        Aug.mat[i][j] /= Aug.mat[i][i];

// backward elimination
for(int i=N-1; i>0; i--)
    for(int j=i-1; j>=0; j--)
        for(int k=2*N-1; k>=i; k--)
            Aug.mat[j][k] -= Aug.mat[i][k] * Aug.mat[j][i] / Aug.mat[i][i];

return Aug;
}

int main() {

    AugmentedMatrix Aug;
    int N; geti(N);
    rep(i, N) rep(j, N) scanf("%lf", &Aug.mat[i][j]);
    for(int i=N; i<2*N; i++) Aug.mat[i-N][i] = 1;

    AugmentedMatrix res = GaussianElimination(N, Aug);

    // Print inversion of A
    for(int i=0; i<N; i++){
        for(int j=N; j<2*N; j++) printf("%f ", res.mat[i][j]);
        printf("\n");
    }

    return 0;
}

```

6.4 Prime Algorithms

```

typedef long long ll;
using namespace std;

ll gcd(ll a, ll b) {
    if (b == 0)
        return a;
    return gcd(b, a%b);
}

namespace miller_rabin {
    ll mul(ll x, ll y, ll mod) { return (__int128)x * y % mod; }
    //ll mul(ll x, ll y, ll mod) { return x * y % mod; }
}

```

```

ll ipow(ll x, ll y, ll p) {
    ll ret = 1, piv = x % p;
    while (y) {
        if (y & 1) ret = mul(ret, piv, p);
        piv = mul(piv, piv, p);
        y >>= 1;
    }
    return ret;
}

bool miller_rabin(ll x, ll a) {
    if (x % a == 0) return 0;
    ll d = x - 1;
    while (1) {
        ll tmp = ipow(a, d, x);
        if (d & 1) return (tmp != 1 && tmp != x - 1);
        else if (tmp == x - 1) return 0;
        d >>= 1;
    }
}

bool isprime(ll x) {
    for (auto &i : { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 }) {
        if (x == i) return 1;
        if (x > 40 && miller_rabin(x, i)) return 0;
    }
    if (x <= 40) return 0;
    return 1;
}

namespace pollard_rho {
    ll f(ll x, ll n, ll c) {
        return (c + miller_rabin::mul(x, x, n)) % n;
    }

    void rec(ll n, vector<ll> &v) {
        if (n == 1) return;
        if (n % 2 == 0) {
            v.push_back(2);
            rec(n / 2, v);
            return;
        }
        if (miller_rabin::isprime(n)) {
            v.push_back(n);
            return;
        }
    }

    ll a, b, c;
    while (1) {
        a = rand() % (n - 2) + 2;
        b = a;
        c = rand() % 20 + 1;
        do {
            a = f(a, n, c);
            b = f(f(b, n, c), n, c);
        } while (gcd(abs(a - b), n) == 1);
        if (a != b) break;
    }
    ll x = gcd(abs(a - b), n);
    rec(x, v);
}

```



```

        rec(n / x, v);
    }
    vector<ll> factorize(ll n) {
        vector<ll> ret;
        rec(n, ret);
        sort(ret.begin(), ret.end());
        return ret;
    }
};

int main() {
    vector<ll> res;
    ll num;
    scanf("%lld", &num);
    res = pollard_rho::factorize(num);
    for (int i = 0; i < res.size(); ++i)
        printf("%lld\n", res[i]);
}

```

7 Miscellaneous

7.1 Grundy number

```

map<set<int>,int> grundy;
map<ll,set<int>> mp;

int get_grundy(set<int> x){
    // base case
    if( sz(x) == 0 ) return 0;
    if( grundy.find(x) != grundy.end() ) return grundy[x];

    set<int> S;
    int res = 0;

    auto iter = x.end(); iter--;
    int mx = *iter;

    // transition : which k to select
    for(int i=1;i<=mx;i++){
        set<int> nxt;
        for(auto e : x){
            if( e < i ) nxt.insert(e);
            else if( e == i ) continue;
            else nxt.insert(e-i);
        }
        S.insert(get_grundy(nxt));
    }

    // find mex and return
    while( S.find(res) != S.end() ) res++;
    grundy[x] = res;
    return res;
}

int main(void){

```

```

int n; geti(n);

// Simple prime factorization
rep(i,n){
    ll x; scanf("%lld",&x);
    for(ll i=2;i*i<=x;i++){
        if( x>0 && x%i == 0 ){
            int cnt = 0;
            while( x>0 && x%i == 0 ){
                cnt++; x/= i;
            }
            mp[i].insert(cnt);
        }
    }
    if( x > 1 ){
        mp[x].insert(1);
    }
}

int res = 0;
for(auto e : mp){
    res ^= get_grundy(e.Se);
}

if( res == 0 ) printf("Arpa");
else printf("Mojtaba");
}

```

7.2 Hungarian

```

// Min cost bipartite matching via shortest augmenting paths
//
// This is an O(n^3) implementation of a shortest augmenting path
// algorithm for finding min cost perfect matchings in dense
// graphs. In practice, it solves 1000x1000 problems in around 1
// second.
//
// cost[i][j] = cost for pairing left node i with right node j
// Lmate[i] = index of right node that left node i pairs with
// Rmate[j] = index of left node that right node j pairs with
//
// The values in cost[i][j] may be positive or negative. To perform
// maximization, simply negate the cost[][] matrix.

typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {
    int n = int(cost.size());

    // construct dual feasible solution
    VD u(n);
    VD v(n);

```

```

for (int i = 0; i < n; i++) {
    u[i] = cost[i][0];
    for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
}
for (int j = 0; j < n; j++) {
    v[j] = cost[0][j] - u[0];
    for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
}

// construct primal solution satisfying complementary slackness
Lmate = VI(n, -1);
Rmate = VI(n, -1);
int mated = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        if (Rmate[j] != -1) continue;
        if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
            Lmate[i] = j;
            Rmate[j] = i;
            mated++;
            break;
        }
    }
}

VD dist(n);
VI dad(n);
VI seen(n);

// repeat until primal solution is feasible
while (mated < n) {

    // find an unmatched left node
    int s = 0;
    while (Lmate[s] != -1) s++;

    // initialize Dijkstra
    fill(dad.begin(), dad.end(), -1);
    fill(seen.begin(), seen.end(), 0);
    for (int k = 0; k < n; k++)
        dist[k] = cost[s][k] - u[s] - v[k];

    int j = 0;
    while (true) {

        // find closest
        j = -1;
        for (int k = 0; k < n; k++) {
            if (seen[k]) continue;
            if (j == -1 || dist[k] < dist[j]) j = k;
        }
        seen[j] = 1;

        // termination condition
        if (Rmate[j] == -1) break;
    }
}

```

```

// relax neighbors
const int i = Rmate[j];
for (int k = 0; k < n; k++) {
    if (seen[k]) continue;
    const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
    if (dist[k] > new_dist) {
        dist[k] = new_dist;
        dad[k] = j;
    }
}

// update dual variables
for (int k = 0; k < n; k++) {
    if (k == j || !seen[k]) continue;
    const int i = Rmate[k];
    v[k] += dist[k] - dist[j];
    u[i] -= dist[k] - dist[j];
}
u[s] += dist[j];

// augment along path
while (dad[j] >= 0) {
    const int d = dad[j];
    Rmate[j] = Rmate[d];
    Lmate[Rmate[j]] = j;
    j = d;
}
Rmate[j] = s;
Lmate[s] = j;

mated++;
}

double value = 0;
for (int i = 0; i < n; i++)
    value += cost[i][Lmate[i]];

return value;
}

```

7.3 Hungarian2

```

/*
Tests
http://www.spoj.com/problems/GREED/
https://www.acmicpc.net/problem/8992
SRM 506 mid

```

Time complexity $O(n^3)$

```

Usage
MinWeightBipartiteMatch matcher(n);
for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) matcher.weights[i][j] =
    SOMETHING;

```

```

cost_t total = matcher.solve();

See matcher.match(row -> col) and matcher.matched(col -> row) for actual match
*/

struct MinWeightBipartiteMatch
{
    typedef long long cost_t;

    cost_t max_cost() const { return numeric_limits<cost_t>::max(); }

    // input
    int n;
    vector<vector<cost_t>> weights;
    // output
    vector<int> match, matched;

    MinWeightBipartiteMatch(int n) :
        n(n), match(n), matched(n), weights(n, vector<cost_t>(n))
    {

    }

    void resize(int n) {
        this->n = n;
        match.resize(n);
        matched.resize(n);
        weights.resize(n);
        for (int i = 0; i < n; i++) {
            weights[i].resize(n);
        }
    }

    /* for solve() */
    vector<cost_t> slack;
    vector<cost_t> potential_row, potential_col;
    vector<int> reach_row, reach_col;
    int rcnt;
    vector<int> from;
    void found_match(int r, int c) {
        do {
            int old_match = match[r];
            match[r] = c;
            matched[c] = r;
            tie(r, c) = make_pair(from[r], old_match);
        } while (r >= 0 && c >= 0);
    }

    void augment(int row_to_match) {
        slack.resize(n);
        for (int c = 0; c < n; c++) {
            slack[c] = weights[row_to_match][c] - potential_row[row_to_match] -
                potential_col[c];
        }
        ++rcnt;
        vector<int> q; q.reserve(n);

```

```

        int h = 0;
        q.push_back(row_to_match);
        reach_row[row_to_match] = rcnt;
        from[row_to_match] = -1;
        for (;;) {
            while (h < q.size()) {
                int r = q[h++];
                for (int c = 0; c < n; c++) {
                    cost_t gap = weights[r][c] - potential_row[r] - potential_col[c];
                    slack[c] = min(slack[c], gap);
                    if (gap != cost_t()) continue;
                    int next = matched[c];
                    if (next < 0) {
                        found_match(r, c);
                        return;
                    }
                    reach_col[c] = rcnt;
                    if (reach_row[next] == rcnt) continue;
                    q.push_back(next);
                    reach_row[next] = rcnt;
                    from[next] = r;
                }
            }
            cost_t delta = max_cost();
            for (int c = 0; c < n; c++) {
                if (reach_col[c] == rcnt) continue; // non-covered -> continue
                delta = min(delta, slack[c]);
            }
            for (int r = 0; r < n; r++) {
                if (reach_row[r] == rcnt) continue;
                potential_row[r] -= delta;
            }
            for (int c = 0; c < n; c++) {
                if (reach_col[c] == rcnt) continue;
                potential_col[c] += delta;
                slack[c] -= delta;
            }
            int lastsize = q.size();
            for (int c = 0; c < n; c++) {
                if (reach_col[c] == rcnt) continue;
                if (slack[c] != cost_t()) continue;
                int next = matched[c];
                if (next >= 0 && reach_row[next] == rcnt) continue;
                for (int qi = 0; qi < lastsize; qi++) {
                    int r = q[qi];
                    cost_t gap = weights[r][c] - potential_row[r] - potential_col[c];
                    if (gap != cost_t()) continue;
                    if (next < 0) {
                        found_match(r, c);
                        return;
                    }
                }
                reach_col[c] = rcnt;
                q.push_back(next);
                reach_row[next] = rcnt;
                from[next] = r;
                break;
            }
        }
    }

```

```

    }
}
}

void initialize() {
    potential_row.assign(n, cost_t());
    potential_col.assign(n, cost_t());
    match.assign(n, -1);
    matched.assign(n, -1);
    reach_row.assign(n, 0);
    reach_col.assign(n, 0);
    from.resize(n);
    rcnt = 1;
    for (int i = 0; i < n; i++) {
        cost_t row_min_weight = *min_element(weights[i].begin(), weights[i].end());
        ;
        potential_row[i] = row_min_weight;
    }
    for (int i = 0; i < n; i++) {
        cost_t col_min_weight = weights[0][i] - potential_row[0];
        for (int j = 1; j < n; j++) col_min_weight = min(col_min_weight, weights[j][i] - potential_row[j]);
        potential_col[i] = col_min_weight;
    }
}

cost_t solve() {
    initialize();
    for (int row_to_match = 0; row_to_match < n; row_to_match++) {
        augment(row_to_match);
    }
    cost_t ans = cost_t();
    for (auto v : potential_row) ans += v;
    for (auto v : potential_col) ans += v;
    return ans;
}
};

```

7.4 LiChao Tree

```

// LiChaoTree for dynamic CHT trick
// This example maintains CHT for finding MAXIMUM of corresponding x
// op=1 : add ax + b into CHT
// op=2 : find max value of position x
// https://cp-algorithms.com/geometry/convex_hull_trick.html
ll f(Pll line, ll x){
    return line.Fi*x + line.Se;
}

vector<ll> xlist;
struct LiChaoTree{
    int n; vector<Pll> d;
    void init(int x){
        n = 1; while (n < x) n *= 2;
    }
}

```

```

        d.resize(n*2+10);
        for(auto& e : d){
            e = {0, -3*(1e18)};
        }
    }

void insert(int node, int nL, int nR, Pll newline){
    if( nL == nR ){
        if( f(d[node], xlist[nL]) < f(newline, xlist[nL]) ) d[node] =
            newline;
        return;
    }
    bool left = f(d[node], xlist[nL]) < f(newline, xlist[nL]);
    bool right = f(d[node], xlist[nR]) < f(newline, xlist[nR]);

    // take upper, lower line based on leftmost point of the segment
    Pll upper = d[node], lower = newline;
    if( left ) swap(upper, lower);

    // one line totally cover another line
    if( left == right ){
        d[node] = upper; return;
    }

    int m = (nL+nR)/2;
    // intersection in left half segment
    if( f(upper, xlist[m]) <= f(lower, xlist[m]) ){
        d[node] = lower;
        insert(node*2, nL, m, upper);
    }
    // intersection in right half segment
    else{
        d[node] = upper;
        insert(node*2+1, m+1, nR, lower);
    }
}

ll query(int node, int nL, int nR, int pos){
    if( nL == nR ) return f(d[node], xlist[pos]);

    int m = (nL+nR)/2;
    ll nval = -3*(1e18);
    if( pos <= m ) nval = query(node*2, nL, m, pos);
    else nval = query(node*2+1, m+1, nR, pos);

    return max(nval, f(d[node], xlist[pos]) );
}

};

int main(){
    int Q; scanf("%d",&Q);
    vector<pair<int,Pll>> qlist;
    repp(q,Q){
        int op; scanf("%d",&op);
        if( op == 1 ){

```

```

        ll a,b; scanf("%lld%lld",&a,&b);
        qlist.push_back({1,{a,b}});
    }
    else{
        ll x; scanf("%lld",&x);
        xlist.push_back(x);
        qlist.push_back({2,{x,x}});
    }
}

xlist.push_back(-2*(1e12) - 10);
sort(all(xlist));
xlist.erase(unique(all(xlist)), xlist.end());
LiChaoTree tree;
tree.init( sz(xlist)+1 );

// careful to put padding into xlist
// so that it fits to tree size
while( sz(xlist) < tree.n+5 ) xlist.push_back(2*(1e12));

for(auto q : qlist){
    if( q.Fi == 1 ){
        tree.insert(1,1,tree.n,q.Se);
    }
    if( q.Fi == 2 ){
        int pos = lower_bound(all(xlist), q.Se.Fi) - xlist.begin();
        printf("%lld\n",tree.query(1,1,tree.n,pos));
    }
}
}

```

7.5 Persistence Segment Tree

```

int n, cnt;
int root[MAXN];

struct node {
    int sum, left, right;
} tree[3 * MAXN * LOGN];

int build(int l = 0, int r = n) {
    int idx = ++cnt;
    if(r - l <= 1) {
        tree[idx] = {0, 0, 0};
        return idx;
    }
    int mid = (l + r) >> 1;
    tree[idx] = {0, build(l, mid), build(mid, r)};
    return idx;
}

int update(int x, int prev, int l = 0, int r = n) {
    if(x < l || r <= x) return prev;
    int idx = ++cnt;

```

```

    if(r - l <= 1) {
        tree[idx] = {1, 0, 0};
        return idx;
    }

    int mid = (l + r) >> 1;
    int L = update(x, tree[prev].left, l, mid);
    int R = update(x, tree[prev].right, mid, r);
    tree[idx] = {tree[L].sum + tree[R].sum, L, R};
    return idx;
}

int query(int x, int y, int k, int l = 0, int r = n) {
    if(r - l <= 1) return l;
    int mid = (l + r) >> 1;
    int leftSum = tree[tree[y].left].sum - tree[tree[x].left].sum;
    if(leftSum >= k)
        return query(tree[x].left, tree[y].left, k, l, mid);
    else
        return query(tree[x].right, tree[y].right, k - leftSum, mid, r);
}

int a[MAXN], rev[MAXN];
map<int, int> M;

int main() {
    int q;
    geti(n, q);
    for(int i = 1; i <= n; i++) {
        geti(a[i]);
        rev[i-1] = a[i];
    }
    sort(rev, rev + n);
    for(int i = 0; i < n; i++)
        M[rev[i]] = i;
    for(int i = 1; i <= n; i++)
        a[i] = M[a[i]];

    root[0] = build();
    for(int i = 1; i <= n; i++)
        root[i] = update(a[i], root[i-1]);

    while(q--) {
        int i, j, k;
        geti(i, j, k);
        printf("%d\n", rev[query(root[i-1], root[j], k)]);
    }
}

```

7.6 XOR FFT

```

#include <cstdio>
#include <complex>

```

```

const int SZ = 20, N = 1 << SZ;

using namespace std;

int Rev(int x) {
    int i, r = 0;
    for (i = 0; i < SZ; i++) {
        r = r << 1 | x & 1;
        x >>= 1;
    }
    return r;
}

void FFT(int *a, bool f) {
    int i, j, k, z;
    for (i = 0; i < N; i++) {
        j = Rev(i);
        if (i < j) {
            z = a[i];
            a[i] = a[j];
            a[j] = z;
        }
    }
    for (i = 1; i < N; i <= 1) for (j = 0; j < N; j += i << 1) for (k = 0; k <
        i; k++) {
        z = a[i + j + k];
        a[i + j + k] = a[j + k] - z;
        a[j + k] += z;
    }
    if (f) for (i = 0; i < N; i++) a[i] /= N;
}

int X[N];

int main() {
    int i, n;
    scanf("%d", &n);
    for (i = 0; i < 1 << n; i++) scanf("%d", &X[i]);
    FFT(X, false);
    for (i = 0; i < N; i++) X[i] *= X[i];
    FFT(X, true);
    for (i = 0; i < 1 << n; i++) printf("%d ", X[i]);
}

```

7.7 NTT

```

#include <cstdio>

const int A = 7, B = 26, P = A << B | 1, R = 3;
const int SZ = 20, N = 1 << SZ;

int Pow(int x, int y) {
    int r = 1;
    while (y) {

```

```

        if (y & 1) r = (long long)r * x % P;
        x = (long long)x * x % P;
        y >>= 1;
    }
    return r;
}

void FFT(int *a, bool f) {
    int i, j, k, x, y, z;
    j = 0;
    for (i = 1; i < N; i++) {
        for (k = N >> 1; j >= k; k >>= 1) j -= k;
        j += k;
        if (i < j) {
            k = a[i];
            a[i] = a[j];
            a[j] = k;
        }
    }
    for (i = 1; i < N; i <= 1) {
        x = Pow(f ? Pow(R, P - 2) : R, P / i >> 1);
        for (j = 0; j < N; j += i << 1) {
            y = 1;
            for (k = 0; k < i; k++) {
                z = (long long)a[i | j | k] * y % P;
                a[i | j | k] = a[j | k] - z;
                if (a[i | j | k] < 0) a[i | j | k] += P;
                a[j | k] += z;
                if (a[j | k] >= P) a[j | k] -= P;
                y = (long long)y * x % P;
            }
        }
    }
    if (f) {
        j = Pow(N, P - 2);
        for (i = 0; i < N; i++) a[i] = (long long)a[i] * j % P;
    }
}

int X[N];

int main() {
    int i, n;
    scanf("%d", &n);
    for (i = 0; i <= n; i++) scanf("%d", &X[i]);
    FFT(X, false);
    for (i = 0; i < N; i++) X[i] = (long long)X[i] * X[i] % P;
    FFT(X, true);
    for (i = 0; i <= n + n; i++) printf("%d ", X[i]);
}

```

7.8 2D FFT

```

const double EPS = 0.00001;

```

```

typedef complex<double> base;

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

void multiply(const vector<int> &a, const vector<int> &b, vector<int> &res){
    vector<base> fa(a.begin(), a.end()), fb(b.begin(),b.end());
    int n = 1;
    while(n < max(a.size(), b.size())) n <= 1;
    n <= 1;
    fa.resize(n); fb.resize(n);
    fft(fa,false);fft(fb,false);
    for(int i=0;i<n;i++) fa[i] *= fb[i];
    fft(fa,true);
    res.resize(n);
    for(int i=0;i<n;i++) res[i] = int(fa[i].real() + (fa[i].real() > 0 ? 0.5 :
        -0.5));
}

void multiply_complex(const vector<base> &a, const vector<base> &b, vector<base>
    &res){
    vector<base> fa(a.begin(), a.end()), fb(b.begin(),b.end());
    int n = 1;
    while(n < max(a.size(), b.size())) n <= 1;
    n <= 1;
    fa.resize(n); fb.resize(n);
    fft(fa,false);fft(fb,false);
    for(int i=0;i<n;i++) fa[i] *= fb[i];
    fft(fa,true);
    res.resize(n);
    for(int i=0;i<n;i++) res[i] = fa[i];
}

```

```

const int MAXN = 405;
const int LOGN = 19;

string S[MAXN], T[MAXN];

int main() {
    int n, m;
    geti(n, m);
    for(int i = 0; i < n; i++)
        cin >> S[i];
    int r, c;
    geti(r, c);
    for(int i = 0; i < r; i++)
        cin >> T[i];

    int p = 1, q = 1;
    while(q < m+c) q <= 1;
    while(p < n+r) p <= 1;

    vector<vector<base>> a(p, vector<base>(q)), b(p, vector<base>(q));
    for(int i = 0; i < p; i++) {
        for(int j = 0; j < q; j++) {
            int t = S[i%n][j%m] - 'a';
            double ang = 2*acos(-1)*t/26;
            a[i][j] = base(cos(ang), sin(ang));
        }
    }

    int cnt = 0;
    for(int i = 0; i < r; i++) {
        for(int j = 0; j < c; j++) {
            if(T[i][j] != '?') {
                cnt++;
                int t = T[i][j] - 'a';
                double ang = 2*acos(-1)*t/26;
                b[(r-1)-i][(c-1)-j] = base(cos(-ang), sin(-ang));
            }
        }
    }

    vector<vector<base>> fa, fb, res;
    for(int i = 0; i < p; i++) {
        vector<base> ta(a[i].begin(), a[i].end()), tb(b[i].begin(), b[i].end());
        fft(ta, false);
        fft(tb, false);
        fa.push_back(ta);
        fb.push_back(tb);
    }

    for(int j = 0; j < q; j++) {
        vector<base> ta(p), tb(p), tmp;
        for(int i = 0; i < p; i++) {
            ta[i] = fa[i][j];
            tb[i] = fb[i][j];
        }
        multiply_complex(ta, tb, tmp);
    }
}

```

```

    if(j == 0)
        res.resize(tmp.size(), vector<base>(q));

    for(int i = 0; i < res.size(); i++)
        res[i][j] = tmp[i];
}

for(int i = 0; i < res.size(); i++)
    fft(res[i], true);

for(int i = 0; i < n; i++) {
    for(int j = 0; j < m; j++) {
        if(abs(res[i+r-1][j+c-1].real() - cnt) < EPS && abs(res[i+r-1][j+c-1].imag()) < EPS) printf("1");
        else printf("0");
    }
    printf("\n");
}
}

```

7.9 Divide and Conquer DP optimization

```

void dfs(int s, int e, int p, int q)
{
    if (s > e) return;
    int m = (s+e)>>1, opt;
    D[m] = 1e18;
    for (int k=p; k<=q&&k<m; k++){
        lld v = E[k] + (lld)(m-k-1)*(S[m]-S[k]);
        if (D[m] > v)
            D[m] = v, opt = k;
    }
    dfs(s, m-1, p, opt);
    dfs(m+1, e, opt, q);
}

```

7.10 Order Statistic Tree

```

#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
    tree_order_statistics_node_update

// Need this
// We can run this code on codeforces
// http://codeforces.com/blog/entry/11080
using namespace __gnu_pbds;

typedef tree<
int,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update>
ordered_set;

```

```

int main(){
    ordered_set X;
    X.insert(1);
    X.insert(2);
    X.insert(4);
    X.insert(8);
    X.insert(16);

    cout<<*X.find_by_order(1)<<endl; // 2
    cout<<*X.find_by_order(2)<<endl; // 4
    cout<<*X.find_by_order(4)<<endl; // 16
    cout<<(end(X)==X.find_by_order(6))<<endl; // true

    cout<<X.order_of_key(-5)<<endl; // 0
    cout<<X.order_of_key(1)<<endl; // 0
    cout<<X.order_of_key(3)<<endl; // 2
    cout<<X.order_of_key(4)<<endl; // 2
    cout<<X.order_of_key(400)<<endl; // 5
}

```

7.11 BITSET

```

#define M 32
int main()
{
    // default constructor initializes with all bits 0
    bitset<M> bset1;

    // bset2 is initialized with bits of 20
    bitset<M> bset2(20);

    // bset3 is initialized with bits of specified binary string
    bitset<M> bset3(string("1100"));

    // cout prints exact bits representation of bitset
    cout << bset1 << endl; // 00000000000000000000000000000000
    cout << bset2 << endl; // 0000000000000000000000000000010100
    cout << bset3 << endl; // 000000000000000000000000000001100
    cout << endl;

    // declaring set8 with capacity of 8 bits

    bitset<8> set8; // 00000000

    // setting first bit (or 6th index)
    set8[1] = 1; // 00000010
    set8[4] = set8[1]; // 00010010
    cout << set8 << endl;

    // count function returns number of set bits in bitset
    int numberof1 = set8.count();

    // size function returns total number of bits in bitset
}

```



```

// so there difference will give us number of unset(0)
// bits in bitset
int numberof0 = set8.size() - numberof1;
cout << set8 << " has " << numberof1 << " ones and "
    << numberof0 << " zeros\n";

// test function return 1 if bit is set else returns 0
cout << "bool representation of " << set8 << " : ";
for (int i = 0; i < set8.size(); i++)
    cout << set8.test(i) << " ";

cout << endl;

// any function returns true, if atleast 1 bit
// is set
if (!set8.any())
    cout << "set8 has no bit set.\n";

if (!bset1.any())
    cout << "bset1 has no bit set.\n";

// none function returns true, if none of the bit
// is set
if (!bset1.none())
    cout << "bset1 has all bit set\n";

// bset.set() sets all bits
cout << set8.set() << endl;

// bset.set(pos, b) makes bset[pos] = b
cout << set8.set(4, 0) << endl;

// bset.set(pos) makes bset[pos] = 1 i.e. default
// is 1
cout << set8.set(4) << endl;

// reset function makes all bits 0
cout << set8.reset(2) << endl;
cout << set8.reset() << endl;

// flip function flips all bits i.e. 1 <-> 0
// and 0 <-> 1
cout << set8.flip(2) << endl;
cout << set8.flip() << endl;

// Converting decimal number to binary by using bitset
int num = 100;
cout << "\nDecimal number: " << num
    << " Binary equivalent: " << bitset<8>(num);

return 0;
}

int main()
{
    bitset<4> bset1(9);    // bset1 contains 1001

```

```

    bitset<4> bset2(3);    // bset2 contains 0011

    // comparison operator
    cout << (bset1 == bset2) << endl;    // false 0
    cout << (bset1 != bset2) << endl;    // true 1

    // bitwise operation and assignment
    cout << (bset1 ^= bset2) << endl;    // 1010
    cout << (bset1 &= bset2) << endl;    // 0010
    cout << (bset1 |= bset2) << endl;    // 0011

    // left and right shifting
    cout << (bset1 <<= 2) << endl;        // 1100
    cout << (bset1 >>= 1) << endl;        // 0110

    // not operator
    cout << (~bset2) << endl;            // 1100

    // bitwise operator
    cout << (bset1 & bset2) << endl;      // 0010
    cout << (bset1 | bset2) << endl;      // 0111
    cout << (bset1 ^ bset2) << endl;      // 0101
}

```


