

# Contents

1	exgcd.cpp	1
2	persist_segtree_on_tree.cpp	1
3	4dir_move.cpp	4
4	ac_automaton.cpp	5
5	automata.cpp	8
6	backpack_on_tree.cpp	11
7	bit_decomposition.cpp	13
8	bi_dir_search.cpp	15
9	bsgs.cpp	16
10	catalan.cpp	17
11	dijkstra.cpp	18
12	discretization.cpp	18
13	disjoint_union.cpp	19
14	dp_on_tree.cpp	19
15	dsu_merge.cpp	22
16	du_sieve.cpp	24
17	euler_path.cpp	26
18	euler_sieve.cpp	28
19	euler_tour.cpp	29
20	fenwick.cpp	34
21	fft.cpp	34
22	heavy_path_decomposition.cpp	36
23	inverse_dp.cpp	41
24	kmp.cpp	43
25	kruskal_smin_span_tree.cpp	44
26	lca.cpp	47
27	leftist_tree.cpp	48

28	manacher.cpp	49
29	matrix.cpp	51
30	min_string.cpp	52
31	mono_queue.cpp	52
32	network_flow.cpp	53
33	network_flow_cost.cpp	55
34	paren_match.cpp	59
35	parser.cpp	60
36	persist_segtree.cpp	62
37	persist_trie.cpp	64
38	plug_dp.cpp	65
39	qpow.cpp	68
40	range_dp.cpp	68
41	rng.cpp	70
42	sa.cpp	70
43	sam.cpp	74
44	search_mem_hash.cpp	76
45	segtree.cpp	80
46	segtree_offline.cpp	83
47	segtree_straddle.cpp	86
48	seg_intersect.cpp	89
49	spfa.cpp	92
50	sqrt_block.cpp	95
51	sqrt_graph.cpp	98
52	st.cpp	102
53	string_hashing.cpp	103
54	tangent_segtree.cpp	105
55	tarjan_bcc.cpp	107

56	tarjan_cut_edge.cpp	111
57	tarjan_cut_vertex.cpp	112
58	tarjan_dcc.cpp	113
59	topo_sort.cpp	117
60	tree_center.cpp	120
61	tree_diameter.cpp	120
62	trie.cpp	120
63	vec.cpp	123

## 1 exgcd.cpp

```
void exgcd(int &x,int &y,int a,int b)
{
    if(!b)
    {
        x=1;
        y=0;
        return;
    }
    exgcd(x,y,b,a%b);
    int t=x;
    x=y;
    y=t-a/b*y;
}
```

## 2 persist\_segtree\_on\_tree.cpp

```
// 1231
#include <cstdio>
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <algorithm>

struct Node
{
    int low;
    int high;
    int count;
};

int to [200000 + 10];
int pre [200000 + 10];
int last [100000 + 10];
```

```

int V [100000 + 10];
int depth[100000 + 10];
int lca [100000 + 10][20];
int root [100000 + 10];
Node tree [2000000 + 10];
int nextId(1);

int increase(int oldNode, int low, int high, int value)
{
    int newNode(nextId);
    ++nextId;
    tree[newNode] = tree[oldNode];
    ++tree[newNode].count;
    if (low != high)
    {
        int mid((low + high) / 2);
        if (value <= mid)
            tree[newNode].low = increase(tree[oldNode].low, low, mid, value);
        else
            tree[newNode].high = increase(tree[oldNode].high, mid + 1, high,
↪ value);
    }
    return newNode;
}

void dfs(int father, int v, int d)
{
    depth[v] = d;
    lca[v][0] = father;
    for (int i(1); (1 << i) <= depth[v]; ++i)
        lca[v][i] = lca[lca[v][i - 1]][i - 1];
    root[v] = increase(root[father], 0, 100000, V[v]);

    for (int i(last[v]); i != 0; i = pre[i])
    {
        if (to[i] != father)
            dfs(v, to[i], d + 1);
    }
}

int getLca(int u, int v)
{
    if (depth[u] < depth[v])
        std::swap(u, v);
    for (int i(19); i >= 0; --i)
    {
        if (depth[u] - (1 << i) >= depth[v])
            u = lca[u][i];
    }
    if (u == v)
        return u;
}

```

```

    for (int i(19); i >= 0; --i)
    {
        if (lca[u][i] != lca[v][i])
        {
            u = lca[u][i];
            v = lca[v][i];
        }
    }
    return lca[u][0];
}

int query(int uNode, int vNode, int lcaNode, int faLcaNode, int low, int high, int k)
{
    if (low == high)
        return low;
    int mid((low + high) / 2);
    int lowCount( tree[tree[uNode].low].count + tree[tree[vNode].low].count
        - tree[tree[lcaNode].low].count - tree[tree[faLcaNode].low].count);
    if (k <= lowCount)
        return query(tree[uNode].low, tree[vNode].low, tree[lcaNode].low,
↪ tree[faLcaNode].low, low, mid, k);
    else
        return query(tree[uNode].high, tree[vNode].high, tree[lcaNode].high,
↪ tree[faLcaNode].high, mid + 1, high, k - lowCount);
}

int main()
{
    int N, M;
    std::cin >> N >> M;
    for (int i(1); i <= N; ++i)
        std::cin >> V[i];
    for (int i(1); i <= N - 1; ++i)
    {
        int x, y;
        std::cin >> x >> y;
        to[i * 2 - 1] = y;
        pre[i * 2 - 1] = last[x];
        last[x] = i * 2 - 1;

        to[i * 2] = x;
        pre[i * 2] = last[y];
        last[y] = i * 2;
    }
    dfs(0, 1, 0);

    int lastAns(0);
    for (int i(1); i <= M; ++i)
    {
        int u, v, k;

```

```

        std::cin >> u >> v >> k;
        u ^= lastAns;
        int ans(query(root[u], root[v], root[getLca(u, v)], root[lca[getLca(u,
↪ v]][0]], 0, 100000, k));
        std::cout << ans << std::endl;
        lastAns = ans;
    }

    return 0;
}

```

### 3 4dir\_move.cpp

```

#include <cstdio>
#include <queue>

struct Point
{
    int x;
    int y;
    Point (int x, int y) : x(x), y(y)
    {
    }
};

int n, m, q, k;
int ans (0);
const int maxN (1000 + 10);
const int maxK (1000000 + 10);
int s [maxN] [maxN];
bool in [maxN] [maxN];
int xDir [4] = {1, 0, -1, 0};
int yDir [4] = {0, 1, 0, -1};
std::queue<Point> queue[maxK];

bool can(const Point &p)
{
    return p.x >= 1 && p.x <= n
        && p.y >= 1 && p.y <= m
        && !in[p.x][p.y];
}

void expand(int color)
{
    while (!queue[color].empty())
    {
        Point p(queue[color].front());
        queue[color].pop();
        ++ans;

        for (int i(0); i < 4; ++i)

```

```

        {
            Point nxt(p.x + xDir[i], p.y + yDir[i]);
            if (can(nxt))
            {
                in[nxt.x][nxt.y] = true;
                queue[s[nxt.x][nxt.y]].push(nxt);
            }
        }
        queue[color] = std::queue<Point>();
    }

int main()
{
    //freopen("color.in", "r", stdin);
    //freopen("color.out", "w", stdout);

    scanf("%d %d %d %d", &n, &m, &q, &k);
    for (int i(1); i <= n; ++i)
    {
        for (int j(1); j <= m; ++j)
        {
            scanf("%d", &s[i][j]);
        }
    }
    in[1][1] = true;
    queue[s[1][1]].push(Point(1, 1));
    expand(s[1][1]);

    for (int qq(1); qq <= q; ++qq)
    {
        int x;
        scanf("%d", &x);
        expand(x);
        printf("%d\n", ans);
    }

    return 0;
}

```

## 4 ac\_automaton.cpp

```

#include <cstdio>
#include <cstring>
#include <queue>
#include <string>
#include <iostream>
#include <vector>

const int maxN (10000000 + 10);
const int maxM (100000 + 10);

```

```

const int maxLen (100 + 10);
const int maxCh  (256);
int      map      [maxCh];
char     in       [maxN];
int      orig     [maxN];
int      pattern[maxM][maxLen];
int      len      [maxM];
int      child    [maxN][5];
int      fail     [maxN];
bool     matched  [maxN];

std::vector<int> sorted;

//char unmap[maxCh];
//std::string str[maxN];

int N, M;

int lastNode(0);
void insert(int index)
{
    for (int i(1), node(0); i <= len[index]; ++i)
    {
        int &ch(child[node][pattern[index][i]]);
        if (!ch)
        {
            ch = ++lastNode;
            //str[ch] = str[node] + unmap[pattern[index][i]];
        }
        node = ch;
    }
}

void build()
{
    std::queue<int> queue;
    queue.push(0);

    while (!queue.empty())
    {
        int node(queue.front());
        queue.pop();
        sorted.push_back(node);

        for (int i(1); i <= 4; ++i)
        {
            int &ch(child[node][i]);
            if (ch)
            {
                if (node)
                {

```



```

        fail[ch] = child[fail[node]][i];
    }
    queue.push(ch);
}
else
{
    ch = child[fail[node]][i];
}
}
}

void match()
{
    for (int i(1), node(0); i <= N; ++i)
    {
        node = child[node][orig[i]];
        matched[node] = true;
        /*std::cout << "---\n";
        for (int k(1); k <= i; ++k)
        {
            printf("%c", unmap[orig[k]]);
        }
        printf("\n");
        for (int k(1); k <= i - int(str[node].length()); ++k)
        {
            printf(" ");
        }
        std::cout << str[node] << '\n'; //*/

    }

    for (int i(lastNode); i >= 0; --i)
    {
        int node(sorted[i]);
        matched[fail[node]] = (matched[fail[node]] || matched[node]);
    }
}

int main()
{
    map[int('E')] = 1;
    map[int('S')] = 2;
    map[int('W')] = 3;
    map[int('N')] = 4;

    /*unmap[1] = 'E';
    unmap[2] = 'S';
    unmap[3] = 'W';
    unmap[4] = 'N'; //*/

    scanf("%d %d", &N, &M);

```

```

scanf("%s", in + 1);
for (int i(1); i <= N; ++i)
{
    orig[i] = map[int(in[i])];
}
for (int i(1); i <= M; ++i)
{
    scanf("%s", in + 1);
    len[i] = strlen(in + 1);
    for (int j(1); j <= len[i]; ++j)
    {
        pattern[i][j] = map[int(in[j])];
    }
    insert(i);
}

build();

match();

/*for (int i(0); i <= lastNode; ++i)
{
    std::cout << i << ' ' << fail[i] << ": " << str[i] << ' ' << str[fail[i]]
↪ << '\n';
}/*/

for (int i(1); i <= M; ++i)
{
    int ans(0);
    int node(0);
    //printf("%d: ", i);
    for (int j(1); j <= len[i]; ++j)
    {
        node = child[node][pattern[i][j]];
        //printf("%d ", matched[node]);
        if (matched[node])
        {
            ans = j;
        }
    }
    //printf("\n");
    printf("%d\n", ans);
}

return 0;
}

```

## 5 automata.cpp

// 5201

```

#include <cstdio>
#include <cstring>
#include <vector>
#include <algorithm>

int      lastNode (0);
int      lastEdge (0);
const int maxN    (1000000 + 10);
int      patternLen;
int      to        [maxN];
int      trans     [maxN];
int      pre       [maxN];
int      last      [maxN];
char     pattern   [maxN];
char     s         [maxN];

void addEdge(int u, int v, int t)
{
    ++lastEdge;
    to[lastEdge] = v;
    trans[lastEdge] = t;
    pre[lastEdge] = last[u];
    last[u] = lastEdge;
}

void addEdge(std::vector<int> &u, int v, int t)
{
    for (std::size_t i(0); i < u.size(); ++i)
    {
        addEdge(u[i], v, t);
    }
}

int build(int pred, int pos)
{
    std::vector<int> branch;
    while (pos <= patternLen && pattern[pos] != ' ' && pattern[pos] != '\n')
    {
        switch (pattern[pos])
        {
            case 'N':
            case 'S':
            case 'E':
            case 'W':
                ++lastNode;
                addEdge(pred, lastNode, pattern[pos]);
                addEdge(branch, lastNode, pattern[pos]);
                branch.clear();
                pred = lastNode;
                break;
        }
    }
}

```

```

        case '*':
            ++lastNode;
            addEdge(pred, lastNode, 0);
            addEdge(branch, lastNode, 0);
            addEdge(lastNode, lastNode, 0);
            branch.clear();
            branch.push_back(lastNode);
            break;

        case '?':
            ++lastNode;
            addEdge(pred, lastNode, 0);
            addEdge(branch, lastNode, 0);
            branch.clear();
            pred = lastNode;
            break;

        case '[':
            pos = build(pred, pos + 1);
            branch.push_back(lastNode);
            break;

        case '\\0':
            ++lastNode;
            addEdge(pred, 2, 0);
            addEdge(branch, 2, 0);
            break;

        default:
            break;
    }

    ++pos;
}

return pos;
}

int main()
{
    scanf("%s", pattern);
    patternLen = strlen(pattern);
    ++lastNode; // S: 1
    ++lastNode; // T: 2
    build(1, 0);

    int n;
    scanf("%d", &n);
    for (int i(1); i <= n; ++i)
    {
        scanf("%s", s);
    }
}

```

```

    int len(strlen(s));
    std::vector<int> queue;
    queue.push_back(1);
    for (int j(0); j <= len + 1; ++j)
    {
        bool can(false);
        std::vector<int> tmp;
        for (std::size_t k(0); k < queue.size(); ++k)
        {
            if (queue[k] == 2)
            {
                can = true;
            }
            for (int e(last[queue[k]]); e; e = pre[e])
            {
                if (!trans[e] || trans[e] == s[j])
                {
                    tmp.push_back(to[e]);
                }
            }
        }

        std::sort(tmp.begin(), tmp.end());
        tmp.erase(std::unique(tmp.begin(), tmp.end()), tmp.end());
        std::swap(queue, tmp);

        if (j > 1)
        {
            printf(can ? "1" : "0");
        }
    }
    printf("\n");
}

return 0;
}

```

## 6 backpack\_on\_tree.cpp

// 1826

```

#include <cstdio>
#include <vector>
#include <limits>

int n, m;

const int maxN(100 + 10);
std::vector<int> edge[maxN];
int dist[maxN];
int w [maxN];

```

```

int          f    [maxN] [maxN] [maxN];
int          g    [maxN];
int          size[maxN];

const int inf(std::numeric_limits<int>::max() / 2);

void dfs(int v)
{
    size[v] = 1;
    for (int i(0); i <= n; ++i)
    {
        f[v][i][0] = (dist[v] - dist[i]) * w[v];
    }

    for (size_t i(0); i < edge[v].size(); ++i)
    {
        int to(edge[v][i]);
        dist[to] += dist[v];
        dfs(to);
        for (int j(0); j <= n; ++j)
        {
            for (int k(0); k <= size[v] + size[to] && k <= m; ++k)
            {
                g[k] = inf;
            }
            for (int k(0); k <= size[v] && k <= m; ++k)
            {
                for (int l(0); l <= size[to] && k + l <= m; ++l)
                {
                    g[k + l] = std::min(g[k + l], f[v][j][k] +
↪ f[to][j][l]);
                }
            }
            for (int k(0); k <= size[v] && k <= m; ++k)
            {
                for (int l(0); l <= size[to] && k + l + 1 <= m; ++l)
                {
                    g[k + l + 1] = std::min(g[k + l + 1], f[v][j][k]
↪ + f[to][to][l]);
                }
            }
            for (int k(0); k <= size[v] + size[to] && k <= m; ++k)
            {
                f[v][j][k] = g[k];
            }
        }
        size[v] += size[to];
    }
}

int main()

```

```

{
    scanf("%d %d", &n, &m);
    for (int i(1); i <= n; ++i)
    {
        int v;
        scanf("%d %d %d", &w[i], &v, &dist[i]);
        edge[v].push_back(i);
    }

    dfs(0);

    printf("%d\n", f[0][0][m]);
    return 0;
}

```

## 7 bit\_decomposition.cpp

```

// 5093

#include <cstdio>
#include <vector>
#include <cctype>

struct Event
{
    int x1;
    int y1;
    int x2;
    int y2;
    int k;
};

const int len(4000000);
int n, m, T;
class Array
{
private:
    int data[len];
public:
    int& operator()(int x, int y)
    {
        return data[x * m + y];
    }
};

Array a;
Array map;
Array killed;
Array delta;
Event event[len];
int mapK[len];

```

```

int read()
{
    char ch(0);
    while (!isdigit(ch))
    {
        ch = getchar();
    }
    int x(0);
    while (isdigit(ch))
    {
        x = x * 10 + int(ch - '0');
        ch = getchar();
    }
    return x;
}

void kill(int color)
{
    for (int i(1); i <= n; ++i)
    {
        for (int j(1); j <= m; ++j)
        {
            delta(i, j) = 0;
        }
    }
    for (int i(1); i <= T; ++i)
    {
        if (mapK[i] == color)
        {
            ++delta(event[i].x1, event[i].y1);
            --delta(event[i].x1, event[i].y2 + 1);
            --delta(event[i].x2 + 1, event[i].y1);
            ++delta(event[i].x2 + 1, event[i].y2 + 1);
        }
    }
    for (int i(1); i <= n; ++i)
    {
        for (int j(1); j <= m; ++j)
        {
            //printf("%d ", delta[i][j]);
            delta(i, j) += delta(i - 1, j) + delta(i, j - 1) - delta(i - 1, j
↵ - 1);

            if (map(i, j) != color && delta(i, j))
            {
                killed(i, j) = true;
            }
        }
        //printf("\n");
    }
}

```



```

        //printf("\n");
    }

    int main()
    {
        n = read(); m = read(); T = read();
        for (int i(1); i <= n; ++i)
        {
            for (int j(1); j <= m; ++j)
            {
                a(i, j) = read();
            }
        }
        for (int i(1); i <= T; ++i)
        {
            event[i].x1 = read();
            event[i].y1 = read();
            event[i].x2 = read();
            event[i].y2 = read();
            event[i].k = read();
        }

        for (int digit(0); digit < 25; ++digit)
        {
            for (int i(1); i <= n; ++i)
            {
                for (int j(1); j <= m; ++j)
                {
                    map(i, j) = (a(i, j) >> digit & 1);
                }
            }

            for (int i(1); i <= T; ++i)
            {
                mapK[i] = (event[i].k >> digit & 1);
            }

            kill(0);
            kill(1);
        }

        int ans(0);
        for (int i(1); i <= n; ++i)
        {
            for (int j(1); j <= m; ++j)
            {
                ans += killed(i, j);
            }
        }
        printf("%d\n", ans);
        return 0;
    }

```

```
}
```

## 8 bi\_dir\_search.cpp

```
#include <iostream>
#include <algorithm>

int N, M;
int p      [40 + 10];
int first [(1 << 20) + 10];
int firstLen;
int second[(1 << 20) + 10];
int secondLen;

void dfs(int pos, int end, int sum, int dst[], int &len)
{
    if (pos == end)
    {
        dst[len++] = sum;
    }
    else
    {
        dfs(pos + 1, end, sum, dst, len);
        if (sum + p[pos] <= M)
            dfs(pos + 1, end, sum + p[pos], dst, len);
    }
}

int main()
{
    std::cin >> N >> M;
    for (int i(0); i < N; ++i)
        std::cin >> p[i];

    dfs(0, N / 2, 0, first, firstLen);
    dfs(N / 2, N, 0, second, secondLen);

    std::sort(first, first + firstLen, std::greater<int>());
    std::sort(second, second + secondLen);

    int ans(0);
    for (int i(0), j(0); i < firstLen; ++i)
    {
        while (j < secondLen - 1 && first[i] + second[j + 1] <= M)
            ++j;
        ans = std::max(ans, first[i] + second[j]);
    }
    std::cout << ans << std::endl;
    return 0;
}
```

## 9 bsgs.cpp

```
int baby_step_giant_step(int a, int b, int p)
{
    map<int, int> hash;
    hash.clear();
    b %= p;
    int t = (int)sqrt(p) + 1;
    for (int j = 0; j < t; ++j)
    {
        int val = (long long)b * power(a, j, p) % p;
        hash[val] = j;
    }
    a = power(a, t, p);
    if (a == 0) return b == 0 ? 1 : -1;
    for (int i = 0; i <= t; ++i)
    {
        int val = power(a, i, p);
        int j = hash.find(val) == hash.end() ? -1 : hash[val];
        if (j >= 0 && i * t - j >= 0) return i * t - j;
    }
    return -1;
}
```

## 10 catalan.cpp

```
//1276

#include <cstdio>

const int maxN(2000000 + 10);
long long fact[maxN];

const long long mod(20100403);

long long qPow(long long base, long long expo)
{
    long long prod(1);
    for (long long i(1); i <= expo; i *= 2)
    {
        if (i & expo)
        {
            prod = prod * base % mod;
        }
        base = base * base % mod;
    }
    return prod;
}

long long inv(long long n)
{

```

```

        return qPow(n, mod - 2);
    }

    long long C(long long n, long long m)
    {
        if (m > n) return 0;
        return fact[n] * inv(fact[m] * fact[n - m] % mod) % mod;
    }

    int main()
    {
        fact[0] = 1;
        for (int i(1); i < maxN; ++i)
        {
            fact[i] = fact[i - 1] * i % mod;
        }
        long long n, m;
        scanf("%lld %lld", &n, &m);
        if (n < m)
        {
            printf("0\n");
        }
        else
        {
            printf("%lld\n", (C(n + m, n) - C(n + m, m - 1) + mod) % mod);
        }
        return 0;
    }

```

## 11 dijkstra.cpp

```

dist[src] = 0;
std::priority_queue<std::pair<int, int>,
                    std::vector<std::pair<int, int> >,
                    std::greater<std::pair<int, int> > > heap;
heap.push(std::make_pair(dist[src], src));

while (!heap.empty())
{
    int v(heap.top().second);
    heap.pop();
    if (visited[v]) continue;
    visited[v] = true;

    for (int nxt(last[v]); nxt; nxt = pre[nxt])
    {
        if (!visited[to[nxt]] && dist[to[nxt]] > dist[v] + cost[nxt])
        {
            dist[to[nxt]] = dist[v] + cost[nxt];
            heap.push(std::make_pair(dist[to[nxt]], to[nxt]));
        }
    }
}

```

```

    }
}

```

## 12 discretization.cpp

```

#include <algorithm>

int *end;
const int maxN(100000);

int a[maxN];

int get(int x)
{
    return std::lower_bound(a, end, x) - a + 1;
}

int main()
{
    int n;
    std::sort(a + 1, a + n + 1);
    end = std::unique(a + 1, a + n + 1);
}

```

## 13 disjoint\_union.cpp

```

int find(int v)
{
    if (belong[v] == v)
    {
        return v;
    }
    else
    {
        return belong[v] = find(belong[v]);
    }
}

```

## 14 dp\_on\_tree.cpp

```

// 5074

#include <cstdio>
#include <algorithm>

int n, m;
int ans;
const int maxN (1000000 + 10);
const int maxLbN (30);
const int maxM (100000 + 10);

```

```

int      color    [maxN];
int      to       [maxN * 2];
int      pre      [maxN * 2];
int      last     [maxN];
int      jump     [maxN][maxLbN];
int      depth    [maxN];
int      max      [maxN];
int      sMax     [maxN];
int      colFirst [maxM];
int      colLast  [maxM];
int      cnt      [maxN];
bool     dom      [maxN];

void addEdge(int u, int v)
{
    static int lastEdge(0);
    ++lastEdge;
    to[lastEdge] = v;
    pre[lastEdge] = last[u];
    last[u] = lastEdge;
}

void prep(int v, int pred)
{
    depth[v] = depth[pred] + 1;
    jump[v][0] = pred;
    for (int i(1); i < maxLbN; ++i)
    {
        jump[v][i] = jump[jump[v][i - 1]][i - 1];
    }

    for (int e(last[v]); e; e = pre[e])
    {
        if (to[e] != pred)
        {
            prep(to[e], v);

            if (max[to[e]] + 1 > max[v])
            {
                sMax[v] = max[v];
                max[v] = max[to[e]] + 1;
            }
            else if (max[to[e]] + 1 > sMax[v])
            {
                sMax[v] = max[to[e]] + 1;
            }
        }
    }
}

int lca(int l, int h)

```

```

{
    if (depth[l] < depth[h]) std::swap(l, h);

    for (int i(maxLbN - 1); i >= 0; --i)
    {
        if (depth[jump[l][i]] >= depth[h])
        {
            l = jump[l][i];
        }
    }

    if (l == h) return l;

    for (int i(maxLbN - 1); i >= 0; --i)
    {
        if (jump[l][i] != jump[h][i])
        {
            l = jump[l][i];
            h = jump[h][i];
        }
    }

    return jump[l][0];
}

void count(int v, int pred)
{
    ++cnt[v];
    if (!colFirst[color[v]])
    {
        colFirst[color[v]] = v;
    }
    else
    {
        --cnt[lca(colLast[color[v]], v)];
    }
    colLast[color[v]] = v;

    for (int e(last[v]); e; e = pre[e])
    {
        if (to[e] != pred)
        {
            count(to[e], v);
        }
    }
}

void solve(int v, int pred, int outDep)
{
    for (int e(last[v]); e; e = pre[e])
    {

```

```

        if (to[e] != pred)
        {
            int curMax(max[to[e]] + 1 == max[v] ? sMax[v] : max[v]);
            solve(to[e], v, std::max(curMax, outDep) + 1);
            cnt[v] += cnt[to[e]];
            dom[v] = (dom[v] || dom[to[e]]);
        }
    }

    if (cnt[v] == m) ans = std::max(ans, outDep + 1);
    if (!dom[v]) ans = std::max(ans, max[v] + 2);
}

int main()
{
    scanf("%d %d", &n, &m);
    for (int v(1); v <= n; ++v)
    {
        scanf("%d", &color[v]);
    }
    for (int e(1); e <= n - 1; ++e)
    {
        int u, v;
        scanf("%d %d", &u, &v);
        addEdge(u, v);
        addEdge(v, u);
    }

    prep(1, 0);
    count(1, 0);
    for (int i(1); i <= m; ++i)
    {
        dom[lca(colFirst[i], colLast[i])] = true;
    }

    solve(1, 0, 0);
    printf("%d\n", ans);
    return 0;
}

```

## 15 dsu\_merge.cpp

// 5287

```

#include <cstdio>
#include <unordered_map>
#include <queue>

const int maxN(300000 + 10);
int f [maxN];
int size[maxN];

```



```

std::unordered_map<int, int> in[maxN];

int unionFind(int v)
{
    if (f[v] == v)
    {
        return v;
    }
    else
    {
        return f[v] = unionFind(f[v]);
    }
}

std::queue<std::pair<int, int> > queue;

void merge(int u, int v)
{
    u = unionFind(u);
    v = unionFind(v);
    if (u == v) return;
    if (size[u] > size[v]) std::swap(u, v);
    size[v] += size[u];
    f[u] = v;

    for (std::unordered_map<int, int>::iterator it(in[u].begin()); it != in[u].end();
    ↪ ++it)
    {
        if (in[v][it->first])
        {
            queue.push(std::make_pair(it->second, in[v][it->first]));
        }
        else
        {
            in[v][it->first] = it->second;
        }
    }
}

int main()
{
    int n, m, k;
    scanf("%d %d %d", &n, &m, &k);
    for (int v(1); v <= n; ++v)
    {
        f[v] = v;
        size[v] = 1;
    }
    for (int e(1); e <= m; ++e)
    {
        int u, v, w;

```

```

        scanf("%d %d %d", &u, &v, &w);
        if (in[v][w])
        {
            queue.push(std::make_pair(u, in[v][w]));
        }
        else
        {
            in[v][w] = u;
        }
    }

    while (!queue.empty())
    {
        std::pair<int, int> pair(queue.front());
        queue.pop();
        merge(pair.first, pair.second);
    }

    long long ans(0);
    for (int v(1); v <= n; ++v)
    {
        if (f[v] == v)
        {
            ans += 1ll * size[v] * (size[v] - 1) / 2;
        }
    }
    printf("%lld\n", ans);
    return 0;
}

```

## 16 du\_sieve.cpp

```

// 4658

#include <cstdio>
#include <map>
#include <vector>

const int maxPre(10000000);
bool sieve [maxPre];
long long phiPre[maxPre];
long long muPre [maxPre];

std::map<int, long long> phiSum;
std::map<int, long long> muSum;

long long getPhiSum(long long n)
{
    if (n < maxPre) return phiPre[n];
    if (phiSum.count(n)) return phiSum[n];
}

```

```

    phiSum[n] = 1ll * n * (n + 1) / 2;
    for (long long l(2), r; l <= n; l = r + 1)
    {
        r = n / (n / l);
        phiSum[n] -= (r - l + 1) * getPhiSum(n / l);
    }
    return phiSum[n];
}

long long getMuSum(int n)
{
    if (n < maxPre) return muPre[n];
    if (muSum.count(n)) return muSum[n];

    muSum[n] = 1;
    for (long long l(2), r; l <= n; l = r + 1)
    {
        r = n / (n / l);
        muSum[n] -= (r - l + 1) * getMuSum(n / l);
    }
    return muSum[n];
}

int main()
{
    phiPre[1] = 1;
    muPre[1] = 1;
    std::vector<long long> prime;
    for (int i(2); i < maxPre; ++i)
    {
        if (!sieve[i])
        {
            phiPre[i] = i - 1;
            muPre[i] = -1;
            prime.push_back(i);
        }

        for (std::size_t j(0); j < prime.size() && i * prime[j] < maxPre; ++j)
        {
            sieve[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                phiPre[i * prime[j]] = phiPre[i] * prime[j];
                muPre[i * prime[j]] = 0;
                break;
            }
            else
            {
                phiPre[i * prime[j]] = phiPre[i] * phiPre[prime[j]];
                muPre[i * prime[j]] = -muPre[i];
            }
        }
    }
}

```

```

        }
    }

    for (int i(2); i < maxPre; ++i)
    {
        phiPre[i] += phiPre[i - 1];
        muPre[i] += muPre[i - 1];
    }

    int T;
    scanf("%d", &T);
    for (int ttt(1); ttt <= T; ++ttt)
    {
        int N;
        scanf("%d", &N);
        printf("%lld %lld\n", getPhiSum(N), getMuSum(N));
    }

    return 0;
}

```

## 17 euler\_path.cpp

// 5186

```

#include <cstdio>
#include <cstring>

int n, m;
const int maxN (400000 + 10);
int to [maxN];
int pre [maxN];
int last [maxN];
int degree[maxN];
int visit [maxN];

int lastEdge;
void addEdge(int u, int v)
{
    ++lastEdge;
    to[lastEdge] = v;
    pre[lastEdge] = last[u];
    last[u] = lastEdge;
}

bool dfs(int v, int pred, int cut)
{
    //printf("!\n", v);
    if (v == cut) return true;
    if (visit[v]) return false;
    visit[v] = true;
}

```

```

    for (int e(last[v]); e; e = pre[e])
    {
        if (to[e] != pred)
        {
            if (!dfs(to[e], v, cut))
            {
                return false;
            }
        }
    }

    return true;
}

bool check(int cut)
{
    //printf("check: %d\n", cut);

    memset(visit, false, sizeof(visit));
    for (int v(1); v <= n; ++v)
    {
        if (!visit[v])
        {
            if (!dfs(v, 0, cut))
            {
                return false;
            }
        }
    }

    return true;
}

int main()
{
    int T;
    scanf("%d", &T);

    for (int ttt(1); ttt <= T; ++ttt)
    {
        scanf("%d %d", &n, &m);
        lastEdge = 0;
        memset(last, 0, sizeof(last));
        memset(degree, 0, sizeof(degree));
        for (int e(1); e <= m; ++e)
        {
            int u, v;
            scanf("%d %d", &u, &v);
            addEdge(u, v);
            addEdge(v, u);
        }
    }
}

```

```

        ++degree[u];
        ++degree[v];
    }

    int cnt(0), max(0);
    for (int v(1); v <= n; ++v)
    {
        if (degree[v] % 2 == 1)
        {
            ++cnt;
        }
        if (degree[v] > degree[max])
        {
            max = v;
        }
    }

    bool exist(false);
    if (cnt <= 2)
    {
        for (int v(1); v <= n; ++v)
        {
            if ((cnt && degree[v] % 2 == 1) || (!cnt && v == max))
            {
                if (check(v))
                {
                    exist = true;
                    break;
                }
            }
        }
    }

    puts(exist ? "YES" : "NO");
}

return 0;
}

```

## 18 euler\_sieve.cpp

```

#include <iostream>
#include <vector>

const int maxN (1000000);

bool sieve[maxN];
int mu [maxN];
int phi [maxN];

int main()

```

```

{
    mu[1] = 1;
    phi[1] = 1;
    std::vector<int> prime;
    for (int i(2); i < maxN; ++i)
    {
        if (!sieve[i])
        {
            prime.push_back(i);
            mu[i] = -1;
            phi[i] = i - 1;
        }

        for (std::size_t j(0); j < prime.size() && i * prime[j] < maxN; ++j)
        {
            sieve[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                mu[i * prime[j]] = 0;
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else
            {
                mu[i * prime[j]] = -mu[i];
                phi[i * prime[j]] = phi[i] * phi[prime[j]];
            }
        }
    }

    for (int i(1); i <= 100; ++i)
        std::cout << phi[i] << ' ';
    std::cout << std::endl;

    return 0;
}

```

## 19 euler\_tour.cpp

```

// 5380

#include <cstdio>
#include <algorithm>
#include <vector>

struct Node
{
    long long sum;
    long long diam;
    int end[2];
};

```

```

int n, m;

const int      maxN  (400000 + 10);
const int      maxLbN(30);
std::vector<int> edge  [maxN];
std::vector<int> cost  [maxN];
std::vector<int> goal  [maxN];
long long      in     [maxN];
long long      prefix[maxN];
int            depth  [maxN];
int            euler  [maxLbN][maxN * 2];
int            enter  [maxN];
int            lb     [maxN * 2];
Node           data   [maxN * maxLbN];
int            lCh    [maxN * maxLbN];
int            rCh    [maxN * maxLbN];
int            root   [maxN];

inline int read()
{
    int x=0;
    char c=getchar();
    while(c<'0' || c>'9')c=getchar();
    while(c>='0' && c<='9')x=x*10+c-'0',c=getchar();
    return x;
}

int curEuler(0);

void dfs(int v, int pred)
{
    depth[v] = depth[pred] + 1;
    euler[0][++curEuler] = v;
    enter[v] = curEuler;
    for (size_t i(0); i < edge[v].size(); ++i)
    {
        int to(edge[v][i]);
        prefix[to] = prefix[v] + cost[v][i];
        dfs(to, v);
        euler[0][++curEuler] = v;
    }
}

int lca(int a, int b)
{
    a = enter[a];
    b = enter[b];
    if (a > b) std::swap(a, b);
    int lbLen(lb[b - a + 1]);
    int x(euler[lbLen][a]), y(euler[lbLen][b - (1 << lbLen) + 1]);

```



```

        return depth[x] < depth[y] ? x : y;
    }

    long long dist(int a, int b)
    {
        if (a == 0 || b == 0) return 0;
        int l(lca(a, b));
        return prefix[a] + prefix[b] - 2 * prefix[l];
    }

    Node operator+(const Node &a, const Node &b)
    {
        if (!b.end[0]) return a;
        if (!a.end[0]) return b;

        Node cur;
        cur.sum = a.sum + b.sum;
        cur.diam = 0;
        cur.end[0] = cur.end[1] = 0;

        int end[4] = {a.end[0], a.end[1], b.end[0], b.end[1]};
        for (int i(0); i < 4; ++i)
        {
            for (int j(i + 1); j < 4; ++j)
            {
                if (end[i] != 0 && end[j] != 0)
                {
                    long long d(dist(end[i], end[j]));
                    if (d >= cur.diam)
                    {
                        cur.diam = d;
                        cur.end[0] = end[i];
                        cur.end[1] = end[j];
                    }
                }
            }
        }

        return cur;
    }

    int curNode(0);

    void insert(int &node, int tl, int tr, int d, int v)
    {
        if (!node) node = ++curNode;

        if (tl == tr)
        {
            data[node].sum = in[v] * 2;
            data[node].end[0] = data[node].end[1] = v;
        }
    }

```

```

    }
    else
    {
        int mid((tl + tr) / 2);
        if (d <= mid)
        {
            insert(lCh[node], tl, mid, d, v);
        }
        else
        {
            insert(rCh[node], mid + 1, tr, d, v);
        }
        data[node] = data[lCh[node]] + data[rCh[node]];
    }
}

int merge(int lNode, int rNode, int tl, int tr)
{
    if (!rNode) return lNode;
    if (!lNode) return rNode;

    if (tl == tr)
    {
        data[lNode] = data[lNode] + data[rNode];
        return lNode;
    }
    else
    {
        int mid((tl + tr) / 2);
        lCh[lNode] = merge(lCh[lNode], lCh[rNode], tl, mid);
        rCh[lNode] = merge(rCh[lNode], rCh[rNode], mid + 1, tr);
        data[lNode] = data[lCh[lNode]] + data[rCh[lNode]];
        return lNode;
    }
}

Node query(int node, int tl, int tr, int l, int r)
{
    if (!node || (l <= tl && tr <= r))
    {
        return data[node];
    }
    else
    {
        int mid((tl + tr) / 2);
        if (r <= mid)
        {
            return query(lCh[node], tl, mid, l, r);
        }
        else if (l >= mid + 1)
        {

```

```

        return query(rCh[node], mid + 1, tr, l, r);
    }
    else
    {
        return query(lCh[node], tl, mid, l, r) + query(rCh[node], mid +
↪ 1, tr, l, r);
    }
}

long long ans(0);

void solve(int v)
{
    insert(root[v], 1, n, depth[v], v);
    for (size_t i(0); i < edge[v].size(); ++i)
    {
        int to(edge[v][i]);
        solve(to);
        root[v] = merge(root[v], root[to], 1, n);
    }

    for (size_t i(0); i < goal[v].size(); ++i)
    {
        Node result(query(root[v], 1, n, depth[v], std::min(depth[v] +
↪ goal[v][i], n)));
        ans ^= result.sum - result.diam - in[v] * 2;
    }
}

int main()
{
    n = read();
    m = read();
    for (int i(2); i <= n; ++i)
    {
        int p(read()), w(read());
        edge[p].push_back(i);
        cost[p].push_back(w);
        in[i] = w;
    }
    for (int i(1); i <= m; ++i)
    {
        int r, d;
        scanf("%d %d", &r, &d);
        goal[r].push_back(d);
    }

    dfs(1, 0);
    lb[0] = -1;
    for (int i(1); i <= curEuler; ++i)

```

```

{
    lb[i] = lb[i / 2] + 1;
}
for (int i(1); i < maxLbN; ++i)
{
    for (int j(1); j + (1 << i) - 1 <= curEuler; ++j)
    {
        int a(euler[i - 1][j]), b(euler[i - 1][j + (1 << (i - 1))]);
        euler[i][j] = (depth[a] <= depth[b] ? a : b);
    }
}

solve(1);
printf("%lld\n", ans);
return 0;
}

```

## 20 fenwick.cpp

```

const int maxN(1000000);

long long BIT[maxN];

int lowbit(int x)
{
    return x & (-x);
}

void add(int pos, int value)
{
    while (pos <= maxN)
    {
        BIT[pos] += value;
        pos += lowbit(pos);
    }
}

int get(int pos)
{
    int sum(0);
    while (pos > 0)
    {
        sum += BIT[pos];
        pos -= lowbit(pos);
    }
    return sum;
}

```

## 21 fft.cpp

```
// 1839
```

```
#include <iostream>
#include <complex>
#include <iomanip>

std::complex<double> q      [400000 + 10];
std::complex<double> qRev   [400000 + 10];
std::complex<double> g      [400000 + 10];
std::complex<double> denom  [400000 + 10];
int      reverse[400000 + 10];
const double pi(std::acos(-1.0));

int init(int len)
{
    len *= 2;
    int bitCount(0);
    int limit(1);
    while (limit < len)
    {
        ++bitCount;
        limit *= 2;
    }
    for (int i(0); i < limit; ++i)
        reverse[i] = (reverse[i >> 1] >> 1) | ((i & 1) << (bitCount - 1));
    return limit;
}

void FFT(std::complex<double> a[], int len, int inv)
{
    for (int i(0); i < len; ++i)
    {
        if (i < reverse[i])
            std::swap(a[i], a[reverse[i]]);
    }

    for (int half(1); half < len; half *= 2)
    {
        std::complex<double> wn(std::cos(pi / half), inv * std::sin(pi / half));
        for (int i(0); i < len; i += half * 2)
        {
            std::complex<double> w(1.0, 0.0);
            for (int j(0); j < half; ++j)
            {
                std::complex<double> x(a[i + j]);
                std::complex<double> y(w * a[i + half + j]);
                a[i + j] = x + y;
                a[i + half + j] = x - y;
                w *= wn;
            }
        }
    }
}
```

```

        }
    }

}

void FFT(std::complex<double> a[], int len)
{
    FFT(a, len, 1);
}

void invFFT(std::complex<double> a[], int len)
{
    FFT(a, len, -1);
    double invLen(1.0 / len);
    for (int i(0); i < len; ++i)
        a[i] *= invLen;
}

int main()
{
    int n;
    std::cin >> n;
    int limit(init(n));
    for (int i(0); i < n; ++i)
    {
        std::cin >> q[i];
        qRev[n - i - 1] = q[i];
        if (i != 0)
            g[i] = 1.0 / i / i;
    }
    FFT(q, limit);
    FFT(qRev, limit);
    FFT(g, limit);
    for (int i(0); i < limit; ++i)
    {
        q[i] *= g[i];
        qRev[i] *= g[i];
    }
    invFFT(q, limit);
    invFFT(qRev, limit);

    for (int i(0); i < n; ++i)
        std::cout << std::fixed << std::setprecision(3) << q[i].real() - qRev[n -
↪ i - 1].real() << std::endl;

    return 0;
}

```

## 22 heavy\_path\_decomposition.cpp

// 2409

```

#include <cstdio>
#include <iostream>
#include <cstring>

int n;
const int maxN(131072);
//const int maxN(5000);
char op[16];
bool test(false);

class Segtree
{
private:
    const int opInstall = 1;
    const int opUninstall = 2;

    int count[maxN * 4 + 10];
    int lazy [maxN * 4 + 10];

    void pushDown(int node, int left, int right);
    void pushUp(int node);

public:
    int install(int node, int left, int right, int oLeft, int oRight);
    int uninstall(int node, int left, int right, int oLeft, int oRight);
    int query(int node, int left, int right, int pos);
};

void Segtree::pushDown(int node, int left, int right)
{
    //if (left == right)
    //if (node >= 2 * n)
    //    std::cout << node << ' ' << left << ' ' << right << '\n';
    //std::cout << "!!!" << opInstall << ' ' << opUninstall << '\n';
    if (lazy[node] == opInstall)
    {
        lazy[node] = 0;
        lazy[node * 2] = opInstall;
        lazy[node * 2 + 1] = opInstall;

        int mid((left + right) / 2);
        int lSize(mid - left + 1);
        int rSize(right - mid);
        count[node * 2] = lSize;
        count[node * 2 + 1] = rSize;
    }

    if (lazy[node] == opUninstall)
    {
        lazy[node] = 0;
        lazy[node * 2] = opUninstall;
    }
}

```

```

        lazy[node * 2 + 1] = opUninstall;

        count[node * 2] = 0;
        count[node * 2 + 1] = 0;
    }
}

void Segtree::pushUp(int node)
{
    count[node] = count[node * 2] + count[node * 2 + 1];
}

int Segtree::install(int node, int left, int right, int oLeft, int oRight)
{
    //std::cout << "###" << left << '-' << right << ' ' << lazy[node] << ' ' <<
    ↪ count[24] << '\n';
    pushDown(node, left, right);
    int sum(0);
    if (left >= oLeft && right <= oRight)
    {
        int size(right - left + 1);
        sum += size - count[node];
        lazy[node] = opInstall;
        count[node] = size;
    }
    else
    {
        int mid((left + right) / 2);
        if (oLeft <= mid) sum += install(node * 2, left, mid, oLeft, oRight);
        if (oRight >= mid + 1) sum += install(node * 2 + 1, mid + 1, right,
    ↪ oLeft, oRight);
        pushUp(node);
    }
    //std::cout << "INSTALL: " << left << '-' << right << ' ' << oLeft << '-' <<
    ↪ oRight << ' ' << count[node] << ' ' << sum << '\n';
    return sum;
}

int Segtree::uninstall(int node, int left, int right, int oLeft, int oRight)
{
    pushDown(node, left, right);
    //std::cout << "UNINSTALL: " << left << '-' << right << ' ' << oLeft << '-' <<
    ↪ oRight << ' ' << count[node] << '\n';
    int sum(0);
    if (left >= oLeft && right <= oRight)
    {
        sum += count[node];
        lazy[node] = opUninstall;
        count[node] = 0;
    }
    else

```



```

        {
            int mid((left + right) / 2);
            if (oLeft <= mid) sum += uninstall(node * 2, left, mid, oLeft, oRight);
            if (oRight >= mid + 1) sum += uninstall(node * 2 + 1, mid + 1, right,
↪ oLeft, oRight);
            pushUp(node);
        }
        return sum;
    }

int Segtree::query(int node, int left, int right, int pos)
{
    pushDown(node, left, right);
    if (left == right)
    {
        return count[node];
    }
    else
    {
        int mid((left + right) / 2);
        if (pos <= mid) return query(node * 2, left, mid, pos);
        else return query(node * 2 + 1, mid + 1, right, pos);
    }
}

Segtree seg;

int to    [maxN * 2 + 10];
int pre   [maxN * 2 + 10];
int last  [maxN      + 10];

void addEdge(int x, int y)
{
    static int e(1);
    to[e] = y;
    pre[e] = last[x];
    last[x] = e;
    ++e;
}

int father[maxN      + 10];
int depth [maxN      + 10];
int size  [maxN      + 10];
int heavy [maxN      + 10];
int top   [maxN      + 10];
int dfn   [maxN      + 10];

void dfsInfo(int v, int f, int d)
{
    father[v] = f;
    depth[v] = d;
}

```

```

    size[v] = 1;
    for (int e(last[v]); e; e = pre[e])
    {
        if (to[e] != f)
        {
            dfsInfo(to[e], v, d + 1);
            if (heavy[v] == -1 || size[to[e]] > size[heavy[v]])
                heavy[v] = to[e];
            size[v] += size[to[e]];
        }
    }
}

int currentDfn(0);
void dfsChain(int v, int begin)
{
    top[v] = begin;
    dfn[v] = currentDfn++;
    if (heavy[v] == -1)
        return;

    dfsChain(heavy[v], begin);
    for (int e(last[v]); e; e = pre[e])
    {
        if (to[e] != father[v] && to[e] != heavy[v])
            dfsChain(to[e], to[e]);
    }
}

int install(int v)
{
    int sum(0);
    while (v != -1)
    {
        sum += seg.install(1, 0, n - 1, dfn[top[v]], dfn[v]);
        v = father[top[v]];
    }
    return sum;
}

int uninstall(int v)
{
    int sum(0);
    sum += seg.uninstall(1, 0, n - 1, dfn[v], dfn[v] + size[v] - 1);
    return sum;
}

int main()
{
    //freopen("manager.in", "r", stdin);
    //freopen("manager.out", "w", stdout);

```

```

scanf("%d", &n);
for (int v(1); v <= n - 1; ++v)
{
    int f;
    scanf("%d", &f);
    addEdge(f, v);
}

memset(heavy, -1, sizeof(heavy));
dfsInfo(0, -1, 0);
dfsChain(0, 0);

//for (int v(0); v <= n - 1; ++v)
//    std::cout << v << "->" << dfn[v] << '\n';

int q;
scanf("%d", &q);
for (int i(1); i <= q; ++i)
{
    int v;
    scanf("%s %d", op, &v);
    if (op[0] == 'i')
    {
        if (seg.query(1, 0, n - 1, dfn[v]) == 1)
            printf("0\n");
        else
            printf("%d\n", install(v));
    }
    else
    {
        if (seg.query(1, 0, n - 1, dfn[v]) == 0)
            printf("0\n");
        else
            printf("%d\n", uninstall(v));
    }
}

return 0;
}

```

## 23 inverse\_dp.cpp

// 5202

```

#include <cstdio>
#include <cctype>

const int      maxN(2000000 + 10);
const long long mod (1000000007);
long long      ans;

```

```

int          to [maxN];
int          pre [maxN];
int          last[maxN];
long long    inv [maxN];
long long    sum [maxN];

int read()
{
    char ch(0);
    while (!isdigit(ch))
    {
        ch = getchar();
    }
    int result(0);
    while (isdigit(ch))
    {
        result = result * 10 + int(ch - '0');
        ch = getchar();
    }
    return result;
}

void addEdge(int u, int v)
{
    static int lastEdge(0);
    ++lastEdge;
    to[lastEdge] = v;
    pre[lastEdge] = last[u];
    last[u] = lastEdge;
}

void dfs(int v, int pred, int depth)
{
    ans = (ans + sum[depth]) % mod;
    for (int e(last[v]); e; e = pre[e])
    {
        if (to[e] != pred)
        {
            dfs(to[e], v, depth + 1);
        }
    }
}

int main()
{
    int n(read());
    for (int v(2); v <= n; ++v)
    {
        int a(read());
        addEdge(a, v);
    }
}

```

```

    inv[1] = 1;
    sum[1] = 1;
    for (int i(2); i <= n; ++i)
    {
        inv[i] = (mod - mod / i) * inv[mod % i] % mod;
        sum[i] = (sum[i - 1] + inv[i]) % mod;
    }

    dfs(1, 0, 0);

    printf("%lld\n", ans);
    return 0;
}

```

## 24 kmp.cpp

```

#include <iostream>
#include <fstream>
#include <string>

const int    maxN {1000000};
int          fail [maxN];
int          match[maxN];

void build(std::string &pat)
{
    fail[0] = -1;
    for (int i{1}, j{-1}; i < pat.length(); ++i)
    {
        while (j >= 0 && pat[i] != pat[j + 1]) j = fail[j];
        if (pat[i] == pat[j + 1]) ++j;
        fail[i] = j;
    }
}

void find(std::string &str, std::string &pat)
{
    for (int i{0}, j{-1}; i < str.length(); ++i)
    {
        while (j >= 0 && (j == pat.length() - 1 || str[i] != str[j + 1]))
        {
            j = fail[j];
        }
        if (str[i] == str[j + 1]) ++j;
        match[i] = j;
    }
}

int main()
{

```

```

    std::string T{"anana"};
    std::string S{"banananban"};

    build(T);
    find(S, T);

    for (int i{0}; i < T.length(); ++i) std::cout << T[i] << ' ';
    std::cout << std::endl;
    for (int i{0}; i < T.length(); ++i) std::cout << fail[i] << ' ';
    std::cout << std::endl;
    for (int i{0}; i < S.length(); ++i) std::cout << S[i] << ' ';
    std::cout << std::endl;
    for (int i{0}; i < S.length(); ++i) std::cout << match[i] << ' ';
    std::cout << std::endl;

    return 0;
}

```

## 25 kruskal\_smin\_span\_tree.cpp

```

#include <iostream>
#include <cstring>
#include <algorithm>

long long n, m;
const long long inf(0x7f7f7f7f);

struct Edge
{
    long long from;
    long long to;
    long long cost;
};

Edge edge [300000 + 10];
bool inMst [300000 + 10];
long long father[100000 + 10];

long long to [300000 * 2 + 10];
long long pre [300000 * 2 + 10];
long long last [100000 + 10];
long long cost [300000 * 2 + 10];

long long depth [100000 + 10];
long long lca [100000 + 10][20];
long long max1 [100000 + 10][20];
long long max2 [100000 + 10][20];

bool operator<(const Edge &a, const Edge &b)
{
    return a.cost < b.cost;
}

```

```

void addEdge(long long x, long long y, long long z)
{
    static long long index(0);
    to[index] = y;
    pre[index] = last[x];
    last[x] = index;
    cost[index] = z;
    ++index;
}

long long getFather(long long vertex)
{
    if (father[vertex] == vertex)
        return vertex;
    else
        return father[vertex] = getFather(father[vertex]);
}

long long kruskal()
{
    for (long long i(1); i <= n; ++i)
        father[i] = i;
    std::sort(edge + 1, edge + m + 1);
    long long sum(0);
    for (long long i(1); i <= m; ++i)
    {
        long long fatherA(getFather(edge[i].from));
        long long fatherB(getFather(edge[i].to));
        if (fatherA != fatherB)
        {
            inMst[i] = true;
            sum += edge[i].cost;
            father[fatherA] = fatherB;
            addEdge(edge[i].from, edge[i].to, edge[i].cost);
            addEdge(edge[i].to, edge[i].from, edge[i].cost);
        }
    }

    return sum;
}

void dfs(long long vertex, long long pred, long long d)
{
    depth[vertex] = d;
    lca[vertex][0] = pred;
    for (long long i(last[vertex]); i != -1; i = pre[i])
    {
        if (to[i] != pred)
        {
            max1[to[i]][0] = cost[i];

```

```

        max2[to[i]][0] = -inf;
        dfs(to[i], vertex, d + 1);
    }
}

long long getLca(long long v1, long long v2)
{
    if (depth[v1] < depth[v2])
        std::swap(v1, v2);
    for (long long i(19); i >= 0; --i)
    {
        if (depth[v1] - (1 << i) >= depth[v2])
            v1 = lca[v1][i];
    }
    if (v1 == v2)
        return v1;
    for (long long i(19); i >= 0; --i)
    {
        if (lca[v1][i] != lca[v2][i])
        {
            v1 = lca[v1][i];
            v2 = lca[v2][i];
        }
    }
    return lca[v1][0];
}

long long calc(long long v, long long f, long long replace)
{
    long long m1(-inf), m2(-inf);
    for (long long i(19); i >= 0; --i)
    {
        if (depth[v] - (1 << i) >= depth[f])
        {
            if (max1[v][i] > m1)
            {
                m2 = m1;
                m1 = max1[v][i];
            }
            m2 = std::max(m2, max2[v][i]);
            v = lca[v][i];
        }
    }

    if (m1 == replace)
        return replace - m2;
    else
        return replace - m1;
}

```



```

int main()
{
    std::cin >> n >> m;
    memset(last, -1, sizeof(last));
    for (long long i(1); i <= m; ++i)
        std::cin >> edge[i].from >> edge[i].to >> edge[i].cost;

    long long sum(kruskal());

    dfs(to[0], 0, 0);
    for (long long i(1); (1 << i) <= n; ++i)
    {
        for (long long j(1); j <= n; ++j)
        {
            lca[j][i] = lca[lca[j][i - 1]][i - 1];
            max1[j][i] = std::max(max1[j][i - 1], max1[lca[j][i - 1]][i -
↪ 1]);

            if (max1[j][i - 1] == max1[lca[j][i - 1]][i - 1])
                max2[j][i] = std::max(max2[j][i - 1], max2[lca[j][i -
↪ 1]][i - 1]);

            else if (max1[j][i - 1] < max1[lca[j][i - 1]][i - 1])
                max2[j][i] = std::max(max1[j][i - 1], max2[lca[j][i -
↪ 1]][i - 1]);

            else
                max2[j][i] = std::max(max2[j][i - 1], max1[lca[j][i -
↪ 1]][i - 1]);
        }
    }

    long long ans(1e18);
    for (long long i(1); i <= m; ++i)
    {
        if (!inMst[i])
        {
            long long f(getLca(edge[i].from, edge[i].to));
            ans = std::min(ans, calc(edge[i].from, f, edge[i].cost));
            ans = std::min(ans, calc(edge[i].to, f, edge[i].cost));
        }
    }
    std::cout << sum + ans << std::endl;

    return 0;
}

```

## 26 lca.cpp

```

void dfs(int father, int v, int d)
{
    depth[v] = d;
    lca[v][0] = father;
    for (int i(1); (1 << i) <= depth[v]; ++i)

```

```

        lca[v][i] = lca[lca[v][i - 1]][i - 1];
    root[v] = increase(root[father], 0, 100000, V[v]);

    for (int i(last[v]); i != 0; i = pre[i])
    {
        if (to[i] != father)
            dfs(v, to[i], d + 1);
    }
}

int getLca(int u, int v)
{
    if (depth[u] < depth[v])
        std::swap(u, v);
    for (int i(19); i >= 0; --i)
    {
        if (depth[u] - (1 << i) >= depth[v])
            u = lca[u][i];
    }
    if (u == v)
        return u;

    for (int i(19); i >= 0; --i)
    {
        if (lca[u][i] != lca[v][i])
        {
            u = lca[u][i];
            v = lca[v][i];
        }
    }
    return lca[u][0];
}

```

## 27 leftist\_tree.cpp

```

#include <bits/stdc++.h>
using namespace std;
#define file(a) freopen(#a".in", "r", stdin), freopen(#a".out", "w", stdout)
#define LL long long
#define N 200010
struct llt{
    int val,lc,rc;
    int fa,d;
}t[N];
#define fa(x) t[x].fa
#define lc(x) t[x].lc
#define rc(x) t[x].rc
int find(int x){
    return x==fa(x)?x:fa(x)=find(fa(x));
}
inline void Swap(int &x,int &y){

```

```

        int tmp=x;x=y;y=tmp;
    }
    int merge(int x,int y){
        if(!x||!y) return x|y;
        if(t[x].val>t[y].val||(t[x].val==t[y].val&&x>y)) Swap(x,y);
        rc(x)=merge(rc(x),y);
        if(t[rc(x)].d>t[lc(x)].d Swap(lc(x),rc(x)));
        fa(x)=fa(lc(x))=fa(rc(x))=x;
        t[x].d=t[rc(x)].d+1;
        return x;
    }
    void pop(int x){
        t[x].val=-1;
        fa(rc(x))=rc(x);fa(lc(x))=lc(x);
        t[x].fa=merge(lc(x),rc(x));
    }
    int n,m;
    int main(){
        scanf("%d%d",&n,&m);
        t[0].d=-1;
        for(int i=1;i<=n;++i){
            int x;scanf("%d",&x);
            t[i].val=x;t[i].fa=i;
        }
        while(m--){
            int opt;
            scanf("%d",&opt);
            if(opt==1){
                int x,y;
                scanf("%d%d",&x,&y);
                int xx=find(x),yy=find(y);
                if(t[x].val==-1||t[y].val==-1) continue;
                if(xx==yy) continue;
                merge(xx,yy);
            }
            if(opt==2){
                int x;scanf("%d",&x);
                int xx=find(x);
                if(t[x].val==-1) printf("%d\n",-1);
                else{
                    printf("%d\n",t[xx].val);
                    pop(xx);
                }
            }
        }
        return 0;
    }
}

```

## 28 manacher.cpp

```
#include <cstdio>
#include <cstring>
#include <algorithm>

const int maxN(2000000 + 10);
char in [maxN];
char str [maxN];
int cnt [maxN];

int main()
{
    scanf("%s", in + 1);
    int len(strlen(in + 1));
    str[0] = '$';
    for (int i(1); i <= len; ++i)
    {
        str[i * 2 - 1] = in[i];
        if (i < len) str[i * 2] = '#';
    }
    len = len * 2 - 1;
    str[len + 1] = '\0';

    int mid(0);
    int r(0);
    for (int i(1); i <= len; ++i)
    {
        if (i <= r)
        {
            cnt[i] = std::min(cnt[mid - (i - mid)], r - i + 1);
        }

        while (str[i - cnt[i]] == str[i + cnt[i]])
        {
            //printf("%c + %d -> %c,%c\n", str[i], cnt[i], str[i - cnt[i]],
↪ str[i + cnt[i]]);
            ++cnt[i];
        }

        if (i + cnt[i] - 1 > r)
        {
            mid = i;
            r = i + cnt[i] - 1;
        }
    }

    for (int i(1); i <= len; ++i)
    {
        if (str[i] == '#' && cnt[i] == 1)
        {
            cnt[i] = 0;
        }
    }
}
```

```

    }
    if (str[i + cnt[i] - 1] == '#')
    {
        --cnt[i];
    }
    printf("%d ", cnt[i]);
}
printf("\n");
return 0;
}

```

## 29 matrix.cpp

```

#include <iostream>

int N, T;
int map[9 * 10 + 10][9 * 10 + 10];
int ans[9 * 10 + 10][9 * 10 + 10];
int tmp[9 * 10 + 10][9 * 10 + 10];
const int mod(2009);

void mult(int a[9 * 10 + 10][9 * 10 + 10], int b[9 * 10 + 10][9 * 10 + 10], int c[9 * 10
↵ + 10][9 * 10 + 10])
{
    for (int i(1); i <= N * 9; ++i)
    {
        for (int j(1); j <= N * 9; ++j)
        {
            tmp[i][j] = 0;
            for (int k(1); k <= N * 9; ++k)
            {
                tmp[i][j] = (tmp[i][j] + a[i][k] * b[k][j]) % mod;
            }
        }
    }

    for (int i(1); i <= N * 9; ++i)
        for (int j(1); j <= N * 9; ++j)
            c[i][j] = tmp[i][j];
}

int main()
{
    std::cin >> N >> T;
    for (int i(0); i < N; ++i)
    {
        for (int j(0); j < N; ++j)
        {
            char ch;
            std::cin >> ch;
            int cost(static_cast<int>(ch - '0'));

```

```

        if (cost != 0)
        {
            for (int k(1); k <= cost - 1; ++k)
                map[i * 9 + k][i * 9 + k + 1] = 1;
            map[i * 9 + cost][j * 9 + 1] = 1;
        }
    }

    for (int i(1); i <= N * 9; ++i)
        ans[i][i] = 1;

    for (int i(1); i <= T; i <= 1)
    {
        if (i & T)
            mult(ans, map, ans);
        mult(map, map, map);

        /*for (int ii(1); ii <= N * 9; ++ii)
        {
            for (int j(1); j <= N * 9; ++j)
            {
                std::cout << map[ii][j] << ' ';
            }
            std::cout << std::endl;
        }*/
        //std::cout << i << ' ' << T << std::endl;
        //system("pause");
    }

    std::cout << ans[0 * 9 + 1][(N - 1) * 9 + 1] << std::endl;
    return 0;
}

```

### 30 min\_string.cpp

```

int k = 0, i = 0, j = 1;
while (k < n && i < n && j < n) {
    if (sec[(i + k) % n] == sec[(j + k) % n]) {
        ++k;
    } else {
        if (sec[(i + k) % n] > sec[(j + k) % n])
            ++i;
        else
            ++j;
        k = 0;
        if (i == j) i++;
    }
}
i = min(i, j);

```

## 31 mono\_queue.cpp

```
// 2840

#include <deque>
#include <algorithm>
#include <iostream>

long long sum[2000000 + 10];

int main()
{
    std::ios::sync_with_stdio(false);
    int n, d;
    long long p;
    std::cin >> n >> p >> d;
    for (int i(1); i <= n; ++i)
    {
        int t;
        std::cin >> t;
        sum[i] = sum[i - 1] + t;
    }

    int ans(0);
    std::deque<int> deque;
    for (int i(d), j(0); i <= n; ++i)
    {
        while (!deque.empty() && sum[i] - sum[i - d] > sum[deque.back()] -
↪ sum[deque.back() - d])
            deque.pop_back();
        deque.push_back(i);
        while (!deque.empty() && sum[i] - sum[j] - (sum[deque.front()] -
↪ sum[deque.front() - d]) > p)
        {
            ++j;
            while (!deque.empty() && deque.front() - d < j)
                deque.pop_front();
        }
        ans = std::max(ans, i - j);
    }

    std::cout << ans << std::endl;
    return 0;
}
```

## 32 network\_flow.cpp

```
#include <iostream>
#include <cstring>
#include <queue>
#include <algorithm>
```

```

const int maxN(5000);

int to [maxN];
int pre [maxN];
int last [maxN];
int cap [maxN];
int depth[maxN];

const int inf(0x3f3f3f3f);

bool bfs(int start, int end)
{
    memset(depth, 0, sizeof(depth));
    depth[start] = 1;

    std::queue<int> queue;
    queue.push(start);
    while (!queue.empty())
    {
        int v(queue.front());
        queue.pop();
        for (int e(last[v]); e; e = pre[e])
        {
            if (!depth[to[e]] && cap[e])
            {
                depth[to[e]] = depth[v] + 1;
                queue.push(to[e]);
            }
        }
    }

    return depth[end];
}

int dfs(int v, int end, int flow)
{
    if (!flow) return 0;
    if (v == end) return flow;

    int sum(0);
    for (int e(last[v]); flow && e; e = pre[e])
    {
        if (depth[v] + 1 == depth[to[e]])
        {
            int cur(dfs(to[e], end, std::min(flow, cap[e])));
            cap[e] -= cur;
            cap[e ^ 1] += cur;
            flow -= cur;
            sum += cur;
        }
    }
}

```



```

    }

    if (!sum) depth[v] = 0;
    return sum;
}

void addEdge(int u, int v, int w)
{
    static int lastEdge(2);

    to[lastEdge] = v;
    pre[lastEdge] = last[u];
    last[u] = lastEdge;
    cap[lastEdge] = w;
    ++lastEdge;

    to[lastEdge] = u;
    pre[lastEdge] = last[v];
    last[v] = lastEdge;
    ++lastEdge;
}

int main()
{
    int m, n;
    std::cin >> m >> n;

    for (int i(1); i <= m; ++i)
        addEdge(0, i, 1);

    while (true)
    {
        int a, b;
        std::cin >> a >> b;
        if (a == -1)
            break;
        addEdge(a, b, 1);
    }

    for (int i(m + 1); i <= n; ++i)
        addEdge(i, n + 1, 1);

    int sum(0);
    while (bfs(0, n + 1))
    {
        sum += dfs(0, n + 1, inf);
    }

    if (sum)
        std::cout << sum << std::endl;
    else

```

```

        std::cout << "No Solution!" << std::endl;

        return 0;
}

```

### 33 network\_flow\_cost.cpp

// 5379

```

#include <cstdio>
#include <limits>
#include <cstdlib>
#include <cmath>
#include <queue>

struct Pig
{
    long long x;
    long long y;
    long long c;

    Pig() : x(0), y(0), c(0)
    {
    }

    Pig(long long x, long long y, long long c = 0) : x(x), y(y), c(c)
    {
    }
};

long long diff(const Pig &a, const Pig &b)
{
    return std::abs(a.x - b.x) + std::abs(a.y - b.y);
}

const int maxN (1000 + 10);
const int maxV (3 * maxN);
const int maxE (40 * maxN);
Pig      a      [maxN];
Pig      b      [maxN];
int      to      [maxE];
int      pre     [maxE];
long long cost   [maxE];
long long cap    [maxE];
int      last    [maxV];
long long flow   [maxV];
long long dist   [maxV];
bool     visit   [maxV];
int      in      [maxV];

const long long inf(std::numeric_limits<long long>::max() / 10);

```

```

void addEdge(int u, int v, long long w, long long c)
{
    static int lastEdge(1);

    ++lastEdge;
    to[lastEdge] = v;
    pre[lastEdge] = last[u];
    last[u] = lastEdge;
    cost[lastEdge] = w;
    cap[lastEdge] = c;

    ++lastEdge;
    to[lastEdge] = u;
    pre[lastEdge] = last[v];
    last[v] = lastEdge;
    cost[lastEdge] = -w;
    cap[lastEdge] = 0;
}

bool SPFA(int S, int T)
{
    for (int i(0); i < maxV; ++i)
    {
        flow[i] = 0;
        dist[i] = -inf;
        visit[i] = false;
    }

    flow[S] = inf;
    dist[S] = 0;
    in[T] = -1;
    std::queue<int> queue;
    queue.push(S);
    visit[S] = true;

    while (!queue.empty())
    {
        int v(queue.front());
        queue.pop();
        visit[v] = false;
        for (int e(last[v]); e; e = pre[e])
        {
            if (cap[e] && dist[to[e]] < dist[v] + cost[e])
            {
                dist[to[e]] = dist[v] + cost[e];
                flow[to[e]] = std::min(flow[v], cap[e]);
                in[to[e]] = e;
                if (!visit[to[e]])
                {
                    queue.push(to[e]);
                }
            }
        }
    }
}

```

```

        visit[to[e]] = true;
    }
}

}

return in[T] != -1;
}

int main()
{
    // freopen("pig.in", "r", stdin);
    // freopen("pig.out", "w", stdout);

    int n;
    scanf("%d", &n);
    for (int i(1); i <= n; ++i)
    {
        scanf("%lld %lld %lld", &a[i].x, &a[i].y, &a[i].c);
    }
    for (int i(1); i <= n; ++i)
    {
        scanf("%lld %lld %lld", &b[i].x, &b[i].y, &b[i].c);
    }

    int S(2 * n + 1), T(2 * n + 2);
    int DL(2 * n + 3), UR(2 * n + 4), DR(2 * n + 5), UL(2 * n + 6);
    Pig cornerDL(0, 0), cornerDR(0, 1000000001);

    for (int i(1); i <= n; ++i)
    {
        addEdge(S, i, 0, a[i].c);
        addEdge(n + i, T, 0, b[i].c);

        addEdge(i, DL, diff(a[i], cornerDL), inf);
        addEdge(DL, n + i, -diff(cornerDL, b[i]), inf);

        addEdge(i, UR, -diff(a[i], cornerDL), inf);
        addEdge(UR, n + i, diff(cornerDL, b[i]), inf);

        addEdge(i, DR, diff(a[i], cornerDR), inf);
        addEdge(DR, n + i, -diff(cornerDR, b[i]), inf);

        addEdge(i, UL, -diff(a[i], cornerDR), inf);
        addEdge(UL, n + i, diff(cornerDR, b[i]), inf);
    }

    long long ans(0);
    while (SPFA(S, T))
    {
        ans += flow[T] * dist[T];
    }
}

```

```

        for (int cur(T); cur != S; cur = to[in[cur] ^ 1])
        {
            cap[in[cur]] -= flow[T];
            cap[in[cur] ^ 1] += flow[T];
        }
    }
    printf("%lld\n", ans);
    return 0;
}

```

## 34 paren\_match.cpp

// 4652

```

#include <cstdio>
#include <vector>

const int      maxN (500000 + 10);
std::vector<int> edge [maxN];
char           in   [maxN];
bool           type [maxN];
int            stack[maxN];
int            match[maxN];
int            cnt   [maxN];
int            cons  [maxN];
long long      sum   [maxN];

long long ans(0);

int top(0);

void dfs(int v, int pre)
{
    ++top;
    stack[top] = type[v];
    if (type[v] == 0 || stack[pre] == 1)
    {
        match[top] = top;
        cons[top] = 0;
        if (top > 0)
        {
            sum[top] = sum[top - 1];
        }
        else
        {
            sum[top] = 0;
        }
    }
    else
    {
        match[top] = match[pre - 1];
    }
}

```

```

        cons[top] = cons[pre - 1] + 1;
        sum[top] = sum[top - 1] + cons[top];
    }
    ans ^= v * sum[top];

    for (size_t i(0); i < edge[v].size(); ++i)
    {
        dfs(edge[v][i], match[top]);
    }

    --top;
}

int main()
{
    int n;
    scanf("%d", &n);
    scanf("%s", in + 1);
    for (int i(1); i <= n; ++i)
    {
        type[i] = (in[i] == '(' ? 0 : 1);
    }
    for (int i(2); i <= n; ++i)
    {
        int f;
        scanf("%d", &f);
        edge[f].push_back(i);
    }

    stack[0] = 1;
    dfs(1, 0);
    printf("%lld\n", ans);
    return 0;
}

```

## 35 parser.cpp

```

#include <iostream>
#include <string>
#include <cctype>
#include <limits>

std::string str;
int pos(0);
int lookup[256];

bool lessPower(char p0, char p1)
{
    if (p0 == '^' && p1 == '^')
        return true;
    else

```

```

        return lookup[p0] < lookup[p1];
    }

    int parse(char oldOp)
    {
        int left(0);
        if (str[pos] == '(')
        {
            ++pos;
            left = parse(0);
            ++pos;
        }
        else
        {
            while (pos < str.size() && isdigit(str[pos]))
            {
                left = left * 10 + static_cast<int>(str[pos] - '0');
                ++pos;
            }

            while (pos < str.size() && str[pos] != ')' && !isdigit(str[pos]) &&
↳ lessPower(oldOp, str[pos]))
            {
                char op(str[pos]);
                ++pos;
                int right(parse(op));
                switch (op)
                {
                    case '+':
                        left += right;
                        break;

                    case '-':
                        left -= right;
                        break;

                    case '*':
                        left *= right;
                        break;

                    case '/':
                        left /= right;
                        break;

                    case '^':
                        {
                            int temp(left);
                            left = 1;
                            for (int i(0); i < right; ++i)
                                left *= temp;
                        }
                    }
                }
            }
        }
    }

```

```

        }
        break;

        default:
            break;
    }
}

return left;
}

int main()
{
    lookup[0] = 0;
    lookup['+'] = 1;
    lookup['-'] = 1;
    lookup['*'] = 2;
    lookup['/'] = 2;
    lookup['^'] = 3;
    std::cin >> str;
    std::cout << parse(0) << std::endl;
    return 0;
}

```

## 36 persist\_segtree.cpp

```

// 1231
#include <cstdio>
#include <cctype>
#include <algorithm>
#include <iostream>

struct Node
{
    int low;
    int high;
    int count;
};

int number[100000 + 10];
int sorted[100000 + 10];
int root [100000 + 10];
Node tree [2000000 + 10];
int nextId(0);

int read()
{
    char ch;
    while (ch = getchar(), !isdigit(ch) && ch != '-')
        ;
    int ret(0);
}

```



```

    int sign(ch == '-' ? -1 : 1);
    if (sign == -1)
        ch = getchar();
    while (isdigit(ch))
    {
        ret = ret * 10 + static_cast<int>(ch - '0');
        ch = getchar();
    }
    return sign * ret;
}

int increase(int oldNode, int low, int high, int value)
{
    int newNode(nextId);
    ++nextId;
    tree[newNode] = tree[oldNode];
    ++tree[newNode].count;
    if (low != high)
    {
        int mid((low + high) / 2);
        if (value <= mid)
            tree[newNode].low = increase(tree[oldNode].low, low, mid, value);
        else
            tree[newNode].high = increase(tree[oldNode].high, mid + 1, high,
↪ value);
    }
    return newNode;
}

int query(int leftNode, int rightNode, int low, int high, int k)
{
    if (low == high)
        return low;
    int mid((low + high) / 2);
    int lowCount(tree[tree[rightNode].low].count - tree[tree[leftNode].low].count);
    if (k <= lowCount)
        return query(tree[leftNode].low, tree[rightNode].low, low, mid, k);
    else
        return query(tree[leftNode].high, tree[rightNode].high, mid + 1, high, k
↪ - lowCount);
}

int main()
{
    int n(read()), m(read());
    for (int i(1); i <= n; ++i)
        sorted[i] = number[i] = read();

    std::sort(sorted + 1, sorted + n + 1);
    int *sortedEnd(std::unique(sorted + 1, sorted + n + 1));
    for (int i(1); i <= n; ++i)

```

```

        number[i] = std::lower_bound(sorted + 1, sortedEnd, number[i]) - sorted;
    int maxNumber(sortedEnd - sorted);

    ++nextId;
    for (int i(1); i <= n; ++i)
        root[i] = increase(root[i - 1], 1, maxNumber, number[i]);

    for (int i(1); i <= m; ++i)
    {
        int l, r, k;
        std::cin >> l >> r >> k;
        std::cout << sorted[query(root[l - 1], root[r], 1, maxNumber, k)] <<
↪ std::endl;
    }

    return 0;
}

```

### 37 persist\_trie.cpp

```

// 1333

#include <iostream>
#include <algorithm>

const int maxN(300000);
int trie[maxN * 2 * 24][2];
int end [maxN * 2 * 24];
int sum [maxN * 2];
int root[maxN * 2];
int newNode;

int insert(int index, int pos, int node)
{
    //std::cout << index << ' ' << (sum[index] >> pos & 1) << ' ' << pos << ' ' <<
↪ node << std::endl;
    int currentNode(++newNode);
    if (pos < 0)
    {
        end[currentNode] = index;
        return currentNode;
    }

    int digit((sum[index] >> pos) & 1);
    trie[currentNode][digit] = insert(index, pos - 1, trie[node][digit]);
    trie[currentNode][digit ^ 1] = trie[node][digit ^ 1];
    end[currentNode] = std::max(end[trie[currentNode][0]],
↪ end[trie[currentNode][1]]);
    //std::cout << '!' << digit << '*' << currentNode << ':' <<
↪ trie[currentNode][digit] << ' ' << trie[currentNode][digit ^ 1] << "^^" <<
↪ end[currentNode] << std::endl;
}

```

```

        return currentNode;
    }

    int query(int value, int pos, int limit, int node)
    {
        int digit((value >> pos) & 1);
        //std::cout << digit << ' ' << pos << ' ' << node << ' ' << limit << ' ' <<
        ↪ end[trie[node][digit ^ 1]] << std::endl;
        if (pos < 0)
            return sum[end[node]] ^ value;
        if (end[trie[node][digit ^ 1]] >= limit)
            return query(value, pos - 1, limit, trie[node][digit ^ 1]);
        else
            return query(value, pos - 1, limit, trie[node][digit]);
    }

    int main()
    {
        int N, M;
        std::cin >> N >> M;

        end[0] = -1;
        root[0] = insert(0, 23, 0);
        for (int i(1); i <= N; ++i)
        {
            int x;
            std::cin >> x;
            sum[i] = sum[i - 1] ^ x;
            root[i] = insert(i, 23, root[i - 1]);
            //std::cout << '!' << root[i] << std::endl << std::endl;
        }

        for (int i(1); i <= M; ++i)
        {
            char op;
            std::cin >> op;
            if (op == 'A')
            {
                int x;
                std::cin >> x;
                ++N;
                sum[N] = sum[N - 1] ^ x;
                root[N] = insert(N, 23, root[N - 1]);
            }
            else
            {
                int l, r, x;
                std::cin >> l >> r >> x;
                std::cout << query(x ^ sum[N], 23, l - 1, root[r - 1]) << '\n';
            }
        }
    }

```

```

        return 0;
}

```

## 38 plug\_dp.cpp

```

// 5128

#include <stdio>
#include <cmath>
#include <string>
#include <algorithm>
#include <vector>

const int maxN (12);
char      in   [maxN];
int       color[maxN][maxN];
int       w    [maxN][maxN];
int       dp   [1 << (2 * maxN)];
int       n;

struct Point
{
    int pos;
    int x;
    int y;
};

void print(int x)
{
    for (int i(0); i < 2 * n; ++i)
    {
        printf("%d", (x >> i) & 1);
    }
    printf("\n");
}

int main()
{
    scanf("%d", &n);
    for (int i(0); i < n; ++i)
    {
        scanf("%s", in);
        for (int j(0); j < n; ++j)
        {
            if (in[j] == 'W')
            {
                color[i][j] = 1;
            }
            else if (in[j] == 'B')
            {

```

```

        color[i][j] = 2;
    }
    else
    {
        color[i][j] = 0;
    }
}
}
for (int i(0); i < n; ++i)
{
    for (int j(0); j < n; ++j)
    {
        scanf("%d", &w[i][j]);
    }
}

const int inf(0x3f3f3f3f);
memset(dp, inf, sizeof(dp));
dp[(1 << (2 * n)) - (1 << n)] = 0;

for (int state(1 << (2 * n)); state; --state)
{
    if (dp[state] != inf)
    {
        std::vector<Point> corner;
        {
            Point p = {0, -1, 0};
            while(p.pos < 2 * n - 1)
            {
                p.x += (((state >> p.pos) & 1) == 0);
                p.y += (((state >> p.pos) & 1) == 1);
                if (0 <= p.x && p.x < n && 0 <= p.y && p.y < n &&
→ ((state >> p.pos) & 3) == 2)
                {
                    corner.push_back(p);
                }
                ++p.pos;
            }
        }

        for (std::size_t i(0); i < corner.size(); ++i)
        {
            int nxt(state ^ (3 << corner[i].pos));
            dp[nxt] = std::min(dp[nxt], dp[state] +
→ w[corner[i].x][corner[i].y]);
        }

        for (std::size_t i(0); i < corner.size(); ++i)
        {
            if (color[corner[i].x][corner[i].y])
            {

```

```

        for (std::size_t j(0); j < corner.size(); ++j)
        {
            if (color[corner[j].x][corner[j].y] &&
↪ color[corner[i].x][corner[i].y] != color[corner[j].x][corner[j].y])
            {
                int nxt(state ^ (3 <<
↪ corner[i].pos) ^ (3 << corner[j].pos));
                dp[nxt] = std::min(dp[nxt],
↪ dp[state] + std::abs(w[corner[i].x][corner[i].y] - w[corner[j].x][corner[j].y]));
            }
        }
    }

    printf("%d\n", dp[(1 << n) - 1]);
    return 0;
}

```

### 39 qpow.cpp

```

long long qpow(long long base, long long ex)
{
    long long prod(1);
    for (long long i(1); i <= ex; i *= 2)
    {
        if (i & ex) prod = prod * base % mod;
        base = base * base % mod;
    }
    return prod;
}

long long inv(long long n)
{
    return qpow(n, mod - 2);
}

long long C(long long n, long long m)
{
    return fact[n] * inv(fact[m] * fact[n - m] % mod) % mod;
}

```

### 40 range\_dp.cpp

```

// 5274

#include <cstdio>
#include <limits>
#include <algorithm>

```

```

#include <cstring>

const int maxN(100 + 10);
int      a      [maxN];
int      f      [maxN][maxN][maxN];

void solve(int l, int r, int k, int zero)
{
    memset(f[l][r], 0, sizeof(f[l][r]));
    if (l > r) return;

    int min(l);
    for (int i(l); i <= r; ++i)
    {
        if (a[i] < a[min])
        {
            min = i;
        }
    }

    solve(l, min - 1, k, zero);
    solve(min + 1, r, k, zero);
    for (int i(l); i <= k; ++i)
    {
        for (int j(0); j <= i; ++j)
        {
            f[l][r][i] = std::max(f[l][r][i], f[l][min - 1][j] + f[min +
↪ 1][r][i - j]);
        }
    }

    /*printf("[%d,%d] %d: ", l, r, zero);
    for (int i(l); i <= k; ++i)
    {
        printf("%d ", f[l][r][i]);
    }
    printf("\n"); */

    solve(l, min - 1, k, a[min]);
    solve(min + 1, r, k, a[min]);
    for (int i(l); i <= k; ++i)
    {
        for (int j(0); j <= i - 1; ++j)
        {
            f[l][r][i] = std::max(f[l][r][i], f[l][min - 1][j] + f[min +
↪ 1][r][i - j - 1] + (r - l + 1) * (a[min] - zero));
        }
    }

    /*printf("[%d,%d] %d: ", l, r, zero);
    for (int i(l); i <= k; ++i)

```

```

    {
        printf("%d ", f[l][r][i]);
    }
    printf("\n"); /**/
}

int main()
{
    int n, k;
    scanf("%d %d", &n, &k);
    for (int i(1); i <= n; ++i)
    {
        scanf("%d", &a[i]);
    }

    solve(1, n, k, 0);
    printf("%d\n", f[1][n][k]);
    return 0;
}

```

## 41 rng.cpp

```

struct Rng
{
    uint64_t state;

    Rng(uint64_t state) : state(state)
    {
    }

    uint32_t getRaw()
    {
        uint64_t oldstate = state;
        state = oldstate * 6364136223846793005ULL + 1;
        uint32_t xorshifted = ((oldstate >> 18u) ^ oldstate) >> 27u;
        uint32_t rot = oldstate >> 59u;
        return (xorshifted >> rot) | (xorshifted << ((-rot) & 31));
    }

    int get(int min, int max)
    {
        return min + 1ll * getRaw() % (max - min + 1);
    }
};

```

## 42 sa.cpp

```

#include <cstdio>
#include <cstring>
#include <algorithm>

```



```

int n;

const int maxN (400000 + 10);
const int maxChar(26);
const int maxLbN (30);
char      S      [maxN];
int       sa      [maxN];
int       rank    [maxN];
int       cnt     [maxN];
int       second  [maxN];
int       rankSec [maxN];
int       oldRank [maxN];
int       height  [maxN];
int       st      [maxLbN][maxN];

void buildSa()
{
    int max(maxChar);
    memset(cnt + 1, 0, max * sizeof(int));
    for (int i(1); i <= n; ++i) ++cnt[rank[i] = int(S[i] - 'a' + 1)];
    for (int i(2); i <= max; ++i) cnt[i] += cnt[i - 1];
    for (int i(n); i >= 1; --i) sa[cnt[rank[i]]--] = i;

    for (int mid(1); mid < n; mid *= 2)
    {
        int front(0);
        for (int i(n - mid + 1); i <= n; ++i)
        {
            second[++front] = i;
        }
        for (int i(1); i <= n; ++i)
        {
            if (sa[i] >= mid + 1)
            {
                second[++front] = sa[i] - mid;
            }
        }

        memset(cnt + 1, 0, max * sizeof(int));
        for (int i(1); i <= n; ++i) ++cnt[rankSec[i] = rank[second[i]]];
        for (int i(2); i <= max; ++i) cnt[i] += cnt[i - 1];
        for (int i(n); i >= 1; --i) sa[cnt[rankSec[i]]--] = second[i];

        memcpy(oldRank + 1, rank + 1, n * sizeof(int));

        int id(0);
        for (int i(1); i <= n; ++i)
        {
            int x(sa[i - 1]), y(sa[i]);

```

```

        if (oldRank[x] != oldRank[y] || oldRank[x + mid] != oldRank[y +
↪ mid])
        {
            ++id;
        }
        rank[y] = id;
    }

    max = id;
    if (max == n) break;
}

/*for (int i(1); i <= n; ++i)
{
    printf("%d ", sa[i]);
}
printf("\n"); //*/
}

void calcHeight()
{
    for (int i(1); i <= n; ++i)
    {
        height[rank[i]] = std::max(height[rank[i - 1]] - 1, 0);
        while (S[i + height[rank[i]]] == S[sa[rank[i] - 1] + height[rank[i]]])
        {
            ++height[rank[i]];
        }
    }

    for (int i(1); i <= n; ++i)
    {
        st[0][i] = height[i];
    }
    for (int i(1); i < maxLbN; ++i)
    {
        for (int j(1); j + (1 << i) - 1 <= n; ++j)
        {
            st[i][j] = std::min(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
        }
    }
}

int lb(int x)
{
    int y(0);
    while ((1 << y) <= x)
    {
        ++y;
    }
    return y - 1;
}

```

```

}

int get(int x, int y)
{
    x = rank[x];
    y = rank[y];
    if (x > y) std::swap(x, y);
    ++x;
    int l(lb(y - x + 1));
    return std::min(st[l][x], st[l][y - (1 << l) + 1]);
}

int main()
{
    int T;
    scanf("%d", &T);
    for (int ttt(1); ttt <= T; ++ttt)
    {
        scanf("%s", S + 1);
        n = strlen(S + 1);

        buildSa();
        calcHeight();

        /*for (int i(1); i <= n; ++i)
        {
            printf("height %d: %d\n", i, height[i]);
        }*/
        /*for (int i(1); i <= n; ++i)
        {
            for (int j(i); j <= n; ++j)
            {
                printf("min [%d,%d]: %d\n", i, j, get(i, j));
            }
        }*/

        int ansPos(sa[1]), ansCnt(1), ansLen(1);
        for (int i(1); i <= n / 2; ++i)
        {
            for (int j(1); j + i <= n; j += i)
            {
                int len(get(j, j + i));
                if (len / i + 1 + 1 >= ansCnt)
                {
                    //printf("%d %d @@ %d %s\n", i, j, len, S + j);
                    for (int k(j); k >= 1 && j - k < i && S[k] == S[k
↪ + i]; --k)
                    {
                        //printf("%d %d %d %d %s\n", i, j, k,
↪ len, S + k);

```

```

↪ == ansCnt && rank[k] < rank[ansPos]))
    {
        ansPos = k;
        ansCnt = len / i + 1;
        ansLen = i;
    }
    ++len;
}
}
}

for (int i(0); i < ansCnt * ansLen; ++i)
{
    putchar(S[ansPos + i]);
}
puts("");
}

return 0;
}

```

## 43 sam.cpp

```

#include <stdio>
#include <cstring>

int n;

const int maxN (50000 + 10);
const int maxCh(256 + 10);
char      str  [maxN];
int       next [maxN * 2][maxCh];
int       link [maxN * 2];
int       len  [maxN * 2];

int lastNode;

void buildSam()
{
    lastNode = 0;
    int last(0);
    link[0] = -1;
    memset(next[0], 0, sizeof(next[0]));

    for (int i(1); i <= n; ++i)
    {
        int cur(++lastNode);
        len[cur] = i;
        memset(next[cur], 0, sizeof(next[cur]));
    }
}

```

```

    int pre(last);
    while (pre != -1 && !next[pre][int(str[i])])
    {
        next[pre][int(str[i])] = cur;
        pre = link[pre];
    }

    if (pre != -1)
    {
        int suc(next[pre][int(str[i])]);
        if (len[suc] == len[pre] + 1)
        {
            link[cur] = suc;
        }
        else
        {
            int clone(++lastNode);
            memcpy(next[clone], next[suc], sizeof(next[clone]));
            link[clone] = link[suc];
            len[clone] = len[pre] + 1;
            link[suc] = clone;

            while (pre != -1 && next[pre][int(str[i])] == suc)
            {
                next[pre][int(str[i])] = clone;
                pre = link[pre];
            }
            link[cur] = clone;
        }
    }
    else
    {
        link[cur] = 0;
    }

    last = cur;
}

int main()
{
    int T;
    scanf("%d", &T);
    for (int ttt(1); ttt <= T; ++ttt)
    {
        scanf("%s", str + 1);
        n = strlen(str + 1);

        buildSam();
        long long ans(0);

```

```

        for (int i(1); i <= lastNode; ++i)
        {
            ans += len[i] - len[link[i]];
        }
        printf("%lld\n", ans);
    }

    return 0;
}

```

## 44 search\_mem\_hash.cpp

// 5237

```

#include <cstdio>
#include <bitset>
#include <cstdlib>

int n, m;

const int maxN(9);

const int mod(10000007);

struct Bitset
{
    std::bitset<maxN * maxN> bs;
    std::bitset<maxN * maxN>::reference operator()(int x, int y)
    {
        return bs[x * m + y];
    }
    bool operator()(int x, int y) const
    {
        return bs[x * m + y];
    }

    const std::bitset<maxN * maxN>& data() const
    {
        return bs;
    }

    int hash() const
    {
        int h(0);
        int cur(1);
        for (int i(0); i < maxN * maxN; ++i)
        {
            if (bs[i])
            {
                h = (h + cur) % mod;
            }
        }
    }
}

```

```

        cur = cur * 2 % mod;
    }
    return h;
}

};

bool    used    [mod];
Bitset  belong[mod];
long long mem    [mod];

int hashFind(const Bitset &bs)
{
    int pos(bs.hash());
    while (used[pos] && belong[pos].data() != bs.data())
    {
        pos = (pos + 1) % mod;
    }
    if (!used[pos])
    {
        belong[pos] = bs;
    }
    return pos;
}

char in    [maxN + 10];

const int typeCnt(14);
int      dx      [typeCnt][4] = {

```

```

{0, 1, 2, 3},
{0, 0, 0, 0},
{0, 0, 1, 2},
{0, 1, 1, 1},
{0, 1, 2, 2},
{0, 0, 0, 1},
{0, 0, 1, 2},
{0, 0, 0, 1},

```

```

{0, 1, 2, 2},
{0, 1, 1, 1},
{0, 1, 1, 1},
{0, 1, 1, 2},
{0, 0, 0, 1},
{0, 1, 1, 2}

```

```

    };
int      dy      [typeCnt][4] = {

```

```

{0, 0, 0, 0},
{0, 1, 2, 3},
{0, 1, 1, 1},
{0, 0, -1, -2},
{0, 0, 0, 1},
{0, 1, 2, 0},

```

```

{0, 1, 0, 0},
{0, 1, 2, 2},

{0, 0, 0, -1},
{0, 0, 1, 2},
{0, -1, 0, 1},
{0, 0, 1, 0},
{0, 1, 2, 1},
{0, -1, 0, 0}

```

```
};
```

```

bool can(const Bitset &bs, int x, int y, int type)
{
    for (int i(0); i < 4; ++i)
    {
        int xp(x + dx[type][i]), yp(y + dy[type][i]);
        if (xp < 0 || xp >= n || yp < 0 || yp >= m || bs(xp, yp))
        {
            return false;
        }
    }
    return true;
}

void toggle(Bitset &bs, int x, int y, int type)
{
    for (int i(0); i < 4; ++i)
    {
        int xp(x + dx[type][i]), yp(y + dy[type][i]);
        bs(xp, yp) = !bs(xp, yp);
    }
}

void find(const Bitset &bs, int x, int y, int &xp, int &yp)
{
    xp = x;
    yp = y;
    while (xp < n && bs(xp, yp))
    {
        //printf("LLLLLL %d, %d\n", xp, yp);
        if (yp < m - 1)
        {
            ++yp;
        }
        else
        {
            yp = 0;
            ++xp;
        }
    }
}

```



```

long long dfs(const Bitset &state, int x, int y)
{
    /*printf("!!%d, %d\n", x, y);
    for (int i(0); i < n; ++i)
    {
        for (int j(0); j < m; ++j)
        {
            printf("%d", int(state(i, j)));
        }
        printf("\n");
    }
    printf("\n");
    system("pause");/*/

    int id(hashFind(state));
    //printf("%d\n", id);

    if (used[id])
    {
        return mem[id];
    }

    used[id] = true;

    if (x == n)
    {
        //printf("!!!!!! %d\n", int(state.data().count()));
        mem[id] = (int(state.data().count()) == n * m);
    }
    else
    {
        for (int i(0); i < typeCnt; ++i)
        {
            Bitset cur(state);
            if (can(cur, x, y, i))
            {
                toggle(cur, x, y, i);
                int xp, yp;
                find(cur, x, y, xp, yp);
                mem[id] += dfs(cur, xp, yp);
            }
        }

        return mem[id];
    }
}

int main()
{

```

```

/*n = 4;
m = 8;
for (int i(0); i < typeCnt; ++i)
{
    Bitset bs;
    toggle(bs, 0, m / 2 - 1, i);
    printf("***%d***\n", i);
    for (int j(0); j < n; ++j)
    {
        for (int k(0); k < m; ++k)
        {
            printf("%d", int(bs(j, k)));
        }
        printf("\n");
    }
    printf("\n");
} //*/

scanf("%d %d", &n, &m);
Bitset empty;
for (int i(0); i < n; ++i)
{
    scanf("%s", in);
    for (int j(0); j < m; ++j)
    {
        empty(i, j) = (in[j] == '*');
    }
}

int x, y;
find(empty, 0, 0, x, y);
printf("%lld\n", dfs(empty, x, y));
return 0;
}

```

## 45 segtree.cpp

```

// 3487

#include <cstdio>
#include <stack>

struct Range
{
    int l;
    int value;
};

const int maxN (300000 + 10);
const int inf (0x3f3f3f3f);
int P [maxN];

```

```

int      min   [maxN * 8];
int      cnt   [maxN * 8];
int      delta[maxN * 8];

void init(int node, int tl, int tr)
{
    min[node] = inf;
    cnt[node] = tr - tl + 1;
    if (tl != tr)
    {
        int mid((tl + tr) / 2);
        init(node * 2, tl, mid);
        init(node * 2 + 1, mid + 1, tr);
    }
}

void pushDown(int node)
{
    min[node * 2] += delta[node];
    delta[node * 2] += delta[node];
    min[node * 2 + 1] += delta[node];
    delta[node * 2 + 1] += delta[node];
    delta[node] = 0;
}

void pushUp(int node)
{
    if (min[node * 2] == min[node * 2 + 1])
    {
        min[node] = min[node * 2];
        cnt[node] = cnt[node * 2] + cnt[node * 2 + 1];
    }
    else if (min[node * 2] < min[node * 2 + 1])
    {
        min[node] = min[node * 2];
        cnt[node] = cnt[node * 2];
    }
    else
    {
        min[node] = min[node * 2 + 1];
        cnt[node] = cnt[node * 2 + 1];
    }
}

void add(int node, int tl, int tr, int l, int r, int value)
{
    //if (node == 1) printf("ADD %d,%d %d\n", l, r, value);
    pushDown(node);
    if (l <= tl && tr <= r)
    {
        min[node] += value;
    }
}

```

```

        delta[node] += value;
    }
    else
    {
        int mid((tl + tr) / 2);
        if (l <= mid) add(node * 2, tl, mid, l, r, value);
        if (r >= mid + 1) add(node * 2 + 1, mid + 1, tr, l, r, value);
        pushUp(node);
    }
}

void modify(int node, int tl, int tr, int l, int r, int value)
{
    //if (node == 1) printf("MODIFY %d,%d %d\n", l, r, value);
    pushDown(node);
    if (l <= tl && tr <= r)
    {
        min[node] = value;
    }
    else
    {
        int mid((tl + tr) / 2);
        if (l <= mid) modify(node * 2, tl, mid, l, r, value);
        if (r >= mid + 1) modify(node * 2 + 1, mid + 1, tr, l, r, value);
        pushUp(node);
    }
}

int main()
{
    int N;
    scanf("%d", &N);
    for (int i(1); i <= N; ++i)
    {
        scanf("%d", &P[i]);
    }

    // max - min - r + l = 0
    long long ans(0);
    init(1, 1, N);
    std::stack<Range> minStack, maxStack;
    for (int r(1); r <= N; ++r)
    {
        modify(1, 1, N, r, r, 1);
        add(1, 1, N, 1, r, -1);

        Range curMin = {r, P[r]};
        while (!minStack.empty() && curMin.value <= minStack.top().value)
        {
            add(1, 1, N, minStack.top().l, curMin.l - 1, -(curMin.value -
↪ minStack.top().value));

```

```

        curMin.l = minStack.top().l;
        minStack.pop();
    }
    minStack.push(curMin);

    Range curMax = {r, P[r]};
    while (!maxStack.empty() && curMax.value >= maxStack.top().value)
    {
        add(1, 1, N, maxStack.top().l, curMax.l - 1, curMax.value -
↪ maxStack.top().value);
        curMax.l = maxStack.top().l;
        maxStack.pop();
    }
    maxStack.push(curMax);

    if (min[1] == 0)
    {
        ans += cnt[1];
    }
}

printf("%lld\n", ans);
return 0;
}

```

## 46 segtree\_offline.cpp

// 1677

```

#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <map>
#include <list>

std::map<std::pair<int, int>, int> t;
std::list<std::pair<int, int> > link [200000 * 4 + 10];
std::pair<int, int> query [200000 + 10];
int result[200000 + 10];
int father[10000 + 10];
int size [10000 + 10];

int read()
{
    char ch;
    do ch = getchar();
    while (ch < '0' || ch > '9');

    int ret(0);
    while (ch >= '0' && ch <= '9')

```

```

    {
        ret = ret * 10 + static_cast<int>(ch - '0');
        ch = getchar();
    }
    return ret;
}

char readCmd()
{
    char ch;
    do ch = getchar();
    while (ch < 'A' || ch > 'Z');
    return ch;
}

void insert(int node, int left, int right, int oLeft, int oRight, const std::pair<int,
↳ int> &pair)
{
    //printf("%d %d %d %d\n", left, right, oLeft, oRight);
    //system("pause");
    if (left >= oLeft && right <= oRight)
    {
        link[node].push_back(pair);
        return;
    }

    int mid((left + right) / 2);
    if (oLeft <= mid)
        insert(node * 2, left, mid, oLeft, oRight, pair);
    if (oRight > mid)
        insert(node * 2 + 1, mid + 1, right, oLeft, oRight, pair);
}

int getFather(int index)
{
    while (father[index] != index)
        index = father[index];
    return index;
}

void simulate(int node, int left, int right, std::list<std::pair<int, int> >::iterator
↳ it)
{
    //printf("%d %d\n", left, right);
    int fa1, fa2;
    bool sameTree(true);
    if (it != link[node].end())
    {
        fa1 = getFather(it->second);
        fa2 = getFather(it->first);
        sameTree = (fa1 == fa2);
    }
}

```

```

    }
    if (!sameTree)
    {
        if (size[fa1] < size[fa2])
            std::swap(fa1, fa2);
        father[fa2] = fa1;
        size[fa1] += size[fa2];
    }

    if (it != link[node].end() && ++it != link[node].end())
    {
        simulate(node, left, right, it);
    }
    else if (left < right)
    {
        int mid((left + right) / 2);
        simulate(node * 2, left, mid, link[node * 2].begin());
        simulate(node * 2 + 1, mid + 1, right, link[node * 2 + 1].begin());
    }
    else if (query[left].first != 0)
    {
        //printf("%d %d\n", getFather(query[left].first),
↪ getFather(query[left].second));
        result[left] = (getFather(query[left].first) ==
↪ getFather(query[left].second));
    }

    if (!sameTree)
    {
        father[fa2] = fa2;
        size[fa1] -= size[fa2];
    }
}

int main()
{
    int n(read()), m(read());
    for (int i(1); i <= m; ++i)
    {
        char cmd(readCmd());
        std::pair<int, int> pair;
        pair.first = read();
        pair.second = read();
        if (pair.first > pair.second)
            std::swap(pair.first, pair.second);
        if (cmd == 'C')
        {
            t[pair] = i;
        }
        else if (cmd == 'D')
        {

```

```

        insert(1, 1, m, t[pair], i, pair);
        t[pair] = 0;
    }
    else
    {
        query[i] = pair;
    }
}

for (std::map<std::pair<int, int>, int>::iterator it(t.begin()); it != t.end();
    ++it)
{
    if (it->second != 0)
        insert(1, 1, m, it->second, m, it->first);
}

memset(result, -1, sizeof(result));
for (int i(1); i <= n; ++i)
{
    father[i] = i;
    size[i] = 1;
}
simulate(1, 1, m, link[1].begin());

for (int i(1); i <= m; ++i)
{
    if (result[i] == 0)
        printf("No\n");
    else if (result[i] == 1)
        printf("Yes\n");
}
return 0;
}

```

## 47 segtree\_straddle.cpp

// 5276

```

#include <cstdio>
#include <algorithm>
#include <deque>

const int maxN(2000 + 10);
char in [maxN];
int max [maxN];
int lMin[maxN][maxN];
int rMin[maxN][maxN];

int n, m, k;

void init(int tl, int tr)

```



```

{
    if (tl > tr) return;

    int mid((tl + tr) / 2);
    max[mid] = std::max(mid - tl + 1, tr - mid);
    for (int y(1); y <= m; ++y)
    {
        lMin[mid][y] = mid - tl + 1;
        rMin[mid][y] = tr - mid;
    }

    if (tl < tr)
    {
        init(tl, mid - 1);
        init(mid + 1, tr);
    }
}

void update(int tl, int tr, int x, int y)
{
    if (tl > tr) return;

    int mid((tl + tr) / 2);
    if (x == mid)
    {
        lMin[mid][y] = rMin[mid][y] = 0;
    }
    else if (x < mid)
    {
        lMin[mid][y] = std::min(lMin[mid][y], mid - x);
    }
    else
    {
        rMin[mid][y] = std::min(rMin[mid][y], x - mid - 1);
    }

    max[mid] = 0;
    if (x < mid)
    {
        update(tl, mid - 1, x, y);
    }
    else if (x > mid)
    {
        update(mid + 1, tr, x, y);
    }

    max[mid] = 0;
    if (tl <= mid - 1)
    {
        max[mid] = std::max(max[mid], max[(tl + mid - 1) / 2]);
    }
}

```

```

    }
    if (mid + 1 <= tr)
    {
        max[mid] = std::max(max[mid], max[(mid + 1 + tr) / 2]);
    }

//    printf("update: [%d,%d]\n", tl, tr);

std::deque<int> lQueue, rQueue;
for (int l(1), r(1); r <= m; ++r)
{
    while (!lQueue.empty() && lMin[mid][r] <= lMin[mid][lQueue.back()])
    {
        lQueue.pop_back();
    }
    lQueue.push_back(r);

    while (!rQueue.empty() && rMin[mid][r] <= rMin[mid][rQueue.back()])
    {
        rQueue.pop_back();
    }
    rQueue.push_back(r);

//    printf("(%d,%d)", lQueue.front(), rQueue.front());
    while (!lQueue.empty() && !rQueue.empty() && r - l + 1 >
↪ lMin[mid][lQueue.front()] + rMin[mid][rQueue.front()])
    {
        ++l;
        if (l > lQueue.front()) lQueue.pop_front();
        if (l > rQueue.front()) rQueue.pop_front();
    }

//    printf("[%d,%d] ", l, r);
    max[mid] = std::max(max[mid], r - l + 1);
}

//    printf("\n");
}

void print(int tl, int tr)
{
    if (tl > tr) return;

    printf("[%d,%d]\n", tl, tr);
    int mid((tl + tr) / 2);
    for (int x(1); x <= n; ++x)
    {
        for (int y(1); y <= m; ++y)
        {
            if (x <= mid)
            {
                if (mid - x + 1 <= lMin[mid][y])
                {

```

```

        printf("L");
    }
    else
    {
        printf(".");
    }
}
else
{
    if (x - mid <= rMin[mid][y])
    {
        printf("R");
    }
    else
    {
        printf(".");
    }
}
}
printf("\n");
}

if (tl < tr)
{
    print(tl, mid - 1);
    print(mid + 1, tr);
}
}

int main()
{
    scanf("%d %d %d", &n, &m, &k);
    init(1, n);
    for (int x(1); x <= n; ++x)
    {
        scanf("%s", in + 1);
        for (int y(1); y <= m; ++y)
        {
            if (in[y] == 'X')
            {
                update(1, n, x, y);
            }
        }
    }
    //print(1, n);

    for (int qq(1); qq <= k; ++qq)
    {
        int x, y;
        scanf("%d %d", &x, &y);
        update(1, n, x, y);
    }
}

```

```

        printf("%d\n", max[(1 + n) / 2]);
    }

    return 0;
}

```

## 48 seg\_intersect.cpp

```

// 2470

#include <cstdio>
#include <utility>
#include <cmath>
#include <algorithm>
#include <set>
#include <iostream>
#include <stdlib.h>

struct Vec
{
    long long x;
    long long y;
};

Vec operator-(const Vec &a, const Vec &b)
{
    Vec t = {a.x - b.x, a.y - b.y};
    return t;
}

long long cross(const Vec &a, const Vec &b)
{
    return a.x * b.y - a.y * b.x;
}

const int maxN(100010);
typedef std::pair<Vec, Vec> Seg;
typedef std::pair<int, bool> EndPoint;
typedef std::set<int>::iterator It;
Seg seg [maxN];
EndPoint endpoint[maxN * 2];

bool cmpEndpoint(const EndPoint &a, const EndPoint &b)
{
    Vec p1(a.second == 0 ? seg[a.first].first : seg[a.first].second);
    Vec p2(b.second == 0 ? seg[b.first].first : seg[b.first].second);
    //std::cout << "!" << p1.x << " " << p2.x << "\n";
    if (p1.x != p2.x)
        return p1.x < p2.x;
    else if (a.second != b.second)
        return a.second < b.second;
}

```

```

        else
            return p1.y < p2.y;
    }

bool intersect(int a, int b)
{
    if (a == b)
    {
        //std::cout << seg[endpoint[ttt].first].first.x << ', ' <<
        ↪ seg[endpoint[ttt].first].first.y << ' '
        // << seg[endpoint[ttt].first].second.x << ', ' <<
        ↪ seg[endpoint[ttt].first].second.y << '\n';
        std::cout << "WTF?\n";
    }

    long long d1(cross(seg[a].first - seg[b].first, seg[b].second - seg[b].first));
    long long d2(cross(seg[a].second - seg[b].first, seg[b].second - seg[b].first));
    long long d3(cross(seg[b].first - seg[a].first, seg[a].second - seg[a].first));
    long long d4(cross(seg[b].second - seg[a].first, seg[a].second - seg[a].first));

    //std::cout << "XXX " << d1 << ' ' << d2 << ' ' << d3 << ' ' << d4 << '\n';
    if (((d1 >= 0 && d2 <= 0) || (d1 <= 0 && d2 >= 0)) && ((d3 >= 0 && d4 <= 0) ||
    ↪ (d3 <= 0 && d4 >= 0)))
    {
        if (a > b) std::swap(a, b);
        printf("%d %d\n", a, b);
        return true;
    }
    else
    {
        return false;
    }
}

bool cmpSeg(int a, int b)
{
    if (a == b)
        return false;
    if (intersect(a, b))
        exit(0);

    if (seg[a].first.x < seg[b].first.x)
    {
        ↪ return cross(seg[a].first - seg[b].first, seg[a].second - seg[b].first)
        ↪ >= 0;
    }
    else
    {
        ↪ return cross(seg[b].first - seg[a].first, seg[b].second - seg[a].first) <
        ↪ 0;
    }
}

```

```

}

std::set<int, bool*>(int, int)> set(cmpSeg);

int main()
{
    int n;
    scanf("%d", &n);
    for (int i(1); i <= n; ++i)
    {
        scanf("%lld %lld %lld %lld", &seg[i].first.x, &seg[i].first.y,
        ↪ &seg[i].second.x, &seg[i].second.y);
        if (seg[i].first.x > seg[i].second.x) std::swap(seg[i].first,
        ↪ seg[i].second);
        endpoint[i * 2 - 1] = std::make_pair(i, 0);
        endpoint[i * 2] = std::make_pair(i, 1);
    }
    std::sort(endpoint + 1, endpoint + n * 2 + 1, cmpEndpoint);

    for (int i(1); i <= n * 2; ++i)
    {
        ↪ << '\n';
        //std::cout << i << ' ' << endpoint[i].first << ' ' << endpoint[i].second
        if (endpoint[i].second == 0)
        {
            set.insert(endpoint[i].first);
        }
        else
        {
            It it(set.find(endpoint[i].first));
            if (it == set.end())
                continue;
            It pre(it);
            set.erase(it);
            if (pre != set.begin())
            {
                --pre;
                It suc(pre);
                ++suc;
                //if (pre == suc)
                //    std::cout << "XXX\n";
                if (suc != set.end())
                {
                    if (intersect(*pre, *suc))
                        return 0;
                }
            }
        }
    }
}

return 0;

```

```
}
```

## 49 spfa.cpp

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <queue>

int      start[20 + 10];
int      to      [400000 + 10];
int      pre      [400000 + 10];
long long len      [400000 + 10];
int      last     [100000 + 10];
long long dist [20 + 10][100000 + 10];
long long minDist [100000 + 10];
bool     far      [100000 + 10];
bool     in       [100000 + 10];
bool     dp       [1 << 20];

int n, m, k;
const long long maxByte(0x30);
const long long mod(998244353);
std::queue<int> queue;

void spfa(int origin, long long d[100000 + 10])
{
    d[origin] = 0;
    queue.push(origin);
    in[origin] = true;
    while (!queue.empty())
    {
        for (int i(last[queue.front()]); i != 0; i = pre[i])
        {
            if (d[to[i]] > d[queue.front()] + len[i])
            {
                d[to[i]] = d[queue.front()] + len[i];
                if (!in[to[i]])
                {
                    queue.push(to[i]);
                    in[to[i]] = true;
                }
            }
        }

        in[queue.front()] = false;
        queue.pop();
    }
}
```

```

int main()
{
    std::cin >> n >> m >> k;
    for (int i(0); i < k; ++i)
        std::cin >> start[i];

    for (int i(1); i <= m; ++i)
    {
        int x, y, l;
        std::cin >> x >> y >> l;
        to[i * 2 - 1] = y;
        len[i * 2 - 1] = l;
        pre[i * 2 - 1] = last[x];
        last[x] = i * 2 - 1;

        to[i * 2] = x;
        len[i * 2] = l;
        pre[i * 2] = last[y];
        last[y] = i * 2;
    }

    memset(minDist, maxByte, sizeof(minDist));
    for (int i(0); i < k; ++i)
    {
        memset(dist[i], maxByte, sizeof(dist[i]));
        spfa(start[i], dist[i]);
        for (int j(1); j <= n; ++j)
            minDist[j] = std::min(minDist[j], dist[i][j]);
    }

    /*for (int i(0); i < k; ++i)
    {
        for (int j(1); j <= n; ++j)
            std::cout << dist[i][j] << ' ';
        std::cout << std::endl;
    }*/

    long long max(0);
    for (int j(1); j <= n; ++j)
        max = std::max(max, minDist[j]);
    for (int j(1); j <= n; ++j)
    {
        int state(0);
        for (int i(0); i < k; ++i)
        {
            if (dist[i][j] <= max)
                state |= 1 << i;
        }
        dp[state] = true;
        //std::cout << state << std::endl;
    }
}

```



```

    int count(0);
    for (int i(0); i < (1 << k); ++i)
    {
        if (dp[i])
            ++count;
        for (int j(0); j < k; ++j)
            dp[i | (1 << j)] = dp[i | (1 << j)] || dp[i];
    }

    long long base(1 << k), power(mod - 2);
    long long denom(1);
    for (long long i(1), mult(base); i <= power; i *= 2, mult = mult * mult % mod)
    {
        if (power & i)
            denom = denom * mult % mod;
    }

    //std::cout << count << std::endl;
    std::cout << ((1 << k) - count) * denom % mod << std::endl;
    return 0;
}

```

## 50 sqrt\_block.cpp

// 4343

```

#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <algorithm>
#include <vector>
#include <cctype>
#include <iostream>

struct Question
{
    int type;
    int l;
    int r;
    long long v;
};

const int maxN (200000 + 10);
long long a [maxN];
long long delta[maxN];
long long data [maxN];
Question quest[maxN];

int n;
int size;

```

```

long long read()
{
    char ch(0), sign(0);
    do
    {
        sign = ch;
        ch = getchar();
    }
    while (!isdigit(ch));
    long long x(0);
    while (isdigit(ch))
    {
        x = x * 10 + int(ch - '0');
        ch = getchar();
    }
    return sign == '-' ? -x : x;
}

double cnt1, cnt2;
double sum1, sum2;

double f(double x)
{
    return 5 * cnt1 * x * std::max(std::log(x) / std::log(2), 1.0) + sum1 / x + 5 *
    ↪ cnt2 * x + sum2 / x * std::max(std::log(x) / std::log(2), 1.0);
}

void getSize()
{
    size = 1;
    for (int i(2); i <= n; ++i)
    {
        if (f(i) < f(size))
        {
            size = i;
        }
    }
}

void rebuild(int block)
{
    for (int i(block * size); i <= n && i / size == block; ++i)
    {
        data[i] = a[i];
    }
    std::sort(&data[block * size], &data[(block + 1) * size]);
}

void add(int l, int r, long long v)
{

```

```

    int blkBeg(l / size), blkEnd(r / size);
    for (int i(l); i <= r && i / size == blkBeg; ++i)
    {
        a[i] += v;
    }
    rebuild(blkBeg);

    if (blkBeg < blkEnd)
    {
        for (int i(r); i >= l && i / size == blkEnd; --i)
        {
            a[i] += v;
        }
        rebuild(blkEnd);
    }
    for (int block(blkBeg + 1); block <= blkEnd - 1; ++block)
    {
        delta[block] += v;
    }
}

int query(int l, int r, long long k)
{
    int ans(0);
    int blkBeg(l / size), blkEnd(r / size);

    for (int i(l); i <= r && i / size == blkBeg; ++i)
    {
        if (delta[blkBeg] + a[i] <= k)
        {
            ++ans;
        }
    }
    if (blkBeg < blkEnd)
    {
        for (int i(r); i >= l && i / size == blkEnd; --i)
        {
            if (delta[blkEnd] + a[i] <= k)
            {
                ++ans;
            }
        }
    }

    for (int block(blkBeg + 1); block <= blkEnd - 1; ++block)
    {
        ans += std::upper_bound(&data[block * size], &data[(block + 1) * size], k
↪ - delta[block]) - &data[block * size];
    }
    return ans;
}

```

```

int main()
{
    //freopen("ds.in", "r", stdin);
    //freopen("ds.out", "w", stdout);

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

    int m(read());
    for (int qq(1); qq <= m; ++qq)
    {
        quest[qq].type = read();
        quest[qq].l = read();
        quest[qq].r = read();
        quest[qq].v = read();
        if (quest[qq].type == 1)
        {
            ++cnt1;
            sum1 += quest[qq].r - quest[qq].l + 1;
        }
        else
        {
            ++cnt2;
            sum2 += quest[qq].r - quest[qq].l + 1;
        }
    }

    getSize();

    std::cerr << size << '\n';

    for (int i(0); i <= (n - 1) / size; ++i)
    {
        rebuild(i);
    }

    for (int qq(1); qq <= m; ++qq)
    {
        if (quest[qq].type == 1)
        {
            add(quest[qq].l - 1, quest[qq].r - 1, quest[qq].v);
        }
        else
        {
            printf("%d\n", query(quest[qq].l - 1, quest[qq].r - 1,
↪ quest[qq].v));
        }
    }
}

```

```

    }
    return 0;
}

```

## 51 sqrt\_graph.cpp

```
// 5192
```

```

#include <cstdio>
#include <algorithm>
#include <vector>
#include <cmath>
#include <utility>

struct Query
{
    int x;
    int v;
};

const int          maxN          (2 * 100000 + 10);
int                n, q;
int                orderedSize;
int                crit;
long long          ans;
int                a              [maxN];
int                cur            [maxN];
long long          ordered        [maxN];
int                degree         [maxN];
Query              query          [maxN];
std::vector<int>    edge           [maxN];
std::vector<int>    large          [maxN];
std::vector<int>    bitCnt         [maxN];
std::vector<long long> bitVal      [maxN];

int myLog2(int x)
{
    int cur(1), result(0);
    while (cur < x)
    {
        cur *= 2;
        ++result;
    }
    return result;
}

int getIndex(long long value)
{
    return std::lower_bound(ordered + 1, ordered + orderedSize + 1, value) - ordered;
}

```

```

int lowbit(int x)
{
    return x & (-x);
}

void add(int id, int pos, int cnt, long long value)
{
    if (pos == 0) return;

    while (pos <= orderedSize)
    {
        bitCnt[id][pos] += cnt;
        bitVal[id][pos] += value;
        pos += lowbit(pos);
    }
}

long long queryCnt(int id, int pos)
{
    long long sum(0);
    while (pos > 0)
    {
        sum += bitCnt[id][pos];
        pos -= lowbit(pos);
    }
    return sum;
}

long long queryVal(int id, int pos)
{
    long long sum(0);
    while (pos > 0)
    {
        sum += bitVal[id][pos];
        pos -= lowbit(pos);
    }
    return sum;
}

long long calc(int v)
{
    long long sum(0);
    if (degree[v] <= crit)
    {
        for (std::size_t e(0); e < edge[v].size(); ++e)
        {
            int to(edge[v][e]);
            sum += std::min(ordered[cur[v]], ordered[cur[to]]);
        }
    }
    else

```

```

        {
            sum += (queryCnt(v, orderedSize) - queryCnt(v, cur[v])) *
↪ ordered[cur[v]];
            sum += queryVal(v, cur[v]);
        }

        return sum;
    }

void update(int v, int value)
{
    //printf("update: %d with %lld\n", v, ordered[cur[v]]);

    //printf("erase: %lld %lld\n", -ordered[cur[v]], calc(v));

    ans -= ordered[cur[v]];
    ans += calc(v);

    for (std::size_t e(0); e < large[v].size(); ++e)
    {
        int to(large[v][e]);
        add(to, cur[v], -1, -ordered[cur[v]]);
        add(to, value, 1, ordered[value]);
    }
    cur[v] = value;

    //printf("add: %lld %lld\n", ordered[cur[v]], -calc(v));

    ans += ordered[cur[v]];
    ans -= calc(v);
}

int main()
{
    // freopen("sorry.in", "r", stdin);
    // freopen("sorry.out", "w", stdout);

    scanf("%d %d", &n, &q);
    for (int v(1); v <= n; ++v)
    {
        scanf("%d", &a[v]);
        ordered[++orderedSize] = a[v];
    }
    for (int e(1); e <= n - 1; ++e)
    {
        int u, v;
        scanf("%d %d", &u, &v);
        edge[u].push_back(v);
        edge[v].push_back(u);
        ++degree[u];
        ++degree[v];
    }
}

```

```

}
for (int qqq(1); qqq <= q; ++qqq)
{
    scanf("%d %d", &query[qqq].x, &query[qqq].v);
    ordered[++orderedSize] = query[qqq].v;
}

std::sort(ordered + 1, ordered + orderedSize + 1);
orderedSize = std::unique(ordered + 1, ordered + orderedSize + 1) - ordered - 1;
for (int v(1); v <= n; ++v)
{
    a[v] = getIndex(a[v]);
}
for (int qqq(1); qqq <= q; ++qqq)
{
    query[qqq].v = getIndex(query[qqq].v);
}

crit = sqrt(2 * (n - 1) * myLog2(orderedSize));
for (int v(1); v <= n; ++v)
{
    for (std::size_t e(0); e < edge[v].size(); ++e)
    {
        int to(edge[v][e]);
        if (degree[to] > crit)
        {
            large[v].push_back(to);
        }
    }
}
for (int v(1); v <= n; ++v)
{
    if (degree[v] > crit)
    {
        bitCnt[v].resize(orderedSize + 1);
        bitVal[v].resize(orderedSize + 1);
    }
}

for (int v(1); v <= n; ++v)
{
    update(v, a[v]);
}

for (int qqq(1); qqq <= q; ++qqq)
{
    update(query[qqq].x, query[qqq].v);
    printf("%lld\n", ans);
}
return 0;
}

```



## 52 st.cpp

```
#include <algorithm>

const int maxN (1000000);
const int maxLbN(20);

int a [maxN];
int lb[maxN];
int f [maxLbN][maxN];

int n;

int query(int l, int r)
{
    int lbLen(lb[r - 1 + 1]);
    return std::max(f[lbLen][l], f[lbLen][r - (1 << lbLen) + 1]);
}

void init()
{
    lb[0] = -1;
    for (int i(1); i <= n; ++i)
    {
        lb[i] = lb[i / 2] + 1;
    }

    for (int i(1); i <= n; ++i)
    {
        f[0][i] = a[i];
    }
    for (int i(1); i < maxLbN; ++i)
    {
        for (int j(1); j + (1 << i) - 1 <= n; ++j)
        {
            f[i][j] = std::max(f[i - 1][j], f[i - 1][j + (1 << (i - 1))]);
        }
    }
}
```

## 53 string\_hashing.cpp

```
#include <iostream>
#include <string>
#include <algorithm>

const int          maxLen(100000 + 10);
unsigned long long hashA;
unsigned long long hashB [maxLen];
unsigned long long weight[maxLen];
```

```

int main()
{
    //freopen("string.in", "r", stdin);
    //freopen("string.out", "w", stdout);

    std::ios::sync_with_stdio(false);
    weight[0] = 1;
    for (int i(1); i < maxLen; ++i)
    {
        weight[i] = weight[i - 1] * 31;
    }

    std::string a, b;
    std::cin >> a >> b;
    bool swap(false);
    if (a.length() < b.length())
    {
        std::swap(a, b);
        swap = true;
    }
    std::size_t delta(a.length() - b.length());

    for (std::size_t i(0); i < a.length(); ++i)
    {
        hashA += (a[i] - 'a') * weight[i];
    }
    for (std::size_t i(0); i < a.length(); ++i)
    {
        hashB[0] += (b[i % b.length()] - 'a') * weight[i];
    }
    //printf("0:%llu\n", hashB[0]);
    for (std::size_t i(1); i < b.length(); ++i)
    {
        hashB[i] = hashB[i - 1];
        hashB[i] -= (b[(b.length() + delta - i) % b.length()] - 'a') *
↪ weight[b.length() - 1];
        hashB[i] *= 31;
        hashB[i] += (b[b.length() - i] - 'a');
        //printf("%zu:%llu %llu\n", i, hashB[i], (b.length() + delta - i) %
↪ b.length());
    }
    //printf("\n");

    std::size_t i(0);
    do
    {
        //printf("%zu:%llu %llu\n", i, hashA, hashB[i]);
        if (hashA != hashB[i])
        {
            for (std::size_t j(0); j < a.length(); ++j)

```

```

        {
            std::size_t bj((j + b.length() - i) % b.length());
            //printf("(%c%c)", a[j], b[bj]);
            char ac(a[j]), bc(b[bj]);
            if (swap) std::swap(ac, bc);
            if (ac < bc)
            {
                std::cout << "<\n";
                return 0;
            }
            else if (ac > bc)
            {
                std::cout << ">\n";
                return 0;
            }
        }
        break;
    }
    //printf("\n");

    i = (b.length() - (a.length() - i) % b.length()) % b.length();
}
while (i != 0);

std::cout << "=\n";
return 0;
}

```

## 54 tangent\_segtree.cpp

// 1363

```

#include <stdio>
#include <algorithm>

```

```

struct Frac
{

```

```

    int y;
    int x;

```

```

    Frac() : y(0), x(1)
    {
    }

```

```

    Frac(int y, int x) : y(y), x(x)
    {
    }

```

```

};

```

```

bool operator<(const Frac &a, const Frac &b)
{

```

```

        return 1ll * a.y * b.x - 1ll * b.y * a.x < 0;
    }

    bool operator==(const Frac &a, const Frac &b)
    {
        return 1ll * a.y * b.x - 1ll * b.y * a.x == 0;
    }

    bool operator<=(const Frac &a, const Frac &b)
    {
        return a < b || a == b;
    }

    const int maxN(100000 + 10);
    Frac      a    [maxN * 4];
    Frac      max  [maxN * 4];
    int       ans  [maxN * 4];

    int pushUp(int node, int tl, int tr, const Frac &f)
    {
        if (max[node] <= f)
        {
            return 0;
        }
        else if (f < a[tl])
        {
            return ans[node];
        }
        else
        {
            int mid((tl + tr) / 2);
            if (max[node * 2] < f)
            {
                return pushUp(node * 2 + 1, mid + 1, tr, f);
            }
            else
            {
                return pushUp(node * 2, tl, mid, f) + ans[node] - ans[node * 2];
            }
        }
    }

    void insert(int node, int tl, int tr, const Frac &f)
    {
        if (tl == tr)
        {
            a[tl] = f;
            max[node] = f;
            ans[node] = 1;
        }
        else
    
```

```

        {
            int mid((tl + tr) / 2);
            if (f.x <= mid) insert(node * 2, tl, mid, f);
            else insert(node * 2 + 1, mid + 1, tr, f);
            max[node] = std::max(max[node * 2], max[node * 2 + 1]);
            ans[node] = ans[node * 2] + pushUp(node * 2 + 1, mid + 1, tr, max[node *
↵ 2]);
        }
    }

int main()
{
    int N, M;
    scanf("%d %d", &N, &M);
    for (int i(1); i <= M; ++i)
    {
        Frac f;
        scanf("%d %d", &f.x, &f.y);
        insert(1, 1, N, f);
        printf("%d\n", ans[1]);
    }

    return 0;
}

```

## 55 tarjan\_bcc.cpp

```

// 1536

#include <iostream>
#include <cstring>
#include <stack>
#include <algorithm>

struct Edges
{
    int to [200000 * 2 + 10];
    int pre [200000 * 2 + 10];
    int last[100000 + 10];

    Edges()
    {
        memset(last, -1, sizeof(last));
    }

    void addEdge(int a, int b)
    {
        static int size(0);
        to[size] = b;
        pre[size] = last[a];
        last[a] = size;
    }
}

```

```

        ++size;
    }
};

Edges map;
Edges tree;
std::stack<int> stack;

int dfn [100000 + 10];
int low [100000 + 10];
int bcc [100000 + 10];
int lca [100000 + 10][20];
int pred [100000 + 10];
int depth[100000 + 10];

int currentDfn(1);
int currentBcc(1);

void tarjan(int vertex, int inEdge)
{
    low[vertex] = dfn[vertex] = currentDfn++;
    stack.push(vertex);
    for (int i(map.last[vertex]); i != -1; i = map.pre[i])
    {
        if (i == (inEdge ^ 1))
            continue;

        if (dfn[map.to[i]])
        {
            low[vertex] = std::min(low[vertex], dfn[map.to[i]]);
        }
        else
        {
            tarjan(map.to[i], i);
            low[vertex] = std::min(low[vertex], low[map.to[i]]);
        }
    }

    if (dfn[vertex] == low[vertex])
    {
        bool below(true);
        while (below)
        {
            bcc[stack.top()] = currentBcc;
            if (stack.top() == vertex)
                below = false;
            stack.pop();
        }
        ++currentBcc;
    }
}

```

```

void dfs(int vertex, int inVertex, int d)
{
    lca[vertex][0] = inVertex;
    for (int i(1); i < 20; ++i)
        lca[vertex][i] = lca[lca[vertex][i - 1]][i - 1];
    depth[vertex] = d;
    for (int i(tree.last[vertex]); i != -1; i = tree.pre[i])
    {
        if (tree.to[i] != inVertex)
            dfs(tree.to[i], vertex, d + 1);
    }
}

int getLca(int v1, int v2)
{
    if (depth[v1] < depth[v2])
        std::swap(v1, v2);
    for (int i(19); i >= 0; --i)
    {
        if (depth[v1] - (1 << i) >= depth[v2])
            v1 = lca[v1][i];
    }
    if (v1 == v2)
        return v1;
    for (int i(19); i >= 0; --i)
    {
        if (lca[v1][i] != lca[v2][i])
        {
            v1 = lca[v1][i];
            v2 = lca[v2][i];
        }
    }
    return lca[v1][0];
}

int get(int vertex)
{
    if (vertex == pred[vertex])
        return vertex;
    else
        return pred[vertex] = get(pred[vertex]);
}

int link(int vertex, int ancestor)
{
    int i(get(vertex));
    int count(0);
    while (depth[i] > depth[ancestor])
    {
        pred[i] = lca[i][0];
    }
}

```

```

        ++count;
        i = get(i);
    }
    //std::cout << "!" << vertex << " " << ancestor << " " << count << std::endl;
    return count;
}

int main()
{
    int N, M;
    std::cin >> N >> M;
    for (int i(1); i <= M; ++i)
    {
        int a, b;
        std::cin >> a >> b;
        map.addEdge(a, b);
        map.addEdge(b, a);
    }

    tarjan(1, -1);

    int bridge(0);
    for (int from(1); from <= N; ++from)
    {
        //std::cout << from << " " << bcc[from] << std::endl;
        for (int i(map.last[from]); i != -1; i = map.pre[i])
        {
            //std::cout << from << "->" << map.to[i]
            // << " " << bcc[from] << "->" << bcc[map.to[i]] << std::endl;
            if (bcc[from] != bcc[map.to[i]])
            {
                tree.addEdge(bcc[from], bcc[map.to[i]]);
                ++bridge;
            }
        }
    }
    bridge /= 2;

    dfs(1, 0, 1);
    for (int i(1); i < currentBcc; ++i)
        pred[i] = i;

    int Q;
    std::cin >> Q;
    for (int i(1); i <= Q; ++i)
    {
        int a, b;
        std::cin >> a >> b;
        if (bcc[a] == bcc[b])
        {
            std::cout << bridge << "\n";
        }
    }
}

```



```

    }
    else
    {
        int ancestor(getLca(bcc[a], bcc[b]));
        bridge -= link(bcc[a], ancestor);
        bridge -= link(bcc[b], ancestor);
        std::cout << bridge << '\n';
    }
}

return 0;
}

```

## 56 tarjan\_cut\_edge.cpp

```

const int SIZE = 100010;
int head[SIZE], ver[SIZE * 2], Next[SIZE * 2];
int dfn[SIZE], low[SIZE], n, m, tot, num;
bool bridge[SIZE * 2];

void add(int x, int y)
{
    ver[++tot] = y, Next[tot] = head[x], head[x] = tot;
}

void tarjan(int x, int in_edge)
{
    dfn[x] = low[x] = ++num;
    for (int i = head[x]; i; i = Next[i])
    {
        int y = ver[i];
        if (!dfn[y])
        {
            tarjan(y, i);
            low[x] = min(low[x], low[y]);

            if (low[y] > dfn[x])
                bridge[i ^ 1] = true;
        }
        else if (i != (in_edge ^ 1))
            low[x] = min(low[x], dfn[y]);
    }
}

int main()
{
    cin >> n >> m;
    tot = 1;
    for (int i = 1; i <= m; ++i)
    {
        int x, y;

```

```

        scanf("%d%d", &x, &y);
        add(x, y), add(y, x);
    }
    for (int i = 1; i <= n; ++i)
        if (!dfn[i]) tarjan(i, 0);
    for (int i = 2; i < tot; i += 2)
        if (bridge[i])
            printf("%d %d\n", ver[i ^ 1], ver[i]);
}

```

## 57 tarjan\_cut\_vertex.cpp

```

const int SIZE = 100010;
int head[SIZE], ver[SIZE * 2], Next[SIZE * 2];
int dfn[SIZE], low[SIZE], stack[SIZE];
int n, m, tot, num, root;
bool cut[SIZE];

void add(int x, int y)
{
    ver[++tot] = y, Next[tot] = head[x], head[x] = tot;
}

void tarjan(int x)
{
    dfn[x] = low[x] = ++num;
    int flag = 0;
    for (int i = head[x]; i; i = Next[i])
    {
        int y = ver[i];
        if (!dfn[y])
        {
            tarjan(y);
            low[x] = min(low[x], low[y]);
            if (low[y] >= dfn[x])
            {
                ++flag;
                if (x != root || flag > 1) cut[x] = true;
            }
        }
        else low[x] = min(low[x], dfn[y]);
    }
}

int main()
{
    cin >> n >> m;
    tot = 1;
    for (int i = 1; i <= m; ++i)
    {
        int x, y;

```

```

        scanf("%d%d", &x, &y);
        if (x == y) continue;
        add(x, y), add(y, x);
    }
    for (int i = 1; i <= n; ++i)
        if (!dfn[i]) root = i, tarjan(i);
    for (int i = 1; i <= n; ++i)
        if (cut[i]) printf("%d ", i);
    puts("are cut-vertexes");
}

```

## 58 tarjan\_dcc.cpp

// 5145

```

#include <cstdio>
#include <cmath>
#include <algorithm>
#include <stack>
#include <bitset>
#include <queue>

struct IO {
#define MAXSIZE (1 << 20)
#define isdigit(x) (x >= '0' && x <= '9')
    char buf[MAXSIZE], *p1, *p2;
    char pbuf[MAXSIZE], *pp;
#ifdef DEBUG
#else
    IO() : p1(buf), p2(buf), pp(pbuf) {}
    ~IO() { fwrite(pbuf, 1, pp - pbuf, stdout); }
#endif
    inline char gc() {
#ifdef DEBUG
        return getchar();
#else
        if (p1 == p2) p2 = (p1 = buf) + fread(buf, 1, MAXSIZE, stdin);
        return p1 == p2 ? ' ' : *p1++;
#endif
    }
    inline bool blank(char ch) {
        return ch == ' ' || ch == '\n' || ch == '\r' || ch == '\t';
    }
}
template <class T>
    inline void read(T &x) {
        register double tmp = 1;
        register bool sign = 0;
        x = 0;
        register char ch = gc();
        for (; !isdigit(ch); ch = gc())
            if (ch == '-') sign = 1;
        for (; isdigit(ch); ch = gc()) x = x * 10 + (ch - '0');
    }

```

```

        if (ch == '.')
            for (ch = gc(); isdigit(ch); ch = gc())
                tmp /= 10.0, x += tmp * (ch - '0');
        if (sign) x = -x;
    }
    inline void read(char *s) {
        register char ch = gc();
        for (; blank(ch); ch = gc())
            ;
        for (; !blank(ch); ch = gc()) *s++ = ch;
        *s = 0;
    }
    inline void read(char &c) {
        for (c = gc(); blank(c); c = gc())
            ;
    }
    inline void push(const char &c) {
#ifdef DEBUG
        putchar(c);
#else
        if (pp - pbuf == MAXSIZE) fwrite(pbuf, 1, MAXSIZE, stdout), pp = pbuf;
        *pp++ = c;
#endif
    }
    template <class T>
    inline void write(T x) {
        if (x < 0) x = -x, push('-');
        static T sta[35];
        T top = 0;
        do {
            sta[int(top++)] = x % 10, x /= 10;
        } while (x);
        while (top) push(sta[int(--top)] + '0');
    }
    template <class T>
    inline void write(T x, char lastChar) {
        write(x), push(lastChar);
    }
} io;

const int      maxN    (2000 + 10);
int             n;
int             lastDfn(0);
int             lastDcc(0);
double          A       [maxN][maxN];
double          E       [maxN][maxN];
bool            adj      [maxN][maxN];
int             dfn       [maxN];
int             low       [maxN];
std::stack<int>  stack;
int             dcc       [maxN];

```

```

int          size    [maxN];
int          edge    [maxN][maxN];
int          in      [maxN];
std::bitset<maxN> connect[maxN];

void tarjan(int v)
{
    dfn[v] = low[v] = ++lastDfn;
    stack.push(v);

    for (int to(1); to <= n; ++to)
    {
        if (adj[v][to] && !dcc[to])
        {
            if (!dfn[to])
            {
                tarjan(to);
            }
            low[v] = std::min(low[v], low[to]);
        }
    }

    //printf("TARJAN %d: %d %d\n", v, dfn[v], low[v]);
    if (dfn[v] == low[v])
    {
        ++lastDcc;
        bool done(false);
        while (!done)
        {
            dcc[stack.top()] = lastDcc;
            ++size[lastDcc];
            if (stack.top() == v) done = true;
            stack.pop();
        }
    }
}

int main()
{
    //freopen("matrix.in", "r", stdin);

    io.read(n);
    for (int i(1); i <= n; ++i)
    {
        E[i][i] = 1;
    }
    for (int i(1); i <= n; ++i)
    {
        for (int j(1); j <= n; ++j)
        {
            io.read(A[i][j]);
        }
    }
}

```

```

        A[i][j] = E[i][j] - A[i][j];
        if (i != j)
        {
            adj[i][j] = (std::abs(A[i][j]) > 1e-9);
            /*if (std::abs(A[i][j]) > 1e-9)
            {
                printf("%d %d\n", i, j);
            }*/
        }
    }

for (int v(1); v <= n; ++v)
{
    if (!dcc[v])
    {
        tarjan(v);
    }
    //printf("%d: %d\n", v, dcc[v]);
}

for (int v(1); v <= n; ++v)
{
    for (int to(1); to <= n; ++to)
    {
        if (dcc[v] != dcc[to] && adj[v][to] && !edge[dcc[to]][dcc[v]])
        {
            edge[dcc[to]][dcc[v]] = true;
            ++in[dcc[v]];
        }
    }
}

std::queue<int> queue;
for (int v(1); v <= lastDcc; ++v)
{
    //printf("%d ", in[v]);
    connect[v][v] = 1;
    if (in[v] == 0)
    {
        queue.push(v);
    }
}
//printf("\n");

while (!queue.empty())
{
    int v(queue.front());
    queue.pop();
    for (int to(1); to <= lastDcc; ++to)
    {

```

```

        if (edge[v][to])
        {
            connect[to] |= connect[v];
            --in[to];
            if (in[to] == 0)
            {
                queue.push(to);
            }
        }
    }

    int ans(0);
    for (int v(1); v <= lastDcc; ++v)
    {
        //printf("%d: ", v);
        for (int to(1); to <= lastDcc; ++to)
        {
            //printf("%d", int(connect[v][to]));
            if (connect[v][to])
            {
                ans += size[v] * size[to];
            }
        }
        //printf("\n");
    }
    io.write(ans, '\n');
    return 0;
}

```

## 59 topo\_sort.cpp

// 4602

```

#include <cstdio>
#include <iostream>
#include <cstring>
#include <queue>
#include <utility>
#include <bitset>
#include <vector>
#include <cstdio>

int n, m;
long long ans(0);
int to [1200000 + 10];
int pre [1200000 + 10];
long long cost[1200000 + 10];
int last[30000 + 10];
long long dist[30000 + 10];
bool go [30000 + 10];

```

```

long long deg [30000 + 10];

struct Cmp
{
    bool operator()(int a, int b)
    {
        return dist[a] > dist[b];
    }
};

std::bitset<30000 + 10> state[30000 + 10];
std::queue<int> queue;
std::priority_queue<int, std::vector<int>, Cmp> pqueue;

void dijkstra()
{
    memset(dist, 0x30, sizeof(dist));
    dist[1] = 0;
    pqueue.push(1);

    while (!pqueue.empty())
    {
        if (!go[pqueue.top()])
        {
            go[pqueue.top()] = true;
            for (int i(last[pqueue.top()]); i != -1; i = pre[i])
            {
                if (!go[to[i]] && dist[to[i]] > dist[pqueue.top()] +
↪ cost[i])
                {
                    dist[to[i]] = dist[pqueue.top()] + cost[i];
                    pqueue.push(to[i]);
                }
            }
            pqueue.pop();
        }
    }
}

void init()
{
    memset(deg, -1, sizeof(deg));
    deg[n] = 0;
    queue.push(n);
    while (!queue.empty())
    {
        for (int i(last[queue.front()]); i != -1; i = pre[i])
        {
            if (dist[queue.front()] == dist[to[i]] + cost[i])
            {
                if (deg[to[i]] == -1)

```



```

        {
            deg[to[i]] = 1;
            queue.push(to[i]);
        }
        else
        {
            ++deg[to[i]];
        }
    }
    queue.pop();
}

void topoSort()
{
    queue.push(n);
    while (!queue.empty())
    {
        ans += state[queue.front()].count();
        state[queue.front()].set(queue.front());
        //std::cout << queue.front() << ' ' << state[queue.front()].to_string()
        << '\n';
        for (int i(last[queue.front()]); i != -1; i = pre[i])
        {
            if (deg[to[i]] > 0)
            {
                --deg[to[i]];
                state[to[i]] |= state[queue.front()];
                if (deg[to[i]] == 0)
                    queue.push(to[i]);
            }
        }
        queue.pop();
    }
}

int main()
{
    memset(last, -1, sizeof(last));

    scanf("%d%d", &n, &m);
    for (int i(0); i < m; ++i)
    {
        int a, b;
        long long c;
        scanf("%d%d%lld", &a, &b, &c);
        to[i * 2] = b;
        pre[i * 2] = last[a];
        last[a] = i * 2;
        cost[i * 2] = c;
    }
}

```

```

        to[i * 2 + 1] = a;
        pre[i * 2 + 1] = last[b];
        last[b] = i * 2 + 1;
        cost[i * 2 + 1] = c;
    }

    dijkstra();
    init();
    topoSort();
    //dfs(n, 1);

    printf("%lld\n", ans);
    return 0;
}

```

## 60 tree\_center.cpp

```

void dfs(int x)
{
    v[x] = 1;
    size[x] = 1;
    int max_part = 0;
    for (int i = head[x]; i; i = next[i])
    {
        int y = ver[i];
        if (v[y]) continue;
        dfs(y);
        size[x] += size[y];
        max_part = max(max_part, size[y]);
    }
    max_part = max(max_part, n - size[x]);
    if (max_part < ans)
    {
        ans = max_part;
        pos = x;
    }
}

```

## 61 tree\_diameter.cpp

*// Or: DFS twice*

```

void dp(int x)
{
    v[x] = 1;
    for (int i = head[x]; i; i = Next[i])
    {
        int y = ver[i];
        if (v[y]) continue;
    }
}

```

```

        dp(y);
        ans = max(ans, d[x] + d[y] + edge[i]);
        d[x] = max(d[x], d[y] + edge[i]);
    }
}

```

## 62 trie.cpp

```

#include <cstdio>
#include <algorithm>
#include <vector>
#include <limits>

const int      maxN      (100000 + 10);
const int      logW      (30);
int            lastNode(0);
int            to         [maxN * 2];
int            pre         [maxN * 2];
int            weight      [maxN * 2];
int            last        [maxN];
int            sum          [maxN];
int            son          [maxN * logW][2];
std::vector<int> sub        [maxN * logW];

void addEdge(int x, int y, int w)
{
    static int lastEdge(0);
    ++lastEdge;
    to[lastEdge] = y;
    pre[lastEdge] = last[x];
    weight[lastEdge] = w;
    last[x] = lastEdge;
}

void dfs(int v, int pred)
{
    for (int e(last[v]); e; e = pre[e])
    {
        if (to[e] != pred)
        {
            sum[to[e]] = sum[v] ^ weight[e];
            dfs(to[e], v);
        }
    }
}

void insert(int value)
{
    int node(0);
    for (int i(logW); i >= 0; --i)
    {

```

```

        int bit(value >> i & 1);
        if (!son[node][bit]) son[node][bit] = ++lastNode;
        node = son[node][bit];
        sub[node].push_back(value);
    }
}

int find(int node, int pos, int value)
{
    int delta(0);
    for (int i(pos); i >= 0; --i)
    {
        int bit(value >> i & 1);
        if (son[node][bit])
        {
            node = son[node][bit];
        }
        else
        {
            delta |= 1 << i;
            node = son[node][bit ^ 1];
        }
    }
    return delta;
}

long long solve(int node, int pos)
{
    long long ans(0);
    int l(son[node][0]), r(son[node][1]);
    if (l && r)
    {
        if (sub[l].size() > sub[r].size()) std::swap(l, r);
        int min(std::numeric_limits<int>::max());
        for (std::size_t i(0); i < sub[l].size(); ++i)
        {
            min = std::min(min, (1 << pos) | find(r, pos - 1, sub[l][i]));
        }
        ans += min;
    }

    if (l) ans += solve(l, pos - 1);
    if (r) ans += solve(r, pos - 1);
    return ans;
}

int main()
{
    int n;
    scanf("%d", &n);
    for (int i(1); i <= n - 1; ++i)

```

```

{
    int x, y;
    int w;
    scanf("%d %d %d", &x, &y, &w);
    ++x, ++y;
    addEdge(x, y, w);
    addEdge(y, x, w);
}

dfs(1, 1);

for (int i(1); i <= n; ++i)
{
    insert(sum[i]);
}

printf("%lld\n", solve(0, logW));
return 0;
}

```

## 63 vec.cpp

```

#include <cmath>

struct Vec
{
    double x;
    double y;

    Vec()
    {
    }

    Vec(double x, double y) : x(x), y(y)
    {
    }

    double norm() const
    {
        return std::sqrt(x * x + y * y);
    }

    Vec normalize() const
    {
        double invNorm(1 / norm());
        return Vec(x * invNorm, y * invNorm);
    }
};

Vec operator+(const Vec &a, const Vec &b)
{

```

```

        return Vec(a.x + b.x, a.y + b.y);
    }

Vec operator-(const Vec &a, const Vec &b)
{
    return Vec(a.x - b.x, a.y - b.y);
}

Vec operator*(double a, const Vec &b)
{
    return Vec(a * b.x, a * b.y);
}

double dot(const Vec &a, const Vec &b)
{
    return a.x * b.x + a.y * b.y;
}

double cross(const Vec &a, const Vec &b)
{
    return a.x * b.y - a.y * b.x;
}

double xtan2(double y, double x)
{
    double p(x / (std::abs(x) + std::abs(y)));
    return y < 0 ? p - 1 : 1 - p;
}

double diff(const Vec &a, const Vec &b)
{
    static const double pi(std::acos(-1));
    double angle(xtan2(cross(a, b), dot(a, b)));
    return angle >= 0 ? angle : pi * 2 + angle;
}

int quad(const Vec &v)
{
    if (v.x >= 0)
    {
        if (v.y >= 0)
            return 3;
        else
            return 2;
    }
    else
    {
        if (v.y >= 0)
            return 4;
        else
            return 1;
    }
}

```

```

    }
}

bool operator<(const Vec &a, const Vec &b)
{
    Vec da(a - origin), db(b - origin);
    return quad(da) < quad(db) || (quad(da) == quad(db) && cross(da, db) > 0);
}

```