

目录

1	基础算法	1
1.1	三分	1
1.2	二分	1
1.3	快速幂	2
1.4	离散化	2
2	图论	2
2.1	Tarjan 割点	2
2.2	Tarjan 割边	3
2.3	Tarjan 强连通分量	4
2.4	Tarjan 点双连通分量	5
2.5	Tarjan 边双连通分量	6
2.6	拓扑排序	6
2.7	最小生成树 kruskal	7
2.8	最小生成树 prim	8
3	数据结构	9
3.1	Splay	9
3.2	ST 表	11
3.3	对顶堆	12

3.4	并查集	13
3.5	树状数组	14
3.6	线段树	14
4	树算法	17
4.1	树剖 LCA	17
5	数论	17
5.1	欧拉筛	17
6	字符串	18
6.1	EXKMP	18
6.2	KMP	19
6.3	字符串哈希	19
6.4	马拉车	20

1 基础算法

1.1 三分

```
1 #include <bits/stdc++.h>
2 constexpr double eps = 1E-6;//eps控制精度
3
4 //三分（实数范围）凸函数
5 //https://www.luogu.com.cn/record/160695683
6 int main() {
7     std::ios::sync_with_stdio(false);
8     std::cin.tie(nullptr);
9     int n;
10    double l, r;
11    std::cin >> n >> l >> r;
12    std::vector<double> v(n + 1);
13    for(int i = n; i >= 0; --i) {
14        std::cin >> v[i];
15    }
16    auto check = [&](double t) ->double {
17        double ans = 0;
18        for(int i = 0; i <= n; ++i) {
19            ans += v[i] * std::pow(t, i);
20        }
21        return ans;
22    };
23    while(l + eps <= r) {
24        double lmid = l + (r - l) / 3;//左三分点
25        double rmid = r - (r - l) / 3;//右三分点
26        if(check(lmid) < check(rmid)) {
27            l = lmid;
28        } else {
29            r = rmid;
30        }
31    }
```

```
32    std::cout << l << '\n';
33    return 0;
34 }
```

1.2 二分

```
1 #include <bits/stdc++.h>
2
3 //二分查找
4 //https://www.luogu.com.cn/record/160694930
5 int binaryFind(std::vector<int> &v, int t) {
6     int l = 1, r = v.size() - 1, ans = -1;
7     while(l <= r) {
8         int mid = l + (r - l) / 2;
9         if(v[mid] >= t) { //此处可换成check函数
10             r = mid - 1;
11             if(v[mid] == t) { //判断什么时候更新答案
12                 ans = mid;
13             }
14         } else {
15             l = mid + 1;
16         }
17     }
18     return ans;
19 }
20
21 int main() {
22     std::ios::sync_with_stdio(false);
23     std::cin.tie(nullptr);
24     int n, m;
25     std::cin >> n >> m;
26     std::vector<int> v(n + 1);
27     for(int i = 1; i <= n; ++i) {
28         std::cin >> v[i];
29     }
```

```

30     for(int i = 1; i <= m; ++i) {
31         int x;
32         std::cin >> x;
33         std::cout << binaryFind(v, x) << " \n"[i == m];
34     }
35     return 0;
36 }

```

1.3 快速幂

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 //快速幂 (Binary Exponentiation)
5 i64 qpow(i64 a, i64 b, i64 p = LLONG_MAX) { //底数, 指数, 模数
6     i64 res = 1;
7     while(b > 0) {
8         if(b & 1) res = res * a % p;
9         a = a * a % p;
10        b >>= 1;
11    }
12    return res;
13 }
14
15 int main() {
16     ____std::ios::sync_with_stdio(false);
17     std::cin.tie(nullptr);
18     std::cout << qpow(2, 20) << '\n';
19     std::cout << std::pow(2, 20) << '\n';
20     return 0;
21 }

```

1.4 离散化

```

1 #include <bits/stdc++.h>
2
3 //离散化
4 int main() {
5     std::vector<int> arr = {1000, 500, 9999, 200, 356, 200};
6     std::vector<int> tmp(arr);
7     std::sort(tmp.begin(), tmp.end()); //排序
8     tmp.erase(std::unique(tmp.begin(), tmp.end()), tmp.end()); //去重
9     for (int i = 0; i < arr.size(); ++i) { //替换
10         arr[i] = std::lower_bound(tmp.begin(), tmp.end(), arr[i]) - tmp.begin() +
11             1;
12     }
13     for(int i = 0; i < arr.size(); ++i) {
14         std::cout << arr[i] << ' ';
15     }
16     return 0;
17 }

```

2 图论

2.1 Tarjan 割点

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 //tarjan求割点
5 //https://www.luogu.com.cn/problem/P3388
6 int main() {
7     std::ios::sync_with_stdio(false);
8     std::cin.tie(nullptr);
9     int n, m;
10    std::cin >> n >> m;
11    std::vector<std::vector<int>> v(n + 1);

```

```

12 for(int i = 1; i <= m; ++i) {
13     int x, y;
14     std::cin >> x >> y;
15     v[x].push_back(y);
16     v[y].push_back(x);
17 }
18 std::vector<int> dfn(n + 1), low(n + 1), bel(n + 1), cutPoint(n + 1);
19 int cnt = 0, root = 0;
20 auto dfs = [&](auto self, int id, int lst) ->void {
21     dfn[id] = low[id] = ++cnt;
22     int sz = 0; //儿子个数
23     for(auto nxt : v[id]) {
24         if(!dfn[nxt]) {
25             sz++;
26             self(self, nxt, id);
27             low[id] = std::min(low[id], low[nxt]);
28             if(low[nxt] >= dfn[id]) {
29                 cutPoint[id] = 1;
30             }
31             } else if(nxt != lst) {
32                 low[id] = std::min(low[id], dfn[nxt]);
33             }
34         }
35     if(num <= 1 && id == root) {
36         cutPoint[id] = 0;
37     }
38 };
39 for(int i = 1; i <= n; ++i) {
40     if(!dfn[i]) {
41         root = i;
42         dfs(dfs, i, 0);
43     }
44 }
45 std::cout << std::count(cutPoint.begin() + 1, cutPoint.end(), 1) << '\n';
46 for(int i = 1; i <= n; ++i) {
47     if(cutPoint[i] == 1) {

```

```

48         std::cout << i << ' ';
49     }
50 }
51 return 0;
52 }

```

2.2 Tarjan 割边

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 //tarjan求割边
5 //https://www.luogu.com.cn/problem/P1656
6 int main() {
7     std::ios::sync_with_stdio(false);
8     std::cin.tie(nullptr);
9     int n, m;
10    std::cin >> n >> m;
11    std::vector<std::vector<std::pair<int, int>>> v(n + 1);
12    for(int i = 1; i <= m; ++i) {
13        int x, y;
14        std::cin >> x >> y;
15        v[x].push_back({y, i}); //记录边id(从1开始), 防止重边
16        v[y].push_back({x, i});
17    }
18    std::vector<int> dfn(n + 1), low(n + 1);
19    std::vector<std::pair<int, int>> bridge;
20    int cnt = 0;
21    auto dfs = [&](auto self, int id, int lid) ->void {
22        dfn[id] = low[id] = ++cnt;
23        for(auto [nxt, eid] : v[id]) {
24            if(!dfn[nxt]) {
25                self(self, nxt, eid);
26                low[id] = std::min(low[id], low[nxt]);
27                if(low[nxt] == dfn[nxt]) { //是割边

```

```

28         bridge.push_back({id, nxt});
29     }
30     } else if(eid != lid) {
31         low[id] = std::min(low[id], dfn[nxt]);
32     }
33 }
34 };
35 for(int i = 1; i <= n; ++i) {
36     if(!dfn[i]) {
37         dfs(dfs, i, 0);
38     }
39 }
40 std::sort(bridge.begin(), bridge.end());
41 for(auto [x, y] : bridge) {
42     std::cout << x << ' ' << y << '\n';
43 }
44 return 0;
45 }

```

2.3 Tarjan 强连通分量

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 //tarjan求强连通分量(scc)
5 //https://www.luogu.com.cn/problem/B3609
6 int main() {
7     std::ios::sync_with_stdio(false);
8     std::cin.tie(nullptr);
9     int n, m;
10    std::cin >> n >> m;
11    std::vector<std::vector<int>> v(n + 1);
12    for(int i = 0; i < m; ++i) {
13        int x, y;
14        std::cin >> x >> y;

```

```

15        v[x].push_back(y);
16    }
17    std::vector<std::vector<int>> scc(n + 1);
18    std::vector<int> dfn(n + 1), low(n + 1), ins(n + 1), bel(n + 1);
19    std::stack<int> stk;
20    int cnt = 0, tot = 0;
21    auto dfs = [&](auto self, int id) ->void {
22        dfn[id] = low[id] = ++cnt;
23        stk.push(id);
24        ins[id] = 1;
25        for(auto nxt : v[id]) {
26            if(!dfn[nxt]) {
27                self(self, nxt);
28                low[id] = std::min(low[id], low[nxt]);
29            } else if(ins[nxt]) {
30                low[id] = std::min(low[id], low[nxt]);
31            }
32        }
33        if(dfn[id] == low[id]) {
34            ++tot;
35            while(true) {
36                int num = stk.top();
37                stk.pop();
38                ins[num] = 0;
39                bel[num] = tot;
40                scc[tot].push_back(num);
41                if(id == num) break;
42            }
43        }
44    };
45    for(int i = 1; i <= n; ++i) {
46        if(!dfn[i]) {
47            dfs(dfs, i);
48        }
49    }
50    for(int i = 1; i <= tot; ++i) {

```

```

51     std::sort(scc[i].begin(), scc[i].end());
52 }
53 std::sort(scc.begin() + 1, scc.begin() + tot + 1);
54 std::cout << tot << '\n';
55 for(int i = 1; i <= tot; ++i) {
56     for(int j = 0; j < scc[i].size(); ++j) {
57         std::cout << scc[i][j] << " \n"[j == scc[i].size() - 1];
58     }
59 }
60 return 0;
61 }

```

2.4 Tarjan 点双连通分量

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 //tarjan求点双连通分量
5 //https://www.luogu.com.cn/problem/P8435
6 int main() {
7     std::ios::sync_with_stdio(false);
8     std::cin.tie(nullptr);
9     int n, m;
10    std::cin >> n >> m;
11    std::vector<std::vector<int>> v(n + 1);
12    for(int i = 1; i <= m; ++i) {
13        int x, y;
14        std::cin >> x >> y;
15        v[x].push_back(y);
16        v[y].push_back(x);
17    }
18    std::vector<std::vector<int>> vcc(n + 1);
19    std::vector<int> dfn(n + 1), low(n + 1);
20    std::stack<int> stk;
21    int cnt = 0, tot = 0;

```

```

22    auto dfs = [&](auto self, int id, int lst) ->void {
23        dfn[id] = low[id] = ++cnt;
24        stk.push(id);
25        int num = 0;
26        for(auto nxt : v[id]) {
27            if(!dfn[nxt]) {
28                num++;
29                self(self, nxt, id);
30                low[id] = std::min(low[id], low[nxt]);
31                if(low[nxt] >= dfn[id]) {
32                    ++tot;
33                    while(true) {
34                        int num = stk.top();
35                        stk.pop();
36                        vcc[tot].push_back(num);
37                        if(num == nxt) break;
38                    }
39                    vcc[tot].push_back(id);
40                }
41            } else if(nxt != lst) {
42                low[id] = std::min(low[id], dfn[nxt]);
43            }
44        }
45        if(lst == 0 && num == 0) {
46            ++tot;
47            vcc[tot].push_back(id);
48        }
49    };
50    for(int i = 1; i <= n; ++i) {
51        if(!dfn[i]) {
52            dfs(dfs, i, 0);
53        }
54    }
55    std::cout << tot << '\n';
56    for(int i = 1; i <= tot; ++i) {
57        std::cout << vcc[i].size() << ' ';

```

```

58     for(int j = 0; j < vcc[i].size(); ++j) {
59         std::cout << vcc[i][j] << " \n"[j == vcc[i].size() - 1];
60     }
61 }
62 return 0;
63 }

```

2.5 Tarjan 边双连通分量

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 //tarjan求边双连通分量
5 //https://www.luogu.com.cn/problem/P8436
6 int main() {
7     std::ios::sync_with_stdio(false);
8     std::cin.tie(nullptr);
9     int n, m;
10    std::cin >> n >> m;
11    std::vector<std::vector<std::pair<int, int>>> v(n + 1);
12    for(int i = 1; i <= m; ++i) {
13        int x, y;
14        std::cin >> x >> y;
15        v[x].push_back({y, i});
16        v[y].push_back({x, i});
17    }
18    std::vector<std::vector<int>> ecc(n + 1);
19    std::vector<int> dfn(n + 1), low(n + 1);
20    std::stack<int> stk;
21    int cnt = 0, tot = 0;
22    auto dfs = [&](auto self, int id, int lid) ->void {
23        dfn[id] = low[id] = ++cnt;
24        stk.push(id);
25        for(auto [nxt, eid] : v[id]) {
26            if(!dfn[nxt]) {

```

```

27                self(self, nxt, eid);
28                low[id] = std::min(low[id], low[nxt]);
29            } else if(lid != eid) {
30                low[id] = std::min(low[id], dfn[nxt]);
31            }
32        }
33        if(dfn[id] == low[id]) {
34            ++tot;
35            while(true) {
36                int num = stk.top();
37                ecc[tot].push_back(num);
38                stk.pop();
39                if(id == num) break;
40            }
41        }
42    };
43    for(int i = 1; i <= n; ++i) {
44        if(!dfn[i]) {
45            dfs(dfs, i, 0);
46        }
47    }
48    std::cout << tot << '\n';
49    for(int i = 1; i <= tot; ++i) {
50        std::cout << ecc[i].size() << ' ';
51        for(int j = 0; j < ecc[i].size(); ++j) {
52            std::cout << ecc[i][j] << " \n"[j == ecc[i].size() - 1];
53        }
54    }
55    return 0;
56 }

```

2.6 拓扑排序

```

1 #include <bits/stdc++.h>
2

```

```

3 //拓扑排序
4 //https://www.luogu.com.cn/problem/B3644
5 int main() {
6     std::ios::sync_with_stdio(false);
7     std::cin.tie(nullptr);
8     int n;
9     std::cin >> n;
10    std::vector<std::vector<int>> v(n + 1); //存图
11    std::vector<int> d(n + 1); //统计入度数量
12    for(int i = 1; i <= n; ++i) { //建图
13        int x;
14        while((std::cin >> x) && x != 0) {
15            v[i].push_back(x);
16            d[x]++;
17        }
18    }
19    std::queue<int> q;
20    for(int i = 1; i <= n; ++i) {
21        if(d[i] == 0) {
22            q.push(i); //将入度为0的放入队列
23        }
24    }
25    while(!q.empty()) {
26        int id = q.front();
27        q.pop();
28        std::cout << id << ' ';
29        for(auto &nxt : v[id]) {
30            d[nxt]--; //更新入度数
31            if(d[nxt] == 0) { //将入度为0的放入队列
32                q.push(nxt);
33            }
34        }
35    }
36    return 0;
37 }

```

2.7 最小生成树 kruskal

```

1 #include <bits/stdc++.h>
2
3 //kruskal算法最小生成树(稀疏图)
4 //https://www.luogu.com.cn/problem/P3366
5 class DSU { //维护并查集
6 public:
7     DSU(int n) { //初始构造
8         v.resize(n + 1);
9         std::iota(v.begin(), v.end(), 0);
10    }
11    int find(int x) { //找根
12        return (v[x] == x ? x : (v[x] = find(v[x])));
13    }
14    void uniset(int x, int y) { //合并集合
15        v[find(x)] = find(y);
16    }
17    bool query(int x, int y) { //是否在同一集合
18        return find(x) == find(y);
19    }
20 private:
21     std::vector<int> v;
22 };
23
24 struct edge { //边
25     int x, y, w; //点, 点, 边权
26     bool operator<(const edge& o) const {
27         return w < o.w;
28     }
29 };
30
31 int main() {
32     int n, m;
33     std::cin >> n >> m;
34     std::vector<edge> v(m);

```



```

35 DSU dsu(n);
36 for(auto &[x, y, w] : v) {
37     std::cin >> x >> y >> w;
38 }
39 std::sort(v.begin(), v.end()); //对边排序
40 int ans = 0, tot = 0;
41 for(auto [x, y, w] : v) {
42     if(!dsu.query(x, y)) {
43         dsu.uniset(x, y);
44         ans += w;
45         tot++;
46     }
47 }
48 if(tot != n - 1) {
49     std::cout << "orz" << '\n';
50 } else {
51     std::cout << ans << '\n';
52 }
53 return 0;
54 }

```

2.8 最小生成树 prim

```

1 #include <bits/stdc++.h>
2
3 //prim算法最小生成树(稠密图)
4 //https://www.luogu.com.cn/problem/P3366
5 struct node {
6     int id, w;
7     bool operator<(const node& o) const {
8         return w > o.w;
9     }
10 };
11
12 int main() {

```

```

13 int n, m;
14 std::cin >> n >> m;
15 std::vector<std::vector<std::pair<int, int>>> v(n + 1);
16 std::vector<int> vis(n + 1);
17 for(int i = 0; i < m; ++i) {
18     int x, y, w;
19     std::cin >> x >> y >> w;
20     v[x].push_back({y, w});
21     v[y].push_back({x, w});
22 }
23 std::priority_queue<node> pq; //利用优先队列不断加入最小边
24 int ans = 0;
25 pq.push({1, 0});
26 while(!pq.empty()) {
27     auto [id, w] = pq.top();
28     pq.pop();
29     if(!vis[id]) {
30         vis[id] = 1;
31         ans += w;
32         for(auto [nxt, w] : v[id]) {
33             if(!vis[nxt]) {
34                 pq.push({nxt, w});
35             }
36         }
37     }
38 }
39 if(!std::min_element(vis.begin() + 1, vis.end())) {
40     std::cout << "orz" << '\n'; //图不连通
41 } else {
42     std::cout << ans << '\n';
43 }
44 return 0;
45 }

```

3 数据结构

3.1 Splay

```
1 #include <bits/stdc++.h>
2
3 class SplayTree {
4 public:
5     SplayTree() {
6         tr.push_back(Node());
7         insert(INF);
8         insert(-INF);
9     }
10    void insert(int t) { //插入值为t的数
11        int id = root, fa = 0;
12        while(id && tr[id].val != t) {
13            fa = id;
14            id = tr[id].nxt[t > tr[id].val];
15        }
16        if(id) {
17            tr[id].cnt++;
18        } else {
19            id = ++size;
20            tr[fa].nxt[t > tr[fa].val] = id;
21            tr.push_back(Node(fa, t));
22        }
23        splay(id);
24    }
25    int get_pre(int t) { //查找t的前驱节点
26        find(t);
27        int id = root;
28        if(tr[id].val < t) return id;
29        id = tr[id].nxt[0];
30        while(tr[id].nxt[1]) {
31            id = tr[id].nxt[1];
```

```
32        }
33        splay(id);
34        return id;
35    }
36    int get_suc(int t) { //查找t的后继节点
37        find(t);
38        int id = root;
39        if(tr[id].val > t) return id;
40        id = tr[id].nxt[1];
41        while(tr[id].nxt[0]) {
42            id = tr[id].nxt[0];
43        }
44        splay(id);
45        return id;
46    }
47    void find(int t) { //查找值为t的节点，并将该节点转到根
48        int id = root;
49        while(tr[id].nxt[t > tr[id].val] && t != tr[id].val) {
50            id = tr[id].nxt[t > tr[id].val];
51        }
52        splay(id);
53    }
54    void erase(int t) { //删除值为t的，只删除1个
55        int pre = get_pre(t);
56        int suc = get_suc(t);
57        splay(pre);
58        splay(suc, pre);
59        int tid = tr[suc].nxt[0]; //目标节点
60        if(tr[tid].cnt > 1) {
61            tr[tid].cnt--;
62            splay(tid); //向上更新其他节点
63        } else {
64            tr[suc].nxt[0] = 0;
65            splay(suc); //向上更新其他节点
66        }
67    }
```

```

68 int get_root() {
69     return root;
70 }
71 int get_rank(int t) { //查一个数t的排名
72     insert(t);
73     int res = tr[tr[root].nxt[0]].size;
74     erase(t);
75     return res;
76 }
77 int get_kth(int t) { //查找第k个节点编号
78     t++; //有哨兵，所以++
79     int id = root;
80     while(true) {
81         pushdown(id); //向下传递懒标记
82         const auto &[x, y] = tr[id].nxt;
83         if(tr[x].size + tr[id].cnt < t) {
84             t -= tr[x].size + tr[id].cnt;
85             id = y;
86         } else {
87             if(tr[x].size >= t) {
88                 id = tr[id].nxt[0];
89             } else {
90                 return id;
91             }
92         }
93     }
94 }
95 int get_val(int t) { //查找排名为t的数的数值
96     int id = get_kth(t);
97     splay(id);
98     return tr[id].val;
99 }
100 void reverse(int l, int r) { //反转区间[l, r]
101     l = get_kth(l - 1), r = get_kth(r + 1);
102     splay(l, 0), splay(r, l);
103     tr[tr[r].nxt[0]].tag ^= 1;

```

```

104 }
105 void output(int id) { //中序遍历
106     pushdown(id);
107     const auto &[x, y] = tr[id].nxt;
108     if(x != 0) output(x);
109     if(std::abs(tr[id].val) != INF) {
110         std::cout << tr[id].val << ' ';
111     }
112     if(y) output(y);
113 }
114 int val(int id) {
115     return tr[id].val;
116 }
117 private:
118 class Node {
119 public:
120     Node() {
121         nxt = {0, 0};
122         lst = val = size = cnt = tag = 0;
123     }
124     Node(int _lst, int _val) : lst(_lst), val(_val) {
125         nxt = {0, 0};
126         tag = 0;
127         size = cnt = 1;
128     }
129     std::array<int, 2> nxt; //左右节点[0左, 1右]
130     int lst; //父亲
131     int val; //权值
132     int cnt; //权值数
133     int size; //子树大小
134     int tag; //懒标记[1翻, 0不翻]
135 };
136 void rotate(int id) {
137     int pid = tr[id].lst, gid = tr[pid].lst; //父节点, 爷节点
138     int k = (tr[pid].nxt[1] == id); //判断id是pid的左节点还是右节点
139     tr[pid].nxt[k] = tr[id].nxt[k ^ 1]; //将父节点的k号子节点设置为id的k

```

```

140     tr[tr[id].nxt[k ^ 1]].lst = pid;          //id的k^1号子节点的父节点设为pid
141     tr[id].nxt[k ^ 1] = pid;                //id的k^1号子节点设置为pid
142     tr[pid].lst = id;                      //pid的父节点设置为id
143     tr[id].lst = gid;                      //id的父节点设置为gid
144     tr[gid].nxt[tr[gid].nxt[1] == pid] = id; //gid的子节点设为id
145     pushup(pid);                          //更新pid
146     pushup(id);                          //更新id
147 }
148 void splay(int id, int t = 0) { //将id旋转到为t的子节点, 为0时id为根
149     while(tr[id].lst != t) {
150         int pid = tr[id].lst, gid = tr[pid].lst;
151         if(gid != t) { //非根做双旋
152             if((tr[pid].nxt[0] == id) == (tr[gid].nxt[0] == pid)) { //直线式
153                 rotate(pid);
154             } else { //折线式转中
155                 rotate(id);
156             }
157         }
158         rotate(id);
159     }
160     if(t == 0) root = id;
161 }
162 void pushup(int id) {
163     const auto &x, y] = tr[id].nxt;
164     tr[id].size = tr[x].size + tr[y].size + tr[id].cnt;
165 }
166 void pushdown(int id) {
167     if(tr[id].tag) {
168         auto &x, y] = tr[id].nxt;
169         std::swap(x, y);
170         tr[x].tag ^= 1;
171         tr[y].tag ^= 1;
172         tr[id].tag = 0;
173     }

```

```

174     }
175     std::vector<Node> tr;
176     int root = 0; //根节点编号
177     int size = 0; //节点个数
178     const int INF = INT_MAX;
179 };
180
181 int main() {
182     std::ios::sync_with_stdio(false);
183     std::cin.tie(nullptr);
184     int n, m;
185     std::cin >> n >> m;
186     SplayTree tr;
187     for(int i = 1; i <= n; ++i) {
188         tr.insert(i);
189     }
190     for(int i = 1; i <= m; ++i) {
191         int l, r;
192         std::cin >> l >> r;
193         tr.reverse(l, r);
194     }
195     tr.output(tr.get_root());
196     return 0;
197 }

```

3.2 ST 表

```

1 #include <bits/stdc++.h>
2
3 //ST表(sparseTable)
4 //https://www.luogu.com.cn/problem/P3865
5 template<typename T>
6 class ST { //下标从0开始
7 public:
8     ST(const std::vector<T> &v) { //数据

```

```

9     int k = std::__lg(v.size());
10    st = std::vector<std::vector<T>>(k + 1, std::vector<T>(v.size()));
11    st[0] = v;
12    for(int i = 0; i < k; ++i) {
13        for(int j = 0; j + (1 << (i + 1)) - 1 < v.size(); ++j) {
14            st[i + 1][j] = std::max(st[i][j], st[i][j + (1 << i)]);
15        }
16    }
17 }
18 T query(int l, int r) { //查询[l, r]的最大值
19     int t = std::__lg(r - l + 1);
20     return std::max(st[t][l], st[t][r + 1 - (1 << t)]);
21 }
22 private:
23     std::vector<std::vector<T>> st;
24 };
25
26 int main() {
27     std::ios::sync_with_stdio(false);
28     std::cin.tie(nullptr);
29     int n, q;
30     std::cin >> n >> q;
31     std::vector<int> v(n);
32     for(int i = 0; i < n; ++i) {
33         std::cin >> v[i];
34     }
35     ST<int> st(v);
36     while(q--) {
37         int l, r;
38         std::cin >> l >> r;
39         l--, r--;
40         std::cout << st.query(l, r) << '\n';
41     }
42     return 0;
43 }

```

3.3 对顶堆

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 //对顶堆，维护第k小/大
5 template<typename T>
6 struct DoubleHeap {
7     DoubleHeap(int _k) : k(_k) {} //第k小，若要第k大，将下面比较函数反转
8     std::priority_queue<T, std::vector<T>, std::less<T>> mpq; //大根堆[1, k - 1]
9     std::priority_queue<T, std::vector<T>, std::greater<T>> Mpq; //小根堆[k, sz]
10    void insert(T x) {
11        mpq.push(x);
12        while(mpq.size() >= k) {
13            Mpq.push(mpq.top());
14            mpq.pop();
15        }
16    }
17    T kth() {
18        assert(Mpq.empty() == false);
19        return Mpq.top();
20    }
21    const int k;
22 };
23
24 struct MINT {
25     int x;
26     bool operator<(const MINT &o) const {
27         return x < o.x;
28     }
29     bool operator>(const MINT &o) const {
30         return x > o.x;
31     }
32 };
33
34 void solve() {

```

```

35     int n, k;
36     std::cin >> n >> k;
37     DoubleHeap<MINT> dpq(k);
38     for(int i = 1; i <= n; ++i) {
39         int opt;
40         std::cin >> opt;
41         if(opt == 1) {
42             int x;
43             std::cin >> x;
44             dpq.insert({x});
45         } else {
46             std::cout << dpq.kth().x << '\n';
47         }
48     }
49 }
50
51
52 int main() {
53     std::ios::sync_with_stdio(false);
54     std::cin.tie(nullptr);
55     int T;
56     std::cin >> T;
57     while(T--) {
58         solve();
59     }
60     return 0;
61 }

```

3.4 并查集

```

1 #include <bits/stdc++.h>
2
3 //并查集(disjoint set union)
4 //https://www.luogu.com.cn/problem/P3367
5 class DSU {

```

```

6 public:
7     DSU(int n) { //初始构造
8         v.resize(n + 1);
9         std::iota(v.begin(), v.end(), 0);
10    }
11    int find(int x) { //找根
12        return (v[x] == x ? x : (v[x] = find(v[x])));
13    }
14    void uniset(int x, int y) { //合并集合
15        v[find(x)] = find(y);
16    }
17    bool query(int x, int y) { //是否在同一集合
18        return find(x) == find(y);
19    }
20 private:
21     std::vector<int> v;
22 };
23
24 int main() {
25     std::ios::sync_with_stdio(false);
26     std::cin.tie(nullptr);
27     int n, m;
28     std::cin >> n >> m;
29     DSU dsu(n);
30     for(int i = 0; i < m; ++i) {
31         int z, x, y;
32         std::cin >> z >> x >> y;
33         if(z == 1) {
34             dsu.uniset(x, y);
35         } else if(z == 2) {
36             std::cout << (dsu.query(x, y) ? 'Y' : 'N') << '\n';
37         }
38     }
39     return 0;
40 }

```

3.5 树状数组

```
1 #include<bits/stdc++.h>
2
3 //树状数组(Fenwick)
4 //https://www.luogu.com.cn/problem/P3374
5 template<typename T>
6 class Fenwick {
7 public:
8     Fenwick(int n) : v(std::vector<T>(n + 1)) {}; //有参构造
9     void update(int x, T dx) { //更新(index, dx)
10         for(int i = x; i < v.size(); i += (i & -i)) {
11             v[i] += dx;
12         }
13     }
14     T query(int x) { //查询前缀和[0, L]
15         T res{};
16         for(int i = x; i > 0; i -= (i & -i)) {
17             res += v[i];
18         }
19         return res;
20     }
21     T range(int l, int r) { //查询区间[L, R]
22         return query(r) - query(l - 1);
23     }
24 private:
25     std::vector<T> v;
26 };
27
28 int main() {
29     std::ios::sync_with_stdio(false);
30     std::cin.tie(nullptr);
31     int n, m;
32     std::cin >> n >> m;
33     Fenwick<int> tr(n);
34     for(int i = 1; i <= n; ++i) {
```

```
35         int x;
36         std::cin >> x;
37         tr.update(i, x);
38     }
39     for(int i = 0; i < m; ++i) {
40         int o, x, y;
41         std::cin >> o >> x >> y;
42         if(o == 1) {
43             tr.update(x, y);
44         } else if (o == 2) {
45             std::cout << tr.range(x, y) << '\n';
46         }
47     }
48     return 0;
49 };
```

3.6 线段树

```
1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 //线段树，区间修改，区间查询
5 //https://www.luogu.com.cn/problem/P3372
6 template<typename Info, typename Tag>
7 struct SegmentTree {
8     #define ls (id<<1)
9     #define rs (id<<1|1)
10     SegmentTree(int n) : n(n), info(n << 2), tag(n << 2) {}
11     SegmentTree(const std::vector<Info> &init) : SegmentTree(init.size()) {
12         auto build = [&](auto self, int id, int l, int r) ->void {
13             if(l == r) {
14                 info[id] = init[l];
15                 return;
16             }
17             int mid = (l + r) / 2;
```

```

18         self(self, ls, l, mid);
19         self(self, rs, mid + 1, r);
20         pushup(id);
21     };
22     build(build, 1, 1, n);
23 }
24 void apply(int id, const Tag &dx) {
25     info[id].apply(dx);
26     tag[id].apply(dx);
27 }
28 void pushup(int id) {
29     info[id] = info[ls] + info[rs];
30 }
31 void pushdown(int id) {
32     apply(ls, tag[id]);
33     apply(rs, tag[id]);
34     tag[id] = Tag();
35 }
36 void update(int t, const Info &val) {
37     update(1, 1, n, t, val);
38 }
39 void rangeUpdate(int l, int r, const Tag &dx) {
40     rangeUpdate(1, 1, n, l, r, dx);
41 }
42 Info rangeQuery(int l, int r) {
43     return rangeQuery(1, 1, n, l, r);
44 }
45 void update(int id, int l, int r, int t, const Info &val) {
46     if(l == r) {
47         info[id] = val;
48         return;
49     }
50     int mid = (l + r) / 2;
51     pushdown(id);
52     if(t <= mid) {
53         update(ls, l, mid, t, val);

```

```

54     } else if(t > mid) {
55         update(rs, mid + 1, r, t, val);
56     }
57     pushup(id);
58 }
59 void rangeUpdate(int id, int l, int r, int x, int y, const Tag &dx) {
60     if(x <= l && r <= y) {
61         apply(id, dx);
62         return;
63     }
64     int mid = (l + r) / 2;
65     pushdown(id);
66     if(x <= mid) {
67         rangeUpdate(ls, l, mid, x, y, dx);
68     }
69     if(y > mid) {
70         rangeUpdate(rs, mid + 1, r, x, y, dx);
71     }
72     pushup(id);
73 }
74 Info rangeQuery(int id, int l, int r, int x, int y) {
75     if(x <= l && r <= y) {
76         return info[id];
77     }
78     int mid = (l + r) / 2;
79     pushdown(id);
80     Info res;
81     if(x <= mid) {
82         res = res + rangeQuery(ls, l, mid, x, y);
83     }
84     if(y > mid) {
85         res = res + rangeQuery(rs, mid + 1, r, x, y);
86     }
87     return res;
88 }
89 #undef ls

```



```

90 #undef rs
91     const int n;
92     std::vector<Info> info;
93     std::vector<Tag> tag;
94 };
95
96 constexpr i64 INF = 1E18;
97
98 struct Tag {
99     i64 add = 0;
100     void apply(const Tag &dx) {
101         add += dx.add;
102     }
103 };
104
105 struct Info {
106     i64 mn = INF;
107     i64 mx = -INF;
108     i64 sum = 0;
109     i64 len = 0;
110     void apply(const Tag &dx) {
111         mn += dx.add;
112         mx += dx.add;
113         sum += len * dx.add;
114     }
115 };
116
117 Info operator+(const Info &x, const Info &y) {
118     Info res;
119     res.mn = std::min(x.mn, y.mn);
120     res.mx = std::max(x.mx, y.mx);
121     res.sum = x.sum + y.sum;
122     res.len = x.len + y.len;
123     return res;
124 }
125

```

```

126 int main() {
127     std::ios::sync_with_stdio(false);
128     std::cin.tie(nullptr);
129     int n, m;
130     std::cin >> n >> m;
131     // std::vector<Info> v(n + 1);
132     // for(int i = 1; i <= n; ++i) {
133     //     int x;
134     //     std::cin >> x;
135     //     v[i] = {x, x, x, 1};
136     // }
137     // SegmentTree<Info, Tag> tr(v);
138     SegmentTree<Info, Tag> tr(n);
139     for(int i = 1; i <= n; ++i) {
140         int x;
141         std::cin >> x;
142         tr.update(i, {x, x, x, 1});
143     }
144     while(m--) {
145         int opt, x, y;
146         std::cin >> opt >> x >> y;
147         if(opt == 1) {
148             int k;
149             std::cin >> k;
150             tr.rangeUpdate(x, y, {k});
151         } else if(opt == 2) {
152             std::cout << tr.rangeQuery(x, y).sum << '\n';
153         }
154     }
155     return 0;
156 }

```

4 树算法

4.1 树剖 LCA

```
1 #include <bits/stdc++.h>
2
3 //树链剖分求LCA
4 //https://www.luogu.com.cn/problem/P3379
5 int main() {
6     std::ios::sync_with_stdio(0);
7     std::cin.tie(nullptr);
8     int n, m, s;
9     std::cin >> n >> m >> s;
10    std::vector<std::vector<int>> v(n + 1);
11    std::vector<int> fa(n + 1), dep(n + 1), son(n + 1), sz(n + 1), top(n + 1, 0);
12    //父节点, 深度, 重儿子, 子树节点数, 所在重链的顶点
13    for(int i = 0; i < n - 1; ++i) {
14        int x, y;
15        std::cin >> x >> y;
16        v[x].push_back(y);
17        v[y].push_back(x);
18    }
19    auto dfs1 = [&](auto self, int id, int lst) ->void { //求fa, dep, son, sz数组
20        fa[id] = lst;
21        dep[id] = dep[lst] + 1;
22        sz[id] = 1;
23        for(auto nxt : v[id]) {
24            if(nxt == lst) continue;
25            self(self, nxt, id);
26            sz[id] += sz[nxt];
27            if(sz[son[id]] < sz[nxt]) {
28                son[id] = nxt;
29            }
30        }
31    };
```

```
32    auto dfs2 = [&](auto self, int id, int t) ->void {
33        top[id] = t;
34        if(son[id] == 0) return;
35        self(self, son[id], t);
36        for(auto nxt : v[id]) {
37            if(nxt != fa[id] && nxt != son[id]) {
38                self(self, nxt, t);
39            }
40        }
41    };
42    auto lca = [&](int x, int y) ->int {
43        while(top[x] != top[y]) {
44            if(dep[top[x]] < dep[top[y]]) {
45                std::swap(x, y);
46            }
47            x = fa[top[x]];
48        }
49        return (dep[x] < dep[y] ? x : y);
50    };
51    dfs1(dfs1, s, 0);
52    dfs2(dfs2, s, s);
53    for(int i = 0; i < m; ++i) {
54        int x, y;
55        std::cin >> x >> y;
56        std::cout << lca(x, y) << '\n';
57    }
58    return 0;
59 }
```

5 数论

5.1 欧拉筛

```

1 #include <bits/stdc++.h>
2
3 int main() {
4     std::ios::sync_with_stdio(false);
5     std::cin.tie(nullptr);
6     int n;
7     std::cin >> n;
8     std::vector<bool> isPrime(n + 1, 1);
9     std::vector<int> res = {2}; //存放质数
10    isPrime[0] = 0;
11    for (int i = 3; i <= n; i += 2) {
12        if (isPrime[i]) { //如果是素数, 则记录
13            res.push_back(i);
14        }
15        for (int j = 0; res[j] * i <= n && j < res.size(); ++j) {
16            isPrime[res[j] * i] = 0; //找出素数的倍数, 标记为合数
17            if (i % res[j] == 0) break;
18        }
19    }
20    std::cout << res.size() << '\n';
21    for(auto x : res) {
22        std::cout << x << ' ';
23    }
24    return 0;
25 }

```

6 字符串

6.1 EXKMP

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3

```

```

4 int main() {
5     std::ios::sync_with_stdio(false);
6     std::cin.tie(nullptr);
7     std::string a, b;
8     std::cin >> a >> b;
9     int n = a.size(), m = b.size();
10    a = '#' + a, b = '#' + b;
11    std::vector<int> z(m + 1), p(n + 1);
12    z[1] = m;
13    for(int i = 2, l = 0, r = 0; i <= m; ++i) {
14        if(i <= r) {
15            z[i] = std::min(z[i - l + 1], r - i + 1);
16        }
17        while(i + z[i] <= m && b[i + z[i]] == b[1 + z[i]]) {
18            z[i]++;
19        }
20        if(i + z[i] - 1 > r) {
21            l = i, r = i + z[i] - 1;
22        }
23    }
24    for(int i = 1, l = 0, r = 0; i <= n; ++i) {
25        if(i <= r) {
26            p[i] = std::min(z[i - l + 1], r - i + 1);
27        }
28        while(1 + p[i] <= m && i + p[i] <= n && b[1 + p[i]] == a[i + p[i]]) {
29            p[i]++;
30        }
31        if(i + p[i] - 1 > r) {
32            l = i, r = i + p[i] - 1;
33        }
34    }
35    i64 ans1 = 0, ans2 = 0;
36    for(int i = 1; i <= m; ++i) {
37        ans1 ^= 1LL * i * (z[i] + 1);
38    }
39    for(int i = 1; i <= n; ++i) {

```

```

40     ans2 ^= 1LL * i * (p[i] + 1);
41 }
42 std::cout << ans1 << '\n' << ans2 << '\n';
43 return 0;
44 }

```

6.2 KMP

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 int main() {
5     std::ios::sync_with_stdio(false);
6     std::cin.tie(nullptr);
7     std::string s, p;
8     std::cin >> s >> p;
9     int n = s.size(), m = p.size();
10    s = '#' + s, p = '#' + p;
11    std::vector<int> kmp(m + 1);
12    for(int i = 2, j = 0; i <= m; ++i) { //求kmp数组
13        while(j > 0 && p[i] != p[j + 1]) {
14            j = kmp[j];
15        }
16        if(p[j + 1] == p[i]) {
17            j++;
18        }
19        kmp[i] = j;
20    }
21    for(int i = 1, j = 0; i <= n; ++i) {
22        while(j > 0 && s[i] != p[j + 1]) {
23            j = kmp[j];
24        }
25        if(s[i] == p[j + 1]) {
26            j++;
27        }

```

```

28        if(j == m) {
29            std::cout << i - j + 1 << '\n';
30            j = kmp[j];
31        }
32    }
33    for(int i = 1; i <= m; ++i) {
34        std::cout << kmp[i] << " \n"[i == m];
35    }
36    return 0;
37 }

```

6.3 字符串哈希

```

1 #include <bits/stdc++.h>
2 using i64 = long long;
3
4 //字符串hash
5 //https://www.luogu.com.cn/problem/P3370
6 struct Hash {
7     std::vector<i64> h1, p1, h2, p2;
8     const i64 base1 = 31, base2 = 37;
9     const i64 mod1 = 2013265921, mod2 = 1004535809;
10    Hash(const std::string &s) //0-index
11    : n(s.size()), h1(s.size() + 1), h2(s.size() + 1), p1(s.size() + 1), p2(s.
12      size() + 1) {
13        p1[0] = p2[0] = 1;
14        for (i64 i = 1; i <= n; i++) {
15            p1[i] = p1[i - 1] * base1 % mod1;
16            p2[i] = p2[i - 1] * base2 % mod2;
17        }
18        for (i64 i = 1; i <= n; i++) {
19            h1[i] = (h1[i - 1] * base1 % mod1 + s[i - 1]) % mod1;
20            h2[i] = (h2[i - 1] * base2 % mod2 + s[i - 1]) % mod2;
21        }
22    }

```

```

22     std::pair<i64, i64> get(int l, int r) { //1-index
23         i64 hash1 = (h1[r] - h1[l - 1] * p1[r - l + 1] % mod1 + mod1) % mod1;
24         i64 hash2 = (h2[r] - h2[l - 1] * p2[r - l + 1] % mod2 + mod2) % mod2;
25         return {hash1, hash2};
26     }
27     int n;
28 };
29
30 int main() {
31     std::ios::sync_with_stdio(false);
32     std::cin.tie(nullptr);
33     int n;
34     std::cin >> n;
35     std::set<std::pair<i64, i64>> st;
36     for(int i = 0; i < n; ++i) {
37         std::string s;
38         std::cin >> s;
39         Hash hs(s);
40         st.insert(hs.get(1, s.size()));
41     }
42     std::cout << st.size() << '\n';
43     return 0;
44 }

```

6.4 马拉车

```

1 #include <bits/stdc++.h>
2
3 //马拉车(manacher)
4 //https://www.luogu.com.cn/problem/P3805
5
6 // 以第i个数为轴的最大回文 v[2 * i + 1]
7 // 以第i个数和i+1个数中间为轴的最大回文 v[2 * i + 2]
8 // 以[L, R] 区间中轴的最大回文为v[L + R + 1]
9 std::vector<int> manacher(const std::string& s) {

```

```

10     int n = 2 * s.length() + 1;
11     std::string t(n, '#'); //处理字符串
12     for(int i = 0; i < s.length(); ++i) {
13         t[2 * i + 1] = s[i];
14     }
15     std::vector<int> v(n); //记录回文半径 [l, r] <=> [mid - v[mid], mid + v[mid]]
16     for(int i = 0, mid = 0; i < n; ++i) { // mid为回文中心
17         if(i <= mid + v[mid]) {
18             v[i] = std::min(v[2 * mid - i], mid + v[mid] - i); // (t + i) / 2 =
19             mid <=> t = 2 * mid - i;
20         }
21         while(t[i - v[i] - 1] == t[i + v[i] + 1] && 0 <= i - v[i] - 1 && i + v[i]
22             + 1 < n) {
23             ++v[i];
24         }
25         if(i + v[i] > mid + v[mid]) {
26             mid = i;
27         }
28     }
29     return v;
30 }
31
32 int main() {
33     std::ios::sync_with_stdio(false);
34     std::cin.tie(nullptr);
35     std::string s;
36     std::cin >> s;
37     std::vector<int> v = manacher(s);
38     int ans = 0;
39     for(int i = 0; i < v.size(); ++i) {
40         ans = std::max(ans, v[i]); //求最长回文子串
41         std::cout << v[i] << " \n"[i == v.size() - 1];
42     }
43     std::cout << ans << '\n';
44     return 0;
45 }

```