目录

1 图论

1.1 Tarjan 割点

```cpp
#include <bits/stdc++.h>
using i64 = long long;

//tarjan求割点
//https://www.luogu.com.cn/problem/P3388
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    std::vector<std::vector<int>> v(n + 1);
    for(int i = 1; i <= m; ++i) {
        int x, y;
        std::cin >> x >> y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    std::vector<int> dfn(n + 1), low(n + 1), bel(n + 1), cutPoint(n + 1);
    int cnt = 0, root = 0;
    auto dfs = [&](auto self, int id, int lst) ->void {
        dfn[id] = low[id] = ++cnt;
        int sz = 0; //儿子个数
        for(auto nxt : v[id]) {
            if(!dfn[nxt]) {
                sz++;
                self(self, nxt, id);
                low[id] = std::min(low[id], low[nxt]);
                if(low[nxt] >= dfn[id]) {
                    cutPoint[id] = 1;
                }
            } else if(nxt != lst) {
                low[id] = std::min(low[id], dfn[nxt]);
            }
        }
        if(num <= 1 && id == root) {
            cutPoint[id] = 0;
        }
    };
    for(int i = 1; i <= n; ++i) {
        if(!dfn[i]) {
            root = i;
            dfs(dfs, i, 0);
        }
    }
    std::cout << std::count(cutPoint.begin() + 1, cutPoint.end(), 1) << '\n';
    for(int i = 1; i <= n; ++i) {
        if(cutPoint[i] == 1) {
            std::cout << i << ' ';
        }
    }
    return 0;
}
```

1.2 Tarjan 割边

```cpp
#include <bits/stdc++.h>
using i64 = long long;

//tarjan求割边
//https://www.luogu.com.cn/problem/P1656
int main() {
␣␣␣␣std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    std::vector<std::vector<std::pair<int, int>>> v(n + 1);
    for(int i = 1; i <= m; ++i) {
        int x, y;
        std::cin >> x >> y;
        v[x].push_back({y, i});//记录边id(从1开始)，防止重边
        v[y].push_back({x, i});
    }
    std::vector<int> dfn(n + 1), low(n + 1);
    std::vector<std::pair<int, int>> bridge;
    int cnt = 0;
    auto dfs = [&](auto self, int id, int lid) ->void {
        dfn[id] = low[id] = ++cnt;
        for(auto [nxt, eid] : v[id]) {
            if(!dfn[nxt]) {
                self(self, nxt, eid);
                low[id] = std::min(low[id], low[nxt]);
                if(low[nxt] == dfn[nxt]) { //是割边
                    bridge.push_back({id, nxt});
                }
            } else if(eid != lid) {
                low[id] = std::min(low[id], dfn[nxt]);
            }
        }
    };
    for(int i = 1; i <= n; ++i) {
        if(!dfn[i]) {
            dfs(dfs, i, 0);
        }
    }
    std::sort(bridge.begin(), bridge.end());
    for(auto [x, y] : bridge) {
        std::cout << x << ' ' << y << '\n';
    }
    return 0;
}
```

1.3 Tarjan 强连通分量

```cpp
#include <bits/stdc++.h>
using i64 = long long;

//tarjan求强连通分量(scc)
//https://www.luogu.com.cn/problem/B3609
int main() {
␣␣␣␣std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    std::vector<std::vector<int>> v(n + 1);
    for(int i = 0; i < m; ++i) {
        int x, y;
        std::cin >> x >> y;
        v[x].push_back(y);
    }
    std::vector<std::vector<int>> scc(n + 1);
    std::vector<int> dfn(n + 1), low(n + 1), ins(n + 1), bel(n + 1);
    std::stack<int> stk;
    int cnt = 0, tot = 0;
    auto dfs = [&](auto self, int id) ->void {
        dfn[id] = low[id] = ++cnt;
        stk.push(id);
        ins[id] = 1;
        for(auto nxt : v[id]) {
            if(!dfn[nxt]) {
                self(self, nxt);
                low[id] = std::min(low[id], low[nxt]);
            } else if(ins[nxt]) {
                low[id] = std::min(low[id], low[nxt]);
            }
        }
        if(dfn[id] == low[id]) {
            ++tot;
            while(true) {
                int num = stk.top();
                stk.pop();
                ins[num] = 0;
                bel[num] = tot;
                scc[tot].push_back(num);
                if(id == num) break;
            }
        }
    };
    for(int i = 1; i <= n; ++i) {
        if(!dfn[i]) {
            dfs(dfs, i);
        }
    }
    for(int i = 1; i <= tot; ++i) {
        std::sort(scc[i].begin(), scc[i].end());
    }
    std::sort(scc.begin() + 1, scc.begin() + tot + 1);
    std::cout << tot << '\n';
    for(int i = 1; i <= tot; ++i) {
        for(int j = 0; j < scc[i].size(); ++j) {
            std::cout << scc[i][j] << " \n"[j == scc[i].size() - 1];
        }
    }
    return 0;
}
```

1.4 Tarjan 点双连通分量

```
#include <bits/stdc++.h>
using i64 = long long;

//tarjan求点双连通分量
//https://www.luogu.com.cn/problem/P8435
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    std::vector<std::vector<int>> v(n + 1);
    for(int i = 1; i <= m; ++i) {
        int x, y;
        std::cin >> x >> y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    std::vector<std::vector<int>> vcc(n + 1);
    std::vector<int> dfn(n + 1), low(n + 1);
    std::stack<int> stk;
    int cnt = 0, tot = 0;
    auto dfs = [&](auto self, int id, int lst) ->void {
        dfn[id] = low[id] = ++cnt;
        stk.push(id);
        int num = 0;
        for(auto nxt : v[id]) {
            if(!dfn[nxt]) {
                num++;
                self(self, nxt, id);
                low[id] = std::min(low[id], low[nxt]);
                if(low[nxt] >= dfn[id]) {
                    ++tot;
                    while(true) {
                        int num = stk.top();
                        stk.pop();
                        vcc[tot].push_back(num);
                        if(num == nxt) break;
                    }
                    vcc[tot].push_back(id);
                }
            } else if(nxt != lst) {
```

```cpp
                low[id] = std::min(low[id], dfn[nxt]);
            }
        }
        if(lst == 0 && num == 0) {
            ++tot;
            vcc[tot].push_back(id);
        }
    };
    for(int i = 1; i <= n; ++i) {
        if(!dfn[i]) {
            dfs(dfs, i, 0);
        }
    }
    std::cout << tot << '\n';
    for(int i = 1; i <= tot; ++i) {
        std::cout << vcc[i].size() << ' ';
        for(int j = 0; j < vcc[i].size(); ++j) {
            std::cout << vcc[i][j] << " \n"[j == vcc[i].size() - 1];
        }
    }
    return 0;
}
```

1.5 Tarjan 边双连通分量

```cpp
#include <bits/stdc++.h>
using i64 = long long;

//tarjan求边双连通分量
//https://www.luogu.com.cn/problem/P8436
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    std::vector<std::vector<std::pair<int, int>>> v(n + 1);
    for(int i = 1; i <= m; ++i) {
        int x, y;
        std::cin >> x >> y;
        v[x].push_back({y, i});
        v[y].push_back({x, i});
    }
    std::vector<std::vector<int>> ecc(n + 1);
    std::vector<int> dfn(n + 1), low(n + 1);
    std::stack<int> stk;
    int cnt = 0, tot = 0;
    auto dfs = [&](auto self, int id, int lid) ->void {
        dfn[id] = low[id] = ++cnt;
        stk.push(id);
        for(auto [nxt, eid] : v[id]) {
            if(!dfn[nxt]) {
                self(self, nxt, eid);
                low[id] = std::min(low[id], low[nxt]);
            } else if(lid != eid) {
                low[id] = std::min(low[id], dfn[nxt]);
            }
        }
        if(dfn[id] == low[id]) {
            ++tot;
            while(true) {
                int num = stk.top();
                ecc[tot].push_back(num);
                stk.pop();
                if(id == num) break;
            }
        }
    };
    for(int i = 1; i <= n; ++i) {
        if(!dfn[i]) {
            dfs(dfs, i, 0);
        }
    }
    std::cout << tot << '\n';
    for(int i = 1; i <= tot; ++i) {
        std::cout << ecc[i].size() << ' ';
        for(int j = 0; j < ecc[i].size(); ++j) {
            std::cout << ecc[i][j] << " \n"[j == ecc[i].size() - 1];
        }
    }
    return 0;
}
```

1.6 拓扑排序

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

//拓扑排序
//https://www.luogu.com.cn/problem/B3644
int main() {
␣␣␣␣std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n;
    std::cin >> n;
    std::vector<std::vector<int>> v(n + 1); //存图
    std::vector<int> d(n + 1); //统计入度数量
    for(int i = 1; i <= n; ++i) { //建图
        int x;
        while((std::cin >> x) && x != 0) {
            v[i].push_back(x);
            d[x]++;
        }
    }
    std::queue<int> q;
    for(int i = 1; i <= n; ++i) {
        if(d[i] == 0) {
            q.push(i); //将入度为0的放入队列
        }
    }
    while(!q.empty()) {
        int id = q.front();
        q.pop();
        std::cout << id << ' ';
        for(auto &nxt  : v[id]) {
            d[nxt]--; //更新入度数
            if(d[nxt] == 0) { //将入度为0的放入队列
                q.push(nxt);
            }
        }
    }
    return 0;
}
```

1.7 最小生成树 kruskal

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

//kruskal算法最小生成树(稀疏图)
//https://www.luogu.com.cn/problem/P3366
class DSU { //维护并查集
public:
    DSU(int n) { //初始构造
        v.resize(n + 1);
        std::iota(v.begin(), v.end(), 0);
    }
    int find(int x) { //找根
        return (v[x] == x ? x : (v[x] = find(v[x])));
    }
    void uniset(int x, int y) { //合并集合
        v[find(x)] = find(y);
    }
    bool query(int x, int y) { //是否在同一集合
```

```cpp
        return find(x) == find(y);
    }
private:
    std::vector<int> v;
};

struct edge { //边
    int x, y, w; //点, 点, 边权
    bool operator<(const edge& o) const {
        return w < o.w;
    }
};

int main() {
    int n, m;
    std::cin >> n >> m;
    std::vector<edge> v(m);
    DSU dsu(n);
    for(auto &[x, y, w] : v) {
        std::cin >> x >> y >> w;
    }
    std::sort(v.begin(), v.end()); //对边排序
    int ans = 0, tot = 0;
    for(auto [x, y, w] : v) {
        if(!dsu.query(x, y)) {
            dsu.uniset(x, y);
            ans += w;
            tot++;
        }
    }
    if(tot != n - 1) {
        std::cout << "orz" << '\n';
    } else {
        std::cout << ans << '\n';
    }
    return 0;
}
```

1.8 最小生成树 prim

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

//prim算法最小生成树(稠密图)
//https://www.luogu.com.cn/problem/P3366
struct node {
    int id, w;
    bool operator<(const node& o) const {
        return w > o.w;
    }
};

int main() {
    int n, m;
    std::cin >> n >> m;
    std::vector<std::vector<std::pair<int, int>>> v(n + 1);
    std::vector<int> vis(n + 1);
    for(int i = 0; i < m; ++i) {
        int x, y, w;
        std::cin >> x >> y >> w;
        v[x].push_back({y, w});
        v[y].push_back({x, w});
    }
    std::priority_queue<node> pq; //利用优先队列不断加入最小边
    int ans = 0;
    pq.push({1, 0});
    while(!pq.empty()) {
        auto [id, w] = pq.top();
        pq.pop();
        if(!vis[id]) {
            vis[id] = 1;
            ans += w;
            for(auto [nxt, w] : v[id]) {
                if(!vis[nxt]) {
                    pq.push({nxt, w});
                }
            }
        }
    }
    if(!*std::min_element(vis.begin() + 1, vis.end())) {
        std::cout << "orz" << '\n'; //图不连通
    } else {
        std::cout << ans << '\n';
    }
    return 0;
}
```

2 基础算法

2.1 三分

```
#include <bits/stdc++.h>
constexpr double eps = 1E-6;//eps控制精度

//三分（实数范围）凸函数
//https://www.luogu.com.cn/record/160695683
int main() {
␣␣␣␣std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n;
    double l, r;
    std::cin >> n >> l >> r;
    std::vector<double> v(n + 1);
    for(int i = n; i >= 0; --i) {
        std::cin >> v[i];
    }
    auto check = [&](double t) ->double {
        double ans = 0;
        for(int i = 0; i <= n; ++i) {
            ans += v[i] * std::pow(t, i);
        }
        return ans;
    };
    while(l + eps <= r) {
        double lmid = l + (r - l) / 3;//左三分点
        double rmid = r - (r - l) / 3;//右三分点
        if(check(lmid) < check(rmid)) {
            l = lmid;
        } else {
            r = rmid;
        }
    }
    std::cout << l << '\n';
    return 0;
}
```

2.2 二分

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

//二分查找
//https://www.luogu.com.cn/record/160694930
int binaryFind(std::vector<int> &v, int t) {
    int l = 1, r = v.size() - 1, ans = -1;
    while(l <= r) {
        int mid = l + (r - l) / 2;
        if(v[mid] >= t) { //此处可换成check函数
            r = mid - 1;
            if(v[mid] == t) { //判断什么时候更新答案
                ans = mid;
            }
```

```cpp
        } else {
            l = mid + 1;
        }
    }
    return ans;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    std::vector<int> v(n + 1);
    for(int i = 1; i <= n; ++i) {
        std::cin >> v[i];
    }
    for(int i = 1; i <= m; ++i) {
        int x;
        std::cin >> x;
        std::cout << binaryFind(v, x) << " \n"[i == m];
    }
    return 0;
}
```

2.3 快速幂

```cpp
#include <bits/stdc++.h>
using i64 = long long;

//快速幂（Binary Exponentiation)
i64 qpow(i64 a, i64 b, i64 p = LLONG_MAX) { //底数，指数，模数
    i64 res = 1;
    while(b > 0) {
        if(b & 1) res = res * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return res;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    std::cout << qpow(2, 20) << '\n';
    std::cout << std::pow(2, 20) << '\n';
    return 0;
}
```

2.4 离散化

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

//离散化
int main() {
    std::vector<int> arr = {1000, 500, 9999, 200, 356, 200};
    std::vector<int> tmp(arr);
    std::sort(tmp.begin(), tmp.end()); //排序
    tmp.erase(std::unique(tmp.begin(), tmp.end()), tmp.end());//去重
    for (int i = 0; i < arr.size(); ++i) { //替换
        arr[i] = std::lower_bound(tmp.begin(), tmp.end(), arr[i]) - tmp.begin() + 1;
    }
    for(int i= 0; i < arr.size(); ++i) {
        std::cout << arr[i] << ' ';
    }
    return 0;
}
```

3 字符串

3.1 EXKMP

```cpp
#include <bits/stdc++.h>
using i64 = long long;

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    std::string a, b;
    std::cin >> a >> b;
    int n = a.size(), m = b.size();
    a = '#' + a, b = '#' + b;
    std::vector<int> z(m + 1), p(n + 1);
    z[1] = m;
    for(int i = 2, l = 0, r = 0; i <= m; ++i) {
        if(i <= r) {
            z[i] = std::min(z[i - l + 1], r - i + 1);
        }
        while(i + z[i] <= m && b[i + z[i]] == b[1 + z[i]]) {
            z[i]++;
        }
        if(i + z[i] - 1 > r) {
            l = i, r = i + z[i] - 1;
        }
    }
    for(int i = 1, l = 0, r = 0; i <= n; ++i) {
        if(i <= r) {
            p[i] = std::min(z[i - l + 1], r - i + 1);
        }
        while(1 + p[i] <= m && i + p[i] <= n && b[1 + p[i]] == a[i + p[i]]) {
            p[i]++;
        }
        if(i + p[i] - 1 > r) {
            l = i, r = i + p[i] - 1;
        }
    }
    i64 ans1 = 0, ans2 = 0;
    for(int i = 1; i <= m; ++i) {
        ans1 ^= 1LL * i * (z[i] + 1);
    }
    for(int i = 1; i <= n; ++i) {
        ans2 ^= 1LL * i * (p[i] + 1);
    }
    std::cout << ans1 << '\n' << ans2 << '\n';
    return 0;
}
```

3.2 KMP

```cpp
#include <bits/stdc++.h>
using i64 = long long;

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    std::string s, p;
    std::cin >> s >> p;
    int n = s.size(), m = p.size();
    s = '#' + s, p = '#' + p;
    std::vector<int> kmp(m + 1);
    for(int i = 2, j = 0; i <= m; ++i) { //求kmp数组
        while(j > 0 && p[i] != p[j + 1]) {
            j = kmp[j];
        }
        if(p[j + 1] == p[i]) {
            j++;
        }
        kmp[i] = j;
    }
    for(int i = 1, j = 0; i <= n; ++i) {
        while(j > 0 && s[i] != p[j + 1]) {
            j = kmp[j];
        }
        if(s[i] == p[j + 1]) {
            j++;
        }
        if(j == m) {
            std::cout << i - j + 1 << '\n';
            j = kmp[j];
        }
    }
    for(int i = 1; i <= m; ++i) {
        std::cout << kmp[i] << " \n"[i == m];
    }
    return 0;
}
```

3.3 字符串哈希

```cpp
#include <bits/stdc++.h>
using i64 = long long;

//字符串hash
//https://www.luogu.com.cn/problem/P3370
struct Hash {
    std::vector<i64> h1, p1, h2, p2;
    const i64 base1 = 31, base2 = 37;
    const i64 mod1 = 2013265921, mod2 = 1004535809;
    Hash(const std::string &s) //0-index
    : n(s.size()), h1(s.size() + 1), h2(s.size() + 1), p1(s.size() + 1), p2(s.
     size() + 1) {
        p1[0] = p2[0] = 1;
        for (i64 i = 1; i <= n; i++) {
            p1[i] = p1[i - 1] * base1 % mod1;
            p2[i] = p2[i - 1] * base2 % mod2;
        }
        for (i64 i = 1; i <= n; i++) {
            h1[i] = (h1[i - 1] * base1 % mod1 + s[i - 1]) % mod1;
            h2[i] = (h2[i - 1] * base2 % mod2 + s[i - 1]) % mod2;
        }
    }
    std::pair<i64, i64> get(int l, int r) {//1-index
        i64 hash1 = (h1[r] - h1[l - 1] * p1[r - l + 1] % mod1 + mod1) % mod1;
        i64 hash2 = (h2[r] - h2[l - 1] * p2[r - l + 1] % mod2 + mod2) % mod2;
        return {hash1, hash2};
    }
    int n;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n;
    std::cin >> n;
    std::set<std::pair<i64, i64>> st;
    for(int i = 0; i < n; ++i) {
        std::string s;
        std::cin >> s;
        Hash hs(s);
        st.insert(hs.get(1, s.size()));
    }
    std::cout << st.size() << '\n';
    return 0;
}
```

3.4 马拉车

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

```cpp

//马拉车(manacher)
//https://www.luogu.com.cn/problem/P3805

// 以第i个数为轴的最大回文 v[2 * i + 1]
// 以第i个数和i+1个数中间为轴的最大回文 v[2 * i + 2]
// 以[L, R] 区间中轴的最大回文为v[L + R + 1]
std::vector<int> manacher(const std::string& s) {
    int n = 2 * s.length() + 1;
    std::string t(n, '#');//处理字符串
    for(int i = 0; i < s.length(); ++i) {
        t[2 * i + 1] = s[i];
    }
    std::vector<int> v(n);//记录回文半径  [l, r] <=> [mid - v[mid], mid - v[mid]]
    for(int i = 0, mid = 0; i < n; ++i) { // mid为回文中心
        if(i <= mid + v[mid]) {
            v[i] = std::min(v[2 * mid - i], mid + v[mid] - i);// (t + i) / 2 = mid <=> t = 2 * mid - i;
        }
        while(t[i - v[i] - 1] == t[i + v[i] + 1] && 0 <= i - v[i] - 1 && i + v[i] + 1 < n) {
            ++v[i];
        }
        if(i + v[i] > mid + v[mid]) {
            mid = i;
        }
    }
    return v;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    std::string s;
    std::cin >> s;
    std::vector<int> v = manacher(s);
    int ans = 0;
    for(int i = 0; i < v.size(); ++i) {
        ans = std::max(ans, v[i]);//求最长回文子串
        std::cout << v[i] << " \n"[i == v.size() - 1];
    }
    std::cout << ans << '\n';
    return 0;
}
```

4 数据结构

4.1 Splay

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

class SplayTree {
public:
    SplayTree() {
        tr.push_back(Node());
        insert(INF);
        insert(-INF);
    }
    void insert(int t) { //插入值为t的数
        int id = root, fa = 0;
        while(id && tr[id].val != t) {
            fa = id;
            id = tr[id].nxt[t > tr[id].val];
        }
        if(id) {
            tr[id].cnt++;
        } else {
            id = ++size;
            tr[fa].nxt[t > tr[fa].val] = id;
            tr.push_back(Node(fa, t));
        }
        splay(id);
    }
    int get_pre(int t) { //查找t的前驱节点
        find(t);
        int id = root;
        if(tr[id].val < t) return id;
        id = tr[id].nxt[0];
        while(tr[id].nxt[1]) {
            id = tr[id].nxt[1];
        }
        splay(id);
        return id;
    }
    int get_suc(int t) { //查找t的后继节点
        find(t);
        int id = root;
        if(tr[id].val > t) return id;
        id = tr[id].nxt[1];
        while(tr[id].nxt[0]) {
            id = tr[id].nxt[0];
        }
        splay(id);
        return id;
    }
    void find(int t) {  //查找值为t的节点，并将该节点转到根
        int id = root;
        while(tr[id].nxt[t > tr[id].val] && t != tr[id].val) {
            id = tr[id].nxt[t > tr[id].val];
        }
        splay(id);
    }
    void erase(int t) { //删除值为t的，只删除1个
        int pre = get_pre(t);
        int suc = get_suc(t);
        splay(pre);
        splay(suc, pre);
        int tid = tr[suc].nxt[0];//目标节点
        if(tr[tid].cnt > 1) {
            tr[tid].cnt--;
            splay(tid);          //向上更新其他节点
        } else {
            tr[suc].nxt[0] = 0;
            splay(suc);          //向上更新其他节点
        }
    }
    int get_root() {
        return root;
    }
    int get_rank(int t) { //查一个数t的排名
        insert(t);
        int res = tr[tr[root].nxt[0]].size;
        erase(t);
        return res;
    }
    int get_kth(int t) {   //查找第k个节点编号
        t++;               //有哨兵，所以++
        int id = root;
        while(true) {
            pushdown(id); //向下传递懒标记
            const auto &[x, y] = tr[id].nxt;
            if(tr[x].size + tr[id].cnt < t) {
                t -= tr[x].size + tr[id].cnt;
                id = y;
            } else {
                if(tr[x].size >= t) {
                    id = tr[id].nxt[0];
                } else {
                    return id;
                }
            }
        }
```

```cpp
    }
    int get_val(int t) { //查找排名为t的数的数值
        int id = get_kth(t);
        splay(id);
        return tr[id].val;
    }
    void reverse(int l, int r) { //反转区间[l, r]
        l = get_kth(l - 1), r = get_kth(r + 1);
        splay(l, 0), splay(r, l);
        tr[tr[r].nxt[0]].tag ^= 1;
    }
    void output(int id) { //中序遍历
        pushdown(id);
        const auto &[x, y] = tr[id].nxt;
        if(x != 0) output(x);
        if(std::abs(tr[id].val) != INF) {
            std::cout << tr[id].val << ' ';
        }
        if(y) output(y);
    }
    int val(int id) {
        return tr[id].val;
    }
private:
    class Node {
    public:
        Node() {
            nxt = {0, 0};
            lst = val = size = cnt = tag = 0;
        }
        Node(int _lst, int _val) : lst(_lst), val(_val) {
            nxt = {0, 0};
            tag = 0;
            size = cnt = 1;
        }
        std::array<int, 2> nxt; //左右节点[0左, 1右]
        int lst;                //父亲
        int val;                //权值
        int cnt;                //权值数
        int size;               //子树大小
        int tag;                //懒标记[1翻, 0不翻]
    };
    void rotate(int id) {
        int pid = tr[id].lst, gid = tr[pid].lst;//父节点, 爷节点
        int k = (tr[pid].nxt[1] == id);         //判断id是pid的左节点还是右节点
        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
^1号子节点
        tr[tr[id].nxt[k ^ 1]].lst = pid;        //id的k^1号子节点的父节点设为pid
        tr[id].nxt[k ^ 1] = pid;                //id的k^1号子节点设置为pid
        tr[pid].lst = id;                       //pid的父节点设置为id
        tr[id].lst = gid;                       //id的父节点设置为gid
        tr[gid].nxt[tr[gid].nxt[1] == pid] = id;//gid的子节点设为id
        pushup(pid);                            //更新pid
        pushup(id);                             //更新id
    }
    void splay(int id, int t = 0) {//将id旋转到为t的子节点, 为0时id为根
        while(tr[id].lst != t) {
            int pid = tr[id].lst, gid = tr[pid].lst;
            if(gid != t) { //非根做双旋
                if((tr[pid].nxt[0] == id) == (tr[gid].nxt[0] == pid)) { //直线式
转中
                    rotate(pid);
                } else {   //折线式转中
                    rotate(id);
                }
            }
            rotate(id);
        }
        if(t == 0) root = id;
    }
    void pushup(int id) {
        const auto &[x, y] = tr[id].nxt;
        tr[id].size = tr[x].size + tr[y].size + tr[id].cnt;
    }
    void pushdown(int id) {
        if(tr[id].tag) {
            auto &[x, y] = tr[id].nxt;
            std::swap(x, y);
            tr[x].tag ^= 1;
            tr[y].tag ^= 1;
            tr[id].tag = 0;
        }
    }
    std::vector<Node> tr;
    int root = 0; //根节点编号
    int size = 0; //节点个数
    const int INF = INT_MAX;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    SplayTree tr;
    for(int i = 1; i <= n; ++i) {
        tr.insert(i);
    }
    for(int i = 1; i <= m; ++i) {
        int l, r;
        std::cin >> l >> r;
        tr.reverse(l, r);
    }
    tr.output(tr.get_root());
    return 0;
}
```

4.2 ST 表

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

//ST表(sparseTable)
//https://www.luogu.com.cn/problem/P3865
template<typename T>
class ST { //下标从0开始
public:
    ST(const std::vector<T> &v) { //数据
        int k = std::__lg(v.size());
        st = std::vector<std::vector<T>>(k + 1, std::vector<T>(v.size()));
        st[0] = v;
        for(int i = 0; i < k; ++i) {
            for(int j = 0; j + (1 << (i + 1)) - 1 < v.size(); ++j) {
                st[i + 1][j] = std::max(st[i][j], st[i][j + (1 << i)]);
            }
        }
    }
    T query(int l, int r) { //查询[l, r]的最大值
        int t = std::__lg(r - l + 1);
        return std::max(st[t][l], st[t][r + 1 - (1 << t)]);
    }
private:
    std::vector<std::vector<T>> st;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, q;
    std::cin >> n >> q;
    std::vector<int> v(n);
    for(int i = 0; i < n; ++i) {
        std::cin >> v[i];
    }
    ST<int> st(v);
    while(q--) {
        int l, r;
        std::cin >> l >> r;
        l--, r--;
        std::cout << st.query(l, r) << '\n';
    }
    return 0;
}
```

4.3 对顶堆

```cpp
#include <bits/stdc++.h>
using i64 = long long;

//对顶堆，维护第k小/大
template<typename T>
struct DoubleHeap {
    DoubleHeap(int _k) : k(_k) {} //第k小，若要第k大，将下面比较函数反转
    std::priority_queue<T, std::vector<T>, std::less<T>> mpq; //大根堆[1, k - 1]
    std::priority_queue<T, std::vector<T>, std::greater<T>> Mpq; //小根堆[k, sz]
    void insert(T x) {
        mpq.push(x);
        while(mpq.size() >= k) {
            Mpq.push(mpq.top());
            mpq.pop();
        }
    }
    T kth() {
        assert(Mpq.empty() == false);
        return Mpq.top();
    }
    const int k;
};

struct MINT {
    int x;
    bool operator<(const MINT &o) const {
        return x < o.x;
    }
    bool operator>(const MINT &o) const {
        return x > o.x;
    }
};

void solve() {
    int n, k;
    std::cin >> n >> k;
    DoubleHeap<MINT> dpq(k);
    for(int i = 1; i <= n; ++i) {
        int opt;
        std::cin >> opt;
        if(opt == 1) {
            int x;
            std::cin >> x;
            dpq.insert({x});
        } else {
            std::cout << dpq.kth().x << '\n';
        }
    }

}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int T;
    std::cin >> T;
    while(T--) {
        solve();
    }
    return 0;
}
```

4.4 并查集

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

//并查集(disjoint set union)
//https://www.luogu.com.cn/problem/P3367
class DSU {
public:
    DSU(int n) { //初始构造
        v.resize(n + 1);
        std::iota(v.begin(), v.end(), 0);
    }
    int find(int x) { //找根
        return (v[x] == x ? x : (v[x] = find(v[x])));
    }
    void uniset(int x, int y) { //合并集合
        v[find(x)] = find(y);
    }
    bool query(int x, int y) { //是否在同一集合
        return find(x) == find(y);
    }
private:
    std::vector<int> v;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    DSU dsu(n);
    for(int i = 0; i < m; ++i) {
        int z, x, y;
        std::cin >> z >> x >> y;
        if(z == 1) {
            dsu.uniset(x, y);
        } else if(z == 2) {
            std::cout << (dsu.query(x, y) ? 'Y' : 'N') << '\n';
        }
    }
    return 0;
}
```

4.5 树状数组

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

//树状数组(Fenwick)
//https://www.luogu.com.cn/problem/P3374
template<typename T>
class Fenwick {
public:
    Fenwick(int n) : v(std::vector<T>(n + 1)) {}; //有参构造
    void update(int x, T dx) { //更新(index, dx)
        for(int i = x; i < v.size(); i += (i & -i)) {
            v[i] += dx;
        }
    }
    T query(int x) { //查询前缀和[0, L]
        T res{};
        for(int i = x; i > 0; i -= (i & -i)) {
            res += v[i];
        }
        return res;
    }
    T range(int l, int r) { //查询区间[L, R]
        return query(r) - query(l - 1);
    }
private:
    std::vector<T> v;
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    Fenwick<int> tr(n);
    for(int i = 1; i <= n; ++i) {
        int x;
        std::cin >> x;
        tr.update(i, x);
    }
    for(int i = 0; i < m; ++i) {
        int o, x, y;
        std::cin >> o >> x >> y;
        if(o == 1) {
            tr.update(x, y);
        } else if (o == 2) {
            std::cout << tr.range(x, y) << '\n';
        }
    }
    return 0;
};
```

4.6 线段树

```cpp
#include <bits/stdc++.h>
using i64 = long long;

//线段树，区间修改，区间查询
//https://www.luogu.com.cn/problem/P3372
template<typename Info, typename Tag>
struct SegmentTree {
#define ls (id<<1)
#define rs (id<<1|1)
    SegmentTree(int n) : n(n), info(n << 2), tag(n << 2) {}
    SegmentTree(const std::vector<Info> &init) : SegmentTree(init.size()) {
        auto build = [&](auto self, int id, int l, int r) ->void {
            if(l == r) {
                info[id] = init[l];
                return;
            }
            int mid = (l + r) / 2;
            self(self, ls, l, mid);
            self(self, rs, mid + 1, r);
            pushup(id);
        };
        build(build, 1, 1, n);
    }
    void apply(int id, const Tag &dx) {
        info[id].apply(dx);
        tag[id].apply(dx);
    }
    void pushup(int id) {
        info[id] = info[ls] + info[rs];
    }
    void pushdown(int id) {
        apply(ls, tag[id]);
        apply(rs, tag[id]);
        tag[id] = Tag();
    }
    void update(int t, const Info &val) {
        update(1, 1, n, t, val);
    }
    void rangeUpdate(int l, int r, const Tag &dx) {
        rangeUpdate(1, 1, n, l, r, dx);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 1, n, l, r);
    }
    void update(int id, int l, int r, int t, const Info &val) {
        if(l == r) {
            info[id] = val;
            return;
        }
        int mid = (l + r) / 2;
        pushdown(id);
        if(t <= mid) {
            update(ls, l, mid, t, val);
        } else if(t > mid) {
            update(rs, mid + 1, r, t, val);
        }
        pushup(id);
    }
    void rangeUpdate(int id, int l, int r, int x, int y, const Tag &dx) {
        if(x <= l && r <= y) {
            apply(id, dx);
            return;
        }
        int mid = (l + r) / 2;
        pushdown(id);
        if(x <= mid) {
            rangeUpdate(ls, l, mid, x, y, dx);
        }
        if(y > mid) {
            rangeUpdate(rs, mid + 1, r, x, y, dx);
        }
        pushup(id);
    }
    Info rangeQuery(int id, int l, int r, int x, int y) {
        if(x <= l && r <= y) {
            return info[id];
        }
        int mid = (l + r) / 2;
        pushdown(id);
        Info res;
        if(x <= mid) {
            res = res + rangeQuery(ls, l, mid, x, y);
        }
        if(y > mid) {
            res = res + rangeQuery(rs, mid + 1, r, x, y);
        }
        return res;
    }
#undef ls
#undef rs
    const int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
};

constexpr i64 INF = 1E18;

struct Tag {
    i64 add = 0;
    void apply(const Tag &dx) {
        add += dx.add;
    }
};

struct Info {
    i64 mn = INF;
    i64 mx = -INF;
    i64 sum = 0;
    i64 len = 0;
    void apply(const Tag &dx) {
        mn += dx.add;
        mx += dx.add;
        sum += len * dx.add;
    }
};

Info operator+(const Info &x, const Info &y) {
    Info res;
    res.mn = std::min(x.mn, y.mn);
    res.mx = std::max(x.mx, y.mx);
    res.sum = x.sum + y.sum;
    res.len = x.len + y.len;
    return res;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    // std::vector<Info> v(n + 1);
    // for(int i = 1; i <= n; ++i) {
    //     int x;
    //     std::cin >> x;
    //     v[i] = {x, x, x, 1};
    // }
    // SegmentTree<Info, Tag> tr(v);
    SegmentTree<Info, Tag> tr(n);
    for(int i = 1; i <= n; ++i) {
        int x;
        std::cin >> x;
        tr.update(i, {x, x, x, 1});
    }
    while(m--) {
        int opt, x, y;
        std::cin >> opt >> x >> y;
        if(opt == 1) {
            int k;
            std::cin >> k;
            tr.rangeUpdate(x, y, {k});
        } else if(opt == 2) {
            std::cout << tr.rangeQuery(x, y).sum << '\n';
        }
    }
    return 0;
}
```

5 数论

5.1 欧拉筛

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

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n;
    std::cin >> n;
    std::vector<bool> isPrime(n + 1, 1);
    std::vector<int> res = {2}; //存放质数
    isPrime[0] = 0;
    for (int i = 3; i <= n; i += 2) {
        if (isPrime[i]) { //如果是素数, 则记录
            res.push_back(i);
        }
        for (int j = 0; res[j] * i <= n && j < res.size(); ++j) {
            isPrime[res[j] * i] = 0;//找出素数的倍数, 标记为合数
            if (i % res[j] == 0) break;
        }
    }
    std::cout << res.size() << '\n';
    for(auto x : res) {
        std::cout << x << ' ';
    }
    return 0;
}
```

6 树算法

6.1 树剖 LCA

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

//树链剖分求LCA
//https://www.luogu.com.cn/problem/P3379
int main() {
    std::ios::sync_with_stdio(0);
    std::cin.tie(nullptr);
    int n, m, s;
    std::cin >> n >> m >> s;
    std::vector<std::vector<int>> v(n + 1);
    std::vector<int> fa(n + 1), dep(n + 1), son(n + 1), sz(n + 1), top(n + 1, 0);
    //父节点，深度，重儿子，子树节点数，所在重链的顶点
    for(int i = 0; i < n - 1; ++i) {
        int x, y;
        std::cin >> x >> y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    auto dfs1 = [&](auto self, int id, int lst) ->void {//求fa, dep, son, sz数组
        fa[id] = lst;
        dep[id] = dep[lst] + 1;
        sz[id] = 1;
        for(auto nxt : v[id]) {
            if(nxt == lst) continue;
            self(self, nxt, id);
            sz[id] += sz[nxt];
            if(sz[son[id]] < sz[nxt]) {
                son[id] = nxt;
            }
        }
    };
    auto dfs2 = [&](auto self, int id, int t) ->void {
        top[id] = t;
        if(son[id] == 0) return;
        self(self, son[id], t);
        for(auto nxt : v[id]) {
            if(nxt != fa[id] && nxt != son[id]) {
                self(self, nxt, nxt);
            }
        }
    };
    auto lca = [&](int x, int y) ->int {
        while(top[x] != top[y]) {
            if(dep[top[x]] < dep[top[y]]) {
                std::swap(x, y);
            }
            x = fa[top[x]];
        }
        return (dep[x] < dep[y] ? x : y);
    };
    dfs1(dfs1, s, 0);
    dfs2(dfs2, s, s);
    for(int i = 0; i < m; ++i) {
        int x, y;
        std::cin >> x >> y;
        std::cout << lca(x, y) << '\n';
    }
    return 0;
}
```