XCPC - Templates

north-h

Contents

1 数据结构

1.1 树状数组（单点修改，区间查询）

```cpp
template <class T>
struct BIT {
    vector<T> tr;
    int n;
    BIT(int N) {
        n = N;
        tr.resize(n + 1);
    }
    void add(int x, T k) {
        for(int i = x; i < n; i += (i & -i))
            tr[i] += k;
    }
    T query(int x) {
        T res = 0;
        for(int i = x; i; i -= (i & -i))
            res += tr[i];
        return res;
    }
    T range_query(int l, int r) {
        return query(r) - query(l - 1);
    }
};
```

1.2 树状数组（区间修改，单点查询）

```cpp
template <class T>
struct BIT {
    vector<T> tr;
    int n;
    BIT(int N) {
        n = N;
        tr.resize(n + 1);
    }
    void add(int x, T k) {
        for(int i = x; i < n; i += (i & -i))
            tr[i] += k;
    }
    void range_add(int l, int r) {
        add(l, k);
        add(r + 1, -k);
    }
    T query(int x) {
        T res = 0;
        for(int i = x; i; i -= (i & -i))
            res += tr[i];
        return res;
    }
};
```

1.3 树状数组（区间修改，区间查询）

```
template <class T>
struct BIT {
    vector<T> sum1, sum2;
    int n;
    BIT() {}
    void init(int N) {
        n = N;
        sum1.resize(n);
        sum2.resize(n);
    }
    void add(int x, T k) {
        for(int i = x; i < n; i += (i & -i))
            sum1[i] += k, sum2[i] += x * k;
    }
    void add(int l, int r, T x) {
        add(l, x), add(r + 1, -x);
    }
    T query(int x) {
        T res = 0;
        for(int i = x; i > 0; i -= (i & -i))
            res += (x + 1) * sum1[i] - sum2[i];
        return res;
    }
    T query(int l, int r) {
        return query(r) - query(l - 1);

    }
};
```

1.4 线段树

```
template <class T>
struct Seg{
    struct Node{ int l, r, sum, lazy; };

    vector<Node> tr;
    vector<int> a;
    int n;

    Seg() {};

    void init(int N) {
        n = N;
        tr.resize(n * 4);
        a.resize(n);
    }

    void add(int x, int k) {
        a[x] = k;
    }
```

```
20
21    void pushup(int u) {
22        tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
23    }
24
25    void pushdown(int u) {
26        if (tr[u].lazy) {
27            tr[u << 1].sum += tr[u].lazy * (tr[u << 1].r - tr[u << 1].l + 1);
28            tr[u << 1 | 1].sum += tr[u].lazy * (tr[u << 1 | 1].r - tr[u << 1 | 1].l +
                  1);
29            tr[u << 1].lazy += tr[u].lazy;
30            tr[u << 1 | 1].lazy += tr[u].lazy;
31            tr[u].lazy = 0;
32        }
33    }
34
35    void build(int u, int l,int r) {
36        tr[u] = {l, r, a[l], 0};
37        if (l == r)return;
38        int mid = l + r >> 1;
39        pushdown(u);
40        build(u << 1, l, mid);
41        build(u << 1 | 1, mid + 1, r);
42        pushup(u);
43    }
44    //区间修改
45    void modify(int u,int l,int r,int k) {
46        if (tr[u].l >= l && tr[u].r <= r) {
47            tr[u].sum += (tr[u].r - tr[u].l + 1) * k;
48            tr[u].lazy += k;
49            return;
50        }
51        pushdown(u);
52        int mid = tr[u].l + tr[u].r >> 1;
53        if (l <= mid)modify(u << 1, l, r, k);
54        if (r > mid) modify(u << 1 | 1, l, r, k);
55        pushup(u);
56    }
57    //单点修改
58    void modify(int u,int x,int k) {
59        if (tr[u].l == tr[u].r) {
60            tr[u].sum += k;
61            return;
62        }
63        int mid = tr[u].l + tr[u].r >> 1;
64        if (x <= mid)modify(u << 1, x, k);
65        else modify(u << 1 | 1, x, k);
66    }
67    //区间查询
68    int query(int u,int l,int r) {
69        if (tr[u].l >= l && tr[u].r <= r) return tr[u].sum;
70        pushdown(u);
71        int sum = 0;
```

```
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid)sum += query(u << 1, l, r);
        if (r > mid) sum += query(u << 1 | 1, l, r);
        return sum;
    }
    //单点查询
    int query(int u,int x) {
        if (tr[u].l == tr[u].r)return tr[u].sum;
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if (x <= mid)return query(u << 1, x);
        else return query(u << 1 | 1, x);
    }
};
```

1.5 ST 表

```
template<class T>
struct ST {
    T n;
    vector<vector<T>> f, g;
    vector<int> lg2;
    ST(const vector<T> &a) {
        n = (int)a.size();
        lg2.resize(n + 1);
        lg2[0] = -1;
        for(int i = 1; i <= n; i ++) {
            lg2[i] = lg2[i >> 1] + 1;
        }
        f.resize(n + 1, vector<T>(lg2[n] + 1));
        g.resize(n + 1, vector<T>(lg2[n] + 1));
        for (int i = 1; i <= n; i++) {
            f[i][0] = a[i];
            g[i][0] = a[i];
        }
        for (int j = 1; (1 << j) <= n; j++) {
            for (int i = 1; i + (1 << j) - 1 <= n; i++) {
                f[i][j] = max(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
                g[i][j] = min(g[i][j - 1], g[i + (1 << (j - 1))][j - 1]);
            }
        }
    }

    T query_max(int l, int r) {
        int k = lg2[r - l + 1];
        return max(f[l][k], f[r - (1 << k) + 1][k]);
    }

    T query_min(int l, int r) {
        int k = lg2[r - l + 1];
        return min(g[l][k], g[r - (1 << k) + 1][k]);
    }
```

```
};
```

1.6 二维树状数组

```
template <class T>
struct BIT_2D {
    vector<vector<T>> tr;
    int n, m;

    BIT_2D(int N, int M) {
        n = , m = M;
        tr.resize(n + 1, vector<T>(m + 1));
    }

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

    void add(int x, int y, T k) {
        for (int i = x; i <= n; i += lowbit(i))
            for (int j = y; j <= m; j += lowbit(j))
                tr[i][j] += k;
    }

    T query(int x, int y) {
        T res = 0;
        for (int i = x; i; i -= lowbit(i))
            for (int j = y; j; j -= lowbit(j))
                res += tr[i][j];
        return res;
    }

    T query(int x1, int y1, int x2, int y2) {
        return query(x2, y2) - query(x2, y1-1) - query(x1-1, y2) + query(x1-1, y1-1);
    }
};
```

1.7 并查集

```
struct DSU {
    vector<int> fa;

    DSU(int n) {
        fa.resize(n + 1);
        for (int i = 1; i <= n; i ++) fa[i] = i;
    }

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

    bool same(int x, int y) {
```

```
        int px = find(x);
        int py = find(y);
        return x == y;
    }

    void merge(int x, int y) {
        int px = find(x);
        int py = find(y);
        if (px != py) {
            fa[px] = py;
        }
    }
};
```

2 字符串

3 杂项

3.1 快读快写

```
template <typename T>
T read() {
    T x = 0;
    char ch = getchar();
    while (ch < '0' || ch > '9') ch = getchar();
    while (ch >= '0' && ch <= '9') x = (x << 3) + (x << 1) + ch - '0', ch = getchar
        ();
    return x;
}

template <typename T>
void write(T x) {
    if (x < 0) putchar('-'), x = -x;
    if (x > 9) write(x / 10);
    putchar(x % 10 + '0');
}
```

3.2 取模

```
template<const int T>
struct ModInt {
    const static int mod = T;
    int x;
    ModInt(int x = 0) : x(x % mod) {}
    ModInt(long long x) : x(int(x % mod)) {}
    int val() {
        return x;
    }
    ModInt operator + (const ModInt &a) const {
        int x0 = x + a.x;
```

```
12      return ModInt(x0 < mod ? x0 : x0 - mod);
13  }
14  ModInt operator - (const ModInt &a) const {
15      int x0 = x - a.x;
16      return ModInt(x0 < 0 ? x0 + mod : x0);
17  }
18  ModInt operator * (const ModInt &a) const {
19      return ModInt(1LL * x * a.x % mod);
20  }
21  ModInt operator / (const ModInt &a) const {
22      return *this * a.inv();
23  }
24  bool operator == (const ModInt &a) const {
25      return x == a.x;
26  };
27  bool operator != (const ModInt &a) const {
28      return x != a.x;
29  };
30  void operator += (const ModInt &a) {
31      x += a.x;
32      if (x >= mod) x -= mod;
33  }
34  void operator -= (const ModInt &a) {
35      x -= a.x;
36      if (x < 0) x += mod;
37  }
38  void operator *= (const ModInt &a) {
39      x = 1LL * x * a.x % mod;
40  }
41  void operator /= (const ModInt &a) {
42      *this = *this / a;
43  }
44  friend ModInt operator + (int y, const ModInt &a) {
45      int x0 = y + a.x;
46      return ModInt(x0 < mod ? x0 : x0 - mod);
47  }
48  friend ModInt operator - (int y, const ModInt &a) {
49      int x0 = y - a.x;
50      return ModInt(x0 < 0 ? x0 + mod : x0);
51  }
52  friend ModInt operator * (int y, const ModInt &a) {
53      return ModInt(1LL * y * a.x % mod);
54  }
55  friend ModInt operator / (int y, const ModInt &a) {
56      return ModInt(y) / a;
57  }
58  friend ostream &operator<<(ostream &os, const ModInt &a) {
59      return os << a.x;
60  }
61  friend istream &operator>>(istream &is, ModInt &t) {
62      return is >> t.x;
63  }
64
```

```
65      ModInt pow(int64_t n) const {
66          ModInt res(1), mul(x);
67          while(n) {
68              if (n & 1) res *= mul;
69              mul *= mul;
70              n >>= 1;
71          }
72          return res;
73      }
74
75      ModInt inv() const {
76          int a = x, b = mod, u = 1, v = 0;
77          while (b) {
78              int t = a / b;
79              a -= t * b;
80              swap(a, b);
81              u -= t * v;
82              swap(u, v);
83          }
84          if (u < 0) u += mod;
85          return u;
86      }
87
88  };
89  using mint = ModInt<998244353>;
```

4 数学

4.1 组合数（递推）

```
1   int C[N][N];
2
3   void comb(){
4       for(int i = 0; i < N; i++) {
5           for(int j = 0; j <= i; j++) {
6               if(!j) {
7                   C[i][j] = 1;
8                   continue;
9               }
10              C[i][j] = C[i - 1][j] + C[i - 1][j - 1];
11              C[i][j] %= mod;
12          }
13      }
14  }
```

4.2 组合数（逆元）

```
1   struct comb2{
2       vector<int> fac,inf;
3
4       comb2(int n) {
```

```
        fac.resize(n + 1);
        inf.resize(n + 1);
        fac[0] = inf[0] = 1;
        for(int i = 1; i <= n; i++){
            fac[i] = fac[i - 1] * i % mod;
            inf[i] = inf[i - 1] * ksm(i, mod - 2) % mod;
        }
    }

    int ksm(int a, int b){ // 快速幂
        int res = 1;
        while(b){
            if(b & 1) res = res * a % mod;
            a = a * a % mod;
            b >>= 1;
        }
        return res;
    }

    int query(int n, int m) {
        int ans;
        ans = fac[n] * inf[n - m] % mod * inf[m] % mod;
        return ans;
    }
};
```

5 Kescholar

5.1 BIT

```
template<typename T>
struct BIT {
#ifndef lowbit
#define lowbit(x) (x & (-x));
#endif
    // static const int maxn = 5e5 + 50;
    int n;
    vector<T> t;

    BIT () {}
    BIT (int _n): n(_n) { t.resize(_n + 1); }
    BIT (int _n, vector<T>& a): n(_n) {
        t.resize(_n + 1);
        for (int i = 1; i <= n; ++ i) {
            t[i] += a[i];
            int j = i + lowbit(i);
            if (j <= n) t[j] += t[i];
        }
    }
    //单点修改
    void update(int i, T x) {
        while (i <= n) {
```

```
            t[i] += x;
            i += lowbit(i);
        }
    }
    //区间查询
    T sum(int i) {
        T ans = 0;
        while (i > 0) {
            ans += t[i];
            i -= lowbit(i);
        }
        return ans;
    }

    T query(int i, int j) {
        return sum(j) - sum(i - 1);
    }
    //区间修改则存入差分数组,[l, r] + k则update(x,k),update(y+1,-k)
    //单点查询则直接求前缀和sum(x)
};
```

5.2 Dijkstra

```
#include <bits/stdc++.h>
using namespace std;

struct DIJ {
    using i64 = long long;
    using PII = pair<i64, i64>;
    vector<i64> dis;
    vector<vector<PII>> G;

    DIJ() {}
    DIJ(int n) {
        dis.assign(n + 1, 1e18);
        G.resize(n + 1);
    }

    void add(int u, int v, int w) {
        G[u].emplace_back(v, w);
    }

    void dijkstra(int s) {
        priority_queue<PII, vector<PII>, greater<PII>> que;
        // priority_queue<PII> que;
        dis[s] = 0;
        que.push({0, s});
        while (!que.empty()) {
            auto p = que.top();
            que.pop();
            int u = p.second;
            if (dis[u] < p.first) continue;
```

```
            for (auto [v, w] : G[u]) {
                if (dis[v] > dis[u] + w) {
                    dis[v] = dis[u] + w;
                    que.push({dis[v], v});
                }
            }
        }
    }
};
```

5.3 Dijkstra 进阶

```
struct DIJ {
    using i64 = long long;
    using PII = pair<i64, i64>;
    vector<i64> dis, path, node;
    vector<vector<array<int, 3>>> G;
    int n;

    DIJ() {}
    DIJ(int n): n(n) {
        node.resize(n + 1, 1);
        dis.assign(n + 1, 1e18);
        G.resize(n + 1);
        path.resize(n + 1, -1);
    }
    //val为权重值,要求最短路径结点数最小时即为1
    void add(int u, int v, int w, int val) {
        G[u].push_back({v, w, val});
    }

    void dijkstra(int s) {
        priority_queue<PII> que;
        dis[s] = 0;
        que.push({0, s});
        while (!que.empty()) {
            auto p = que.top();
            que.pop();
            int u = p.second;
            if (dis[u] < p.first) continue;
            for (auto [v, w, val] : G[u]) {
                if (dis[v] > dis[u] + w) {
                    node[v] = node[u] + val;
                    dis[v] = dis[u] + w;
                    que.push({ -dis[v], v});
                    path[v] = u;
                } else if (dis[v] == dis[u] + w) {
                    if (node[v] > node[u] + val) {
                        node[v] = node[u] + val;
                        path[v] = u;
                    }
                }
```

```
        }
    }
}
};
```

5.4 eulerrange

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

using namespace std;

//欧拉函数,质数
vector<int> euler_range(int n) {
    vector<int> phi(n + 1), prime;
    vector<bool> is_prime(n + 1, true);
    is_prime[1] = 0, phi[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (is_prime[i]) prime.push_back(i), phi[i] = i - 1;
        for (int j = 0; j < (int)prime.size() && i * prime[j] <= n; j++) {
            is_prime[i * prime[j]] = 0;
            if (i % prime[j]) phi[i * prime[j]] = phi[i] * (prime[j] - 1);
            else {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
        }
    }
    return phi;
}
```

5.5 hashpair

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef pair<i64, i64> PII;

template <typename T>
inline void hash_combine(std::size_t &seed, const T &val) {
    seed ^= std::hash<T>()(val) + 0x9e3779b9 + (seed << 6) + (seed >> 2);
}
// auxiliary generic functions to create a hash value using a seed
template <typename T> inline void hash_val(std::size_t &seed, const T &val) {
    hash_combine(seed, val);
}
template <typename T, typename... Types>
inline void hash_val(std::size_t &seed, const T &val, const Types &... args) {
    hash_combine(seed, val);
    hash_val(seed, args...);
}

template <typename... Types>
```

```cpp
inline std::size_t hash_val(const Types &... args) {
    std::size_t seed = 0;
    hash_val(seed, args...);
    return seed;
}

struct pair_hash {
    template <class T1, class T2>
    std::size_t operator()(const std::pair<T1, T2> &p) const {
        return hash_val(p.first, p.second);
    }
};
unordered_map<PII, int, pair_hash> mp;
```

5.6 KMP

```cpp
struct Kmp {
    vector<int> next;

    void getNext(string s) {
        int j = 0, k = -1, n = s.size();
        next.resize(n + 1);
        next[0] = -1;
        while (j < n) {
            if (k == -1 || s[j] == s[k]) {
                next[++j] = ++k;
            } else
                k = next[k];
        }
    }

    int kmp(string s, string p) {
        int n = s.size(), m = p.size();
        int i = 0, j = 0;

        while (i < n && j < m) {
            if (j == -1 || s[i] == p[j]) {
                i ++, j ++;
            } else {
                j = next[j];
            }
        }
        if (j == m) return i;
        else return -1;
    }
};
```

5.7 ksm

```cpp
i64 ksm(i64 a, i64 n, i64 mod) {
    i64 ans = 1;
```

```
    while (n) {
        if (n & 1)
            ans = (ans * a) % mod;
        a = a * a % mod;
        n >>= 1;
    }
    return ans % mod;
}

i64 ksm(i64 a, i64 n) {
    i64 ans = 1;
    while (n) {
        if (n & 1)
            ans = (ans * a);
        a = a * a;
        n >>= 1;
    }
    return ans ;
}
```

5.8 LCA

```
include <bits/stdc++.h>
using namespace std;

struct LCA {
    int n;
    vector<int> dep;
    vector<vector<int>> e;
    vector<array<int, 21>> fa;
    LCA() {}
    LCA(int n) {
        dep.resize(n + 1);
        e.resize(n + 1);
        fa.resize(n + 1);
    }

    void add(int u, int v) {
        e[u].push_back(v);
        e[v].push_back(u);
    }

    //计算深度,处理各点祖先
    void dfs(int u, int father) {
        dep[u] = dep[father] + 1;

        fa[u][0] = father;
        for (int i = 1; i < 20; i ++)
            fa[u][i] = fa[fa[u][i - 1]][i - 1];

        for (auto v : e[u])
            if (v != father)
```

```
            dfs(v, u);
    }

    //最近公共祖先
    //两点集并的最近公共祖先为两点几分别的最近公共祖先的最近公共祖先,
    //即LCA(A∪B) = LCA(LCA(A), LCA(B));
    int lca(int u, int v) {
        if (dep[u] < dep[v]) swap(u, v);
        for (int i = 19; i >= 0; i --)
            if (dep[fa[u][i]] >= dep[v])
                u = fa[u][i];

        if (u == v) return v;

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

    //d(u,v) = h(u) + h(v) - 2h(LCA(u,v));
    //其中d是树上两点间的距离,h代表某点到树根的距离
    int get_dis(int u, int v) {
        return dep[u] + dep[v] - 2 * dep[lca(u, v)];
    }
};
```

5.9 Matrix

```
#include <bits/stdc++.h>
using namespace std;


struct Matrix {
    using i64 = long long;
    i64 N;
    vector<vector<i64>> A;
    Matrix() { N = 0;}
    Matrix(int n) {
        N = n;
        A.resize(N + 1);
        for (int i = 0; i <= N; i ++)
            A[i].resize(N + 1, 0);
    }
    Matrix operator*(const Matrix &b) const {
        Matrix res(b.N);

        for (int i = 1; i <= b.N; ++i)
            for (int j = 1; j <= b.N; ++j)
                for (int k = 1; k <= b.N; ++k)
                    res.A[i][j] = (res.A[i][j] + A[i][k] * b.A[k][j]);
        return res;
```

```
    }
    Matrix qpow(i64 k) {
        Matrix res(N);

        //斐波那契数列初始化
        //res.A[1][1] = res.A[1][2] = 1;
        //A[1][1] = A[1][2] = A[2][1] = 1;

        //单位矩阵
        for (int i = 0; i <= N; i ++)
            res.A[i][i] = 1;

        while (k) {
            if (k & 1) res = res * (*this);
            (*this) = (*this) * (*this);
            k >>= 1;
        }
        return res;
    }
};
```

5.10 phi

```
//欧拉定理求质因数
long long phi(long long x) {
    long long i;
    long long res = x;

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

    return res;
}
```

5.11 Prim

```
#include <bits/stdc++.h>
#define debug(a) cout<<#a<<"="<<a<<'\n';

using namespace std;
using i64 = long long;

typedef pair<i64, i64> PII;

//prim适合稠密图，Kruskal适合稀疏图
int main() {
```

```
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, m;
    cin >> n >> m;
    vector g(n + 1, vector<PII>());
    vector<int> dis(n + 1);
    vector<bool> vis(n + 1);
    i64 ans = 0, cnt = 0;
    for (int i = 0; i < m; i ++) {
        int u, v, w;
        cin >> u >> v >> w;
        g[u].emplace_back(v, w);
        g[v].emplace_back(u, w);
    }

    priority_queue<PII, vector<PII>, greater<PII>> Q;
    Q.push({0, 1});
    while (Q.size()) {
        auto [w, u] = Q.top();
        Q.pop();

        if (vis[u]) continue;
        vis[u] = true;
        ans += w;
        cnt ++;
        dis[u] = w;
        for (auto [v, d] : g[u]) {
            if (!vis[v]) {
                Q.push({d, v});
            }
        }
    }

    if (cnt != n) {
        cout << "orz\n";
    } else
        cout << ans << '\n';

    return 0;
}
```

5.12 ST

```
#include <bits/stdc++.h>
using namespace std;

template <typename T>
class SparseTable {
  using VT = vector<T>;
  using VVT = vector<VT>;
  using func_type = function<T(const T &, const T &)>;
```

```

  VVT ST;

  static T default_func(const T &t1, const T &t2) { return min(t1, t2); }

  func_type op;

public:
  SparseTable(const vector<T> &v, func_type _func = default_func) {
    op = _func;
    int len = v.size(), l1 = ceil(log2(len)) + 1;
    ST.assign(len, VT(l1, 0));
    for (int i = 0; i < len; ++i) {
      ST[i][0] = v[i];
    }

    for (int j = 1; j < l1; ++j) {
      int pj = (1 << (j - 1));
      for (int i = 0; i + pj < len; ++i) {
        ST[i][j] = op(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]);
      }
    }
  }

  T query(int l, int r) {
    if (r < l) return INT_MAX;
    int lt = r - l + 1;
    int q = floor(log2(lt));
    return op(ST[l][q], ST[r - (1 << q) + 1][q]);
  }
};
```

5.13 toposort

```
#include <bits/stdc++.h>
using namespace std;

struct toposort {
   vector<vector<int>> e;
   vector<int> tp , din;
   int n ;

   toposort() {}
   toposort(int n) {
      this->n = n;
      din.resize(n + 1);
      e.resize(n + 1);
   }

   void add(int u, int v) {
      e[u].push_back(v);
      din[v] ++;
```

```
    }

    // Kahn(卡恩算法)判环
    bool topo() {
        queue<int> Q;
        for (int i = 1; i <= n; i ++)
            if (!din[i]) Q.push(i);
        while (Q.size()) {
            auto u = Q.front();
            Q.pop();
            tp.push_back(u);
            for (auto v : e[u])
                if (!--din[v])
                    Q.push(v);
        }
        return tp.size() == n;
    }

};
```

5.14 trie

```
#include <bits/stdc++.h>
using namespace std;

struct trie {
  int n;
  vector<array<int, 52>> trans;
  vector<int> cnt;
  trie() : n(0) { new_node(); }
  int new_node() {
    trans.push_back({});
    trans.back().fill(0);
    cnt.push_back(0);
    return n++;
  }
  int insert(const string &s) {
    int now = 0;
    for (char c : s) {
      int i = c - 'a';
      if (c >= 'A') i = c - 'A' + 26;
      if (!trans[now][i]) {
        trans[now][i] = new_node();
      }
      now = trans[now][i];
    }
    cnt[now]++;//以该结点结尾的单词数+1
    return now;
  }
};

struct trie2 {//常规
```

```
  int trans[100010][26], cnt;
  bool exist[100010];

  void insert(string s) {
    int p = 0;
    for (int i = 0; i < s.size(); i++) {
      int c = s[i] - 'a';
      if (!trans[p][c]) trans[p][c] = ++cnt;
      p = trans[p][c];
    }
    exist[p] = 1;
  }

  bool find(string s) {
    int p = 0;
    for (int i = 0; i < s.size(); i++) {
      int c = s[i] - 'a';
      if (!trans[p][c]) return 0;
      p = trans[p][c];
    }
    return exist[p];
  }
};
```

5.15 并查集

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

using namespace std;

struct UFS {
    int sz;
    vector<int> rank, p;
    void link(int x, int y) {
        if (x == y)
            return;
        if (rank[x] > rank[y])
            p[y] = x;
        else
            p[x] = y;
        if (rank[x] == rank[y])
            rank[y]++;
    }
    void init(int n) {
        sz = n;
        rank.resize(n + 1);
        p.resize(n + 1);
        for (int i = 0; i <= sz; i++) {
            p[i] = i;
            rank[i] = 0;
        }
    }
```

```
    int find(int x) {
        return x == p[x] ? x : (p[x] = find(p[x]));
    }
    void unin(int x, int y) {
        link(find(x), find(y));
    }
    void compress() {
        for (int i = 0; i < sz; i++)
            find(i);
    }
};
//种类并查集 merge(y + n, x),merge(x + n, y)
```

5.16 二叉搜索树

```
struct Node {
    int data;
    Node* left;
    Node* right;

    Node* newNode(int v) {
        Node* node = new Node;
        node->data = v;
        node->right = node->left = NULL;
        return node;
    }

    void insert(Node* &root, int x) {
        if (root == NULL) {
            root = newNode(x);
            return ;
        }
        if (root->data > x)
            insert(root->left, x);
        else
            insert(root->right, x);
    }

    Node* Create(vector<int> tree) {
        Node* root = NULL;
        for (auto i : tree) {
            insert(root, i);
        }
        return root;
    }
};
```

5.17 二分图最大匹配

```
struct augment_path {
    vector<vector<int> > g;
```

```
3     vector<int> pa; // 匹配
4     vector<int> pb;
5     vector<int> vis; // 访问
6     int n, m; // 两个点集中的顶点数量
7     int dfn; // 时间戳记
8     int res; // 匹配数
9
10    augment_path(int _n, int _m) : n(_n), m(_m) {
11        assert(0 <= n && 0 <= m);
12        pa = vector<int>(n, -1);
13        pb = vector<int>(m, -1);
14        vis = vector<int>(n);
15        g.resize(n);
16        res = 0;
17        dfn = 0;
18    }
19
20    void add(int from, int to) {
21        assert(0 <= from && from < n && 0 <= to && to < m);
22        g[from].push_back(to);
23    }
24
25    bool dfs(int v) {
26        vis[v] = dfn;
27        for (int u : g[v]) {
28            if (pb[u] == -1) {
29                pb[u] = v;
30                pa[v] = u;
31                return true;
32            }
33        }
34        for (int u : g[v]) {
35            if (vis[pb[u]] != dfn && dfs(pb[u])) {
36                pa[v] = u;
37                pb[u] = v;
38                return true;
39            }
40        }
41        return false;
42    }
43
44    int solve() {
45        while (true) {
46            dfn++;
47            int cnt = 0;
48            for (int i = 0; i < n; i++) {
49                if (pa[i] == -1 && dfs(i)) {
50                    cnt++;
51                }
52            }
53            if (cnt == 0) {
54                break;
55            }
```

```
            res += cnt;
        }
        return res;
    }
};
```

5.18 二维坐标离散化

```
struct Two_D_Discrete {
    int n, tot1 = 1, tot2 = 1;
    vector<vector<int>> mp;
    vector<int> x, y, nx, ny;
    vector<pair<i64, i64>> a;
    vector<PII> New;

    Two_D_Discrete (int _n, vector<pair<i64, i64>>& _a): n(_n), a(_a) {
        x.resize(n), y.resize(n);
        nx.resize(n * 2 + 5), ny.resize(n * 2 + 5);
        vector<vector<int>>(n * 2 + 5, vector<int>(n * 2 + 5)).swap(mp);
        for (int i = 0; i < n; i ++) {
            x[i] = a[i].first;
            y[i] = a[i].second;
        }
    }

    void work() {
        //排序
        sort(x.begin(), x.end());
        sort(y.begin(), y.end());
        // 去重 并得到有多少个点
        int len1 = unique(x.begin(), x.end()) - x.begin();
        int len2 = unique(y.begin(), y.end()) - y.begin();
        // 离散化 x 轴
        for (int i = 0; i < len1; i++) {
            if (i && x[i] != x[i - 1] + 1)
                nx[tot1++] = x[i] - 1, nx[tot1++] = x[i];
            else
                nx[tot1++] = x[i];
        }
        // 离散化 y 轴
        for (int i = 0; i < len2; i++) {
            if (i && y[i] != y[i - 1] + 1)
                ny[tot2++] = y[i] - 1, ny[tot2++] = y[i];
            else
                ny[tot2++] = y[i];
        }
        //映射关系将需离散的点放入离散图中
        for (int i = 0; i < n; i++) {
            int newx = lower_bound(nx.begin(), nx.begin() + tot1, a[i].first) - nx.
                begin();
            int newy = lower_bound(ny.begin(), ny.begin() + tot2, a[i].second) - ny.
                begin();
```

```
43          mp[newx][newy] = 1;
44          // cout << "(" << newx << ',' << newy << ")\n";
45          New.emplace_back(newx, newy);
46      }
47   }
48 };
```

5.19 区间素数筛

```
1  template<typename T>
2  struct segment_sieve {
3     vector<bool> is_prime, is_prime_small;
4     vector<T> prime;
5     segment_sieve() {
6        is_prime.resize(1000010);
7        is_prime_small.resize(1000100);
8     }
9
10    //对区间[a,b]内的整数执行筛法，is_prime[i-a]=true --- 表示i是素数 注意这里下标偏移了a
          ，所以从0开始。
11    void get_num(T a, T b) {
12       if (a == 1) a = 2;
13       for (T i = 0; i * i <= b; ++i)
14          is_prime_small[i] = true; //对[2,sqrt(b))的初始化全为质数
15       for (T i = 0; i <= b - a; ++i)
16          is_prime[i] = true; //对下标偏移后的[a,b)进行初始化
17
18       for (T i = 2; i * i <= b; ++i) {
19          if (is_prime_small[i]) {
20             for (T j = 2 * i; j * j <= b; j += i)
21                is_prime_small[j] = false; //筛选[2,sqrt(b));
22             //(a+i-1)/i得到最接近a的i的倍数，最低是i的2倍，然后筛选
23             for (T j = max((T)2, (a + i - 1) / i) * i; j <= b; j += i)
24                is_prime[j - a] = false;
25          }
26       }
27       for (T i = 0; i <= b - a; ++i) //统计个数
28          if (is_prime[i])
29             prime.push_back(i + a);
30    }
31 };
```

5.20 树状数组区间查询单点修改

```
1  #include <bits/stdc++.h>
2  #define debug(a) cout<<#a<<"="<<a<<'\n';
3
4  using namespace std;
5  using i64 = long long;
6
7  typedef pair<i64, i64> PII;
```

```cpp

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, m;
    cin >> n >> m;
    vector<int> h1(n + 1), a(n + 1), h2(n + 1);

#ifndef lowbit
#define lowbit(x) (x & (-x))
#endif

    auto updata = [&](int x, int val) {
        while (x <= n) {
            h1[x] = h2[x] = val;
            int lx = lowbit(x);
            for (int i = 1; i < lx; i <<= 1) {
                h1[x] = max(h1[x], h1[x - i]);
                h2[x] = min(h2[x], h2[x - i]);
            }
            x += lowbit(x);
        }
    };

    auto RMQ = [&](int l, int r) {
        int x = 0, y = a[l];
        while (l <= r) {
            x = max(x, a[r]);
            y = min(y, a[r]);
            r --;
            for (; r - lowbit(r) >= l; r -= lowbit(r)) {
                x = max(x, h1[r]);
                y = min(y, h2[r]);
            }
        }
        return x - y;
    };

    for (int i = 1; i <= n; i ++) {
        cin >> a[i];
        updata(i, a[i]);
    }

    while (m --) {
        int l, r, op;
        cin >> op;
        if (op) {
            cin >> l >> r;
            cout << RMQ(l, r) << '\n';

        } else {
            int y, x;
```

```
61          cin >> y >> x;
62          a[y] = x;
63          updata(y, x);
64       }
65    }
66
67
68    return 0;
69 }
```