OI 模板

panda_2134

2018 年 4 月 3 日

目录

1　图的 DFS 树

1.1　强连通分量

一有向图上每个点有非负权值，求一条路径，使得路径上点权值和最大。点和边都可以多次经过，但是权值只计入答案一次。

Solution: 缩点后直接在 DAG 上 DP.

GraphTheory/TarjanSCC.cpp

```cpp
#include <bits/stdc++.h>
#define fst first
#define snd second
using namespace std;

typedef pair<int, int> pii;
const int MAXN = 1e5, INF = 0x3f3f3f3f;

struct Graph {
    struct Edge {
        int v, next;
    };

    int n, m, e_ptr = 1, head[MAXN+10]; Edge E[(MAXN+10)<<1];

    void add_edge(int u, int v) {
        E[++e_ptr] = (Edge) { v, head[u] }; head[u] = e_ptr;
    }
} G1, G2;

int dfs_clock, scc_cnt, sccno[MAXN+10], dfn[MAXN+10], low[MAXN+10];
int ans, topo_cnt, topo_seq[MAXN+10], w[MAXN+10],
    tot[MAXN+10], vis[MAXN+10], dp[MAXN+10];

stack<int> S;
void dfs(int u) {
    dfn[u] = low[u] = ++dfs_clock;
    S.push(u);
    for(int j=G1.head[u]; j; j=G1.E[j].next) {
        int v = G1.E[j].v;
        if(!dfn[v]) {
            dfs(v);
            low[u] = min(low[u], low[v]);
        } else if(!sccno[v])
            low[u] = min(low[u], dfn[v]);
    }
    if(low[u] == dfn[u]) {
        int v; ++scc_cnt;
        do {
            v = S.top(); S.pop();
            sccno[v] = scc_cnt;
            tot[scc_cnt] += w[v];
        } while(u != v);
    }
}

void Tarjan() {
    for(int u = 1; u <= G1.n; u++)
        if(!dfn[u]) dfs(u);
}

void scc_graph() {
    set<pii> evis;
    for(int u = 1; u <= G1.n; u++)
        for(int j=G1.head[u]; j; j=G1.E[j].next) {
            int v = G1.E[j].v;
            if(sccno[u] == sccno[v] || evis.count(make_pair(sccno[u], sccno[v])))
                continue;
            else {
                evis.insert(make_pair(sccno[u], sccno[v]));
                G2.add_edge(sccno[u], sccno[v]);
            }
        }
    G2.n = scc_cnt;
}

bool topo_dfs(int u) {
    vis[u] = -1;
    for(int j=G2.head[u]; j; j=G2.E[j].next) {
        int v = G2.E[j].v;
        if(vis[v] == -1 || (vis[v] == 0 && !topo_dfs(v)))
            return false;
    }
    vis[u] = 1;
    topo_seq[topo_cnt--] = u;
    return true;
}

bool toposort() {
    topo_cnt = G2.n;
    for(int u = G2.n; u >= 1; u--)
        if(vis[u] == 0 && !topo_dfs(u)) return false;
    return true;
}

inline int readint() {
    int f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    return f*r;
}

void init() {
    int u, v;
    G1.n = readint(); G1.m = readint();
    for(int i = 1; i <= G1.n; i++)
        w[i] = readint();
    for(int i = 1; i <= G1.m; i++) {
        u = readint(); v = readint();
        G1.add_edge(u, v);
    }
    Tarjan(); scc_graph();
    assert(toposort());
}
```

```

void work() {
    for(int i = G2.n; i >= 1; i--) {
        int u = topo_seq[i], maxv = 0;
        for(int j=G2.head[u]; j; j=G2.E[j].next) {
            int v = G2.E[j].v;
            if(dp[v] > maxv) maxv = dp[v];
        }
        dp[u] = tot[u] + maxv;
        ans = max(ans, dp[u]);
    }
    printf("%d", ans);
}

int main() {
    init(); work();
    return 0;
}
```

1.2 桥和割点

1.3 点双连通分量

1.4 边双连通分量

2 最短路

2.1 Dijkstra

2.1.1 Using std::priority_queue

GraphTheory/Dijkstra–STL.cpp

```cpp
#include <bits/stdc++.h>
#define fst first
#define snd second
using namespace std;

typedef pair<int, int> HeapNode;

struct Edge {
    int v, len, next;
};

const int MAXN = 1e4, MAXM = 5e5, INF = 0x3f3f3f3f;
int n, m, s, e_ptr = 1, head[MAXN+10]; Edge E[(MAXM+10)<<1];
int dist[MAXN+10], done[MAXN+10];

void add_edge(int u, int v, int len) {
    E[++e_ptr] = (Edge) { v, len, head[u] }; head[u] = e_ptr;
}

void add_pair(int u, int v, int len) {
    add_edge(u, v, len); add_edge(v, u, len);
}

void Dijkstra() {
    priority_queue<HeapNode, vector<HeapNode>, greater<HeapNode> > pq;
    memset(done, 0, sizeof(done));
    memset(dist, 0x3f, sizeof(dist));
    dist[s] = 0; pq.push(make_pair(dist[s], s));
    while(!pq.empty()) {
        HeapNode p = pq.top(); pq.pop();
        int u = p.snd;
        if(done[u]) continue;
        done[u] = true;
        for(int j=head[u]; j; j=E[j].next) {
            int v = E[j].v, len = E[j].len;
            if(dist[v] > dist[u] + len) {
                dist[v] = dist[u] + len;
                pq.push(make_pair(dist[v], v));
            }
        }
    }
}

inline int readint() {
    int f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    return f*r;
}
```

```cpp

int main() {
    int u, v, w;
    n = readint(); m = readint(); s = readint();
    for(int i = 1; i <= m; i++) {
        u = readint(); v = readint(); w = readint();
        add_edge(u, v, w);
    }
    Dijkstra();
    for(int i = 1; i <= n; i++) {
        if(dist[i] < INF)
            printf("%d ", dist[i]);
        else
            printf("%d ", INT_MAX);
    }
    return 0;
}
```

2.1.2 Using `__gnu_pbds::priority_queue`

使用了扩展库 pb_ds 中的配对堆，自带修改堆内元素操作，速度更快。仅在允许使用 STL 扩展时才使用。

GraphTheory/Dijkstra–pb_ds.cpp

```cpp
#include <bits/stdc++.h>
#include <bits/extc++.h>
#define fst first
#define snd second
using namespace std;

typedef pair<int, int> HeapNode;
typedef __gnu_pbds::priority_queue<HeapNode, greater<HeapNode>,
        __gnu_pbds::pairing_heap_tag> PairingHeap;

struct Edge {
    int v, len, next;
};

const int MAXN = 1e4, MAXM = 5e5, INF = 0x3f3f3f3f;
int n, m, s, e_ptr = 1, head[MAXN+10]; Edge E[(MAXM+10)<<1];
int dist[MAXN+10]; PairingHeap pq; PairingHeap::point_iterator it[MAXN+10];

void add_edge(int u, int v, int len) {
    E[++e_ptr] = (Edge) { v, len, head[u] }; head[u] = e_ptr;
}

void add_pair(int u, int v, int len) {
    add_edge(u, v, len); add_edge(v, u, len);
}

void Dijkstra() {
    memset(it, 0, sizeof(it));
    memset(dist, 0x3f, sizeof(dist));
    dist[s] = 0; it[s] = pq.push(make_pair(dist[s], s));
    while(!pq.empty()) {
        HeapNode p = pq.top(); pq.pop();
        int u = p.snd;
        for(int j=head[u]; j; j=E[j].next) {
            int v = E[j].v, len = E[j].len;
            if(dist[v] > dist[u] + len) {
                dist[v] = dist[u] + len;
                if(it[v] == NULL)
                    it[v] = pq.push(make_pair(dist[v], v));
                else
                    pq.modify(it[v], make_pair(dist[v], v));
            }
        }
    }
}

inline int readint() {
    int f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    return f*r;
}

int main() {
    int u, v, w;
    n = readint(); m = readint(); s = readint();
    for(int i = 1; i <= m; i++) {
        u = readint(); v = readint(); w = readint();
        add_edge(u, v, w);
    }
    Dijkstra();
    for(int i = 1; i <= n; i++) {
        if(dist[i] < INF)
            printf("%d ", dist[i]);
        else
            printf("%d ", INT_MAX);
    }
    return 0;
}
```

3 网络流

3.1 最大流

3.2 Dinic

NetworkFlow/Dinic.cpp

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

struct Edge {
    int v, flow, cap, next;
};

const int MAXN = 1e4, MAXM = 1e5, INF = 0x3f3f3f3f;
```

```
int n, m, s, t, e_ptr = 1, head[MAXN+10]; Edge E[(MAXM+10)<<1];
int d[MAXN+10], cur[MAXN+10];

void AddEdge(int u, int v, int cap) {
    E[++e_ptr] = (Edge) { v, 0, cap, head[u] }; head[u] = e_ptr;
    E[++e_ptr] = (Edge) { u, 0,   0, head[v] }; head[v] = e_ptr;
}

bool BFS() {
    queue<int> Q;
    memset(d, 0xff, sizeof(d));
    Q.push(s); d[s] = 0;
    while(!Q.empty()) {
        int u = Q.front(); Q.pop();
        for(int j=head[u]; j; j=E[j].next) {
            int v = E[j].v, f = E[j].flow, c = E[j].cap;
            if(f < c && d[v] == -1) {
                d[v] = d[u] + 1;
                if(v == t) return true;
                else Q.push(v);
            }
        }
    }
    return false;
}

int DFS(int u, int flow) {
    if(u == t || flow == 0) return flow; // !!!!!
    int res = flow;
    for(int &j=cur[u]; j; j=E[j].next) { // !!!!!
        int v = E[j].v, f = E[j].flow, c = E[j].cap;
        if(f < c && d[v] == d[u] + 1) {
            int aug = DFS(v, min(res, c-f));
            E[j].flow += aug; E[j^1].flow -= aug;
            res -= aug;
            if(res == 0) break; // !!!!!
        }
    }
    return flow - res;
}

int Dinic() {
    int MaxFlow = 0, CurFlow = 0;
    while(BFS()) {
        memcpy(cur, head, sizeof(head));
        while((CurFlow = DFS(s, INF)))
            MaxFlow += CurFlow;
    }
    return MaxFlow;
}

int main() {
    int u, v, c;
    scanf("%d%d%d%d", &n, &m, &s, &t);
    for(int i = 1; i <= m; i++) {
        scanf("%d%d%d", &u, &v, &c);
        AddEdge(u, v, c);
    }
    printf("%d", Dinic());
    return 0;
}
```

3.3 最小费用最大流

3.3.1 zkw 费用流

NetworkFlow/zkw.cpp

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

typedef long long int64;
struct Edge {
    int u, v;
    int64 flow, cap, cost;
    int next;
};

const int MAXN = 5e3, MAXM = 5e4;
const int64 LL_INF = 0x3f3f3f3f3f3f3f3fLL;
int n, m, s, t, e_ptr = 1, head[MAXN+10]; Edge E[(MAXM+10)<<1]; // ** E[(MAXM+10)<<1] **
int64 MaxFlow, MinCost, dist[MAXN+10], inq[MAXN+10], vis[MAXN+10];

void add_edge(int u, int v, int64 cap, int64 cost) {
    E[++e_ptr] = (Edge) { u, v, 0, cap, cost, head[u] }; head[u] = e_ptr;
    E[++e_ptr] = (Edge) { v, u, 0,  0, -cost, head[v] }; head[v] = e_ptr;
}

bool spfa() {
    queue<int> Q;
    memset(dist, 0x3f, sizeof(dist));
    Q.push(t); dist[t] = 0; inq[t] = true;
    while(!Q.empty()) {
        int u = Q.front(); Q.pop(); inq[u] = false;
        for(int j=head[u]; j; j=E[j].next) {
            int v = E[j].v; int64 f = E[j^1].flow, c = E[j^1].cap, len = E[j^1].cost;
            if(f < c && dist[v] > dist[u] + len) {
                dist[v] = dist[u] + len;
                if(!inq[v]) {
                    inq[v] = true; Q.push(v);
                }
            }
        }
    }
    return dist[s] != LL_INF;
}

int64 dfs(int u, int64 flow) {
    if(u == t || flow == 0) return flow;
    vis[u] = true;
    int64 res = flow;
    for(int j=head[u]; j; j=E[j].next) {
        int v = E[j].v; int64 f = E[j].flow, c = E[j].cap, len = E[j].cost;
        if(f < c && !vis[v] && dist[v] == dist[u] - len) {
            int64 aug = dfs(v, min(res, c-f));
            E[j].flow += aug; E[j^1].flow -= aug;
            res -= aug;
            if(res == 0LL) break;
        }
    }
    return flow - res;
}

void zkw() {
    int64 CurFlow = 0LL;
    while(spfa()) {
        while(memset(vis, 0, sizeof(vis)),
            CurFlow = dfs(s, LL_INF)) {
            MaxFlow += CurFlow;
            MinCost += dist[s] * CurFlow;
        }
    }
}

template<typename T>
inline void readint(T &x) {
    T f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    x = f*r;
}

int main() {
    int u, v; int64 w, c;
    readint(n); readint(m); readint(s); readint(t);
    for(int i = 1; i <= m; i++) {
        readint(u); readint(v); readint(w); readint(c);
        add_edge(u, v, w, c);
    }
    zkw();
    printf("%lld %lld", MaxFlow, MinCost);
    return 0;
}
```

3.3.2 Primal Dual

NetworkFlow/PrimalDual.cpp

```
#include <bits/stdc++.h>
#include <bits/extc++.h>
#define fst first
#define snd second

using namespace std;

typedef long long int64;
typedef pair<int64, int> HeapNode;
typedef __gnu_pbds::priority_queue<HeapNode, greater<HeapNode>,
        __gnu_pbds::pairing_heap_tag> PairingHeap;
```

```
const int MAXN = 5e3, MAXM = 5e4;
const int64 LL_INF = 0x3f3f3f3f3f3f3f3fLL;

struct Edge {
    int u, v;
    int64 flow, cap, cost;
    int next;
};

int n, m, s, t, e_ptr = 1, head[MAXN+10]; Edge E[(MAXM+10)<<1];
int64 MaxFlow, MinCost, delta, dist[MAXN+10], vis[MAXN+10], inq[MAXN+10];
PairingHeap::point_iterator it[MAXN+10];

void add_edge(int u, int v, int64 cap, int64 cost) {
    E[++e_ptr] = (Edge) { u, v, 0, cap, cost, head[u] }; head[u] = e_ptr;
    E[++e_ptr] = (Edge) { v, u, 0,  0, -cost, head[v] }; head[v] = e_ptr;
}

void Reduce() {
    for(int i = 2; i <= e_ptr; i++)
        E[i].cost -= (dist[E[i].u] - dist[E[i].v]);
    delta += dist[s];
}

bool BellmanFord() {
    queue<int> Q;
    memset(dist, 0x3f, sizeof(dist));
    Q.push(t); dist[t] = 0; inq[t] = true;
    while(!Q.empty()) {
        int u = Q.front(); Q.pop(); inq[u] = false;
        for(int j=head[u]; j; j=E[j].next) {
            int v = E[j].v; int64 f = E[j^1].flow, c = E[j^1].cap, len = E[j^1].cost;
            if(f < c && dist[v] > dist[u] + len) {
                dist[v] = dist[u] + len;
                if(!inq[v]) {
                    inq[v] = true; Q.push(v);
                }
            }
        }
    }
    return dist[s] != LL_INF;
}

bool Dijkstra() {
    PairingHeap pq;
    memset(dist, 0x3f, sizeof(dist));
    memset(it, 0, sizeof(it));
    dist[t] = 0; it[t] = pq.push(make_pair(dist[t], t));
    while(!pq.empty()) {
        HeapNode t = pq.top(); pq.pop();
        int u = t.snd;
        for(int j=head[u]; j; j=E[j].next) {
            int v = E[j].v; int64 f = E[j^1].flow, c = E[j^1].cap, len = E[j^1].cost;
            if(f < c && dist[v] > dist[u] + len) {
                dist[v] = dist[u] + len;
                if(it[v] == NULL)
                    it[v] = pq.push(make_pair(dist[v], v));
                else
                    pq.modify(it[v], make_pair(dist[v], v));
            }
        }
    }
    return dist[s] != LL_INF;
}

int64 dfs(int u, int64 flow) {
    if(u == t || flow == 0) return flow;
    vis[u] = true;
    int64 res = flow;
    for(int j=head[u]; j; j=E[j].next) {
        int v = E[j].v; int64 f = E[j].flow, c = E[j].cap, len = E[j].cost;
        if(f < c && !vis[v] && len == 0) {
            int64 aug = dfs(v, min(res, c-f));
            E[j].flow += aug; E[j^1].flow -= aug;
            res -= aug;
            if(res == 0) break;
        }
    }
    return flow - res;
}

void Augment() {
    int64 CurFlow = 0;
    while( memset(vis, 0, sizeof(vis)),
        (CurFlow = dfs(s, LL_INF)) ) {
        MaxFlow += CurFlow;
        MinCost += delta * CurFlow;
    }
}

void PrimalDual() {
    if(!BellmanFord()) return;
    Reduce(); Augment();
    while(Dijkstra()) {
        Reduce(); Augment();
    }
}

template<typename T>
inline void readint(T &x) {
    T f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    x = f*r;
}

int main() {
    int u, v; int64 w, c;
    readint(n); readint(m); readint(s); readint(t);
    for(int i = 1; i <= m; i++) {
        readint(u); readint(v); readint(w); readint(c);
        add_edge(u, v, w, c);
    }
    PrimalDual();
    printf("%lld %lld", MaxFlow, MinCost);
    return 0;
}
```

4 树

4.1 倍增 LCA

Tree/DoublingLCA.cpp

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

struct Edge { int v, next; };

const int MAXN = 1e6, LOG = 20;
int n, q, s, e_ptr = 1, head[MAXN+10]; Edge E[(MAXN+10)<<1];
int dep[MAXN+10], anc[MAXN+10][LOG+1];

void add_edge(int u, int v) { E[++e_ptr] = (Edge) { v, head[u] }; head[u] = e_ptr; }
void add_pair(int u, int v) { add_edge(u, v); add_edge(v, u); }

void dfs(int u) {
    for(int i = 1; i <= LOG; i++)
        anc[u][i] = anc[anc[u][i-1]][i-1];
    for(int j=head[u]; j; j=E[j].next) {
        int v = E[j].v;
        if(v == anc[u][0]) continue;
        anc[v][0] = u; dep[v] = dep[u] + 1;
        dfs(v);
    }
}

int lca(int u, int v) {
    if(dep[u] < dep[v]) swap(u, v);
    for(int i = LOG; i >= 0; i--)
        if(dep[anc[u][i]] >= dep[v])
            u = anc[u][i];
    if(u == v) return u;
    for(int i = LOG; i >= 0; i--)
        if(anc[u][i] != anc[v][i])
            u = anc[u][i], v = anc[v][i];
    u = anc[u][0], v = anc[v][0];
    return u;
}

inline int readint() {
    int f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    return f*r;
}

int main() {
    int u, v;
    n = readint(); q = readint(); s = readint();
    for(int i = 1; i <= n-1; i++) {
        u = readint(); v = readint();
        add_pair(u, v);
    }
    dep[s] = 1; dfs(s);
    while(q--) {
        u = readint(); v = readint();
        printf("%d\n", lca(u, v));
    }
    return 0;
}
```

4.2 欧拉序列求 LCA

Tree/EulerTourLCA.cpp

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

const int MAXN = 1e6;

struct Edge {
    int v, next;
};

int n, q, s, e_ptr = 1, dfs_clock, head[MAXN+10]; Edge E[(MAXN+10)<<1];
int dfn[MAXN+10], dfs_seq[MAXN+10], idx[MAXN+10], euler_seq[(MAXN+10)<<1], st[(MAXN+10)<<1][22];
/*
    dfn: dfs-clock of vertex u
    idx: the index of vertex u in euler-tour sequence
    dfs_seq: the dfs sequence
*/

void add_edge(int u, int v) {
    E[++e_ptr] = (Edge) { v, head[u] }; head[u] = e_ptr;
}

void add_pair(int u, int v) {
    add_edge(u, v); add_edge(v, u);
}

inline int readint() {
    int f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    return f*r;
}

void dfs(int u, int fa) {
    euler_seq[++euler_seq[0]] = dfn[u] = ++dfs_clock;
    idx[u] = euler_seq[0]; dfs_seq[dfs_clock] = u;
    for(int j=head[u]; j; j=E[j].next) {
        int v = E[j].v;
        if(v == fa) continue;
        dfs(v, u);
        euler_seq[++euler_seq[0]] = dfn[u];
    }
}

void init_lca() {
    memset(st, 0x3f, sizeof(st));
    for(int i = 1; i <= euler_seq[0]; i++)
        st[i][0] = euler_seq[i];
    for(int j = 1; j <= 21; j++)
        for(int i = 1; i <= euler_seq[0] - (1<<j) + 1; i++) // bounds of sparse-table!
            st[i][j] = min(st[i][j-1], st[i + (1 << (j-1))][j-1]);
}

int query(int l, int r) {
    if(l > r) swap(l, r);
    int j;
    for(j = 0; (1 << (j+1)) <= (r-l+1); j++);
    return min(st[l][j], st[r - (1<<j) + 1][j]);
}

int lca(int u, int v) {
    return dfs_seq[query(idx[u], idx[v])];
}

int main() {
    int u, v;
    n = readint(); q = readint(); s = readint();
    for(int i = 1; i <= n-1; i++) {
        u = readint(); v = readint();
        add_pair(u, v);
    }
    dfs(s, -1); init_lca();
    while(q--) {
        u = readint(); v = readint();
        printf("%d\n", lca(u, v));
    }
    return 0;
}
```

4.3 树链剖分

Tree/HLD.cpp

```cpp
// call Dfs1(1) and Dfs2(1, 1)
const int MAXN = 1e5;
int dfs_clock, Fa[MAXN+10], Son[MAXN+10], Sz[MAXN+10],
    Dep[MAXN+10], Top[MAXN+10], Dfn[MAXN+10];

void Dfs1(int u) { // Fa Son Sz Dep
    int maxsz = 0; Sz[u] = 1;
    for(int j=head[u]; j; j=E[j].next) {
        int v = E[j].v;
        if(v == Fa[u]) continue;
        Fa[v] = u; Dep[v] = Dep[u] + 1; // !
        Dfs1(v); Sz[u] += Sz[v];
        if(Sz[v] > maxsz) {
            maxsz = Sz[v];
            Son[u] = v;
        }
    }
}

```

```
void Dfs2(int u, int anc) { // Top Dfn
    Dfn[u] = ++dfs_clock; Top[u] = anc;
    if(Son[u]) Dfs2(Son[u], anc);
    for(int j=head[u]; j; j=E[j].next) {
        int v = E[j].v;
        if(v == Fa[u] || v == Son[u]) continue;
        Dfs2(v, v);
    }
}

int LCA(int u, int v) {
    while(Top[u] != Top[v]) {
        if(Dep[Top[u]] < Dep[Top[v]]) swap(u, v);
        u = Fa[Top[u]];
    }
    if(Dep[u] > Dep[v]) swap(u, v);
    return u;
}

int HLDQuery(int u, int v) {
    int ret = -INF;
    while(Top[u] != Top[v]) {
        if(Dep[Top[u]] < Dep[Top[v]]) swap(u, v);
        ret = max(ret, st_query(1, 1, n, Dfn[Top[u]], Dfn[u]));
        u = Fa[Top[u]];
    }
    if(Dep[u] > Dep[v]) swap(u, v);
    ret = max(ret, st_query(1, 1, n, Dfn[u], Dfn[v]));
    return ret;
}
```

4.4 点分治

5　单调数据结构

5.1　单调队列 (滑动窗口)

Monotonic/SlidingWindow.cpp

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

const int MAXN = 1e6;
int n, k, Hd, Tl, A[MAXN+10], Q[MAXN+10];

void SlideMin() {
    Hd = 1, Tl = 0;
    for(int i = 1; i <= k; i++) {
        while(Hd <= Tl && A[Q[Tl]] >= A[i]) Tl--;
        Q[++Tl] = i;
    }
    printf("%d ", A[Q[Hd]]);
    for(int i = k+1; i <= n; i++) {
        while(Hd <= Tl && Q[Hd] < i-k+1) Hd++;
        while(Hd <= Tl && A[Q[Tl]] >= A[i]) Tl--;
        Q[++Tl] = i;
        printf("%d ", A[Q[Hd]]);
    }
}

void SlideMax() {
    Hd = 1, Tl = 0;
    for(int i = 1; i <= k; i++) {
        while(Hd <= Tl && A[Q[Tl]] <= A[i]) Tl--;
        Q[++Tl] = i;
    }
    printf("%d ", A[Q[Hd]]);
    for(int i = k+1; i <= n; i++) {
        while(Hd <= Tl && Q[Hd] < i-k+1) Hd++;
        while(Hd <= Tl && A[Q[Tl]] <= A[i]) Tl--;
        Q[++Tl] = i;
        printf("%d ", A[Q[Hd]]);
    }
}

inline int readint() {
    int f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    return f*r;
}

int main() {
    n = readint(); k = readint();
    for(int i = 1; i <= n; i++) A[i] = readint();
    SlideMin(); putchar(10); SlideMax();
    return 0;
}
```

5.2 单调栈

6 线段树

6.1 Lazy-Tag

Solution: 暴力拆开式子后（或者根据《重难点手册》的结论），发现要维护区间的 $\sum x_i$, $\sum y_i$, $\sum x_i y_i$, $\sum x_i^2$, 同时要支持区间加和区间设置为 $S+i$ 和T_j. 在线段树上维护 $add_s, add_t, set_s, set_t$，然后推一推式子找出 Lazy-tag 更新主 Tag 的公式即可。几个坑点：

1. add_s, add_t 标记在下推的时候，不能赋值，要累加!!! 累加!!! 累加!!!
2. 只有 set_s, set_t 用 $-\infty$ 来标记不存在，add_s, add_t 必须用 0 标记不存在！不然是给自己找麻烦，多出来各种特判!!!

SegTree/CorrelationAnalyse.cpp

```cpp
/*
    [SDOI2017] 相关分析
    Coded by panda_2134
*/
#include <bits/stdc++.h>
#define LC(o) ((o)*2)
#define RC(o) ((o)*2+1)
#define Mid(x, y) (((x) + (y)) / 2)
using namespace std;

const double eps = 1e-6, NONE = -1e6;
const int MAXN = 1e5;

int dcmp(double x) {
    return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1);
}

struct Info {
    double x, y, xy, x2;
    Info() { x = y = xy = x2 = .0; }
    Info(double a, double b, double c, double d):
        x(a), y(b), xy(c), x2(d) {}
    Info operator+(const Info &rhs) const {
        return Info(x + rhs.x, y + rhs.y, xy + rhs.xy, x2 + rhs.x2);
    }
    Info operator+=(const Info &rhs) { return *this = *this + rhs; }
};

struct Node {
    double x, y, xy, x2, add_s, add_t, set_s, set_t;
    Node() {
        x = y = xy = x2 = .0;
        add_s = add_t = .0;
        set_s = set_t = NONE;
    }
    void clear() { x = y = xy = x2 = .0; }
} nd[(MAXN+10)<<2];

int n, q;
double X[MAXN+10], Y[MAXN+10];

```

```
void Maintain(int o, double L, double R) {
    if(dcmp(nd[o].set_s - NONE) == 0) {
        // no set (if set_s exists, then set_t exists, and vice versa)
        assert(dcmp(nd[o].set_t - NONE) == 0);
        nd[o].clear();
        if(L != R) {
            nd[o].x = nd[LC(o)].x + nd[RC(o)].x;
            nd[o].y = nd[LC(o)].y + nd[RC(o)].y;
            nd[o].xy = nd[LC(o)].xy + nd[RC(o)].xy;
            nd[o].x2 = nd[LC(o)].x2 + nd[RC(o)].x2;
        }
    } else {
        nd[o].x2 = (R-L+1) * nd[o].set_s * nd[o].set_s
            + R * (R+1) * (2*R+1) / 6 - L * (L-1) * (2*L-1) / 6
            + nd[o].set_s * (L+R) * (R-L+1);
        nd[o].xy = (R-L+1) * nd[o].set_s * nd[o].set_t
            + (nd[o].set_s + nd[o].set_t) * (L+R) * (R-L+1) / 2
            + R * (R+1) * (2*R+1) / 6 - L * (L-1) * (2*L-1) / 6;
        nd[o].x = (R-L+1) * nd[o].set_s + (L+R) * (R-L+1) / 2;
        nd[o].y = (R-L+1) * nd[o].set_t + (L+R) * (R-L+1) / 2;
    }
    nd[o].x2 += 2 * nd[o].add_s * nd[o].x + (R-L+1) * nd[o].add_s * nd[o].add_s;
    nd[o].xy += nd[o].add_t * nd[o].x
             + nd[o].add_s * nd[o].y + (R-L+1) * nd[o].add_s * nd[o].add_t;
    nd[o].x += (R-L+1) * nd[o].add_s;
    nd[o].y += (R-L+1) * nd[o].add_t; // update last
}

void Pushdown(int o) {
    if(dcmp(nd[o].set_s - NONE) != 0) { // mark exist
        assert(dcmp(nd[o].set_t - NONE) != 0);
        nd[LC(o)].set_s = nd[RC(o)].set_s = nd[o].set_s;
        nd[LC(o)].set_t = nd[RC(o)].set_t = nd[o].set_t;
        nd[LC(o)].add_s = nd[RC(o)].add_s = .0;
        nd[LC(o)].add_t = nd[RC(o)].add_t = .0;
        nd[o].set_s = NONE;
        nd[o].set_t = NONE;
    }
    if(dcmp(nd[o].add_s) != 0) {
        nd[LC(o)].add_s += nd[o].add_s; //add 标记要累加!!!!!!!!!!!
        nd[RC(o)].add_s += nd[o].add_s;
        nd[o].add_s = .0;
    }
    if(dcmp(nd[o].add_t) != 0) {
        nd[LC(o)].add_t += nd[o].add_t;
        nd[RC(o)].add_t += nd[o].add_t;
        nd[o].add_t = .0;
    }
}

Info Query(int o, int L, int R, int qL, int qR) {
    Maintain(o, L, R);
    if(qL <= L && R <= qR)
        return Info(nd[o].x, nd[o].y, nd[o].xy, nd[o].x2);
    else {
        Info ret;
        Pushdown(o);
        if(qL <= Mid(L, R)) ret += Query(LC(o), L, Mid(L, R), qL, qR);
        else Maintain(LC(o), L, Mid(L, R));
        if(qR >= Mid(L, R)+1) ret += Query(RC(o), Mid(L, R)+1, R, qL, qR);
        else Maintain(RC(o), Mid(L, R)+1, R);
        return ret;
    }
}

void BuildTree(int o, int L, int R) {
    if(L == R) {
        nd[o].add_s = X[L];
        nd[o].add_t = Y[L];
    } else {
        BuildTree(LC(o), L, Mid(L, R));
        BuildTree(RC(o), Mid(L, R)+1, R);
    }
    Maintain(o, L, R);
}

void Add(int o, int L, int R, int qL, int qR, double S, double T) {
    if(qL <= L && R <= qR) {
        nd[o].add_s += S;
        nd[o].add_t += T;
    } else {
        Pushdown(o);
        if(qL <= Mid(L, R)) Add(LC(o), L, Mid(L, R), qL, qR, S, T);
        else Maintain(LC(o), L, Mid(L, R));
        if(qR >= Mid(L, R)+1) Add(RC(o), Mid(L, R)+1, R, qL, qR, S, T);
        else Maintain(RC(o), Mid(L, R)+1, R);
    }
    Maintain(o, L, R);
}

void Set(int o, int L, int R, int qL, int qR, double S, double T) {
    if(qL <= L && R <= qR) {
        nd[o].add_s = nd[o].add_t = .0; // override 'add' mark
        nd[o].set_s = S;
        nd[o].set_t = T;
    } else {
        Pushdown(o);
        if(qL <= Mid(L, R)) Set(LC(o), L, Mid(L, R), qL, qR, S, T);
        else Maintain(LC(o), L, Mid(L, R));
        if(qR >= Mid(L, R)+1) Set(RC(o), Mid(L, R)+1, R, qL, qR, S, T);
        else Maintain(RC(o), Mid(L, R)+1, R);
    }
    Maintain(o, L, R);
}

void init() {
    scanf("%d%d", &n, &q);
    for(int i = 1; i <= n; i++)
        scanf("%lf", &X[i]);
    for(int i = 1; i <= n; i++)
        scanf("%lf", &Y[i]);
    BuildTree(1, 1, n);
}

```

```
void work() {
    int op, L, R; double S, T;
    Info res;
    while(q--) {
        scanf("%d", &op);
        switch(op) {
            case 1:
                scanf("%d%d", &L, &R);
                res = Query(1, 1, n, L, R);
                printf("%.12lf\n",
                    (res.xy - res.x * res.y / (R-L+1)) / (res.x2 - res.x * res.x / (R-L+1)));
                break;
            case 2:
                scanf("%d%d%lf%lf", &L, &R, &S, &T);
                Add(1, 1, n, L, R, S, T);
                break;
            case 3:
                scanf("%d%d%lf%lf", &L, &R, &S, &T);
                Set(1, 1, n, L, R, S, T);
                break;
        }
    }
}

int main() {
    init(); work();
    return 0;
}
```

6.2 动态开点线段树

6.3 可持久化线段树

7 离线二维数点

7.1 带修改

7.1.1 静态：线段树 + 扫描线

7.1.2 动态：CDQ 分治

8 在线二维数点

8.0.1 动态：二维线段树

时间复杂度 插入$O(\lg^2 n)$ – 查询$O(\lg n)$ 空间复杂度 $O(n^2)$

8.0.2 动态：树状数组套动态开点线段树

8.0.3 动态：树状数组套平衡树

9 平衡树

9.1 Treap

BalancedTree/Treap.cpp

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

struct Node *null, *rt;
struct Node {
    int v, r, sz, cnt;
    Node *ch[2];
    Node(int v_) {
        v = v_; r = rand(); sz = cnt = 1;
        ch[0] = ch[1] = null;
    }
    int cmp(int val) {
        return val == v ? -1 : (val > v ? 1 : 0);
    }
    void maintain() {
        if(this == null) return;
        sz = ch[0]->sz + ch[1]->sz + cnt;
    }
};

void init_null() {
    null = new Node(0); null->sz = null->cnt = 0;
    rt = null;
}

void rotate(Node* &o, int d) {
    Node* k = o->ch[d^1];
    o->ch[d^1] = k->ch[d];
    k->ch[d] = o;
    o->maintain(); k->maintain();
    o = k;
}

void insert(Node* &o, int val) {
    if(o == null) {
        o = new Node(val);
        return;
    } else {
        int d = o->cmp(val);
        if(d == -1) {
            ++o->cnt; o->maintain();
        } else {
            insert(o->ch[d], val);
            o->maintain();
            if(o->ch[d]->r < o->r) rotate(o, d^1);
        }
    }
}

void erase(Node* &o, int val) {
    int d = o->cmp(val);
    if(d == -1) {
        if(o->cnt == 1) {
            if(o->ch[1] == null) {
                Node* lhs = o->ch[0];
                delete o;
                o = lhs;
            } else if(o->ch[0] == null) {
                Node* rhs = o->ch[1];
                delete o;
                o = rhs;
            } else {
                int d2 = (o->ch[0]->r) > (o->ch[1]->r);
                rotate(o, d2^1);
                erase(o->ch[d2^1], val);
            }
        } else
            --o->cnt;
    } else
        erase(o->ch[d], val);
    o->maintain();
}

Node* kth(Node* o, int k) {
    int d = (k >= o->ch[0]->sz + 1 && k <= o->ch[0]->sz + o->cnt) ? -1 :
            (k <= o->ch[0]->sz ? 0 : 1);
    if(d == -1) return o;
    if(d == 1) k -= (o->sz - o->ch[1]->sz);
    return kth(o->ch[d], k);
}

int get_rank(Node* o, int val) {
    if(o == null) return 1;
    int d = o->cmp(val);
    if(d == -1) return o->ch[0]->sz + 1;
    return get_rank(o->ch[d], val) + d * (o->sz - o->ch[1]->sz);
}

Node* find(Node* o, int val) {
    if(o == null) return o;
    int d = o->cmp(val);
    if(d == -1) return o;
    else return find(o->ch[d], val);
}

Node* pre(int val) {
    int rk = get_rank(rt, val);
    return rk != 1 ? kth(rt, rk-1) : null;
}

Node* succ(int val) {
    int rk = get_rank(rt, val);  // !!!!!!!!!
    return rk != (rt->sz) ? kth(rt, rk+find(rt, val)->cnt) : null;
}

inline int readint() {
    int f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
```

```cpp
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    return f*r;
}

int main() {
    srand(66623333);
    int q, op, x;
    init_null();
    q = readint();
    while(q--) {
        op = readint();
        switch(op) {
            case 1:
                x = readint(); insert(rt, x);
                break;
            case 2:
                x = readint(); erase(rt, x);
                break;
            case 3:
                x = readint(); insert(rt, x);
                printf("%d\n", get_rank(rt, x));
                erase(rt, x);
                break;
            case 4:
                x = readint();
                printf("%d\n", kth(rt, x)->v);
                break;
            case 5:
                x = readint(); insert(rt, x);
                assert(pre(x) != null);
                printf("%d\n", pre(x)->v);
                erase(rt, x);
                break;
            case 6:
                x = readint(); insert(rt, x);
                assert(succ(x) != null);
                printf("%d\n", succ(x)->v);
                erase(rt, x);
                break;
        }
    }
    return 0;
}
```

9.2 Splay

BalancedTree/Splay.cpp

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

const int MAXN = 1e5;

struct Node *null, *rt;
struct Node {
    int v, sz; bool flip;
    Node* ch[2];
    Node(int v_) { v = v_, sz = 1; flip = false; ch[0] = ch[1] = null; }
    int cmp(int k) {
        return k == ch[0]->sz + 1 ? -1 : (k > ch[0]->sz + 1 ? 1 : 0);
    }
    void rev() {
        if(this == null) return;
        flip ^= 1;
    }
    void maintain() {
        if(this == null) return;
        sz = ch[0]->sz + ch[1]->sz + 1;
    }
    void pushdown() {
        if(flip) {
            flip = false;
            ch[0]->rev(); ch[1]->rev();
            swap(ch[0], ch[1]);
        }
    }
};
int n, m;

void init_null() {
    null = new Node(0); null->sz = 0;
    rt = null;
}

void rotate(Node* &o, int d) {
    Node* k = o->ch[d^1];
    o->pushdown(); k->pushdown();
    o->ch[d^1] = k->ch[d];
    k->ch[d] = o;
    o->maintain(); k->maintain();
    o = k;
}

void splay(Node* &o, int k) {
    o->pushdown();
    int d = o->cmp(k);
    if(d == 1) k -= (o->ch[0]->sz + 1);
    if(d != -1) {
        Node* p = o->ch[d];
        p->pushdown();
        int d2 = p->cmp(k);
        if(d2 == 1) k -= (p->ch[0]->sz + 1);
        if(d2 != -1) {
            splay(p->ch[d2], k);
            if(d == d2) rotate(o, d^1);
            else rotate(o->ch[d], d);
        }
        rotate(o, d^1);
    }
}

Node* merge(Node* lhs, Node* rhs) {
    splay(lhs, lhs->sz);
    lhs->pushdown();
    lhs->ch[1] = rhs;
    lhs->maintain();
    return lhs;
}

void split(Node* o, int k, Node* &lhs, Node* &rhs) {
    splay(o, k);
    o->pushdown();
    lhs = o, rhs = o->ch[1];
    o->ch[1] = null; o->maintain(); // 赋值后再断开和右儿子的连接，并维护 sz!
}

void traverse(Node* o) {
    if(o == null) return;
    o->pushdown();
    traverse(o->ch[0]);
    if(o->v > 0) printf("%d ", o->v);
    traverse(o->ch[1]);
}

inline int readint() {
    int f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    return f*r;
}

int main() {
    int l, r; Node *a, *b, *c;
    init_null();

    n = readint(); m = readint();

    rt = new Node(0); // dummy
    for(int i = 1; i <= n; i++) rt = merge(rt, new Node(i));
    rt = merge(rt, new Node(0)); // dummy

    while(m--) {
        l = readint() + 1, r = readint() + 1;
        split(rt, l-1, a, b); split(b, r-l+1, b, c);
        b->rev();
        rt = merge(a, merge(b, c));
    }

    traverse(rt);
    return 0;
}
```

10 动态树

10.1 Link-cut Tree

11 字符串

11.1 KMP 字符串匹配

1-indexed

11.2 AC 自动机

0-indexed

String/ACAutomaton.cpp

```cpp
#include <bits/stdc++.h>
#define CLEAR(x) memset((x), 0, sizeof(x))
using namespace std;

const int SIGMA = 26, MAX_TEMP_LEN = 70, MAXN = 150,
MAX_LEN = 1e6, MAX_NODE = MAXN * MAX_TEMP_LEN;

int N, sz, ch[MAX_NODE + 10][SIGMA + 2], f[MAX_NODE + 10], last[MAX_NODE+10],
    val[MAX_NODE + 10], found_cnt[MAX_NODE+10];
char str[MAX_LEN+10], tpl[MAXN+10][MAX_TEMP_LEN+10];
unordered_map<string, int> ms;

inline int idx(char c) { return c - 'a' + 1; }

void insert(char *str) {
    int u = 0, len = strlen(str);
    for(int i = 0; i < len; i++) {
        int c = idx(str[i]);
        if(!ch[u][c]) ch[u][c] = ++sz;
        u = ch[u][c];
    }
    ms[string(str)] = u;
    ++val[u];
}

void get_fail() {
    queue<int> Q;
    f[0] = 0;
    for(int c = 1; c <= SIGMA; c++) if(ch[0][c]) {
        int v = ch[0][c];
        f[v] = last[v] = 0;
        Q.push(v);
    }
    while(!Q.empty()) {
        int u = Q.front(); Q.pop();
        for(int c = 1; c <= SIGMA; c++) {
            int v = ch[u][c];
            if(!v) {
                ch[u][c] = ch[f[u]][c];
                continue;
            }

            Q.push(v);

            int u2 = f[u];
            while(u2 && !ch[u2][c]) u2 = f[u2];
            f[v] = ch[u2][c];
            last[v] = val[f[v]] ? f[v] : last[f[v]];
        }
    }
}

void found(int u) {
    for(; u; u = last[u])
        found_cnt[u] += val[u];
}

void search(char *str) {
    int u = 0, len = strlen(str);
    for(int i = 0; i < len; i++) {
        int c = idx(str[i]);
        u = ch[u][c];
        if(val[u]) found(u);
        else if(last[u]) found(last[u]);
    }
}

inline void readstr(char *str) {
    char c=getchar(); int p=0;
    while(!isalnum(c) && !ispunct(c)) c = getchar();
    while(isalnum(c) || ispunct(c)) {
        str[p++] = c;
        c = getchar();
    }
    str[p++] = '\0';
}

int main() {
    while(true) {
        int ans = 0;
        sz = 0; CLEAR(ch); CLEAR(f); CLEAR(found_cnt);
        CLEAR(last); CLEAR(tpl); CLEAR(val); CLEAR(str);

        scanf("%d", &N); if(N == 0) break;
        for(int i = 1; i <= N; i++) {
            readstr(tpl[i]); insert(tpl[i]);
        }
        get_fail();

        readstr(str); search(str);

        for(int i = 0; i <= sz; i++)
            ans = max(ans, found_cnt[i]);
        printf("%d\n", ans);
        for(int i = 1; i <= N; i++)
            if(found_cnt[ms[string(tpl[i])]] == ans)
                printf("%s\n", tpl[i]);
    }
    return 0;
}
```

11.3 后缀数组

0-indexed

String/SuffixArray.cpp

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

const int MAXLEN = 1e6, SIGMA = 100;

inline int idx(char c) {
    if(!c) return 0;
    else if(isdigit(c)) return c - '0' + 1;
    else if(isupper(c)) return c - 'A' + 1 + 10;
    else if(islower(c)) return c - 'a' + 1 + 10 + 26;
    else throw "Invalid Character";
}

struct SuffixArray {
    int sa[MAXLEN+10], rk[MAXLEN+10], buf[3][MAXLEN+10], height[MAXLEN+10], c[MAXLEN+10];
    void build_sa(char *s, int len) {
        int m = SIGMA + 10, n = len + 1, *x = buf[0], *y = buf[1];
        for(int i = 0; i < m; i++) c[i] = 0;
        for(int i = 0; i < n; i++) ++c[x[i] = idx(s[i])];
        for(int i = 1; i < m; i++) c[i] += c[i-1];
        for(int i = n-1; i >= 0; i--) sa[--c[x[i]]] = i;
        for(int k = 1; k <= n; k <<= 1) {
            int p = 0;
            for(int i = n-k; i < n; i++) y[p++] = i;
            for(int i = 0; i < n; i++) if(sa[i] >= k) y[p++] = sa[i] - k;
            for(int i = 0; i < m; i++) c[i] = 0;
            for(int i = 0; i < n; i++) ++c[x[y[i]]];
            for(int i = 1; i < m; i++) c[i] += c[i-1];
            for(int i = n-1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
            swap(x, y);
            p = 1, x[sa[0]] = 0;
            for(int i = 1; i < n; i++)
                x[sa[i]] = (y[sa[i]] == y[sa[i-1]] && y[sa[i] + k] == y[sa[i-1] + k] ? p-1 : p++);
            if(p >= n) break;
            m = p;
        }
        memcpy(rk, x, sizeof(rk));
        int k = 0;
        for(int i = 0; i < n; i++) {
            if(!rk[i]) continue;
            if(k) k--;
            int j = sa[rk[i]-1];
            while(s[i+k] == s[j+k]) k++;
            height[rk[i]] = k;
        }
    }
} SA;
inline void readstr(char* str) {
    char c=getchar(); int p=0;
    while(!isalnum(c) && !ispunct(c)) c=getchar();
    while(isalnum(c) || ispunct(c)) {
        str[p++] = c;
        c = getchar();
    }
    str[p++] = '\0';
}

int len;
char str[MAXLEN+10];

int main() {
    readstr(str); len = strlen(str);
    SA.build_sa(str, len);
    for(int i = 1; i <= len; i++)
        printf("%d ", SA.sa[i]+1);
    return 0;
}
```

12 Miscellaneous

12.1 ST 表

12.2 Fenwick Tree

Misc/BIT.cpp

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

const int MAXN = 5e5;

int n, q, a[MAXN+10];

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

void add(int p, int val) {
    while(p <= n) {
        a[p] += val;
        p += lowbit(p);
    }
}

int query(int p) {
    int ret = 0;
    while(p > 0) {
        ret += a[p];
        p -= lowbit(p);
    }
    return ret;
}

inline int readint() {
    int f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    return f*r;
}

int main() {
    n = readint(); q = readint();
    for(int i = 1; i <= n; i++)
        add(i, readint());
    while(q--) {
        int op, x, y;
        op = readint(); x = readint(); y = readint();
        switch(op) {
            case 1:
                add(x, y);
                break;
            case 2:
                printf("%d\n", query(y) - query(x-1));
                break;
        }
    }
    return 0;
}
```

12.3 左偏树

Misc/LefiestTree.cpp

```
#include <bits/stdc++.h>
#define fst first
#define snd second
using namespace std;

typedef pair<int, int> pii;
const int MAXN = 1e5;

extern struct Node *null;
struct Node {
    pii val; int dist;
    Node *ch[2];
    Node() {
        ch[0] = ch[1] = null;
        dist = -1; //!!!
    };
    Node(pii v_) {
        ch[0] = ch[1] = null;
        dist = -1; val = v_;
    }
} Tnull, *null=&Tnull, *rt[MAXN+10];
int n, q, fa[MAXN+10], del[MAXN+10];

int get_fa(int x) { return x == fa[x] ? x : fa[x] = get_fa(fa[x]); }
void union_set(int x, int y) { fa[get_fa(y)] = get_fa(x); } // 顺序

Node* merge(Node* lhs, Node* rhs) {
    if(lhs == null) return rhs;
    else if(rhs == null) return lhs;
    else {
        if(lhs->val > rhs->val) swap(lhs, rhs);
        lhs->ch[1] = merge(lhs->ch[1], rhs);
        if(lhs->ch[0]->dist < lhs->ch[1]->dist)
            swap(lhs->ch[0], lhs->ch[1]);
        lhs->dist = lhs->ch[1]->dist + 1; // 距离应该是左右儿子的最小 dist + 1 （定义）
        return lhs;
    }
}

void pop(Node* &o) {
    Node *lhs = o->ch[0], *rhs = o->ch[1];
    delete o;
    o = merge(lhs, rhs);
}

void push(Node* &o, pii val) {
    o = merge(o, new Node(val));
}

inline int readint() {
```

```cpp
    int f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    return f*r;
}

int main() {
    int op, x, y;
    n = readint(); q = readint();
    for(int i = 1; i <= n; i++) {
        fa[i] = i;
        rt[i] = new Node(make_pair(readint(), i));
    }
    while(q--) {
        op = readint();
        switch(op) {
            case 1:
                x = readint(); y = readint();
                if(del[x] || del[y] || get_fa(x) == get_fa(y))
                    continue;
                rt[get_fa(x)] = merge(rt[get_fa(x)], rt[get_fa(y)]);
                union_set(x, y);
                break;
            case 2:
                x = readint();
                if(del[x]) puts("-1");
                else {
                    pii u = rt[get_fa(x)]->val;
                    printf("%d\n", u.fst);
                    del[u.snd] = true;
                    pop(rt[get_fa(x)]);
                }
                break;
        }
    }
    return 0;
}
```

13 悬线法

13.1 Algorithm 1

13.2 Algorithm 2

14 莫队

14.1 普通莫队

14.2 带修改莫队

15　分块相关

15.1　分块

例题：教主的魔法

15.2　区间众数

16　多项式

（为何比别人多了 4 倍常数……）

16.1　快速傅里叶变换

Poly/FFT.cpp

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

const int MAXN = 4e6;
const double PI = 3.141592653589793238462643383;

struct cpx {
    double real, imag;
    cpx() { real = imag = .0; }
    cpx(double x) { real = x, imag = .0; }
    cpx(double x, double y) { real = x, imag = y; }
    friend cpx operator+(const cpx &lhs, const cpx &rhs) {
        return cpx(lhs.real + rhs.real, lhs.imag + rhs.imag);
    }
    friend cpx operator-(const cpx &lhs, const cpx &rhs) {
        return cpx(lhs.real - rhs.real, lhs.imag - rhs.imag);
    }
    friend cpx operator*(const cpx &lhs, const cpx &rhs) {
        return cpx(lhs.real * rhs.real - lhs.imag * rhs.imag,
            lhs.imag * rhs.real + lhs.real * rhs.imag);
    }
    cpx operator*=(const cpx &rhs) { return (*this) = (*this) * rhs; }
    cpx conj() const { return cpx(real, -imag); }
    friend cpx operator/(const cpx &lhs, double rhs) {
        return cpx(lhs.real / rhs, lhs.imag / rhs);
    }
    friend cpx operator/(const cpx &lhs, const cpx &rhs) {
        cpx ret = lhs * rhs.conj();
        ret.real /= (rhs.real * rhs.real + rhs.imag * rhs.imag);
        ret.imag /= (rhs.real * rhs.real + rhs.imag * rhs.imag);
        return ret;
    }
    cpx operator/=(const cpx &rhs) { return (*this) = (*this) / rhs; }
};

int n, m, R[MAXN+10]; double A[MAXN+10], B[MAXN+10], C[MAXN+10];

inline cpx get_rt(int step, bool inv) { // rotation factor
    return inv ? cpx(std::cos(2*PI / step), -std::sin(2*PI / step)) :
                 cpx(std::cos(2*PI / step), std::sin(2*PI / step));
}

void fft(cpx A[], int len, bool inv) {
    for(int i = 0; i < len; i++)
        if(R[i] > i) std::swap(A[i], A[R[i]]);
    for(int step = 1; step < len; step <<= 1) {
        for(int i = 0; i < len; i += (step<<1)) {
            cpx omega = 1, rt = get_rt(step<<1, inv);
            for(int j = 0; j < step; j++, omega *= rt) {
                    cpx t = omega * A[i+j+step];
                    A[i+j+step] = A[i+j] - t;
                    A[i+j] = A[i+j] + t;
                }
            }
        }
    if(inv)
        for(int i = 0; i < len; i++) A[i] /= len;
}

void conv(double dA[], double dB[], int deg1, int deg2, double dC[]) { // deg: 输入多项式的度数
    int len;
    static cpx A[MAXN+10], B[MAXN+10], C[MAXN+10];
    for(len = 1; len < deg1+deg2+1; len <<= 1); // 考虑乘完后的长度
    for(int i = 0; i < len; i++) {
        A[i] = dA[i], B[i] = dB[i];
    }

    R[0] = 0;
    for(int i = 1; i < len; i++)
        R[i] = ((R[i>>1]>>1) | (len >> (i&1))) & (len-1);

    fft(A, len, false); fft(B, len, false);
    for(int i = 0; i < len; i++) C[i] = A[i] * B[i];
    fft(C, len, true);
    for(int i = 0; i < len; i++) dC[i] = C[i].real;
}

inline int readint() {
    int f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    return f*r;
}

int main() {
    n = readint(); m = readint();
    for(int i = 0; i <= n; i++) A[i] = readint();
    for(int i = 0; i <= m; i++) B[i] = readint();
    conv(A, B, n, m, C);
    for(int i = 0; i <= n+m; i++) std::printf("%d ", int(round(C[i])));
    return 0;
}
```

16.2 快速数论变换

998244353 的原根是 3

Poly/NTT.cpp

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

typedef long long int64;
const int MAXN = 4e6, MOD = 998244353, G = 3;

```

```cpp
int n, m, A[MAXN+10], B[MAXN+10], C[MAXN+10], R[MAXN+10];

int64 fastpow(int64 a, int64 x) {
    int64 ret = 1; a %= MOD;
    while(x) {
        if(x & 1) ret = ret * a % MOD;
        x >>= 1; a = a * a % MOD;
    }
    return ret;
}

int get_rt(int step, bool inv) {
    return !inv ? fastpow(G, (MOD-1) / step) : fastpow(G, (MOD-1) / step * (step-1));
}

void ntt(int A[], int len, bool inv) {
    for(int i = 0; i < len; i++)
        if(R[i] > i) swap(A[R[i]], A[i]);
    for(int step = 1; step < len; step <<= 1) {
        for(int i = 0; i < len; i += (step << 1)) {
            int64 omega = 1, rt = get_rt(step << 1, inv);
            for(int j = 0; j < step; j++, omega = (omega * rt) % MOD) {
                int t = omega * A[i+j+step] % MOD;
                A[i+j+step] = ((A[i+j] - t) % MOD + MOD) % MOD;
                A[i+j] = (A[i+j] + t) % MOD;
            }
        }
    }
    if(inv) {
        int64 inv_ele = fastpow(len, MOD-2);
        for(int i = 0; i < len; i++) A[i] = (A[i] * inv_ele) % MOD;
    }
}

void conv(int A[], int B[], int deg1, int deg2, int C[]) {
    int len; for(len = 1; len < deg1+deg2+1; len <<= 1);
    R[0] = 0;
    for(int i = 1; i < len; i++)
        R[i] = ((R[i>>1]>>1) | (len >> (i & 1))) & (len-1);
    ntt(A, len, false); ntt(B, len, false);
    for(int i = 0; i < len; i++) C[i] = int64(A[i]) * B[i] % MOD;
    ntt(A, len, true); ntt(B, len, true); ntt(C, len, true);
}

template<typename T>
inline void readint(T &x) {
    int f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    x = f*r;
}

template<typename T>
inline void writeint(T &x) {
    static char buf[32];
    char *ptr = buf;
    if(x < 0) putchar('-'), x = -x;
    do {
        *ptr++ = (x % 10) + '0';
        x /= 10;
    } while(x);
    do
        putchar(*--ptr);
    while(ptr != buf);
}

int main() {
    readint(n); readint(m);
    for(int i = 0; i <= n; i++) readint(A[i]);
    for(int i = 0; i <= m; i++) readint(B[i]);
    conv(A, B, n, m, C);
    for(int i = 0; i <= n+m; i++) writeint(C[i]), putchar(' ');
    return 0;
}
```

17 数论

17.1 线性求逆元

推导 令 $p = ki + r (0 \le r < i)$

则 $0 \equiv ki + r \pmod p$

$\Rightarrow ki \cdot i^{-1} r^{-1} + r \cdot i^{-1} r^{-1} \equiv 0$

$\Rightarrow i^{-1} \equiv -k \cdot r^{-1} \equiv p - \left\lfloor \frac{p}{i} \right\rfloor + p \bmod i \pmod p$

Math/inv.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 3e6;
int n, p, inv[MAXN+10];
int main() {
    cin >> n >> p;
    inv[1] = 1; printf("%d\n", inv[1]);
    for(int i = 2; i <= n; i++)
        printf("%d\n", inv[i] = (p - (long long)(p / i) * inv[p % i] % p) % p);
    return 0;
}
```

17.2 线性筛

17.2.1 求 $\varphi(n)$

17.2.2 求 $\mu(n)$

17.2.3 求 $d(n)$

17.3 扩展欧几里得定理

17.4 中国剩余定理

17.5 扩展欧拉定理

17.6 Lucas 定理