All-in at the River

Standard Code Library

Shanghai Jiao Tong University

Desprado2

fstqwq

AntiLeaf

“

我清楚的记得那是 icpc2020 银川
当时我看着队友差 100 分钟出线，趴在魄罗上泣不成声
这个画面我永生难忘
那一刻我在想，如果能再给我一次机会，我一定要赢回所有
如今沈阳就在眼前，我必须考虑这是不是我此生仅有的机会
重振交大荣光，我辈义不容辞

”

Contents

1. 图论

1.1 最小生成树

1.1.1 动态最小生成树

```cpp
// 动态最小生成树的离线算法比较容易▯而在线算法通常极为复杂
// 一个跑得比较快的离线做法是对时间分治▯在每层分治时找出一定在/不在MST上的边▯只带着不确定边继续递归
// 简单起见▯找确定边的过程用Kruskal算法实现▯过程中的两种重要操作如下▯
// - Reduction▯待修改边标为+INF▯跑MST后把非树边删掉▯减少无用边
// - Contraction▯待修改边标为-INF▯跑MST后缩除待修改边之外的所有MST边▯计算必须边
// 每轮分治需要Reduction-Contraction▯借此减少不确定边▯从而保证复杂度
// 复杂度证明▯假设当前区间有k条待修改边▯n和m表示点数和边数▯那么最坏情况下R-C的效果为(n, m) -> (n, n + k - 1) -> (k + 1, 2k)


// 全局结构体与数组定义
struct edge { //边的定义
    int u, v, w, id; // id表示边在原图中的编号
    bool vis; // 在Kruskal时用▯记录这条边是否是树边
    bool operator < (const edge &e) const { return w < e.w; }
} e[20][maxn], t[maxn]; // 为了便于回滚▯在每层分治存一个副本


// 用于存储修改的结构体▯表示第id条边的权值从u修改为v
struct A {
    int id, u, v;
} a[maxn];


int id[20][maxn]; // 每条边在当前图中的编号
int p[maxn], size[maxn], stk[maxn], top; // p和size是并查集数组▯stk是用来撤销的栈
int n, m, q; // 点数▯边数▯修改数


// 方便起见▯附上可能需要用到的预处理代码
for (int i = 1; i <= n; i++) { // 并查集初始化
    p[i] = i;
    size[i] = 1;
}

for (int i = 1; i <= m; i++) { // 读入与预标号
    scanf("%d%d%d", &e[0][i].u, &e[0][i].v, &e[0][i].w);
    e[0][i].id = i;
    id[0][i] = i;
}

for (int i = 1; i <= q; i++) { // 预处理出调用数组
    scanf("%d%d", &a[i].id, &a[i].v);
    a[i].u = e[0][a[i].id].w;
    e[0][a[i].id].w = a[i].v;
}

for(int i = q; i; i--)
    e[0][a[i].id].w = a[i].u;

CDQ(1, q, 0, m, 0); // 这是调用方法


// 分治主过程 O(nlog^2n)
// 需要调用Reduction和Contraction
void CDQ(int l, int r, int d, int m, long long ans) { // CDQ分治
    if (l == r) { // 区间长度已减小到1▯输出答案▯退出
        e[d][id[d][a[l].id]].w = a[l].v;
        printf("%lld\n", ans + Kruskal(m, e[d]));
        e[d][id[d][a[l].id]].w=a[l].u;
        return;
    }

    int tmp = top;

    Reduction(l, r, d, m);
    ans += Contraction(l, r, d, m); // R-C

    int mid = (l + r) / 2;

    copy(e[d] + 1, e[d] + m + 1, e[d + 1] + 1);
    for (int i = 1; i <= m; i++)
        id[d + 1][e[d][i].id] = i; // 准备好下一层要用的数组

    CDQ(l, mid, d + 1, m, ans);

    for (int i = l; i <= mid; i++)
        e[d][id[d][a[i].id]].w = a[i].v; // 进行左边的修改

    copy(e[d] + 1, e[d] + m + 1, e[d + 1] + 1);
    for (int i = 1; i <= m; i++)
        id[d + 1][e[d][i].id] = i; // 重新准备下一层要用的数组

    CDQ(mid + 1, r, d + 1, m, ans);

    for (int i = top; i > tmp; i--)
        cut(stk[i]);//撤销所有操作
    top = tmp;
}


// Reduction▯减少无用边▯▯待修改边标为+INF▯跑MST后把非树边删掉▯减少无用边
// 需要调用Kruskal
void Reduction(int l, int r, int d, int &m) {
    for (int i = l; i <= r; i++)
        e[d][id[d][a[i].id]].w = INF;//待修改的边标为INF

    Kruskal(m, e[d]);

    copy(e[d] + 1, e[d] + m + 1, t + 1);

    int cnt = 0;
    for (int i = 1; i <= m; i++)
        if (t[i].w == INF || t[i].vis){ // 非树边扔掉
            id[d][t[i].id] = ++cnt; // 给边重新编号
            e[d][cnt] = t[i];
        }

    for (int i = r; i >= l; i--)
        e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边改回去

    m=cnt;
}
```

```cpp


// Contraction，缩必须边，，待修改边标为-INF，跑MST后缩除待修改边之外的所有树边
// 返回缩掉的边的总权值
// 需要调用Kruskal
long long Contraction(int l, int r, int d, int &m) {
    long long ans = 0;

    for (int i = l; i <= r; i++)
        e[d][id[d][a[i].id]].w = -INF; // 待修改边标为-INF

    Kruskal(m, e[d]);
    copy(e[d] + 1, e[d] + m + 1, t + 1);

    int cnt = 0;
    for (int i = 1; i <= m ; i++) {

        if (t[i].w != -INF && t[i].vis) { // 必须边
            ans += t[i].w;
            mergeset(t[i].u, t[i].v);
        }
        else { // 不确定边
            id[d][t[i].id]=++cnt;
            e[d][cnt]=t[i];
        }
    }

    for (int i = r ; i >= l; i--) {
        e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边改回去
        e[d][id[d][a[i].id]].vis = false;
    }

    m = cnt;

    return ans;
}


// Kruskal算法 O(mlogn)
// 方便起见，这里直接沿用进行过缩点的并查集，在过程结束后撤销即可
long long Kruskal(int m, edge *e) {
    int tmp = top;
    long long ans = 0;

    sort(e + 1, e + m + 1); // 比较函数在结构体中定义过了

    for (int i = 1; i <= m; i++) {
        if (findroot(e[i].u) != findroot(e[i].v)) {
            e[i].vis = true;
            ans += e[i].w;
            mergeset(e[i].u, e[i].v);
        }
        else
            e[i].vis = false;
    }

    for(int i = top; i > tmp; i--)
        cut(stk[i]); // 撤销所有操作
    top = tmp;

    return ans;
}


// 以下是并查集相关函数
int findroot(int x) { // 因为需要撤销，不写路径压缩
    while (p[x] != x)
        x = p[x];

    return x;
}

void mergeset(int x, int y) { // 按size合并，如果想跑得更快就写一个按秩合并
    x = findroot(x); // 但是按秩合并要再开一个栈记录合并之前的秩
    y = findroot(y);

    if (x == y)
        return;

    if (size[x] > size[y])
        swap(x, y);

    p[x] = y;
    size[y] += size[x];
    stk[++top] = x;
}

void cut(int x) { // 并查集撤销
    int y = x;

    do
        size[y = p[y]] -= size[x];
    while (p[y]! = y);

    p[x] = x;
}
```

1.2 最短路

1.2.1 k短路

```cpp
//注意这是个多项式算法，在k比较大时很有优势，但k比较小时最好还是用A*
//DAG和有环的情况都可以，有重边或自环也无所谓，但不能有零环
//以下代码以Dijkstra+可持久化左偏树为例

const int maxn=1005,maxe=10005,maxm=maxe*30;//点数，边数，左偏树结点数

//需要用到的结构体定义
struct A{//用来求最短路
    int x,d;
    A(int x,int d):x(x),d(d){}
    bool operator<(const A &a)const{return d>a.d;}
};

struct node{//左偏树结点
    int w,i,d;//i：最后一条边的编号 d：左偏树附加信息
    node *lc,*rc;
    node(){}
    node(int w,int i):w(w),i(i),d(0){}
    void refresh(){d=rc->d+1;}
}null[maxm],*ptr=null,*root[maxn];

struct B{//维护答案用
    int x,w;//x是结点编号，w表示之前已经产生的权值
```

```
    node *rt;//这个答案对应的堆顶□注意可能不等于任何一个
      ↪ 结点的堆
    B(int x,node *rt,int w):x(x),w(w),rt(rt){}
    bool operator<(const B &a)const{return
      ↪ w+rt->w>a.w+a.rt->w;}
};

//全局变量和数组定义
vector<int>G[maxn],W[maxn],id[maxn];//最开始要存反向图□然
  ↪ 后把G清空作为儿子列表
bool vis[maxn],used[maxe];//used表示边是否在最短路树上
int u[maxe],v[maxe],w[maxe];//存下每条边□注意是有向边
int d[maxn],p[maxn];//p表示最短路树上每个点的父边
int n,m,k,s,t;//s,t分别表示起点和终点

//以下是主函数中较关键的部分
for(int i=0;i<=n;i++)root[i]=null;//一定要加上□□□
//(读入&建反向图)
Dijkstra();
//(清空G,W,id)
for(int i=1;i<=n;i++)
    if(p[i]){
        used[p[i]]=true;//在最短路树上
        G[v[p[i]]].push_back(i);
    }
for(int i=1;i<=m;i++){
    w[i]-=d[u[i]]-d[v[i]];//现在的w[i]表示这条边能使路径长
      ↪ 度增加多少
    if(!used[i])
        root[u[i]]=merge(root[u[i]],newnode(w[i],i));
}
dfs(t);
priority_queue<B>heap;
heap.push(B(s,root[s],0));//初始状态是找贡献最小的边加进
  ↪ 去
printf("%d\n",d[s]);//第1短路需要特判
while(--k){//其余k-1短路径用二叉堆维护
    if(heap.empty())printf("-1\n");
    else{
        int x=heap.top().x,w=heap.top().w;
        node *rt=heap.top().rt;
        heap.pop();
        printf("%d\n",d[s]+w+rt->w);
        if(rt->lc!=null||rt->rc!=null)
            heap.push(B(x,merge(rt->lc,rt->rc),w));//pop掉
              ↪ 当前边□换成另一条贡献大一点的边
        if(root[v[rt->i]]!=null)
            heap.push(B(v[rt->i],root[v[rt->i]],w+rt->w));//保
              ↪ 留当前边□往后面再接上另一条边
    }
}
//主函数到此结束

//Dijkstra预处理最短路 O(m\log n)
void Dijkstra(){
    memset(d,63,sizeof(d));
    d[t]=0;
    priority_queue<A>heap;
    heap.push(A(t,0));
    while(!heap.empty()){
        int x=heap.top().x;
        heap.pop();
        if(vis[x])continue;
        vis[x]=true;
        for(int i=0;i<(int)G[x].size();i++)
            if(!vis[G[x][i]]&&d[G[x][i]]>d[x]+W[x][i]){
                d[G[x][i]]=d[x]+W[x][i];
                p[G[x][i]]=id[x][i];
                heap.push(A(G[x][i],d[G[x][i]]));
            }
    }
}

//dfs求出每个点的堆 总计O(m\log n)
//需要调用merge□同时递归调用自身
void dfs(int x){
    root[x]=merge(root[x],root[v[p[x]]]);
    for(int i=0;i<(int)G[x].size();i++)
        dfs(G[x][i]);
}

//包装过的new node() O(1)
node *newnode(int w,int i){
    *++ptr=node(w,i);
    ptr->lc=ptr->rc=null;
    return ptr;
}

//带可持久化的左偏树合并 总计O(\log n)
//递归调用自身
node *merge(node *x,node *y){
    if(x==null)return y;
    if(y==null)return x;
    if(x->w>y->w)swap(x,y);
    node *z=newnode(x->w,x->i);
    z->lc=x->lc;
    z->rc=merge(x->rc,y);
    if(z->lc->d>z->rc->d)swap(z->lc,z->rc);
    z->refresh();
    return z;
}
```

1.3 仙人掌

1.3.1 仙人掌DP

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

using namespace std;

const int maxn = 200005;

struct edge{
    int to, w, prev;
}e[maxn * 2];

vector<pair<int, int> > v[maxn];

vector<long long> d[maxn];

stack<int> stk;

int p[maxn];

bool vis[maxn], vise[maxn * 2];

int last[maxn], cnte;

long long f[maxn], g[maxn], sum[maxn];

int n, m, cnt;

void addedge(int x, int y, int w) {
```

```
    v[x].push_back(make_pair(y, w));
}

void dfs(int x) {

    vis[x] = true;

    for (int i = last[x]; ~i; i = e[i].prev) {
        if (vise[i ^ 1])
            continue;

        int y = e[i].to, w = e[i].w;

        vise[i] = true;

        if (!vis[y]) {
            stk.push(i);
            p[y] = x;
            dfs(y);

            if (!stk.empty() && stk.top() == i) {
                stk.pop();
                addedge(x, y, w);
            }
        }

        else {
            cnt++;

            long long tmp = w;
            while (!stk.empty()) {
                int i = stk.top();
                stk.pop();

                int yy = e[i].to, ww = e[i].w;

                addedge(cnt, yy, 0);

                d[cnt].push_back(tmp);

                tmp += ww;

                if (e[i ^ 1].to == y)
                    break;
            }

            addedge(y, cnt, 0);

            sum[cnt] = tmp;
        }
    }
}

void dp(int x) {

    for (auto o : v[x]) {
        int y = o.first, w = o.second;
        dp(y);
    }

    if (x <= n) {
        for (auto o : v[x]) {
            int y = o.first, w = o.second;

            f[x] += 2 * w + f[y];
        }

        g[x] = f[x];

        for (auto o : v[x]) {
            int y = o.first, w = o.second;

            g[x] = min(g[x], f[x] - f[y] - 2 * w + g[y] +
              ↪ w);
        }
    }
    else {
        f[x] = sum[x];
        for (auto o : v[x]) {
            int y = o.first;

            f[x] += f[y];
        }

        g[x] = f[x];

        for (int i = 0; i < (int)v[x].size(); i++) {
            int y = v[x][i].first;

            g[x] = min(g[x], f[x] - f[y] + g[y] + min(d[x]
              ↪ [i], sum[x] - d[x][i]));
        }
    }
}

signed main() {

    memset(last, -1, sizeof(last));

    ios::sync_with_stdio(false);

    cin >> n >> m;

    cnt = n;

    while (m--) {
        int x, y, w;
        cin >> x >> y >> w;

        e[cnte].to = y;
        e[cnte].w = w;
        e[cnte].prev = last[x];
        last[x] = cnte++;

        e[cnte].to = x;
        e[cnte].w = w;
        e[cnte].prev = last[y];
        last[y] = cnte++;
    }

    dfs(1);
    dp(1);

    cout << g[1] << endl;

    return 0;
}
```

1.4 二分图

1.4.1 KM二分图最大权匹配

```cpp
// KM Weighted Bio-Graph Matching KM二分图最大权匹配
// By AntiLeaf
// O(n^3)

const long long INF = 0x3f3f3f3f3f3f3f3f;

long long w[maxn][maxn], lx[maxn], ly[maxn], slack[maxn];
// 边权 顶标 slack
// 如果要求最大权完美匹配就把不存在的边设为-INF 否则所有
    ↪ 边对0取max

bool visx[maxn], visy[maxn];

int boy[maxn], girl[maxn], p[maxn], q[maxn], head, tail; //
    ↪ p : pre

int n, m, N, e;

// 增广
bool check(int y) {
    visy[y] = true;

    if (boy[y]) {
        visx[boy[y]] = true;
        q[tail++] = boy[y];
        return false;
    }

    while (y) {
        boy[y] = p[y];
        swap(y, girl[p[y]]);
    }

    return true;
}

// bfs每个点
void bfs(int x) {
    memset(q, 0, sizeof(q));
    head = tail = 0;

    q[tail++] = x;
    visx[x] = true;

    while (true) {
        while (head != tail) {
            int x = q[head++];

            for (int y = 1; y <= N; y++)
                if (!visy[y]) {
                    long long d = lx[x] + ly[y] - w[x][y];

                    if (d < slack[y]) {
                        p[y] = x;
                        slack[y] = d;

                        if (!slack[y] && check(y))
                            return;
                    }
                }
        }

        long long d = INF;
        for (int i = 1; i <= N; i++)
            if (!visy[i])
                d = min(d, slack[i]);

        for (int i = 1; i <= N; i++) {
            if (visx[i])
                lx[i] -= d;

            if (visy[i])
                ly[i] += d;
            else
                slack[i] -= d;
        }

        for (int i = 1; i <= N; i++)
            if (!visy[i] && !slack[i] && check(i))
                return;
    }
}

// 主过程
long long KM() {
    for (int i = 1; i <= N; i++) {
        // lx[i] = 0;
        ly[i] = -INF;
        // boy[i] = girl[i] = -1;

        for (int j = 1; j <= N; j++)
            ly[i] = max(ly[i], w[j][i]);
    }

    for (int i = 1; i <= N; i++) {
        memset(slack, 0x3f, sizeof(slack));
        memset(visx, 0, sizeof(visx));
        memset(visy, 0, sizeof(visy));
        bfs(i);
    }

    long long ans = 0;
    for (int i = 1; i <= N; i++)
        ans += w[i][girl[i]];
    return ans;
}

// 为了方便贴上主函数
int main() {

    scanf("%d%d%d", &n, &m, &e);
    N = max(n, m);

    while (e--) {
        int x, y, c;
        scanf("%d%d%d", &x, &y, &c);
        w[x][y] = max(c, 0);
    }

    printf("%lld\n", KM());

    for (int i = 1; i <= n; i++) {
        if (i > 1)
            printf(" ");
        printf("%d", w[i][girl[i]] > 0 ? girl[i] : 0);
    }
    printf("\n");

    return 0;
```

```
}
```

1.5 一般图匹配

1.5.1 高斯消元

```
// Graph Matching Based on Linear Algebra 基于线性代数的一般图匹配 O(n^3)
// By ysf
// 通过题目□UOJ#79 一般图最大匹配

// 这个算法基于Tutte定理和高斯消元□思维难度相对小一些□也更方便进行可行边的判定
// 注意这个算法复杂度是满的□并且常数有点大□而带花树通常是跑不满的
// 以及□根据Tutte定理□如果求最大匹配的大小的话直接输出Tutte矩阵的秩/2即可
// 需要输出方案时才需要再写后面那些乱七八糟的东西


// 复杂度和常数所限□1s之内500已经是这个算法的极限了
const int maxn = 505, p = 1000000007;//p可以是任意10^9以内的质数

// 全局数组和变量定义
int A[maxn][maxn], B[maxn][maxn], t[maxn][maxn], id[maxn], a[maxn];
bool row[maxn] = {false}, col[maxn] = {false};
int n, m, girl[maxn]; // girl是匹配点□用来输出方案

// 为了方便使用□贴上主函数
// 需要调用高斯消元和eliminate
int main() {
    srand(19260817); // 膜蛤专用随机种子□换一个也无所谓

    scanf("%d%d", &n, &m); // 点数和边数
    while (m--) {
        int x, y;
        scanf("%d%d", &x, &y);
        A[x][y] = rand() % p;
        A[y][x] = -A[x][y]; // Tutte矩阵是反对称矩阵
    }

    for (int i = 1; i <= n; i++)
        id[i] = i; // 输出方案用的□因为高斯消元的时候会交换列
    memcpy(t, A, sizeof(t));
    Gauss(A, NULL, n);

    m = n;
    n = 0; // 这里变量复用纯属个人习惯......

    for (int i = 1; i <= m; i++)
        if (A[id[i]][id[i]])
            a[++n] = i; // 找出一个极大满秩子矩阵

    for (int i = 1;i <= n; i++)
        for (int j = 1; j <= n; j++)
            A[i][j]=t[a[i]][a[j]];

    Gauss(A,B,n);

    for (int i = 1; i <= n; i++)
        if (!girl[a[i]])
            for (int j = i + 1; j <= n; j++)
                if (!girl[a[j]] && t[a[i]][a[j]] && B[j][i]) {
                    // 注意上面那句if的写法□现在t是邻接矩阵的备份□
                    // 逆矩阵j行i列不为0当且仅当这条边可行
                    girl[a[i]] = a[j];
                    girl[a[j]] = a[i];
                    eliminate(i, j);
                    eliminate(j, i);
                    break;
                }

    printf("%d\n", n >> 1);
    for (int i = 1; i <= m; i++)
        printf("%d ", girl[i]);

    return 0;
}

// 高斯消元 O(n^3)
// 在传入B时表示计算逆矩阵□传入NULL则只需计算矩阵的秩
void Gauss(int A[][maxn], int B[][maxn], int n){
    if(B) {
        memset(B, 0, sizeof(t));
        for (int i = 1; i <= n; i++)
            B[i][i] = 1;
    }

    for (int i = 1; i <= n; i++) {
        if (!A[i][i]) {
            for (int j = i + 1; j <= n; j++)
                if (A[j][i]) {
                    swap(id[i], id[j]);
                    for (int k = i; k <= n; k++)
                        swap(A[i][k], A[j][k]);

                    if (B)
                        for (int k = 1; k <= n; k++)
                            swap(B[i][k], B[j][k]);
                    break;
                }

            if (!A[i][i])
                continue;
        }

        int inv = qpow(A[i][i], p - 2);

        for (int j = 1; j <= n; j++)
            if (i != j && A[j][i]){
                int t = (long long)A[j][i] * inv % p;

                for (int k = i; k <= n; k++)
                    if (A[i][k])
                        A[j][k] = (A[j][k] - (long long)t * A[i][k]) % p;

                if (B)
                    for (int k = 1; k <= n; k++)
                        if (B[i][k])
                            B[j][k] = (B[j][k] - (long long)t * B[i][k])%p;
            }
    }

    if (B)
```

```
        for (int i = 1; i <= n; i++) {
            int inv = qpow(A[i][i], p - 2);

            for (int j = 1; j <= n; j++)
                if (B[i][j])
                    B[i][j] = (long long)B[i][j] * inv % p;
        }
}

// 消去一行一列 O(n^2)
void eliminate(int r, int c) {
    row[r] = col[c] = true; // 已经被消掉

    int inv = qpow(B[r][c], p - 2);

    for (int i = 1; i <= n; i++)
        if (!row[i] && B[i][c]) {
            int t = (long long)B[i][c] * inv % p;

            for (int j = 1; j <= n; j++)
                if (!col[j] && B[r][j])
                    B[i][j] = (B[i][j] - (long long)t * B[r][j]) % p;
        }
}
```

1.5.2 带花树

```
// Blossom 带花树 O(nm)
// By ysf
// 通过题目□UOJ#79 一般图最大匹配

// 带花树通常比高斯消元快很多□但在只需要求最大匹配大小的时候并没有高斯消元好写
// 当然输出方案要方便很多

// 全局数组与变量定义
vector<int> G[maxn];
int girl[maxn], f[maxn], t[maxn], p[maxn], vis[maxn], tim, q[maxn], head, tail;
int n, m;


// 封装好的主过程 O(nm)
int blossom() {
    int ans = 0;

    for (int i = 1; i <= n; i++)
        if (!girl[i])
            ans += bfs(i);

    return ans;
}


// bfs找增广路 O(m)
bool bfs(int s) {
    memset(t, 0, sizeof(t));
    memset(p, 0, sizeof(p));

    for (int i = 1; i <= n; i++)
        f[i] = i; // 并查集

    head = tail = 0;
    q[tail++] = s;
    t[s] = 1;

    while (head != tail){
        int x = q[head++];
        for (int y : G[x]){
            if (findroot(y) == findroot(x) || t[y] == 2)
                continue;

            if (!t[y]){
                t[y] = 2;
                p[y] = x;

                if (!girl[y]){
                    for (int u = y, t; u; u = t) {
                        t = girl[p[u]];
                        girl[p[u]] = u;
                        girl[u] = p[u];
                    }
                    return true;
                }
                t[girl[y]] = 1;
                q[tail++] = girl[y];
            }
            else if (t[y] == 1) {
                int z = LCA(x, y);
                shrink(x, y, z);
                shrink(y, x, z);
            }
        }
    }

    return false;
}

//缩奇环 O(n)
void shrink(int x, int y, int z) {
    while (findroot(x) != z){
        p[x] = y;
        y = girl[x];

        if (t[y] == 2) {
            t[y] = 1;
            q[tail++] = y;
        }

        if(findroot(x) == x)
            f[x] = z;
        if(findroot(y) == y)
            f[y] = z;

        x = p[y];
    }
}

//暴力找LCA O(n)
int LCA(int x, int y) {
    tim++;
    while (true) {
        if (x) {
            x = findroot(x);

            if (vis[x] == tim)
                return x;
            else {
                vis[x] = tim;
                x = p[girl[x]];
            }
```

```
        }
        swap(x, y);
    }
}

//并查集的查找 O(1)
int findroot(int x) {
    return x == f[x] ? x : (f[x] = findroot(f[x]));
}
```

1.6 最大流

1.6.1 Dinic

```
// 注意Dinic适用于二分图或分层图,对于一般稀疏图ISAP更优,稠密图则HLPP更优

struct edge{int to,cap,prev;}e[maxe<<1];


int last[maxn],len=0,d[maxn],cur[maxn],q[maxn];
memset(last,-1,sizeof(last));


void addedge(int x,int y,int z){
    AddEdge(x,y,z);
    AddEdge(y,x,0);
}


void AddEdge(int x,int y,int z){
    e[len].to=y;
    e[len].cap=z;
    e[len].prev=last[x];
    last[x]=len++;
}


int Dinic(){
    int flow=0;
    while(bfs(),d[t]!=-1){
        memcpy(cur,last,sizeof(int)*(t+5));
        flow+=dfs(s,(~0u)>>1);
    }
    return flow;
}


void bfs(){
    int head=0,tail=0;
    memset(d,-1,sizeof(int)*(t+5));
    q[tail++]=s;
    d[s]=0;
    while(head!=tail){
        int x=q[head++];
        for(int i=last[x];i!=-1;i=e[i].prev)
            if(e[i].cap>0&&d[e[i].to]==-1){
                d[e[i].to]=d[x]+1;
                q[tail++]=e[i].to;
            }
    }
}


int dfs(int x,int a){
    if(x==t||!a)return a;
    int flow=0,f;
    for(int &i=cur[x];i!=-1;i=e[i].prev)
        if(e[i].cap>0&&d[e[i].to]==d[x]+1&&(f=dfs(e[i].to,min(
        {
            e[i].cap-=f;
            e[i^1].cap+=f;
            flow+=f;
            a-=f;
            if(!a)break;
        }
    return flow;
}
```

1.6.2 ISAP

```
// 注意ISAP适用于一般稀疏图,对于二分图或分层图情况Dinic比较优,稠密图则HLPP更优


// 边的定义
// 这里没有记录起点和反向边,因为反向边即为正向边xor 1,起点即为反向边的终点
struct edge{
    int to, cap, prev;
} e[maxe * 2];


// 全局变量和数组定义
int last[maxn], cnte = 0, d[maxn], p[maxn], c[maxn], cur[maxn], q[maxn];
int n, m, s, t; // s, t一定要开成全局变量


// 重要!!!
// main函数最前面一定要加上如下初始化
memset(last, -1, sizeof(last));


// 加边函数 O(1)
// 包装了加反向边的过程,方便调用
// 需要调用AddEdge
void addedge(int x, int y, int z) {
    AddEdge(x, y, z);
    AddEdge(y, x, 0);
}


// 真·加边函数 O(1)
void AddEdge(int x, int y, int z){
    e[cnte].to = y;
    e[cnte].cap = z;
    e[cnte].prev = last[x];
    last[x] = cnte++;
}


// 主过程 O(n^2 m)
// 返回最大流的流量
// 需要调用bfs,augment
// 注意这里的n是编号最大值,在这个值不为n的时候一定要开个变量记录下来并修改代码
// 非递归
int ISAP() {
    bfs();

    memcpy(cur, last, sizeof(cur));
```

```cpp

    int x = s, flow = 0;

    while (d[s] < n) {
        if (x == t) {//如果走到了t就增广一次,并返回s重新找增广路
            flow += augment();
            x = s;
        }

        bool ok = false;
        for (int &i = cur[x]; ~i; i = e[i].prev)
            if (e[i].cap && d[x] == d[e[i].to] + 1) {
                p[e[i].to] = i;
                x = e[i].to;

                ok = true;
                break;
            }

        if (!ok) { // 修改距离标号
            int tmp = n - 1;
            for (int i = last[x]; ~i; i = e[i].prev)
                if (e[i].cap)
                    tmp = min(tmp, d[e[i].to] + 1);

            if (!--c[d[x]])
                break; // gap优化,一定要加上

            c[d[x] = tmp]++;
            cur[x] = last[x];

            if(x != s)
                x = e[p[x] ^ 1].to;
        }
    }
    return flow;
}

// bfs函数 O(n+m)
// 预处理到t的距离标号
// 在测试数据组数较少时可以省略,把所有距离标号初始化为0
void bfs(){
    memset(d, -1, sizeof(d));

    int head = 0, tail = 0;
    d[t] = 0;
    q[tail++] = t;

    while (head != tail) {
        int x = q[head++];
        c[d[x]]++;

        for (int i = last[x]; ~i; i = e[i].prev)
            if (e[i ^ 1].cap && d[e[i].to] == -1) {
                d[e[i].to] = d[x] + 1;
                q[tail++] = e[i].to;
            }
    }
}

// augment函数 O(n)
// 沿增广路增广一次,返回增广的流量
int augment() {
    int a = (~0u) >> 1; // INT_MAX

    for (int x = t; x != s; x = e[p[x] ^ 1].to)
        a = min(a, e[p[x]].cap);

    for (int x = t; x != s; x = e[p[x] ^ 1].to){
        e[p[x]].cap -= a;
        e[p[x] ^ 1].cap += a;
    }

    return a;
}
```

1.6.3 HLPP最高标号预流推进

```cpp
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
using std::min;
using std::vector;
using std::queue;
using std::priority_queue;
const int N=2e4+5,M=2e5+5,inf=0x3f3f3f3f;
int n,s,t,tot;
int v[M<<1],w[M<<1],first[N],next[M<<1];
int h[N],e[N],gap[N<<1],inq[N];//节点高度是可以到达2n-1的
struct cmp
{
    inline bool operator()(int a,int b) const
    {
        return h[a]<h[b];//因为在优先队列中的节点高度不会改变所以可以直接比较
    }
};
queue<int> Q;
priority_queue<int,vector<int>,cmp> pQ;
inline void add_edge(int from,int to,int flow)
{
    tot+=2;
    v[tot+1]=from;v[tot]=to;w[tot]=flow;w[tot+1]=0;
    next[tot]=first[from];first[from]=tot;
    next[tot+1]=first[to];first[to]=tot+1;
    return;
}
inline bool bfs()
{
    int now;
    register int go;
    memset(h+1,0x3f,sizeof(int)*n);
    h[t]=0;Q.push(t);
    while(!Q.empty())
    {
        now=Q.front();Q.pop();
        for(go=first[now];go;go=next[go])
            if(w[go^1]&&h[v[go]]>h[now]+1)
                h[v[go]]=h[now]+1,Q.push(v[go]);
    }
    return h[s]!=inf;
}
inline void push(int now)//推送
{
    int d;
    register int go;
    for(go=first[now];go;go=next[go])
        if(w[go]&&h[v[go]]+1==h[now])
        {
            d=min(e[now],w[go]);
            w[go]-=d;w[go^1]+=d;e[now]-=d;e[v[go]]+=d;
```

```cpp
            if(v[go]!=s&&v[go]!=t&&!inq[v[go]])
                pQ.push(v[go]),inq[v[go]]=1;
            if(!e[now])//已经推送完毕可以直接退出
                break;
        }
    return;
}
inline void relabel(int now)//重贴标签
{
    register int go;
    h[now]=inf;
    for(go=first[now];go;go=next[go])
        if(w[go]&&h[v[go]]+1<h[now])
            h[now]=h[v[go]]+1;
    return;
}
inline int hlpp()
{
    int now,d;
    register int i,go;
    if(!bfs())//s和t不连通
        return 0;
    h[s]=n;
    memset(gap,0,sizeof(int)*(n<<1));
    for(i=1;i<=n;i++)
        if(h[i]<inf)
            ++gap[h[i]];
    for(go=first[s];go;go=next[go])
        if(d=w[go])
        {
            w[go]-=d;w[go^1]+=d;e[s]-=d;e[v[go]]+=d;
            if(v[go]!=s&&v[go]!=t&&!inq[v[go]])
                pQ.push(v[go]),inq[v[go]]=1;
        }
    while(!pQ.empty())
    {
        inq[now=pQ.top()]=0;pQ.pop();push(now);
        if(e[now])
        {
            if(!--gap[h[now]])//gap优化□因为当前节点是最高的所以修改的节点一定不在优先队列中□不必担心修改对优先队列会造成影响
                for(i=1;i<=n;i++)
                    if(i!=s&&i!=t&&h[i]>h[now]&&h[i]<n+1)
                        h[i]=n+1;
            relabel(now);++gap[h[now]];
            pQ.push(now);inq[now]=1;
        }
    }
    return e[t];
}
int m;
signed main()
{
    int u,v,w;
    scanf("%d%d%d%d",&n,&m,&s,&t);
    while(m--)
    {
        scanf("%d%d%d",&u,&v,&w);
        add_edge(u,v,w);
    }
    printf("%d\n",hlpp());
    return 0;
}
```

2. 字符串

2.1 AC自动机

```cpp
// Aho-Corasick Automata AC自动机
// By AntiLeaf
// 通过题目□bzoj3881 Divljak


// 全局变量与数组定义
int ch[maxm][26] = {{0}}, f[maxm][26] = {{0}}, q[maxm] = {0}, sum[maxm] = {0}, cnt = 0;


// 在字典树中插入一个字符串 O(n)
int insert(const char *c) {
    int x = 0;
    while (*c) {
        if (!ch[x][*c - 'a'])
            ch[x][*c - 'a'] = ++cnt;
        x = ch[x][*c++ - 'a'];
    }
    return x;
}


// 建AC自动机 O(n*sigma)
void getfail() {
    int x, head = 0, tail = 0;

    for (int c = 0; c < 26; c++)
        if (ch[0][c])
            q[tail++] = ch[0][c]; // 把根节点的儿子加入队列

    while (head != tail) {
        x = q[head++];

        G[f[x][0]].push_back(x);
        fill(f[x] + 1, f[x] + 26, cnt + 1);

        for (int c = 0; c < 26; c++) {
            if (ch[x][c]) {
                int y = f[x][0];

                while (y&&!ch[y][c])
                    y=f[y][0];

                f[ch[x][c]][0] = ch[y][c];
                q[tail++] = ch[x][c];
            }
            else
                ch[x][c] = ch[f[x][0]][c];
        }
    }
    fill(f[0], f[0] + 26, cnt + 1);
}
```

2.2 后缀数组

2.2.1 SAMSA

```cpp
#include<bits/stdc++.h>
using namespace std;
const int maxn=100005;
void expand(int);
void dfs(int);
```

```
int root,last,cnt=0,val[maxn<<1]={0},par[maxn<<1]={0},go[maxn<<1][26]={{0}};
bool vis[maxn<<1]={0};
char s[maxn];
int n,id[maxn<<1]={0},ch[maxn<<1][26]={{0}},height[maxn],tim=0;
int main(){
    root=last=++cnt;
    scanf("%s",s+1);
    n=strlen(s+1);
    for(int i=n;i;i--){
        expand(s[i]-'a');
        id[last]=i;
    }
    vis[1]=true;
    for(int i=1;i<=cnt;i++)if(id[i])for(int x=i,pos=n;x&&!vis[x];x=par[x]){
        vis[x]=true;
        pos-=val[x]-val[par[x]];
        ch[par[x]][s[pos+1]-'a']=x;
    }
    dfs(root);
    printf("\n");
    for(int i=1;i<n;i++)printf("%d ",height[i]);
    return 0;
}
void expand(int c){
    int p=last,np=++cnt;
    val[np]=val[p]+1;
    while(p&&!go[p][c]){
        go[p][c]=np;
        p=par[p];
    }
    if(!p)par[np]=root;
    else{
        int q=go[p][c];
        if(val[q]==val[p]+1)par[np]=q;
        else{
            int nq=++cnt;
            val[nq]=val[p]+1;
            memcpy(go[nq],go[q],sizeof(go[q]));
            par[nq]=par[q];
            par[np]=par[q]=nq;
            while(p&&go[p][c]==q){
                go[p][c]=nq;
                p=par[p];
            }
        }
    }
    last=np;
}
void dfs(int x){
    if(id[x]){
        printf("%d ",id[x]);
        height[tim++]=val[last];
        last=x;
    }
    for(int c=0;c<26;c++)if(ch[x][c])dfs(ch[x][c]);
    last=par[x];
}
```

2.3 后缀自动机

```
//Suffix Automaton 后缀自动机 O(n)
//By ysf
//通过题目□Bzoj3473 字符串

//在字符集比较小的时候可以直接开go数组□否则需要用map或者哈希表替换
//注意□□□结点数要开成串长的两倍

//全局变量与数组定义
int last,val[maxn],par[maxn],go[maxn][26],cnt;
int c[maxn],q[maxn];//用来桶排序

//在主函数开头加上这句初始化
last=cnt=1;

//以下是按val进行桶排序的代码
for(int i=1;i<=cnt;i++)c[val[i]+1]++;
for(int i=1;i<=n;i++)c[i]+=c[i-1];//这里n是串长
for(int i=1;i<=cnt;i++)q[++c[val[i]]]=i;

//加入一个字符 均摊O(1)
void extend(int c){
    int p=last,np=++cnt;
    val[np]=val[p]+1;
    while(p&&!go[p][c]){
        go[p][c]=np;
        p=par[p];
    }
    if(!p)par[np]=1;
    else{
        int q=go[p][c];
        if(val[q]==val[p]+1)par[np]=q;
        else{
            int nq=++cnt;
            val[nq]=val[p]+1;
            memcpy(go[nq],go[q],sizeof(go[q]));
            par[nq]=par[q];
            par[np]=par[q]=nq;
            while(p&&go[p][c]==q){
                go[p][c]=nq;
                p=par[p];
            }
        }
    }
    last=np;
}
```

2.4 回文树

```
//Palindromic Tree/EERTREE 回文树 O(n)
//By ysf
//通过题目□APIO2014 回文串

//定理□一个字符串本质不同的回文子串个数是O(n)的
//注意回文树只需要开一倍结点□另外结点编号是一个可用的bfs序

//全局数组定义
int val[maxn],par[maxn],go[maxn][26],last,cnt;
char s[maxn];

//重要□在主函数最前面一定要加上以下初始化
par[0]=cnt=1;
val[1]=-1;

//extend函数 均摊O(1)
//向后扩展一个字符
//传入对应下标
void extend(int n){
    int p=last,c=s[n]-'a';
```

```
    while(s[n-val[p]-1]!=s[n])p=par[p];
    if(!go[p][c]){
        int q=++cnt,now=p;
        val[q]=val[p]+2;
        do p=par[p];while(s[n-val[p]-1]!=s[n]);
        par[q]=go[p][c];
        last=go[now][c]=q;
    }
    else last=go[p][c];
    a[last]++;
}
```

2.4.1 广义回文树

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

using namespace std;

constexpr int maxn = 1000005, mod = 1000000007;

int val[maxn], par[maxn], go[maxn][26], fail[maxn][26], pam_last[maxn], pam_cnt;
int weight[maxn], pow_26[maxn];

int trie[maxn][26], trie_cnt, d[maxn], mxd[maxn], son[maxn], top[maxn], len[maxn], sum[maxn];
char chr[maxn];
int f[25][maxn], log_tbl[maxn];
vector<int> v[maxn];

vector<int> queries[maxn];

char str[maxn];
int n, m, ans[maxn];

int add(int x, int c) {
    if (!trie[x][c]) {
        trie[x][c] = ++trie_cnt;
        f[0][trie[x][c]] = x;
        chr[trie[x][c]] = c + 'a';
    }

    return trie[x][c];
}

int del(int x) {
    return f[0][x];
}

void dfs1(int x) {
    mxd[x] = d[x] = d[f[0][x]] + 1;

    for (int i = 0; i < 26; i++)
        if (trie[x][i]) {
            int y = trie[x][i];

            dfs1(y);

            mxd[x] = max(mxd[x], mxd[y]);
            if (mxd[y] > mxd[son[x]])
                son[x] = y;
        }
}

void dfs2(int x) {
    if (x == son[f[0][x]])
        top[x] = top[f[0][x]];
    else
        top[x] = x;

    for (int i = 0; i < 26; i++)
        if (trie[x][i]) {
            int y = trie[x][i];
            dfs2(y);
        }

    if (top[x] == x) {
        int u = x;
        while (top[son[u]] == x)
            u = son[u];

        len[x] = d[u] - d[x];

        for (int i = 0; i < len[x]; i++) {
            v[x].push_back(u);
            u = f[0][u];
        }

        u = x;
        for (int i = 0; i < len[x]; i++) { // 梯子剖分要延长一倍
            v[x].push_back(u);
            u = f[0][u];
        }
    }
}

int get_anc(int x, int k) {
    if (!k)
        return x;
    if (k > d[x])
        return 0;

    x = f[log_tbl[k]][x];
    k ^= 1 << log_tbl[k];

    return v[top[x]][d[top[x]] + len[top[x]] - d[x] + k];
}

char get_char(int x, int k) { // 查询x前面k个的字符是哪个
    return chr[get_anc(x, k)];
}

int getfail(int x, int p) {
    if (get_char(x, val[p] + 1) == chr[x])
        return p;
    return fail[p][chr[x] - 'a'];
}

int extend(int x) {

    int p = pam_last[f[0][x]], c = chr[x] - 'a';

    p = getfail(x, p);

    int new_last;

    if (!go[p][c]) {
        int q = ++pam_cnt, now = p;
        val[q] = val[p] + 2;

        p = getfail(x, par[p]);

        par[q] = go[p][c];
        new_last = go[now][c] = q;

```

```cpp
        for (int i = 0; i < 26; i++)
            fail[q][i] = fail[par[q]][i];

        if (get_char(x, val[par[q]]) >= 'a')
            fail[q][get_char(x, val[par[q]]) - 'a'] = par[q];

        if (val[q] <= n)
            weight[q] = (weight[par[q]] + (long long)(n - val[q] + 1) * pow_26[n - val[q]]) % mod;
        else
            weight[q] = weight[par[q]];
    }
    else
        new_last = go[p][c];

    pam_last[x] = new_last;

    return weight[pam_last[x]];
}

void bfs() {

    queue<int> q;

    q.push(1);

    while (!q.empty()) {
        int x = q.front();
        q.pop();

        sum[x] = sum[f[0][x]];
        if (x > 1)
            sum[x] = (sum[x] + extend(x)) % mod;

        for (int i : queries[x])
            ans[i] = sum[x];

        for (int i = 0; i < 26; i++)
            if (trie[x][i])
                q.push(trie[x][i]);
    }

}

int main() {

    pow_26[0] = 1;
    log_tbl[0] = -1;

    for (int i = 1; i <= 1000000; i++) {
        pow_26[i] = 26ll * pow_26[i - 1] % mod;
        log_tbl[i] = log_tbl[i / 2] + 1;
    }

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

    while (T--) {
        scanf("%d%d%s", &n, &m, str);

        trie_cnt = 1;
        chr[1] = '#';

        int last = 1;
        for (char *c = str; *c; c++)
            last = add(last, *c - 'a');

        queries[last].push_back(0);

        for (int i = 1; i <= m; i++) {
            int op;
            scanf("%d", &op);

            if (op == 1) {
                char c;
                scanf(" %c", &c);

                last = add(last, c - 'a');
            }
            else
                last = del(last);

            queries[last].push_back(i);
        }

        dfs1(1);
        dfs2(1);

        for (int j = 1; j <= log_tbl[trie_cnt]; j++)
            for (int i = 1; i <= trie_cnt; i++)
                f[j][i] = f[j - 1][f[j - 1][i]];

        par[0] = pam_cnt = 1;


        for (int i = 0; i < 26; i++)
            fail[0][i] = fail[1][i] = 1;

        val[1] = -1;
        pam_last[1] = 1;

        bfs();

        for (int i = 0; i <= m; i++)
            printf("%d\n", ans[i]);

        for (int j = 0; j <= log_tbl[trie_cnt]; j++)
            memset(f[j], 0, sizeof(f[j]));

        for (int i = 1; i <= trie_cnt; i++) {
            chr[i] = 0;
            d[i] = mxd[i] = son[i] = top[i] = len[i] = pam_last[i] = sum[i] = 0;
            v[i].clear();
            queries[i].clear();

            memset(trie[i], 0, sizeof(trie[i]));
        }
        trie_cnt = 0;

        for (int i = 0; i <= pam_cnt; i++) {
            val[i] = par[i] = weight[i];

            memset(go[i], 0, sizeof(go[i]));
            memset(fail[i], 0, sizeof(fail[i]));
        }
        pam_cnt = 0;

    }

    return 0;
}
```

2.5 Manacher马拉车

```
//Manacher O(n)
//By ysf
//通过题目□51Nod1089 最长回文子串V2

//n为串长□回文半径输出到p数组中
//数组要开串长的两倍
void manacher(const char *t, int n) {
    static char s[maxn * 2];

    for (int i = n; i; i--)
        s[i * 2] = t[i];
    for (int i = 0; i <= n; i++)
        s[i * 2 + 1]='#';

    s[0] = '$';
    s[(n + 1) * 2] = '\0';
    n = n * 2 + 1;

    int mx = 0, j = 0;

    for (int i = 1; i <= n; i++) {
        p[i] = (mx > i ? min(p[j * 2 - i], mx - i) : 1);
        while (s[i - p[i]] == s[i + p[i]])
            p[i]++;

        if(i + p[i] > mx){
            mx = i + p[i];
            j = i;
        }
    }
}
```

2.6 KMP

2.6.1 ex-KMP

```
//Extended KMP 扩展KMP
//By AntiLeaf
//通过题目□小作业OJ 4182

//全局变量与数组定义
char s[maxn], t[maxn];
int n, m, a[maxn];

//主过程 O(n + m)
//把t的每个后缀与s的LCP输出到a中□s的后缀和自己的LCP存
    ↪ 在nx中
//0-based□s的长度是m□t的长度是n
void exKMP(const char *s, const char *t, int *a) {
    static int nx[maxn];

    memset(nx, 0, sizeof(nx));

    int j = 0;
    while (j + 1 < m && s[j] == s[j + 1])
        j++;
    nx[1] = j;

    for (int i = 2, k = 1;i < m; i++) {
        int pos = k + nx[k], len = nx[i - k];

        if (i + len < pos)
            nx[i] = len;
        else {
            j = max(pos - i, 0);
            while (i + j < m && s[j] == s[i + j])
                j++;

            nx[i] = j;
            k = i;
        }
    }

    j = 0;
    while (j < n && j < m && s[j] == t[j])
        j++;
    a[0] = j;

    for (int i = 1, k = 0; i < n; i++) {
        int pos = k + a[k], len = nx[i - k];
        if (i + len < pos)
            a[i] = len;
        else {
            j = max(pos - i, 0);
            while(j < m && i + j < n && s[j] == t[i + j])
                j++;

            a[i] = j;
            k = i;
        }
    }
}
```

3. 数学

3.1 插值

3.1.1 牛顿插值

3.1.2 拉格朗日插值

3.2 多项式

3.2.1 FFT

```
//Fast Fourier Transform 快速傅里叶变换 O(n\log n)
//By ysf
//通过题目□COGS2294 释迦□作为拆系数FFT的组成部分□
//使用时一定要注意double的精度是否足够□极限大概是10^14□

const double pi=acos((double)-1.0);

//手写复数类
//支持加减乘三种运算
//+=运算符如果用的不多可以不重载
struct Complex{
    double a,b;//由于long double精度和double几乎相同□通常
      ↪ 没有必要用long double
    Complex(double a=0.0,double b=0.0):a(a),b(b){}
    Complex operator+(const Complex &x)const{return
      ↪ Complex(a+x.a,b+x.b);}
    Complex operator-(const Complex &x)const{return
      ↪ Complex(a-x.a,b-x.b);}
    Complex operator*(const Complex &x)const{return
      ↪ Complex(a*x.a-b*x.b,a*x.b+b*x.a);}
    Complex &operator+=(const Complex &x){return
      ↪ *this=*this+x;}
}w[maxn],w_inv[maxn];

```

```
//FFT初始化 O(n)
//需要调用sin、cos函数
void FFT_init(int n){
    for(int i=0;i<n;i++)//根据单位根的旋转性质可以节省计算单位根逆元的时间
        w[i]=w_inv[n-i-1]=Complex(cos(2*pi/n*i),sin(2*pi/n*i));
    //当然不存单位根也可以□只不过在FFT次数较多时很可能会增大常数
}

//FFT主过程 O(n\log n)
void FFT(Complex *A,int n,int tp){
    for(int i=1,j=0,k;i<n-1;i++){
        k=n;
        do j^=(k>>=1);while(j<k);
        if(i<j)swap(A[i],A[j]);
    }
    for(int k=2;k<=n;k<<=1)
        for(int i=0;i<n;i+=k)
            for(int j=0;j<(k>>1);j++){
                Complex a=A[i+j],b=(tp>0?w:w_inv)[n/k*j]*A[i+j+(k>>1)];
                A[i+j]=a+b;
                A[i+j+(k>>1)]=a-b;
            }
    if(tp<0)for(int i=0;i<n;i++)A[i].a/=n;
}
```

3.2.2 NTT

```
// Number Theory Transform 快速数论变换 O(n\log n)
// By AntiLeaf
// 通过题目□UOJ#34 多项式乘法
// 要求模数为10^9以内的NTT模数

const int p = 998244353, g = 3; // p为模数□g为p的任意一个原根

void NTT(int *A, int n, int tp) { // n为变换长度□tp为1或-1□表示正/逆变换
    for (int i = 1, j = 0, k; i < n - 1; i++) { // O(n)旋转算法□原理是模拟二进制加一
        k = n;
        do
            j ^= (k >>= 1);
        while (j < k);

        if(i < j)
            swap(A[i], A[j]);
    }

    for (int k = 2; k <= n; k <<= 1) {
        int wn = qpow(g, (tp > 0 ? (p - 1) / k : (p - 1) / k * (long long)(p - 2) % (p - 1)));
        for (int i = 0; i < n; i += k) {
            int w = 1;
            for (int j = 0; j < (k >> 1); j++, w = (long long)w * wn % p){
                int a = A[i + j], b = (long long)w * A[i + j + (k >> 1)] % p;
                A[i + j] = (a + b) % p;
                A[i + j + (k >> 1)] = (a - b + p) % p;
            } // 更好的写法是预处理单位根的次幂□参照FFT的代码
        }
    }

    if (tp < 0) {
        int inv = qpow(n, p - 2); // 如果预处理过逆元的话就不用快速幂了
        for (int i = 0; i < n; i++)
            A[i] = (long long)A[i] * inv % p;
    }
}
```

3.2.3 任意模数卷积(三模数NTT)

```
//只要求模数在2^30-1以内,无其他特殊要求
//常数很大,慎用
//在卷积结果不超过10^14时可以直接double暴力乘,这时就不要写任意模数卷积了
//这里有三模数NTT和拆系数FFT两个版本,通常后者常数要小一些
//但在答案不超过10^18时可以改成双模数NTT,这时就比拆系数FFT快一些了

//以下为三模数NTT,原理是选取三个乘积大于结果的NTT模数,最后中国剩余定理合并
//以对23333333(不是质数)取模为例
const int maxn=262200,Mod=23333333,g=3,m[]={998244353,1004535809,1045430
    m0_inv=669690699,m1_inv=332747959,M_inv=942377029;//这三个模数最小原根都是3
const long long M=(long long)m[0]*m[1];

//主函数(当然更多时候包装一下比较好)
//用来卷积的是A和B
//需要调用mul
int n,N=1,A[maxn],B[maxn],C[maxn],D[maxn],ans[3][maxn];
int main(){
    scanf("%d",&n);
    while(N<(n<<1))N<<=1;
    for(int i=0;i<n;i++)scanf("%d",&A[i]);
    for(int i=0;i<n;i++)scanf("%d",&B[i]);
    for(int i=0;i<3;i++)mul(m[i],ans[i]);
    for(int i=0;i<n;i++)printf("%d ",China(ans[0][i],ans[1][i],ans[2][i]));
    return 0;
}

//mul O(n\log n)
//包装了模NTT模数的卷积
//需要调用NTT
void mul(int p,int *ans){
    copy(A,A+N,C);
    copy(B,B+N,D);
    NTT(C,N,1,p);
    NTT(D,N,1,p);
    for(int i=0;i<N;i++)ans[i]=(long long)C[i]*D[i]%p;
    NTT(ans,N,-1,p);
}

//中国剩余定理 O(1)
//由于直接合并会爆long long,采用神奇的方法合并
//需要调用O(1)快速乘
inline int China(int a0,int a1,int a2){
    long long A=(mul((long long)a0*m1_inv,m[1],M)
        +mul((long long)a1*m0_inv,m[0],M))%M;
    int k=((a2-A)%m[2]+m[2])%m[2]*M_inv%m[2];
    return (k%Mod*(M%Mod)%Mod+A%Mod)%Mod;
}
```

```
//-----------------------分割线-------------------------

//以下为拆系数FFT,原理是减小结果范围使得double精度能够承受
//仍然以模23333333为例
const int maxn=262200,p=23333333,M=4830;//M取值要使得结果不超过10^14

//需要开的数组
struct Complex{//内容略
}w[maxn],w_inv[maxn],A[maxn],B[maxn],C[maxn],D[maxn],F[maxn],G[maxn],H[maxn];

//主函数(当然更多时候包装一下比较好)
//需要调用FFT初始化,FFT
int main(){
    scanf("%d",&n);
    int N=1;
    while(N<(n<<1))N<<=1;
    for(int i=0,x;i<n;i++){
        scanf("%d",&x);
        A[i]=x/M;
        B[i]=x%M;
    }
    for(int i=0,x;i<n;i++){
        scanf("%d",&x);
        C[i]=x/M;
        D[i]=x%M;
    }
    FFT_init(N);
    FFT(A,N,1);
    FFT(B,N,1);
    FFT(C,N,1);
    FFT(D,N,1);
    for(int i=0;i<N;i++){
        F[i]=A[i]*C[i];
        G[i]=A[i]*D[i]+B[i]*C[i];
        H[i]=B[i]*D[i];
    }
    FFT(F,N,-1);
    FFT(G,N,-1);
    FFT(H,N,-1);
    for(int i=0;i<n;i++)
        printf("%d\n",(int)((M*M*((long long)(F[i].a+0.5)%p)%p+
        M*((long long)(G[i].a+0.5)%p)%p+(long long)(H[i].a+0.5)%p)%p));
    return 0;
}
```

3.2.4 多项式操作

```
//Polymial Operations 多项式操作
//By ysf
//通过题目⊠COGS2189 帕秋莉的超级多项式⊠板子题⊠

const int maxn=262200;//以下所有代码均为NTT版本
//以下所有代码均满足⊠A为输入⊠不进行修改⊠⊠C为输出⊠n为所需长度

//多项式求逆 O(n\log n)
//要求A常数项不为0
void getinv(int *A,int *C,int n){
    static int B[maxn];
    memset(C,0,sizeof(int)*(n<<1));
    C[0]=qpow(A[0],p-2);//一般题目直接赋值为1就可以
    for(int k=2;k<=n;k<<=1){
        memcpy(B,A,sizeof(int)*k);
        memset(B+k,0,sizeof(int)*k);
        NTT(B,k<<1,1);
        NTT(C,k<<1,1);
        for(int i=0;i<(k<<1);i++)
            C[i]=((2-(long long)B[i]*C[i])%p*C[i]%p+p)%p;
        NTT(C,k<<1,-1);
        memset(C+k,0,sizeof(int)*k);
    }
}

//多项式开根 O(n\log n)
//要求A常数项可以开根/存在二次剩余
//需要调用多项式求逆⊠且需要预处理2的逆元
void getsqrt(int *A,int *C,int n){
    static int B[maxn],D[maxn];
    memset(C,0,sizeof(int)*(n<<1));
    C[0]=(int)(sqrt(A[0])+1e-7);//一般题目直接赋值为1就可以
    for(int k=2;k<=n;k<<=1){
        memcpy(B,A,sizeof(int)*k);
        memset(B+k,0,sizeof(int)*k);
        getinv(C,D,k);
        NTT(B,k<<1,1);
        NTT(D,k<<1,1);
        for(int i=0;i<(k<<1);i++)B[i]=(long long)B[i]*D[i]%p;
        NTT(B,k<<1,-1);
        for(int i=0;i<k;i++)C[i]=(long long)(C[i]+B[i])*inv_2%p;//inv_2是2的逆元
    }
}

//求导 O(n)
void getderivative(int *A,int *C,int n){
    for(int i=1;i<n;i++)C[i-1]=(long long)A[i]*i%p;
    C[n-1]=0;
}

//不定积分 O(n\log n)⊠如果预处理过逆元可以降到O(n)
void getintegrate(int *A,int *C,int n){
    for(int i=1;i<n;i++)C[i]=(long long)A[i-1]*qpow(i,p-2)%p;
    C[0]=0;//由于是不定积分⊠结果没有常数项
}

//多项式\ln O(n\log n)
//要求A常数项不为0/存在离散对数
//需要调用多项式求逆⊠求导⊠不定积分
void getln(int *A,int *C,int n){//通常情况下A常数项都是1
    static int B[maxn];
    getderivative(A,B,n);
    memset(B+n,0,sizeof(int)*n);
    getinv(A,C,n);
    NTT(B,n<<1,1);
    NTT(C,n<<1,1);
    for(int i=0;i<(n<<1);i++)B[i]=(long long)B[i]*C[i]%p;
    NTT(B,n<<1,-1);
    getintegrate(B,C,n);
    memset(C+n,0,sizeof(int)*n);
}

//多项式\exp O(n\log n)
//要求A没有常数项
//需要调用多项式\ln
```

```cpp
//常数很大且总代码较长□在时间效率要求不高时最好替换为分治FFT
//分治FFT依据□设G(x)=\exp F(x)□则有g_i=\sum_{k=1}^i f_k g_{i-k}
void getexp(int *A,int *C,int n){
    static int B[maxn];
    memset(C,0,sizeof(int)*(n<<1));
    C[0]=1;
    for(int k=2;k<=n;k<<=1){
        getln(C,B,k);
        for(int i=0;i<k;i++){
            B[i]=A[i]-B[i];
            if(B[i]<0)B[i]+=p;
        }
        (++B[0])%=p;
        NTT(B,k<<1,1);
        NTT(C,k<<1,1);
        for(int i=0;i<(k<<1);i++)C[i]=(long long)C[i]*B[i]%p;
        NTT(C,k<<1,-1);
        memset(C+k,0,sizeof(int)*k);
    }
}

//多项式k次幂 O(n\log n)
//在A常数项不为1时需要转化
//需要调用多项式/exp、\ln
//常数较大且总代码较长□在时间效率要求不高时最好替换为暴力快速幂
void getpow(int *A,int *C,int n,int k){
    static int B[maxn];
    getln(A,B,n);
    for(int i=0;i<n;i++)B[i]=(long long)B[i]*k%p;
    getexp(B,C,n);
}
```

3.2.5 拉格朗日反演

3.2.6 半在线卷积

```cpp
// Half-Online Convolution 半在线卷积
// By AntiLeaf
// O(n\log^2 n)
// 通过题目□自己出的题


// 主过程□递归调用自身
void solve(int l, int r) {
    if (r <= m)
        return;

    if (r - l == 1) {
        if (l == m)
            f[l] = a[m];
        else
            f[l] = (long long)f[l] * inv[l - m] % p;

        for (int i = l, t = (long long)l * f[l] % p; i <= n; i += l)
            g[i] = (g[i] + t) % p;

        return;
    }

    int mid = (l + r) / 2;

    solve(l, mid);

    if (l == 0) {
        for (int i = 1; i < mid; i++) {
            A[i] = f[i];
            B[i] = (c[i] + g[i]) % p;
        }
        NTT(A, r, 1);
        NTT(B, r, 1);
        for (int i = 0; i < r; i++)
            A[i] = (long long)A[i] * B[i] % p;
        NTT(A, r, -1);

        for (int i = mid; i < r; i++)
            f[i] = (f[i] + A[i]) % p;
    }
    else {
        for (int i = 0; i < r - l; i++)
            A[i] = f[i];
        for (int i = l; i < mid; i++)
            B[i - l] = (c[i] + g[i]) % p;
        NTT(A, r - l, 1);
        NTT(B, r - l, 1);
        for (int i = 0; i < r - l; i++)
            A[i] = (long long)A[i] * B[i] %p;
        NTT(A, r - l, -1);

        for (int i = mid; i < r; i++)
            f[i] = (f[i] + A[i - l]) % p;

        memset(A, 0, sizeof(int) * (r - l));
        memset(B, 0, sizeof(int) * (r - l));

        for (int i = l; i < mid; i++)
            A[i - l] = f[i];
        for (int i = 0; i < r - l; i++)
            B[i] = (c[i] + g[i]) % p;
        NTT(A, r - l, 1);
        NTT(B, r - l, 1);
        for (int i = 0; i < r - l; i++)
            A[i] = (long long)A[i] * B[i] % p;
        NTT(A, r - l, -1);

        for (int i = mid; i < r; i++)
            f[i] = (f[i] + A[i - l]) % p;
    }

    memset(A, 0, sizeof(int) * (r - l));
    memset(B, 0, sizeof(int) * (r - l));

    solve(mid, r);
}
```

3.3 FWT快速沃尔什变换

```cpp
//Fast Walsh-Hadamard Transform 快速沃尔什变换 O(n\log n)
//By ysf
//通过题目□COGS上几道板子题

//注意FWT常数比较小□这点与FFT/NTT不同
//以下代码均以模质数情况为例□其中n为变换长度□tp表示正/逆变换

//按位或版本
void FWT_or(int *A,int n,int tp){
```

```
    for(int k=2;k<=n;k<<=1)
        for(int i=0;i<n;i+=k)
            for(int j=0;j<(k>>1);j++){
                if(tp>0)A[i+j+(k>>1)]=(A[i+j+(k>>1)]+A[i+j])%p;
                else A[i+j+(k>>1)]=(A[i+j+(k>>1)]-A[i+j]+p)%p;
            }
}

//按位与版本
void FWT_and(int *A,int n,int tp){
    for(int k=2;k<=n;k<<=1)
        for(int i=0;i<n;i+=k)
            for(int j=0;j<(k>>1);j++){
                if(tp>0)A[i+j]=(A[i+j]+A[i+j+(k>>1)])%p;
                else A[i+j]=(A[i+j]-A[i+j+(k>>1)]+p)%p;
            }
}

//按位异或版本
void FWT_xor(int *A,int n,int tp){
    for(int k=2;k<=n;k<<=1)
        for(int i=0;i<n;i+=k)
            for(int j=0;j<(k>>1);j++){
                int a=A[i+j],b=A[i+j+(k>>1)];
                A[i+j]=(a+b)%p;
                A[i+j+(k>>1)]=(a-b+p)%p;
            }
    if(tp<0){
        int inv=qpow(n%p,p-2);//n的逆元□在不取模时需要用每层除以2代替
        for(int i=0;i<n;i++)A[i]=A[i]*inv%p;
    }
}
```

3.4 单纯形

```
//Simplex Method 单纯形方法求解线性规划
//By ysf
//通过题目□UOJ#179 线性规划□然而被hack了QAQ……□

//单纯形其实是指数算法□但实践中跑得飞快□所以复杂度什么的也就无所谓了


const double eps=1e-10;

double A[maxn][maxn],x[maxn];
int n,m,t,id[maxn<<1];

//方便起见□这里附上主函数
int main(){
    scanf("%d%d%d",&n,&m,&t);
    for(int i=1;i<=n;i++){
        scanf("%lf",&A[0][i]);
        id[i]=i;
    }
    for(int i=1;i<=m;i++){
        for(int j=1;j<=n;j++)scanf("%lf",&A[i][j]);
        scanf("%lf",&A[i][0]);
    }
    if(!initalize())printf("Infeasible");
    else if(!simplex())printf("Unbounded");
    else{
        printf("%.15lf\n",-A[0][0]);
        if(t){
            for(int i=1;i<=m;i++)x[id[i+n]]=A[i][0];
            for(int i=1;i<=n;i++)printf("%.15lf ",x[i]);
        }
    }
    return 0;
}

//初始化
//对于初始解可行的问题□可以把初始化省略掉
bool initalize(){
    for(;;){
        double t=0.0;
        int l=0,e=0;
        for(int i=1;i<=m;i++)if(A[i][0]+eps<t){
            t=A[i][0];
            l=i;
        }
        if(!l)return true;
        for(int i=1;i<=n;i++)if(A[l][i]<-eps&&(!e||id[i]<id[e]))e=i;
        if(!e)return false;
        pivot(l,e);
    }
}

//求解
bool simplex(){
    for(;;){
        int l=0,e=0;
        for(int i=1;i<=n;i++)if(A[0][i]>eps&&(!e||id[i]<id[e]))e=i;
        if(!e)return true;
        double t=1e50;
        for(int i=1;i<=m;i++)if(A[i][e]>eps&&A[i][0]/A[i][e]<t){
            l=i;
            t=A[i][0]/A[i][e];
        }
        if(!l)return false;
        pivot(l,e);
    }
}

//转轴操作□本质是
void pivot(int l,int e){
    swap(id[e],id[n+l]);
    double t=A[l][e];
    A[l][e]=1.0;
    for(int i=0;i<=n;i++)A[l][i]/=t;
    for(int i=0;i<=m;i++)if(i!=l){
        t=A[i][e];
        A[i][e]=0.0;
        for(int j=0;j<=n;j++)A[i][j]-=t*A[l][j];
    }
}
```

3.5 线性代数

3.5.1 线性基

4. 数论

4.1 $O(n)$预处理逆元

```
//Mutiply Inversation 预处理乘法逆元 O(n)
//By ysf
//要求p为质数(?)

inv[0]=inv[1]=1;
for(int i=2;i<=n;i++)
    inv[i]=(long long)(p-(p/i))*inv[p%i]%p;//p为模数
//$i^{-1} \equiv -\left\lfloor \frac{p}{i} \right\rfloor \times (p \bmod i)^{-1} \pmod p$
//i^-1 = -(p/i) * (p%i)^-1
```

4.2 杜教筛

```
//Yuhao Du's Sieve 杜教筛 O(n^{2/3})
//By ysf
//通过题目?51Nod1239 欧拉函数之和

//用于求可以用狄利克雷卷积构造出好求和的东西的函数的前缀和?有点绕?
//有些题只要求n<=10^9?这时就没必要开long long了?但记得乘法时强转

//常量/全局变量/数组定义
const int maxn=5000005,table_size=5000000,p=1000000007,inv_2=(p+1)/2
bool notp[maxn];
int prime[maxn/20],phi[maxn],tbl[100005];
//tbl用来顶替哈希表?其实开到n^{1/3}就够了?不过保险起见开成\sqrt n比较好
long long N;

//主函数前面加上这么一句
memset(tbl,-1,sizeof(tbl));

//线性筛预处理部分略去

//杜教筛主过程 总计O(n^{2/3})
//递归调用自身
//递推式还需具体情况具体分析?这里以求欧拉函数前缀和(mod 10^9+7)为例
int S(long long n){
    if(n<=table_size)return phi[n];
    else if(~tbl[N/n])return tbl[N/n];
    //原理?n除以所有可能的数的结果一定互不相同
    int ans=0;
    for(long long i=2,last;i<=n;i=last+1){
        last=n/(n/i);
        ans=(ans+(last-i+1)%p*S(n/i))%p;//如果n是int范围的话记得强转
    }
    ans=(n%p*((n+1)%p)%p*inv_2-ans+p)%p;//同上
    return tbl[N/n]=ans;
}
```

4.3 线性筛

```
//Extended Euler's Sieving 扩展线性筛 O(n)
//By ysf
//通过题目?51Nod1220 约数之和?预处理部分?

//此代码以计算约数之和函数\sigma_1?对10^9+7取模?为例
//适用于任何f(p^k)便于计算的积性函数
const int p=1000000007;

int prime[maxn/10],sigma_one[maxn],f[maxn],g[maxn];
//f?除掉最小质因子后剩下的部分
//g?最小质因子的幂次?在f(p^k)比较复杂时很有用?但f(p^k)可以递推时就可以省略了
//这里没有记录最小质因子?但根据线性筛的性质?每个合数只会被它最小的质因子筛掉
bool notp[maxn];//顾名思义

void get_table(int n){
    sigma_one[1]=1;//积性函数必有f(1)=1
    for(int i=2;i<=n;i++){
        if(!notp[i]){//质数情况
            prime[++prime[0]]=i;
            sigma_one[i]=i+1;
            f[i]=g[i]=1;
        }
        for(int j=1;j<=prime[0]&&i*prime[j]<=n;j++){
            notp[i*prime[j]]=true;
            if(i%prime[j]){//加入一个新的质因子?这种情况很简单
                sigma_one[i*prime[j]]=(long long)sigma_one[i]*(prime[j]+1)%p;
                f[i*prime[j]]=i;
                g[i*prime[j]]=1;
            }
            else{//再加入一次最小质因子?需要再进行分类讨论
                f[i*prime[j]]=f[i];
                g[i*prime[j]]=g[i]+1;
                //对于f(p^k)可以直接递推的函数?这里的判断可以改成
                //i/prime[j]%prime[j]!=0?这样可以省下f[]的空间?
                //但常数很可能会稍大一些
                if(f[i]==1)//质数的幂次?这里\sigma_1可以递推
                    sigma_one[i*prime[j]]=(sigma_one[i]+i*prime[j]
                    //对于更一般的情况?可以借助g[]计算f(p^k)
                else sigma_one[i*prime[j]]=//否则直接利用积性?两半乘起来
                    (long long)sigma_one[i*prime[j]/f[i]]*sigma_one[
                break;
            }
        }
    }
}
```

4.4 Miller-Rabin

```
//Miller-Rabin Primality Test  Miller-Rabin素性检测算法
//By ysf
//通过题目?Bzoj4802 欧拉函数?作为Pollard's Rho的子算法?

//复杂度可以认为是常数
```

```cpp

//封装好的函数体
//需要调用check
bool Miller_Rabin(long long n){
    if(n==1)return false;
    if(n==2)return true;
    if(n%2==0)return false;
    for(int i:{2,3,5,7,11,13,17,19,23,29,31,37}){
        if(i>n)break;
        if(!check(n,i))return false;
    }
    return true;
}

//用一个数检测
//需要调用long long快速幂和O(1)快速乘
bool check(long long n,long long b){//b是base
    long long a=n-1;
    int k=0;
    while(a%2==0){
        a>>=1;
        k++;
    }
    long long t=qpow(b,a,n);//这里的快速幂函数需要写O(1)快
      ↪ 速乘
    if(t==1||t==n-1)return true;
    while(k--){
        t=mul(t,t,n);//mul是O(1)快速乘函数
        if(t==n-1)return true;
    }
    return false;
}
```

4.5 Pollard's Rho

```cpp
//Miller-Rabin Primality Test  Miller-Rabin素性检测算法
//By ysf
//通过题目Bzoj4802 欧拉函数作为Pollard's Rho的子算法

//复杂度可以认为是常数

//封装好的函数体
//需要调用check
bool Miller_Rabin(long long n){
    if(n==1)return false;
    if(n==2)return true;
    if(n%2==0)return false;
    for(int i:{2,3,5,7,11,13,17,19,23,29,31,37}){
        if(i>n)break;
        if(!check(n,i))return false;
    }
    return true;
}

//用一个数检测
//需要调用long long快速幂和O(1)快速乘
bool check(long long n,long long b){//b是base
    long long a=n-1;
    int k=0;
    while(a%2==0){
        a>>=1;
        k++;
    }
    long long t=qpow(b,a,n);//这里的快速幂函数需要写O(1)快
      ↪ 速乘
    if(t==1||t==n-1)return true;
    while(k--){
        t=mul(t,t,n);//mul是O(1)快速乘函数
        if(t==n-1)return true;
    }
    return false;
}
```

5. 数据结构

5.1 线段树

5.1.1 主席树

参见GREAD07加强版

5.2 陈丹琦分治

```cpp
// Division of Danqi Chen CDQ分治
// By AntiLeaf
// 通过题目四维偏序

void CDQ1(int l,int r){
    if(l>=r)return;
    int mid=(l+r)>>1;
    CDQ1(l,mid);CDQ1(mid+1,r);
    int i=l,j=mid+1,k=l;
    while(i<=mid&&j<=r){
        if(a[i].x<a[j].x){
            a[i].ins=true;
            b[k++]=a[i++];
        }
        else{
            a[j].ins=false;
            b[k++]=a[j++];
        }
    }
    while(i<=mid){
        a[i].ins=true;
        b[k++]=a[i++];
    }
    while(j<=r){
        a[j].ins=false;
        b[k++]=a[j++];
    }
    copy(b+l,b+r+1,a+l);
    CDQ2(l,r);
}
void CDQ2(int l,int r){
    if(l>=r)return;
    int mid=(l+r)>>1;
    CDQ2(l,mid);CDQ2(mid+1,r);
    int i=l,j=mid+1,k=l;
    while(i<=mid&&j<=r){
        if(b[i].y<b[j].y){
            if(b[i].ins)add(b[i].z,1);
            t[k++]=b[i++];
        }
        else{
            if(!b[j].ins)ans+=query(b[j].z-1);
            t[k++]=b[j++];
        }
    }
    while(i<=mid){
        if(b[i].ins)add(b[i].z,1);
        t[k++]=b[i++];
```

```
    }
    while(j<=r){
        if(!b[j].ins)ans+=query(b[j].z-1);
        t[k++]=b[j++];
    }
    for(i=l;i<=mid;i++)if(b[i].ins)add(b[i].z,-1);
    copy(t+l,t+r+1,b+l);
}
```

5.3 Splay

参见LCT

5.4 树分治

5.4.1 动态树分治

```
//Dynamic Divide and Couquer on Tree 动态树分治 O(n\log n)-O(\log n)
//By ysf
//通过题目□COGS2278 树黑白

//为了减小常数□这里采用bfs写法□实测预处理比dfs快将近一半
//以下以维护一个点到每个黑点的距离之和为例

//全局数组定义
vector<int>G[maxn],W[maxn];
int size[maxn],son[maxn],q[maxn];
int p[maxn],depth[maxn],id[maxn][20],d[maxn][20];//id是对应层所在子树的根
int a[maxn],ca[maxn],b[maxn][20],cb[maxn][20];//维护距离和用的
bool vis[maxn]={false},col[maxn]={false};

//建树 总计O(n\log n)
//需要调用找重心□预处理距离□同时递归调用自身
void build(int x,int k,int s,int pr){//结点□深度□连通块大小□点分树上的父亲
    x=getcenter(x,s);
    vis[x]=true;
    depth[x]=k;
    p[x]=pr;
    for(int i=0;i<(int)G[x].size();i++)
        if(!vis[G[x][i]]){
            d[G[x][i]][k]=W[x][i];
            p[G[x][i]]=x;
            getdis(G[x][i],k,G[x][i]);
        }
    for(int i=0;i<(int)G[x].size();i++)
        if(!vis[G[x][i]])build(G[x][i],k+1,size[G[x][i]],x);
}

//找重心 O(n)
int getcenter(int x,int s){
    int head=0,tail=0;
    q[tail++]=x;
    while(head!=tail){
        x=q[head++];
        size[x]=1;
        son[x]=0;
        for(int i=0;i<(int)G[x].size();i++)
            if(!vis[G[x][i]]&&G[x][i]!=p[x]){
                p[G[x][i]]=x;
                q[tail++]=G[x][i];
            }
    }
    for(int i=tail-1;i;i--){
        x=q[i];
        size[p[x]]+=size[x];
        if(size[x]>size[son[p[x]]])son[p[x]]=x;
    }
    x=q[0];
    while(son[x]&&(size[son[x]]<<1)>=s)x=son[x];
    return x;
}

//预处理距离 O(n)
//方便起见□这里直接用了笨一点的方法□O(n\log n)全存下来
void getdis(int x,int k,int rt){
    int head=0,tail=0;
    q[tail++]=x;
    while(head!=tail){
        x=q[head++];
        size[x]=1;
        id[x][k]=rt;
        for(int i=0;i<(int)G[x].size();i++)
            if(!vis[G[x][i]]&&G[x][i]!=p[x]){
                p[G[x][i]]=x;
                d[G[x][i]][k]=d[x][k]+W[x][i];
                q[tail++]=G[x][i];
            }
    }
    for(int i=tail-1;i;i--)
        size[p[q[i]]]+=size[q[i]];
}

//修改 O(\log n)
void modify(int x){
    if(col[x])ca[x]--;
    else ca[x]++;//记得先特判自己作为重心的那层
    for(int u=p[x],k=depth[x]-1;u;u=p[u],k--){
        if(col[x]){
            a[u]-=d[x][k];
            ca[u]--;
            b[id[x][k]][k]-=d[x][k];
            cb[id[x][k]][k]--;
        }
        else{
            a[u]+=d[x][k];
            ca[u]++;
            b[id[x][k]][k]+=d[x][k];
            cb[id[x][k]][k]++;
        }
    }
    col[x]^=true;
}

//询问 O(\log n)
int query(int x){
    int ans=a[x];//特判自己是重心的那层
    for(int u=p[x],k=depth[x]-1;u;u=p[u],k--)
        ans+=a[u]-b[id[x][k]][k]+d[x][k]*(ca[u]-cb[id[x][k]][k]);
    return ans;
}
```

5.4.2 紫荆花之恋

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

```cpp
#include<algorithm>
#include<vector>
using namespace std;
const int maxn=100010;
const double alpha=0.7;
struct node{
    static int randint(){
        static int a=1213,b=97818217,p=998244353,x=751815431;
        x=a*x+b;x%=p;
        return x<0?(x+=p):x;
    }
    int data,size,p;
    node *ch[2];
    node(int d):data(d),size(1),p(randint()){}
    inline void refresh(){size=ch[0]->size+ch[1]->size+1;}
}*null=new node(0),*root[maxn],*root1[maxn][50];
void addnode(int,int);
void rebuild(int,int,int,int);
void dfs_getcenter(int,int,int&);
void dfs_getdis(int,int,int,int);
void dfs_destroy(int,int);
void insert(int,node*&);
int order(int,node*);
void destroy(node*&);
void rot(node*&,int);
vector<int>G[maxn],W[maxn];
int size[maxn]={0},siz[maxn][50]={0},son[maxn];
bool vis[maxn];
int depth[maxn],p[maxn],d[maxn][50],id[maxn][50];
int n,m,w[maxn],tmp;
long long ans=0;
int main(){
    freopen("flowera.in","r",stdin);
    freopen("flowera.out","w",stdout);
    null->size=0;
    null->ch[0]=null->ch[1]=null;
    scanf("%*d%d",&n);
    fill(vis,vis+n+1,true);
    fill(root,root+n+1,null);
    for(int i=0;i<=n;i++)fill(root1[i],root1[i]+50,null);
    scanf("%*d%*d%d",&w[1]);
    insert(-w[1],root[1]);
    size[1]=1;
    printf("0\n");
    for(int i=2;i<=n;i++){
        scanf("%d%d%d",&p[i],&tmp,&w[i]);
        p[i]^=(ans%(int)1e9);
        G[i].push_back(p[i]);
        W[i].push_back(tmp);
        G[p[i]].push_back(i);
        W[p[i]].push_back(tmp);
        addnode(i,tmp);
        printf("%lld\n",ans);
    }
    return 0;
}
void addnode(int x,int z){//wj-dj>=di-wi
    depth[x]=depth[p[x]]+1;
    size[x]=1;
    insert(-w[x],root[x]);
    int rt=0;
    for(int u=p[x],k=depth[p[x]];u;u=p[u],k--){
        if(u==p[x]){
            id[x][k]=x;
            d[x][k]=z;
        }
        else{
            id[x][k]=id[p[x]][k];
            d[x][k]=d[p[x]][k]+z;
        }
        ans+=order(w[x]-d[x][k],root[u])-order(w[x]-d[x][k],root1[id[x][k]][k]);
        insert(d[x][k]-w[x],root[u]);
        insert(d[x][k]-w[x],root1[id[x][k]][k]);
        size[u]++;
        siz[id[x][k]][k]++;
        if(siz[id[x][k]][k]>size[u]*alpha+5)rt=u;
    }
    id[x][depth[x]]=0;
    d[x][depth[x]]=0;
    if(rt){
        dfs_destroy(rt,depth[rt]);
        rebuild(rt,depth[rt],size[rt],p[rt]);
    }
}
void rebuild(int x,int k,int s,int pr){
    int u=0;
    dfs_getcenter(x,s,u);
    vis[x=u]=true;
    p[x]=pr;
    depth[x]=k;
    size[x]=s;
    d[x][k]=id[x][k]=0;
    destroy(root[x]);
    insert(-w[x],root[x]);
    if(s<=1)return;
    for(int i=0;i<(int)G[x].size();i++)if(!vis[G[x][i]]){
        p[G[x][i]]=0;
        d[G[x][i]][k]=W[x][i];
        siz[G[x][i]][k]=p[G[x][i]]=0;
        destroy(root1[G[x][i]][k]);
        dfs_getdis(G[x][i],x,G[x][i],k);
    }
    for(int i=0;i<(int)G[x].size();i++)if(!vis[G[x][i]])rebuild(G[x][i],k+1,size[G[x][i]],x);
}
void dfs_getcenter(int x,int s,int &u){
    size[x]=1;
    son[x]=0;
    for(int i=0;i<(int)G[x].size();i++)if(!vis[G[x][i]]&&G[x][i]!=p[x]){
        p[G[x][i]]=x;
        dfs_getcenter(G[x][i],s,u);
        size[x]+=size[G[x][i]];
        if(size[G[x][i]]>size[son[x]])son[x]=G[x][i];
    }
    if(!u||max(s-size[x],size[son[x]])<max(s-size[u],size[son[u]])
}
void dfs_getdis(int x,int u,int rt,int k){
    insert(d[x][k]-w[x],root[u]);
    insert(d[x][k]-w[x],root1[rt][k]);
    id[x][k]=rt;
    siz[rt][k]++;
    size[x]=1;
    for(int i=0;i<(int)G[x].size();i++)if(!vis[G[x][i]]&&G[x][i]!=p[x]){
        p[G[x][i]]=x;
        d[G[x][i]][k]=d[x][k]+W[x][i];
        dfs_getdis(G[x][i],u,rt,k);
        size[x]+=size[G[x][i]];
```

```cpp
    }
}
void dfs_destroy(int x,int k){
    vis[x]=false;
    for(int i=0;i<(int)G[x].size();i++)if(depth[G[x][i]]>=k&&G[x][i]!=p[x]){
        p[G[x][i]]=x;
        dfs_destroy(G[x][i],k);
    }
}
void insert(int x,node *&rt){
    if(rt==null){
        rt=new node(x);
        rt->ch[0]=rt->ch[1]=null;
        return;
    }
    int d=x>=rt->data;
    insert(x,rt->ch[d]);
    rt->refresh();
    if(rt->ch[d]->p<rt->p)rot(rt,d^1);
}
int order(int x,node *rt){
    int ans=0,d;
    x++;
    while(rt!=null){
        if((d=x>rt->data))ans+=rt->ch[0]->size+1;
        rt=rt->ch[d];
    }
    return ans;
}
void destroy(node *&x){
    if(x==null)return;
    destroy(x->ch[0]);
    destroy(x->ch[1]);
    delete x;
    x=null;
}
void rot(node *&x,int d){
    node *y=x->ch[d^1];
    x->ch[d^1]=y->ch[d];
    y->ch[d]=x;
    x->refresh();
    (x=y)->refresh();
}
```

5.5 LCT

5.5.1 不换根(弹飞绵羊)

```cpp
//Link-Cut Trees without Changing Root LCT不换根版本 O((n+m)\log n)
//By ysf
//通过题目，弹飞绵羊

//常数较大，请根据数据范围谨慎使用

#define isroot(x) ((x)!=(x)->p->ch[0]&&(x)!=(x)->p->ch[1])//判断是不是Splay的根
#define dir(x) ((x)==(x)->p->ch[1])//判断它是它父亲的左/右儿子

struct node{//结点类定义
    int size;//Splay的子树大小
    node *ch[2],*p;
    node():size(1){}
    void refresh(){size=ch[0]->size+ch[1]->size+1;}//附加信息维护
}null[maxn];

//在主函数开头加上这句初始化
null->size=0;

//初始化结点
void initalize(node *x){x->ch[0]=x->ch[1]=x->p=null;}//

//Access 均摊O(\log n)
//LCT核心操作，把结点到根的路径打通，顺便把与重儿子的连边变成轻边
//需要调用splay
node *access(node *x){
    node *y=null;
    while(x!=null){
        splay(x);
        x->ch[1]=y;
        (y=x)->refresh();
        x=x->p;
    }
    return y;
}

//Link 均摊O(\log n)
//把x的父亲设为y
//要求x必须为所在树的根节点，否则会导致后续各种莫名其妙的问题
//需要调用splay
void link(node *x,node *y){
    splay(x);
    x->p=y;
}

//Cut 均摊O(\log n)
//把x与其父亲的连边断掉
//x可以是所在树的根节点，这时此操作没有任何实质效果
//需要调用access和splay
void cut(node *x){
    access(x);
    splay(x);
    x->ch[0]->p=null;
    x->ch[0]=null;
    x->refresh();
}

//Splay 均摊O(\log n)
//需要调用旋转
void splay(node *x){
    while(!isroot(x)){
        if(isroot(x->p)){
            rot(x->p,dir(x)^1);
            break;
        }
        if(dir(x)==dir(x->p))rot(x->p->p,dir(x->p)^1);
        else rot(x->p,dir(x)^1);
        rot(x->p,dir(x)^1);
    }
}

//旋转，LCT版本，O(1)
//平衡树基本操作
//要求对应儿子必须存在，否则会导致后续各种莫名其妙的问题
void rot(node *x,int d){
    node *y=x->ch[d^1];
    y->p=x->p;
```

```cpp
    if(!isroot(x))x->p->ch[dir(x)]=y;
    if((x->ch[d^1]=y->ch[d])!=null)y->ch[d]->p=x;
    (y->ch[d]=x)->p=y;
    x->refresh();
    y->refresh();
}
```

5.5.2 换根/维护生成树(GREALD07加强版)

```cpp
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<map>
#define isroot(x) ((x)->p==null||((x)->p->ch[0]!=(x)&&(x)->p->ch[1]!=(x)))
#define dir(x) ((x)==(x)->p->ch[1])
using namespace std;
const int maxn=200010;
struct node{
    int key,mn,pos;
    bool rev;
    node *ch[2],*p;
    node(int key=(~0u)>>1):key(key),mn(key),pos(-1),rev(false){}
    inline void pushdown(){
        if(!rev)return;
        ch[0]->rev^=true;
        ch[1]->rev^=true;
        swap(ch[0],ch[1]);
        if(pos!=-1)pos^=1;
        rev=false;
    }
    inline void refresh(){
        mn=key;
        pos=-1;
        if(ch[0]->mn<mn){
            mn=ch[0]->mn;
            pos=0;
        }
        if(ch[1]->mn<mn){
            mn=ch[1]->mn;
            pos=1;
        }
    }
}null[maxn<<1],*ptr=null;
node *newnode(int);
node *access(node*);
void makeroot(node*);
void link(node*,node*);
void cut(node*,node*);
node *getroot(node*);
node *getmin(node*,node*);
void splay(node*);
void rot(node*,int);
void build(int,int,int&,int);
void query(int,int,int,int);
int sm[maxn<<5]={0},lc[maxn<<5]={0},rc[maxn<<5]={0},root[maxn]={0},cnt=0;
map<node*,pair<node*,node*> >mp;
node *tmp;
int n,m,q,tp,x,y,k,l,r,t,ans=0;
int main(){
    null->ch[0]=null->ch[1]=null->p=null;
    scanf("%d%d%d%d",&n,&m,&q,&tp);
    for(int i=1;i<=n;i++)newnode((~0u)>>1);
    for(int i=1;i<=m;i++){
        scanf("%d%d",&x,&y);
        if(x==y){
            root[i]=root[i-1];
            continue;
        }
        if(getroot(null+x)!=getroot(null+y)){
            tmp=newnode(i);
            k=0;
        }
        else{
            tmp=getmin(null+x,null+y);
            cut(tmp,mp[tmp].first);
            cut(tmp,mp[tmp].second);
            k=tmp->key;
            tmp->key=i;
            tmp->refresh();
        }
        link(tmp,null+x);
        link(tmp,null+y);
        mp[tmp]=make_pair(null+x,null+y);
        build(0,m-1,root[i],root[i-1]);
    }
    while(q--){
        scanf("%d%d",&l,&r);
        if(tp){
            l^=ans;
            r^=ans;
        }
        ans=n;
        t=--l;
        query(0,m-1,root[r],root[l]);
        printf("%d\n",ans);
    }
    return 0;
}
node *newnode(int x){
    *++ptr=node(x);
    ptr->ch[0]=ptr->ch[1]=ptr->p=null;
    return ptr;
}
node *access(node *x){
    node *y=null;
    while(x!=null){
        splay(x);
        x->ch[1]=y;
        (y=x)->refresh();
        x=x->p;
    }
    return y;
}
void makeroot(node *x){
    access(x);
    splay(x);
    x->rev^=true;
}
void link(node *x,node *y){
    makeroot(x);
    x->p=y;
}
void cut(node *x,node *y){
    makeroot(x);
    access(y);
    splay(y);
    y->ch[0]->p=null;
    y->ch[0]=null;
    y->refresh();
}
node *getroot(node *x){
    x=access(x);
    while(x->pushdown(),x->ch[0]!=null)x=x->ch[0];
```

```cpp
    splay(x);
    return x;
}
node *getmin(node *x,node *y){
    makeroot(x);
    x=access(y);
    while(x->pushdown(),x->pos!=-1)x=x->ch[x->pos];
    splay(x);
    return x;
}
void splay(node *x){
    x->pushdown();
    while(!isroot(x)){
        if(!isroot(x->p))x->p->p->pushdown();
        x->p->pushdown();
        x->pushdown();
        if(isroot(x->p)){
            rot(x->p,dir(x)^1);
            break;
        }
        if(dir(x)==dir(x->p))rot(x->p->p,dir(x->p)^1);
        else rot(x->p,dir(x)^1);
        rot(x->p,dir(x)^1);
    }
}
void rot(node *x,int d){
    node *y=x->ch[d^1];
    if((x->ch[d^1]=y->ch[d])!=null)y->ch[d]->p=x;
    y->p=x->p;
    if(!isroot(x))x->p->ch[dir(x)]=y;
    (y->ch[d]=x)->p=y;
    x->refresh();
    y->refresh();
}
void build(int l,int r,int &rt,int pr){
    sm[rt=++cnt]=sm[pr]+1;
    if(l==r)return;
    lc[rt]=lc[pr];
    rc[rt]=rc[pr];
    int mid=(l+r)>>1;
    if(k<=mid)build(l,mid,lc[rt],lc[pr]);
    else build(mid+1,r,rc[rt],rc[pr]);
}
void query(int l,int r,int rt,int pr){
    if(!rt&&!pr)return;
    if(t>=r){
        ans-=sm[rt]-sm[pr];
        return;
    }
    int mid=(l+r)>>1;
    query(l,mid,lc[rt],lc[pr]);
    if(t>mid)query(mid+1,r,rc[rt],rc[pr]);
}
```

5.5.3 维护子树信息

```cpp
//Link-Cut Trees with subtree values LCT维护子树信息
  ↪ O((n+m)\log n)
//By ysf
//通过题目□LOJ#558 我们的CPU遭到攻击□维护黑点到根距离和□

//这个东西虽然只需要抄板子但还是极其难写□常数极其巨大□没
  ↪ 必要的时候就不要用
//如果维护子树最小值就需要套一个可删除的堆来维护□复杂度
  ↪ 会变成O(n\log^2 n)
//注意由于这道题与边权有关□需要边权拆点变点权

//宏定义
#define isroot(x) ((x)->p==null||((x)!=(x)->p->ch[0]&&(x)!
  ↪ =(x)->p->ch[1]))
#define dir(x) ((x)==(x)->p->ch[1])

//节点类定义
struct node{//以维护子树中黑点到根距离和为例
    int w,chain_cnt,tree_cnt;
    long long sum,suml,sumr,tree_sum;//由于换根需要子树反
      ↪ 转□需要维护两个方向的信息
    bool rev,col;
    node *ch[2],*p;
    node():w(0),chain_cnt(0),tree_cnt(0),sum(0),suml(0),sumr(0),tre
      ↪ {}
    inline void pushdown(){
        if(!rev)return;
        ch[0]->rev^=true;
        ch[1]->rev^=true;
        swap(ch[0],ch[1]);
        swap(suml,sumr);
        rev=false;
    }
    inline void refresh(){//不多解释了……这毒瘤题恶心的要
      ↪ 死□我骂我自己.png
        sum=ch[0]->sum+ch[1]->sum+w;
        suml=(ch[0]->rev?ch[0]->sumr:ch[0]->suml)+(ch[1]->rev?ch[1
            +(tree_cnt+ch[1]->chain_cnt)*(ch[0]->sum+w)+tree_sum;
        sumr=(ch[0]->rev?ch[0]->suml:ch[0]->sumr)+(ch[1]->rev?ch[1
            +(tree_cnt+ch[0]->chain_cnt)*(ch[1]->sum+w)+tree_sum;
        chain_cnt=ch[0]->chain_cnt+ch[1]->chain_cnt+tree_cnt;
    }
}null[maxn<<1];//如果没有边权变点权就不用乘2了

//封装构造函数
node *newnode(int w){
    node *x=nodes.front();
    nodes.pop();
    initalize(x);
    x->w=w;
    x->refresh();
    return x;
}

//封装初始化函数
//记得在进行操作之前对所有结点调用一遍
inline void initalize(node *x){
    *x=node();
    x->ch[0]=x->ch[1]=x->p=null;
}

//Access函数
//注意一下在Access的同时更新子树信息的方法
node *access(node *x){
    node *y=null;
    while(x!=null){
        splay(x);
        x->tree_cnt+=x->ch[1]->chain_cnt-y->chain_cnt;
        x->tree_sum+=(x->ch[1]->rev?x->ch[1]->sumr:x->ch[1]->suml)
        x->ch[1]=y;
        (y=x)->refresh();
        x=x->p;
    }
    return y;
}

//找到一个点所在连通块的根
//对比原版没有变化
```

```cpp
node *getroot(node *x){
    x=access(x);
    while(x->pushdown(),x->ch[0]!=null)x=x->ch[0];
    splay(x);
    return x;
}

//换根，同样没有变化
void makeroot(node *x){
    access(x);
    splay(x);
    x->rev^=true;
    x->pushdown();
}

//连接两个点
//注意这里必须把两者都变成根，因为只能修改根结点
void link(node *x,node *y){
    makeroot(x);
    makeroot(y);
    x->p=y;
    y->tree_cnt+=x->chain_cnt;
    y->tree_sum+=x->suml;
    y->refresh();
}

//删除一条边
//对比原版没有变化
void cut(node *x,node *y){
    makeroot(x);
    access(y);
    splay(y);
    y->ch[0]->p=null;
    y->ch[0]=null;
    y->refresh();
}

//修改/询问一个点，这里以询问为例
//如果是修改就在换根之后搞一些操作
long long query(node *x){
    makeroot(x);
    return x->suml;
}

//Splay函数
//对比原版没有变化
void splay(node *x){
    x->pushdown();
    while(!isroot(x)){
        if(!isroot(x->p))x->p->p->pushdown();
        x->p->pushdown();
        x->pushdown();
        if(isroot(x->p)){
            rot(x->p,dir(x)^1);
            break;
        }
        if(dir(x)==dir(x->p))rot(x->p->p,dir(x->p)^1);
        else rot(x->p,dir(x)^1);
        rot(x->p,dir(x)^1);
    }
}

//旋转函数
//对比原版没有变化
void rot(node *x,int d){
    node *y=x->ch[d^1];
    if((x->ch[d^1]=y->ch[d])!=null)y->ch[d]->p=x;
    y->p=x->p;
    if(!isroot(x))x->p->ch[dir(x)]=y;
    (y->ch[d]=x)->p=y;
    x->refresh();
    y->refresh();
}
```

5.5.4 模板题:动态QTREE4(询问树上相距最远点)

```cpp
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
#include<ext/pb_ds/priority_queue.hpp>

#define isroot(x) ((x)->p==null||((x)!=(x)->p->ch[0]&&(x)!=(x)->p->ch[1]))
#define dir(x) ((x)==(x)->p->ch[1])

using namespace std;
using namespace __gnu_pbds;

const int maxn=100010;
const long long INF=1000000000000000000ll;

struct binary_heap{
    __gnu_pbds::priority_queue<long long,less<long long>,binary_heap_tag>q1,q2;
    binary_heap(){}
    void push(long long x){if(x>(-INF)>>2)q1.push(x);}
    void erase(long long x){if(x>(-INF)>>2)q2.push(x);}
    long long top(){
        if(empty())return -INF;
        while(!q2.empty()&&q1.top()==q2.top()){
            q1.pop();
            q2.pop();
        }
        return q1.top();
    }
    long long top2(){
        if(size()<2)return -INF;
        long long a=top();
        erase(a);
        long long b=top();
        push(a);
        return a+b;
    }
    int size(){return q1.size()-q2.size();}
    bool empty(){return q1.size()==q2.size();}
}heap;//全局堆维护每条链的最大子段和
struct node{
    long long sum,maxsum,prefix,suffix;
    int key;
    binary_heap heap;//每个点的堆存的是它的子树中到它的最远距离，如果它是黑点的话还会包括自己
    node *ch[2],*p;
    bool rev;
    node(int k=0):sum(k),maxsum(-INF),prefix(-INF),suffix(-INF),key(k),
    {}
    inline void pushdown(){
        if(!rev)return;
        ch[0]->rev^=true;
        ch[1]->rev^=true;
        swap(ch[0],ch[1]);
        swap(prefix,suffix);
        rev=false;
    }
```

```
    inline void refresh(){
        pushdown();
        ch[0]->pushdown();
        ch[1]->pushdown();
        sum=ch[0]->sum+ch[1]->sum+key;
        prefix=max(ch[0]->prefix,ch[0]->sum+key+ch[1]->prefix);
        suffix=max(ch[1]->suffix,ch[1]->sum+key+ch[0]->suffix);
        maxsum=max(max(ch[0]->maxsum,ch[1]->maxsum),ch[0]->suffix+key+ch[1]->prefix);
        if(!heap.empty()){
            prefix=max(prefix,ch[0]->sum+key+heap.top());
            suffix=max(suffix,ch[1]->sum+key+heap.top());
            maxsum=max(maxsum,max(ch[0]->suffix,ch[1]->prefix)+key+heap.top());
            if(heap.size()>1){
                maxsum=max(maxsum,heap.top2()+key);
            }
        }
    }
}null[maxn<<1],*ptr=null;
void addedge(int,int,int);
void deledge(int,int);
void modify(int,int,int);
void modify_color(int);
node *newnode(int);
node *access(node*);
void makeroot(node*);
void link(node*,node*);
void cut(node*,node*);
void splay(node*);
void rot(node*,int);
queue<node*>freenodes;
tree<pair<int,int>,node*>mp;
bool col[maxn]={false};
char c;
int n,m,k,x,y,z;
int main(){
    null->ch[0]=null->ch[1]=null->p=null;
    scanf("%d%d%d",&n,&m,&k);
    for(int i=1;i<=n;i++){
        newnode(0);
    }
    heap.push(0);
    while(k--){
        scanf("%d",&x);
        col[x]=true;
        null[x].heap.push(0);
    }
    for(int i=1;i<n;i++){
        scanf("%d%d%d",&x,&y,&z);
        if(x>y)swap(x,y);
        addedge(x,y,z);
    }
    while(m--){
        scanf(" %c%d",&c,&x);
        if(c=='A'){
            scanf("%d",&y);
            if(x>y)swap(x,y);
            deledge(x,y);
        }
        else if(c=='B'){
            scanf("%d%d",&y,&z);
            if(x>y)swap(x,y);
            addedge(x,y,z);
        }
        else if(c=='C'){
            scanf("%d%d",&y,&z);
```

```
            if(x>y)swap(x,y);
            modify(x,y,z);
        }
        else modify_color(x);
        printf("%lld\n",(heap.top()>0?heap.top():-1));
    }
    return 0;
}
void addedge(int x,int y,int z){
    node *tmp;
    if(freenodes.empty())tmp=newnode(z);
    else{
        tmp=freenodes.front();
        freenodes.pop();
        *tmp=node(z);
    }
    tmp->ch[0]=tmp->ch[1]=tmp->p=null;heap.push(tmp->maxsum);
    link(tmp,null+x);
    link(tmp,null+y);
    mp[make_pair(x,y)]=tmp;
}
void deledge(int x,int y){
    node *tmp=mp[make_pair(x,y)];
    cut(tmp,null+x);
    cut(tmp,null+y);
    freenodes.push(tmp);
    heap.erase(tmp->maxsum);
    mp.erase(make_pair(x,y));
}
void modify(int x,int y,int z){
    node *tmp=mp[make_pair(x,y)];
    makeroot(tmp);
    tmp->pushdown();
    heap.erase(tmp->maxsum);
    tmp->key=z;
    tmp->refresh();
    heap.push(tmp->maxsum);
}
void modify_color(int x){
    makeroot(null+x);
    col[x]^=true;
    if(col[x])null[x].heap.push(0);
    else null[x].heap.erase(0);
    heap.erase(null[x].maxsum);
    null[x].refresh();
    heap.push(null[x].maxsum);
}
node *newnode(int k){
    *(++ptr)=node(k);
    ptr->ch[0]=ptr->ch[1]=ptr->p=null;
    return ptr;
}
node *access(node *x){
    splay(x);
    heap.erase(x->maxsum);
    x->refresh();
    if(x->ch[1]!=null){
        x->ch[1]->pushdown();
        x->heap.push(x->ch[1]->prefix);x->refresh();
        heap.push(x->ch[1]->maxsum);
    }
    x->ch[1]=null;
    x->refresh();
    node *y=x;
    x=x->p;
    while(x!=null){
        splay(x);
```

```cpp
        heap.erase(x->maxsum);
        if(x->ch[1]!=null){
            x->ch[1]->pushdown();
            x->heap.push(x->ch[1]->prefix);
            heap.push(x->ch[1]->maxsum);
        }
        x->heap.erase(y->prefix);
        x->ch[1]=y;
        (y=x)->refresh();
        x=x->p;
    }
    heap.push(y->maxsum);
    return y;
}
void makeroot(node *x){
    access(x);
    splay(x);
    x->rev^=true;
}
void link(node *x,node *y){//新添一条虚边□维护y对应的堆
    makeroot(x);
    makeroot(y);
    x->pushdown();
    x->p=y;
    heap.erase(y->maxsum);
    y->heap.push(x->prefix);
    y->refresh();
    heap.push(y->maxsum);
}
void cut(node *x,node *y){//断开一条实边□一条链变成两条链□需要维护全局堆
    makeroot(x);
    access(y);
    splay(y);
    heap.erase(y->maxsum);
    heap.push(y->ch[0]->maxsum);
    y->ch[0]->p=null;
    y->ch[0]=null;
    y->refresh();
    heap.push(y->maxsum);
}
void splay(node *x){
    x->pushdown();
    while(!isroot(x)){
        if(!isroot(x->p))x->p->p->pushdown();
        x->p->pushdown();
        x->pushdown();
        if(isroot(x->p)){
            rot(x->p,dir(x)^1);
            break;
        }
        if(dir(x)==dir(x->p))rot(x->p->p,dir(x->p)^1);
        else rot(x->p,dir(x)^1);
        rot(x->p,dir(x)^1);
    }
}
void rot(node *x,int d){
    node *y=x->ch[d^1];
    if((x->ch[d^1]=y->ch[d])!=null)y->ch[d]->p=x;
    y->p=x->p;
    if(!isroot(x))x->p->ch[dir(x)]=y;
    (y->ch[d]=x)->p=y;
    x->refresh();
    y->refresh();
}
```

5.6 长链剖分,梯子剖分

```cpp
//Long-chain Subdivision 长链剖分 O(n)
//By ysf
//通过题目□vijos Lxhgww的奇思妙想□板子题□、Codeforces 1009F

//顾名思义□长链剖分是取最深的儿子作为重儿子
//长链剖分的两个应用□
//O(1)在线求一个点的第k祖先
//O(n)维护以深度为下标的子树信息

//-----------------------------分割线-----------------------------

//在线求一个点的第k祖先 O(n\log n)-O(1)
//其中O(n\log n)预处理是因为需要用到倍增
//理论基础□任意一个点x的k级祖先y所在长链长度一定>=k

//全局数组定义
vector<int>G[maxn],v[maxn];
int d[maxn],mxd[maxn],son[maxn],top[maxn],len[maxn];
int f[maxn][19],log_tbl[maxn];

//在主函数中两遍dfs之后加上如下预处理
log_tbl[0]=-1;
for(int i=1;i<=n;i++)log_tbl[i]=log_tbl[i>>1]+1;
for(int j=1;(1<<j)<n;j++)
    for(int i=1;i<=n;i++)
        f[i][j]=f[f[i][j-1]][j-1];

//第一遍dfs□用于计算深度和找出重儿子
//递归调用自身
void dfs1(int x){
    mxd[x]=d[x];
    for(int i=0;i<(int)G[x].size();i++)
        if(G[x][i]!=f[x][0]){
            f[G[x][i]][0]=x;
            d[G[x][i]]=d[x]+1;
            dfs1(G[x][i]);
            mxd[x]=max(mxd[x],mxd[G[x][i]]);
            if(mxd[G[x][i]]>mxd[son[x]])son[x]=G[x][i];
        }
}

//第二遍dfs□用于进行剖分和预处理梯子剖分□每条链向上延伸一倍□数组
//递归调用自身
void dfs2(int x){
    top[x]=(x==son[f[x][0]]?top[f[x][0]]:x);
    for(int i=0;i<(int)G[x].size();i++)
        if(G[x][i]!=f[x][0])dfs2(G[x][i]);
    if(top[x]==x){
        int u=x;
        while(top[son[u]]==x)u=son[u];
        len[x]=d[u]-d[x];
        for(int i=0;i<len[x];i++,u=f[u][0])v[x].push_back(u);
        u=x;
        for(int i=0;i<len[x]&&u;i++,u=f[u][0])v[x].push_back(u);
    }
}

//在线询问x的k级祖先 O(1)
//不存在时返回0
int query(int x,int k){
```

```cpp
    if(!k)return x;
    if(k>d[x])return 0;
    x=f[x][log_tbl[k]];
    k^=1<<log_tbl[k];
    return v[top[x]][d[top[x]]+len[top[x]]-d[x]+k];
}

//----------------------------分割线----------------------------

//O(n)维护以深度为下标的子树信息

vector<int>G[maxn],v[maxn];
int n,p[maxn],h[maxn],son[maxn],ans[maxn];

//原题题意，求每个点的子树中与它距离是几的点最多，相同的取最大深度
//由于vector只能在后面加入元素，为了写代码方便，这里反过来存
void dfs(int x){
    h[x]=1;
    for(int i=0;i<(int)G[x].size();i++)
        if(G[x][i]!=p[x]){
            p[G[x][i]]=x;
            dfs(G[x][i]);
            if(h[G[x][i]]>h[son[x]])son[x]=G[x][i];
        }
    if(!son[x]){
        v[x].push_back(1);
        ans[x]=0;
        return;
    }
    //printf("x=%d h=%d son=%d\n",x,h[x],son[x]);
    h[x]=h[son[x]]+1;
    swap(v[x],v[son[x]]);
    if(v[x][ans[son[x]]]==1)ans[x]=h[x]-1;
    else ans[x]=ans[son[x]];
    v[x].push_back(1);
    int mx=v[x][ans[x]];
    for(int i=0;i<(int)G[x].size();i++)
        if(G[x][i]!=p[x]&&G[x][i]!=son[x]){
            for(int j=1;j<=h[G[x][i]];j++){
                v[x][h[x]-j-1]+=v[G[x][i]][h[G[x][i]]-j];
                int t=v[x][h[x]-j-1];
                if(t>mx||(t==mx&&h[x]-j-1>ans[x])){
                    mx=t;
                    ans[x]=h[x]-j-1;
                }
            }
            v[G[x][i]].clear();
        }
}
```

5.7 左偏树

参见k短路

5.8 STL

5.9 pb_ds

5.10 rope

5.11 常见根号思路

6. 动态规划

6.1 决策单调性 $O(n \log n)$

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

using namespace std;

const int maxn = 300005;

int a[maxn], q[maxn], p[maxn], g[maxn]; // 存左端点，右端点就是下一个左端点 - 1

long long f[maxn], s[maxn];

int n, m;

long long calc(int l, int r) {
    if (r < l)
        return 0;

    int mid = (l + r) / 2;
    if ((r - l + 1) % 2 == 0)
        return (s[r] - s[mid]) - (s[mid] - s[l - 1]);
    else
        return (s[r] - s[mid]) - (s[mid - 1] - s[l - 1]);
}

int solve(long long tmp) {
    memset(f, 63, sizeof(f));
    f[0] = 0;

    int head = 1, tail = 0;

    // printf("---------- solve(%lld) ----------\n", tmp);

    for (int i = 1; i <= n; i++) {
        f[i] = calc(1, i);
        g[i] = 1;

        while (head < tail && p[head + 1] <= i)
            head++;
        if (head <= tail) {
            if (f[q[head]] + calc(q[head] + 1, i) < f[i]) {
                f[i] = f[q[head]] + calc(q[head] + 1, i);
                g[i] = g[q[head]] + 1;
            }
            while (head < tail && p[head + 1] <= i + 1)
                head++;
            if (head <= tail)
                p[head] = i + 1;
        }
        f[i] += tmp;
        // printf("f[%d] = %lld g[%d] = %d\n", i, f[i], i, g[i]);

        /*
        if (head <= tail && f[q[tail]] + calc(q[tail] + 1, n) <= f[i] + calc(i + 1, n))
            continue;
```

```
        */

        int r = n;

        while(head <= tail) {
            if (f[q[tail]] + calc(q[tail] + 1, p[tail]) > f[i] + calc(i + 1, p[tail])) {
                r = p[tail] - 1;
                tail--;
            }
            else if (f[q[tail]] + calc(q[tail] + 1, r) <= f[i] + calc(i + 1, r)) {
                if (r < n) {
                    q[++tail] = i;
                    p[tail] = r + 1;
                }
                break;
            }
            else {
                int L = p[tail], R = r;
                while (L < R) {
                    int M = (L + R) / 2;

                    if (f[q[tail]] + calc(q[tail] + 1, M) <= f[i] + calc(i + 1, M))
                        L = M + 1;
                    else
                        R = M;
                }

                q[++tail] = i;
                p[tail] = L;

                break;
            }
        }
        if (head > tail) {
            q[++tail] = i;
            p[tail] = i + 1;
        }
    }

    return g[n];
}

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

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

    long long L = 0, R = 1e16;

    while (L < R) {
        long long M = (L + R) / 2;
        if (solve(M) > m)
            L = M + 1;
        else
            R = M;
    }

    solve(L);

    printf("%lld\n", f[n] - m * L);

    return 0;
}
```

7. 其他算法

7.1 $O(1)$快速乘

7.2 $O(n^2)$高精度

7.3 xorshift

8. 参考资料

8.1 常见数列

8.1.1 伯努利数

$$B(x) = \sum_{i\geq 0} \frac{B_i x^i}{i!} = \frac{x}{e^x - 1}$$

$$B_n = [n = 0] - \sum_{i=0}^{n-1} \binom{n}{i} \frac{B_i}{n - k + 1}$$

$$\sum_{i=0}^{n} \binom{n+1}{i} B_i = 0$$

$$S_n(m) = \sum_{i=0}^{m-1} i^n = \sum_{i=0}^{n} \binom{n}{i} B_{n-i} \frac{m^{i+1}}{i+1}$$

9. 注意事项

9.1 常见下毒手法

- 高精度高低位搞反了吗
- 线性筛抄对了吗
- sort比较函数是不是比了个寂寞
- 该取模的地方都取模了吗
- 边界情况(+1-1之类的)有没有想清楚
- 特判是否有必要,确定写对了吗

9.2 场外相关

- 安顿好之后查一下附近的咖啡店,打印店,便利店之类的位置,以备不时之需
- 热身赛记得检查一下编译注意事项中的代码能否过编译,还有熟悉比赛场地,清楚洗手间在哪儿,测试打印机(如果可以)
- 比赛前至少要翻一遍板子,尤其要看原理与例题
- 比赛前一两天不要摸鱼,要早睡,有条件最好洗个澡;比赛当天不要起太晚,维持好的状态
- 赛前记得买咖啡,最好直接安排三人份,记得要咖啡因比较足的;如果主办方允许,就带些巧克力之类的高热量零食
- 入场之后记得检查机器,尤其要逐个检查键盘按键有没有坏的;如果可以的话,调一下gedit设置
- 开赛之前调整好心态,比赛而已,不必心急.

9.3 做题策略与心态调节

- 拿到题后立刻按照商量好的顺序读题,前半小时最好跳过题意太复杂的题(除非被过穿了)
- 签到题写完不要激动,稍微检查一下最可能的下毒点再交,避免无谓的罚时
  - 一两行的那种傻逼题就算了
- 读完题及时输出题意,一方面避免重复读题,一方面也可以让队友有一个初步印象,方便之后决定开题顺序
- 一个题如果卡了很久又有其他题可以写,那不妨先放掉写更容易的题,不要在一棵树上吊死
  - 不要被一两道题搞得心态爆炸,一方面急也没有意义,一方面你很可能真的离AC就差一步
- 榜是不会骗人的,一个题如果被不少人过了就说明这个题很可能并没有那么难;如果不是有十足的把握就不要轻易开没什么人交的题;另外不要忘记最后一小时会封榜
- 想不出题/找不出毒自然容易犯困,一定不要放任自己昏昏欲睡,最好去洗手间冷静一下,没有条件就站起来踱步
- 思考的时候不要挂机,一定要在草稿纸上画一画,最好说出声来最不容易断掉思路
- 出完算法一定要check一下样例和一些trivial的情况,不然容易写了半天发现写了个假算法
- 上机前有时间就提前给需要思考怎么写的地方打草稿,不要浪费机时
- 查毒时如果最难的地方反复check也没有问题,就从头到脚仔仔细细查一遍,不要放过任何细节,即使是并查集和sort这种东西也不能想当然
- 后半场如果时间不充裕就不要冒险开难题,除非真的无事可做
  - 如果是没写过的东西也不要轻举妄动,在有其他好写的题的时候就等一会再说
- 大多数时候都要听队长安排,虽然不一定最正确但可以保持组织性
- 最好注意一下影响,就算忍不住嘴臭也不要太大声
- 任何时候都不要着急,着急不能解决问题,不要当喆国王
- 输了游戏,还有人生;赢了游戏,还有人生.