Standard Code Library

AntiLeaf

Contents

1 图论 2
1.1 最小生成树 2
1.1.1 动态最小生成树 2
1.2 最短路 4
1.2.1 k短路 4
1.3 仙人掌 5
1.3.1 仙人掌DP 5
1.4 二分图 6
1.4.1 KM二分图最大权匹配 6
1.5 一般图匹配 7
1.5.1 高斯消元 7
1.5.2 带花树 8
1.6 最大流 9
1.6.1 Dinic 9
1.6.2 ISAP 10
1.6.3 HLPP最高标号预流推进 11

2 字符串 12
2.1 AC自动机 12
2.2 后缀数组 13
2.2.1 SAMSA 13
2.3 后缀自动机 13
2.4 回文树 14
2.4.1 广义回文树 14
2.5 Manacher马拉车 16
2.6 KMP 16
2.6.1 ex-KMP 16

3 数学 17
3.1 插值 17
3.1.1 牛顿插值 17
3.1.2 拉格朗日插值 17
3.2 多项式 17
3.2.1 FFT 17
3.2.2 NTT 17
3.2.3 任意模数卷积 三模数NTT 18
3.2.4 多项式操作 18
3.2.5 拉格朗日反演 19
3.2.6 半在线卷积 19
3.3 FWT快速沃尔什变换 20
3.4 单纯形 20
3.5 线性代数 21
3.5.1 线性基 21

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

```

```cpp
    m=cnt;
}


// 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];

```

```cpp
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

```
// 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);
```

```cpp
    }

    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

```cpp
// 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;
```

```cpp

                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

```cpp
// 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)
```

```cpp
            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

```cpp
// 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[{
            e[i].cap-=f;
            e[i^1].cap+=f;
            flow+=f;
            a-=f;
            if(!a)break;
        }
    return flow;
}
```

1.6.2 ISAP

```cpp
// 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));

    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

```
#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0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;
            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

```
// 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

```

```cpp
    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

```cpp
//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{
```

```cpp
            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

```cpp
//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

```cpp
#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;

        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
```

```cpp
//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[]={998244353,1004535809,1045430273},
    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

```cpp
//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
//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

```
// 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

```
//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