

# 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

```

1 // 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
2 // 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
3 // 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
4 // - Reduction待修改边标为+INF跑MST后把非树边删掉减少无
  ↳ 用边
5 // - 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
6 // 每轮分治需要Reduction-Contraction借此减少不确定边从而
  ↳ 保证复杂度
7 // 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)
8
9 // 全局结构体与数组定义
10 struct edge { //边的定义
11     int u, v, w, id; // id表示边在原图中的编号
12     bool vis; // 在Kruskal时用记录这条边是否是树边
13     bool operator < (const edge &e) const { return w < e.w; }
14 } e[20][maxn], t[maxn]; // 为了便于回滚在每层分治存一个副
  ↳ 本
15
16 // 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
17 struct A {
18     int id, u, v;
19 } a[maxn];
20
21 int id[20][maxn]; // 每条边在当前图中的编号
22 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
23 int n, m, q; // 点数边数修改数
24
25 // 方便起见附上可能需要用到的预处理代码
26 for (int i = 1; i <= n; i++) { // 并查集初始化
27     p[i] = i;
28     size[i] = 1;
29 }
30
31 for (int i = 1; i <= m; i++) { // 读入与预标号
32     scanf("%d%d%d", &e[0][i].u, &e[0][i].v, &e[0][i].w);
33     e[0][i].id = i;
34     id[0][i] = i;
35 }
36
37 for (int i = 1; i <= q; i++) { // 预处理出调用数组
38     scanf("%d%d", &a[i].id, &a[i].v);
39     a[i].u = e[0][a[i].id].w;
40     e[0][a[i].id].w = a[i].v;
41 }
42
43 for (int i = q; i >= 1; i--)
44     e[0][a[i].id].w = a[i].u;
45
46 CDQ(1, q, 0, m, 0); // 这是调用方法

```

```

51
52
53 // 分治主过程 O(nLog^2n)
54 // 需要调用Reduction和Contraction
55 void CDQ(int l, int r, int d, int m, long long ans) { //
  ↳ CDQ分治
56     if (l == r) { // 区间长度已减小到1输出答案退出
57         e[d][id[d][a[l].id]].w = a[l].v;
58         printf("%lld\n", ans + Kruskal(m, e[d]));
59         e[d][id[d][a[l].id]].w = a[l].u;
60         return;
61     }
62
63     int tmp = top;
64
65     Reduction(l, r, d, m);
66     ans += Contraction(l, r, d, m); // R-C
67
68     int mid = (l + r) / 2;
69
70     copy(e[d] + 1, e[d] + m + 1, e[d + 1] + 1);
71     for (int i = 1; i <= m; i++)
72         id[d + 1][e[d][i].id] = i; // 准备好下一层要用的数组
73
74     CDQ(l, mid, d + 1, m, ans);
75
76     for (int i = 1; i <= mid; i++)
77         e[d][id[d][a[i].id]].w = a[i].v; // 进行左边的修改
78
79     copy(e[d] + 1, e[d] + m + 1, e[d + 1] + 1);
80     for (int i = 1; i <= m; i++)
81         id[d + 1][e[d][i].id] = i; // 重新准备下一层要用的数
  ↳ 组
82
83     CDQ(mid + 1, r, d + 1, m, ans);
84
85     for (int i = top; i > tmp; i--)
86         cut(stk[i]); // 撤销所有操作
87     top = tmp;
88 }
89
90
91 // 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
92 // 需要调用Kruskal
93 void Reduction(int l, int r, int d, int &m) {
94     for (int i = l; i <= r; i++)
95         e[d][id[d][a[i].id]].w = INF; // 待修改的边标为INF
96
97     Kruskal(m, e[d]);
98
99     copy(e[d] + 1, e[d] + m + 1, t + 1);
100
101     int cnt = 0;
102     for (int i = 1; i <= m; i++)
103         if (t[i].w == INF || t[i].vis) { // 非树边扔掉
104             id[d][t[i].id] = ++cnt; // 给边重新编号
105             e[d][cnt] = t[i];
106         }
107
108     for (int i = r; i >= l; i--)
109         e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边改回
  ↳ 去
110
111

```

```

111     m=cnt;
112 }
113
114
115 // 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
116 // 返回缩掉的边的总权值
117 // 需要调用Kruskal
118 long long Contraction(int l, int r, int d, int &m) {
119     long long ans = 0;
120
121     for (int i = l; i <= r; i++)
122         e[d][id[d][a[i].id]].w = -INF; // 待修改边标为-INF
123
124     Kruskal(m, e[d]);
125     copy(e[d] + 1, e[d] + m + 1, t + 1);
126
127     int cnt = 0;
128     for (int i = 1; i <= m; i++) {
129
130         if (t[i].w != -INF && t[i].vis) { // 必须边
131             ans += t[i].w;
132             mergeset(t[i].u, t[i].v);
133         }
134         else { // 不确定边
135             id[d][t[i].id] = ++cnt;
136             e[d][cnt] = t[i];
137         }
138     }
139
140     for (int i = r; i >= l; i--) {
141         e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边改回
            ↳ 去
142         e[d][id[d][a[i].id]].vis = false;
143     }
144
145     m = cnt;
146
147     return ans;
148 }
149
150
151 // Kruskal算法 O(mlogn)
152 // 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
153 long long Kruskal(int m, edge *e) {
154     int tmp = top;
155     long long ans = 0;
156
157     sort(e + 1, e + m + 1); // 比较函数在结构体中定义过了
158
159     for (int i = 1; i <= m; i++) {
160         if (findroot(e[i].u) != findroot(e[i].v)) {
161             e[i].vis = true;
162             ans += e[i].w;
163             mergeset(e[i].u, e[i].v);
164         }
165         else
166             e[i].vis = false;
167     }
168
169     for(int i = top; i > tmp; i--)
170         cut(stk[i]); // 撤销所有操作
171     top = tmp;

```

```

172     return ans;
173 }
174
175
176 // 以下是并查集相关函数
177 int findroot(int x) { // 因为需要撤销 不写路径压缩
178     while (p[x] != x)
179         x = p[x];
180
181     return x;
182 }
183
184 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
186     x = findroot(x); // 但是按秩合并要再开一个栈记录合并之
        ↳ 前的秩
187     y = findroot(y);
188
189     if (x == y)
190         return;
191
192     if (size[x] > size[y])
193         swap(x, y);
194
195     p[x] = y;
196     size[y] += size[x];
197     stk[++top] = x;
198 }
199
200 void cut(int x) { // 并查集撤销
201     int y = x;
202
203     do
204         size[y = p[y]] -= size[x];
205     while (p[y] != y);
206
207     p[x] = x;
208 }

```

## 1.2 最短路

### 1.2.1 k短路

```

1 //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*
2 //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
3 //以下代码以Dijkstra+可持久化左偏树为例
4
5 const int maxn=1005, maxe=10005, maxm=maxe*30; //点数 边数 左偏
    ↳ 树结点数
6
7 //需要用到的结构体定义
8 struct A { //用来求最短路
9     int x, d;
10     A(int x, int d): x(x), d(d) {}
11     bool operator<(const A &a) const { return d > a.d; }
12 };
13
14 struct node { //左偏树结点
15     int w, i, d; // i 最后一条边的编号 d 左偏树附加信息
16     node *lc, *rc;
17     node() {}
18     node(int w, int i): w(w), i(i), d(0) {}
19     void refresh() { d = rc->d + 1; }

```

```

20 }null[maxm],*ptr=null,*root[maxn];
21
22 struct B{//维护答案用
23     int x,w;//x是结点编号w表示之前已经产生的权值
24     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
25     B(int x,node *rt,int w):x(x),w(w),rt(rt){}
26     bool operator<(const B &a)const{return
        ↳ w+rt->w>a.w+a.rt->w;}
27 };
28
29 //全局变量和数组定义
30 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
31 bool vis[maxn],used[maxe];//used表示边是否在最短路树上
32 int u[maxe],v[maxe],w[maxe];//存下每条边注意是有向边
33 int d[maxn],p[maxn];//p表示最短路树上每个点的父边
34 int n,m,k,s,t;//s,t分别表示起点和终点
35
36 //以下是主函数中较关键的部分
37 for(int i=0;i<=n;i++)root[i]=null;//一定要加上root[0]
38 //(读入&建反向图)
39 Dijkstra();
40 //(清空G,W,id)
41 for(int i=1;i<=n;i++){
42     if(p[i]){
43         used[p[i]]=true;//在最短路树上
44         G[v[p[i]]].push_back(i);
45     }
46     for(int i=1;i<=m;i++){
47         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
48         if(!used[i])
49             root[u[i]]=merge(root[u[i]],newnode(w[i],i));
50     }
51     dfs(t);
52     priority_queue<B>heap;
53     heap.push(B(s,root[s],0));//初始状态是找贡献最小的边加进去
54     printf("%d\n",d[s]);//第1短路需要特判
55     while(--k){//其余k-1短路径用二叉堆维护
56         if(heap.empty())printf("-1\n");
57         else{
58             int x=heap.top().x,w=heap.top().w;
59             node *rt=heap.top().rt;
60             heap.pop();
61             printf("%d\n",d[s]+w+rt->w);
62             if(rt->lc!=null||rt->rc!=null)
63                 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
64             if(root[v[rt->i]]!=null)
65                 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
66         }
67     }
68 //主函数到此结束
69
70 //Dijkstra预处理最短路 O(m\log n)
71 void Dijkstra(){
72     memset(d,63,sizeof(d));
73     d[t]=0;
74     priority_queue<A>heap;
75     heap.push(A(t,0));
76     while(!heap.empty()){
77         int x=heap.top().x;

```

```

78         heap.pop();
79         if(vis[x])continue;
80         vis[x]=true;
81         for(int i=0;i<(int)G[x].size();i++){
82             if(!vis[G[x][i]]&&d[G[x][i]]>d[x]+W[x][i]){
83                 d[G[x][i]]=d[x]+W[x][i];
84                 p[G[x][i]]=id[x][i];
85                 heap.push(A(G[x][i],d[G[x][i]]));
86             }
87         }
88     }
89
90 //dfs求出每个点的堆 总计O(m\log n)
91 //需要调用merge同时递归调用自身
92 void dfs(int x){
93     root[x]=merge(root[x],root[v[p[x]]]);
94     for(int i=0;i<(int)G[x].size();i++)
95         dfs(G[x][i]);
96 }
97
98 //包装过的new node() O(1)
99 node *newnode(int w,int i){
100     *++ptr=node(w,i);
101     ptr->lc=ptr->rc=null;
102     return ptr;
103 }
104
105 //带可持久化的左偏树合并 总计O(\log n)
106 //递归调用自身
107 node *merge(node *x,node *y){
108     if(x==null)return y;
109     if(y==null)return x;
110     if(x->w>y->w)swap(x,y);
111     node *z=newnode(x->w,x->i);
112     z->lc=x->lc;
113     z->rc=merge(x->rc,y);
114     if(z->lc->d>z->rc->d)swap(z->lc,z->rc);
115     z->refresh();
116     return z;
117 }

```

## 1.3 仙人掌

### 1.3.1 仙人掌DP

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int maxn = 200005;
6
7 struct edge{
8     int to, w, prev;
9 }e[maxn * 2];
10
11 vector<pair<int, int>> v[maxn];
12
13 vector<long long> d[maxn];
14
15 stack<int> stk;
16
17 int p[maxn];
18
19 bool vis[maxn], vise[maxn * 2];
20

```

```

21 int last[maxn], cnte;
22
23 long long f[maxn], g[maxn], sum[maxn];
24
25 int n, m, cnt;
26
27 void addedge(int x, int y, int w) {
28     v[x].push_back(make_pair(y, w));
29 }
30
31 void dfs(int x) {
32
33     vis[x] = true;
34
35     for (int i = last[x]; ~i; i = e[i].prev) {
36         if (vise[i ^ 1])
37             continue;
38
39         int y = e[i].to, w = e[i].w;
40
41         vise[i] = true;
42
43         if (!vis[y]) {
44             stk.push(i);
45             p[y] = x;
46             dfs(y);
47
48             if (!stk.empty() && stk.top() == i) {
49                 stk.pop();
50                 addedge(x, y, w);
51             }
52         }
53
54         else {
55             cnt++;
56
57             long long tmp = w;
58             while (!stk.empty()) {
59                 int i = stk.top();
60                 stk.pop();
61
62                 int yy = e[i].to, ww = e[i].w;
63
64                 addedge(cnt, yy, 0);
65
66                 d[cnt].push_back(tmp);
67
68                 tmp += ww;
69
70                 if (e[i ^ 1].to == y)
71                     break;
72             }
73
74             addedge(y, cnt, 0);
75
76             sum[cnt] = tmp;
77         }
78     }
79 }
80
81 void dp(int x) {
82
83     for (auto o : v[x]) {
84         int y = o.first, w = o.second;
85
86         dp(y);
87     }
88
89     if (x <= n) {
90         for (auto o : v[x]) {
91             int y = o.first, w = o.second;
92
93             f[x] += 2 * w + f[y];
94         }
95
96         g[x] = f[x];
97
98         for (auto o : v[x]) {
99             int y = o.first, w = o.second;
100
101             g[x] = min(g[x], f[x] - f[y] - 2 * w + g[y] + w);
102         }
103     }
104     else {
105         f[x] = sum[x];
106         for (auto o : v[x]) {
107             int y = o.first;
108
109             f[x] += f[y];
110         }
111
112         g[x] = f[x];
113
114         for (int i = 0; i < (int)v[x].size(); i++) {
115             int y = v[x][i].first;
116
117             g[x] = min(g[x], f[x] - f[y] + g[y] + min(d[x]
118                 ↳ [i], sum[x] - d[x][i]));
119         }
120     }
121 }
122
123 signed main() {
124
125     memset(last, -1, sizeof(last));
126
127     ios::sync_with_stdio(false);
128
129     cin >> n >> m;
130
131     cnt = n;
132
133     while (m--) {
134         int x, y, w;
135         cin >> x >> y >> w;
136
137         e[cnt].to = y;
138         e[cnt].w = w;
139         e[cnt].prev = last[x];
140         last[x] = cnt++;
141
142         e[cnt].to = x;
143         e[cnt].w = w;
144         e[cnt].prev = last[y];
145         last[y] = cnt++;
146     }
147
148     dfs(1);

```

```

147     dp(1);
148
149     cout << g[1] << endl;
150
151     return 0;
152 }

```

## 1.4 二分图

### 1.4.1 KM二分图最大权匹配

```

1 // KM Weighted Bio-Graph Matching KM二分图最大权匹配
2 // By AntiLeaf
3 // O(n^3)
4
5 const long long INF = 0x3f3f3f3f3f3f3f3f;
6
7 long long w[maxn][maxn], lx[maxn], ly[maxn], slack[maxn];
8 // 边权 顶标 slack
9 // 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
10
11 bool visx[maxn], visy[maxn];
12
13 int boy[maxn], girl[maxn], p[maxn], q[maxn], head, tail; // p
  ↪ : pre
14
15 int n, m, N, e;
16
17 // 增广
18 bool check(int y) {
19     visy[y] = true;
20
21     if (boy[y]) {
22         visx[boy[y]] = true;
23         q[tail++] = boy[y];
24         return false;
25     }
26
27     while (y) {
28         boy[y] = p[y];
29         swap(y, girl[p[y]]);
30     }
31
32     return true;
33 }
34
35 // bfs每个点
36 void bfs(int x) {
37     memset(q, 0, sizeof(q));
38     head = tail = 0;
39
40     q[tail++] = x;
41     visx[x] = true;
42
43     while (true) {
44         while (head != tail) {
45             int x = q[head++];
46
47             for (int y = 1; y <= N; y++)
48                 if (!visy[y]) {
49                     long long d = lx[x] + ly[y] - w[x][y];
50
51                     if (d < slack[y]) {
52                         p[y] = x;

```

```

53         slack[y] = d;
54
55         if (!slack[y] && check(y))
56             return;
57     }
58 }
59
60
61 long long d = INF;
62 for (int i = 1; i <= N; i++)
63     if (!visy[i])
64         d = min(d, slack[i]);
65
66 for (int i = 1; i <= N; i++) {
67     if (visx[i])
68         lx[i] -= d;
69
70     if (visy[i])
71         ly[i] += d;
72     else
73         slack[i] -= d;
74 }
75
76 for (int i = 1; i <= N; i++)
77     if (!visy[i] && !slack[i] && check(i))
78         return;
79 }
80 }
81
82 // 主过程
83 long long KM() {
84     for (int i = 1; i <= N; i++) {
85         // lx[i] = 0;
86         ly[i] = -INF;
87         // boy[i] = girl[i] = -1;
88
89         for (int j = 1; j <= N; j++)
90             ly[i] = max(ly[i], w[j][i]);
91     }
92
93     for (int i = 1; i <= N; i++) {
94         memset(slack, 0x3f, sizeof(slack));
95         memset(visx, 0, sizeof(visx));
96         memset(visy, 0, sizeof(visy));
97         bfs(i);
98     }
99
100     long long ans = 0;
101     for (int i = 1; i <= N; i++)
102         ans += w[i][girl[i]];
103     return ans;
104 }
105
106 // 为了方便贴上主函数
107 int main() {
108
109     scanf("%d%d", &n, &m, &e);
110     N = max(n, m);
111
112     while (e--) {
113         int x, y, c;
114         scanf("%d%d", &x, &y, &c);
115         w[x][y] = max(c, 0);

```

```

116     }
117
118     printf("%lld\n", KM());
119
120     for (int i = 1; i <= n; i++) {
121         if (i > 1)
122             printf(" ");
123         printf("%d", w[i][girl[i]] > 0 ? girl[i] : 0);
124     }
125     printf("\n");
126
127     return 0;
128 }

```

## 1.5 一般图匹配

### 1.5.1 高斯消元

```

1 // Graph Matching Based on Linear Algebra 基于线性代数的一般
  ↳ 图匹配  $O(n^3)$ 
2 // By ysf
3 // 通过题目 UOJ#79 一般图最大匹配
4
5 // 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
6 // 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
7 // 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
8 // 需要输出方案时才需要再写后面那些乱七八糟的东西
9
10
11 // 复杂度和常数所限 1s 之内 500 已经是这个算法的极限了
12 const int maxn = 505, p = 1000000007; // p 可以是任意  $10^9$  以内的
  ↳ 质数
13
14 // 全局数组和变量定义
15 int A[maxn][maxn], B[maxn][maxn], t[maxn][maxn], id[maxn],
  ↳ a[maxn];
16 bool row[maxn] = {false}, col[maxn] = {false};
17 int n, m, girl[maxn]; // girl 是匹配点 用来输出方案
18
19 // 为了方便使用 贴上主函数
20 // 需要调用高斯消元和 eliminate
21 int main() {
22     srand(19260817); // 膜蛤专用随机种子 换一个也无所谓
23
24     scanf("%d%d", &n, &m); // 点数和边数
25     while (m--) {
26         int x, y;
27         scanf("%d%d", &x, &y);
28         A[x][y] = rand() % p;
29         A[y][x] = -A[x][y]; // Tutte 矩阵是反对称矩阵
30     }
31
32     for (int i = 1; i <= n; i++)
33         id[i] = i; // 输出方案用的 因为高斯消元的时候会交换
  ↳ 列
34     memcpy(t, A, sizeof(t));
35     Gauss(A, NULL, n);
36
37     m = n;
38     n = 0; // 这里变量复用纯属个人习惯.....
39
40     for (int i = 1; i <= m; i++)

```

```

41         if (A[id[i]][id[i]])
42             a[++n] = i; // 找出一个极大满秩子矩阵
43
44     for (int i = 1; i <= n; i++)
45         for (int j = 1; j <= n; j++)
46             A[i][j] = t[a[i]][a[j]];
47
48     Gauss(A, B, n);
49
50     for (int i = 1; i <= n; i++)
51         if (!girl[a[i]])
52             for (int j = i + 1; j <= n; j++)
53                 if (!girl[a[j]] && t[a[i]][a[j]] && B[j][i])
54                     ↳ {
55                         ↳ // 注意上面那句 if 的写法 现在 t 是邻接矩阵
56                         ↳ 的备份
57                         ↳ // 逆矩阵 j 行 i 列不为 0 当且仅当这条边可行
58                         girl[a[i]] = a[j];
59                         girl[a[j]] = a[i];
60                         eliminate(i, j);
61                         eliminate(j, i);
62                         break;
63                     }
64
65     printf("%d\n", n >> 1);
66     for (int i = 1; i <= m; i++)
67         printf("%d ", girl[i]);
68
69     return 0;
70 }
71
72 // 高斯消元  $O(n^3)$ 
73 // 在传入 B 时表示计算逆矩阵 传入 NULL 则只需计算矩阵的秩
74 void Gauss(int A[][maxn], int B[][maxn], int n) {
75     if (B) {
76         memset(B, 0, sizeof(t));
77         for (int i = 1; i <= n; i++)
78             B[i][i] = 1;
79     }
80
81     for (int i = 1; i <= n; i++) {
82         if (!A[i][i]) {
83             for (int j = i + 1; j <= n; j++)
84                 if (A[j][i]) {
85                     swap(id[i], id[j]);
86                     for (int k = i; k <= n; k++)
87                         swap(A[i][k], A[j][k]);
88
89                     if (B)
90                         for (int k = 1; k <= n; k++)
91                             swap(B[i][k], B[j][k]);
92                     break;
93                 }
94
95         if (!A[i][i])
96             continue;
97     }
98
99     int inv = qpow(A[i][i], p - 2);
100
101     for (int j = 1; j <= n; j++)
102         if (i != j && A[j][i]) {
103             int t = (long long)A[j][i] * inv % p;

```



```

102     for (int k = i; k <= n; k++)
103     {
104         if (A[i][k])
105             A[j][k] = (A[j][k] - (long long)t *
106                 A[i][k]) % p;
107     }
108     if (B)
109     {
110         for (int k = 1; k <= n; k++)
111             if (B[i][k])
112                 B[j][k] = (B[j][k] - (long long)t
113                     A[i][k]) % p;
114     }
115     if (B)
116     {
117         for (int i = 1; i <= n; i++) {
118             int inv = qpow(A[i][i], p - 2);
119             for (int j = 1; j <= n; j++)
120                 if (B[i][j])
121                     B[i][j] = (long long)B[i][j] * inv % p;
122         }
123     }
124 // 消去一行一列  $O(n^2)$ 
125 void eliminate(int r, int c) {
126     row[r] = col[c] = true; // 已经被消掉
127     int inv = qpow(B[r][c], p - 2);
128     for (int i = 1; i <= n; i++)
129         if (!row[i] && B[i][c]) {
130             int t = (long long)B[i][c] * inv % p;
131             for (int j = 1; j <= n; j++)
132                 if (!col[j] && B[r][j])
133                     B[i][j] = (B[i][j] - (long long)t * B[r][j]
134                         A[j][c]) % p;
135         }
136 }
137 }
138

```

### 1.5.2 带花树

```

1 // Blossom 带花树  $O(nm)$ 
2 // By ysf
3 // 通过题目 P4017 一般图最大匹配
4
5 // 带花树通常比高斯消元快很多但在只要求最大匹配大小的时
6 // 候并没有高斯消元好写
7 // 当然输出方案要方便很多
8
9 // 全局数组与变量定义
10 vector<int> G[maxn];
11 int girl[maxn], f[maxn], t[maxn], p[maxn], vis[maxn], tim,
12     q[maxn], head, tail;
13 int n, m;
14
15 // 封装好的主过程  $O(nm)$ 
16 int blossom() {
17     int ans = 0;
18     for (int i = 1; i <= n; i++)
19         if (!girl[i])
20             ans += bfs(i);
21

```

```

21     return ans;
22 }
23
24 // bfs找增广路  $O(m)$ 
25 bool bfs(int s) {
26     memset(t, 0, sizeof(t));
27     memset(p, 0, sizeof(p));
28     for (int i = 1; i <= n; i++)
29         f[i] = i; // 并查集
30     head = tail = 0;
31     q[tail++] = s;
32     t[s] = 1;
33     while (head != tail) {
34         int x = q[head++];
35         for (int y : G[x]) {
36             if (findroot(x) == findroot(y) || t[y] == 2)
37                 continue;
38             if (!t[y]) {
39                 t[y] = 2;
40                 p[y] = x;
41             }
42             if (!girl[y]) {
43                 for (int u = y; t[u] < 2; u = p[u]) {
44                     t[u] = girl[p[u]];
45                     girl[p[u]] = u;
46                     girl[u] = p[u];
47                 }
48                 return true;
49             }
50             t[girl[y]] = 1;
51             q[tail++] = girl[y];
52         }
53     }
54     else if (t[y] == 1) {
55         int z = LCA(x, y);
56         shrink(x, y, z);
57         shrink(y, x, z);
58     }
59 }
60
61 return false;
62 }
63
64 // 缩奇环  $O(n)$ 
65 void shrink(int x, int y, int z) {
66     while (findroot(x) != z) {
67         p[x] = y;
68         y = girl[x];
69     }
70     if (t[y] == 2) {
71         t[y] = 1;
72         q[tail++] = y;
73     }
74     if (findroot(x) == x)
75         f[x] = z;
76     if (findroot(y) == y)
77

```

```

84     f[y] = z;
85
86     x = p[y];
87 }
88 }
89
90 //暴力找LCA O(n)
91 int LCA(int x, int y) {
92     tim++;
93     while (true) {
94         if (x) {
95             x = findroot(x);
96
97             if (vis[x] == tim)
98                 return x;
99             else {
100                 vis[x] = tim;
101                 x = p[girl[x]];
102             }
103         }
104         swap(x, y);
105     }
106 }
107
108 //并查集的查找 O(1)
109 int findroot(int x) {
110     return x == f[x] ? x : (f[x] = findroot(f[x]));
111 }

```

## 1.6 最大流

### 1.6.1 Dinic

```

1 // 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
2
3 struct edge{int to, cap, prev;}e[maxe<<1];
4
5
6 int last[maxn], len=0, d[maxn], cur[maxn], q[maxn];
7 memset(last, -1, sizeof(last));
8
9
10 void addedge(int x, int y, int z){
11     AddEdge(x, y, z);
12     AddEdge(y, x, 0);
13 }
14
15
16 void AddEdge(int x, int y, int z){
17     e[len].to=y;
18     e[len].cap=z;
19     e[len].prev=last[x];
20     last[x]=len++;
21 }
22
23
24 int Dinic(){
25     int flow=0;
26     while(bfs(), d[t]!=-1){
27         memcpy(cur, last, sizeof(int)*(t+5));
28         flow+=dfs(s, (~0u)>>1);
29     }
30     return flow;
31 }

```

```

32
33
34 void bfs(){
35     int head=0, tail=0;
36     memset(d, -1, sizeof(int)*(t+5));
37     q[tail++]=s;
38     d[s]=0;
39     while(head!=tail){
40         int x=q[head++];
41         for(int i=last[x]; i!=-1; i=e[i].prev)
42             if(e[i].cap>0&&d[e[i].to]==-1){
43                 d[e[i].to]=d[x]+1;
44                 q[tail++]=e[i].to;
45             }
46     }
47 }
48
49
50 int dfs(int x, int a){
51     if(x==t||!a)return a;
52     int flow=0, f;
53     for(int &i=cur[x]; i!=-1; i=e[i].prev)
54         ↳ if(e[i].cap>0&&d[e[i].to]==d[x]+1&&(f=dfs(e[i].to, min(e[
55         ↳ {
56             e[i].cap-=f;
57             e[i^1].cap+=f;
58             flow+=f;
59             a-=f;
60             if(!a)break;
61         }
62     }
63     return flow;
64 }

```

### 1.6.2 ISAP

```

1 // 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
2
3
4 // 边的定义
5 // 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
6 struct edge{
7     int to, cap, prev;
8 } e[maxe * 2];
9
10
11 // 全局变量和数组定义
12 int last[maxn], cnte = 0, d[maxn], p[maxn], c[maxn],
  ↳ cur[maxn], q[maxn];
13 int n, m, s, t; // s, t一定要开成全局变量
14
15
16 // 重要!!!
17 // main函数最前面一定要加上如下初始化
18 memset(last, -1, sizeof(last));
19
20
21 // 加边函数 O(1)
22 // 包装了加反向边的过程,方便调用
23 // 需要调用AddEdge
24 void addedge(int x, int y, int z) {
25     AddEdge(x, y, z);

```

```

26     AddEdge(y, x, 0);
27 }
28
29 // 真·加边函数  $O(1)$ 
30 void AddEdge(int x, int y, int z){
31     e[cnte].to = y;
32     e[cnte].cap = z;
33     e[cnte].prev = last[x];
34     last[x] = cnte++;
35 }
36
37
38 // 主过程  $O(n^2 m)$ 
39 // 返回最大流的流量
40 // 需要调用 bfs, augment
41 // 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
42 // 非递归
43 int ISAP() {
44     bfs();
45
46     memcpy(cur, last, sizeof(cur));
47
48     int x = s, flow = 0;
49
50     while (d[s] < n) {
51         if (x == t) { // 如果走到了  $t$  就增广一次, 并返回  $s$  重新找增广路
52             flow += augment();
53             x = s;
54         }
55
56         bool ok = false;
57         for (int &i = cur[x]; ~i; i = e[i].prev)
58             if (e[i].cap && d[x] == d[e[i].to] + 1) {
59                 p[e[i].to] = i;
60                 x = e[i].to;
61             }
62
63         ok = true;
64         break;
65     }
66
67     if (!ok) { // 修改距离标号
68         int tmp = n - 1;
69         for (int i = last[x]; ~i; i = e[i].prev)
70             if (e[i].cap)
71                 tmp = min(tmp, d[e[i].to] + 1);
72
73         if (!--c[d[x]])
74             break; // gap 优化, 一定要加上
75
76         c[d[x]] = tmp++;
77         cur[x] = last[x];
78
79         if (x != s)
80             x = e[p[x] ^ 1].to;
81     }
82 }
83 return flow;
84 }
85
86 // bfs 函数  $O(n+m)$ 
87 // 预处理到  $t$  的距离标号

```

```

88 // 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
89 void bfs(){
90     memset(d, -1, sizeof(d));
91
92     int head = 0, tail = 0;
93     d[t] = 0;
94     q[tail++] = t;
95
96     while (head != tail) {
97         int x = q[head++];
98         c[d[x]]++;
99
100         for (int i = last[x]; ~i; i = e[i].prev)
101             if (e[i] ^ 1).cap && d[e[i].to] == -1) {
102                 d[e[i].to] = d[x] + 1;
103                 q[tail++] = e[i].to;
104             }
105     }
106 }
107
108 // augment 函数  $O(n)$ 
109 // 沿增广路增广一次, 返回增广的流量
110 int augment() {
111     int a = (~0u) >> 1; // INT_MAX
112
113     for (int x = t; x != s; x = e[p[x] ^ 1].to)
114         a = min(a, e[p[x]].cap);
115
116     for (int x = t; x != s; x = e[p[x] ^ 1].to){
117         e[p[x]].cap -= a;
118         e[p[x] ^ 1].cap += a;
119     }
120
121     return a;
122 }

```

### 1.6.3 HLPP 最高标号预流推进

```

1 #include<cstdio>
2 #include<cstring>
3 #include<algorithm>
4 #include<queue>
5 using std::min;
6 using std::vector;
7 using std::queue;
8 using std::priority_queue;
9 const int N=2e4+5,M=2e5+5,inf=0x3f3f3f3f;
10 int n,s,t,tot;
11 int v[M<<1],w[M<<1],first[N],next[M<<1];
12 int h[N],e[N],gap[N<<1],inq[N]; // 节点高度是可以到达  $2n-1$  的
13 struct cmp
14 {
15     inline bool operator()(int a,int b) const
16     {
17         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
18     }
19 };
20 queue<int> Q;
21 priority_queue<int,vector<int>,cmp> pQ;
22 inline void add_edge(int from,int to,int flow)
23 {
24     tot+=2;
25     v[tot+1]=from;v[tot]=to;w[tot]=flow;w[tot+1]=0;

```

```

26     next[tot]=first[from];first[from]=tot;
27     next[tot+1]=first[to];first[to]=tot+1;
28     return;
29 }
30 inline bool bfs()
31 {
32     int now;
33     register int go;
34     memset(h+1,0x3f,sizeof(int)*n);
35     h[t]=0;Q.push(t);
36     while(!Q.empty())
37     {
38         now=Q.front();Q.pop();
39         for(go=first[now];go;go=next[go])
40             if(w[go^1]&&h[v[go]]>h[now]+1)
41                 h[v[go]]=h[now]+1,Q.push(v[go]);
42     }
43     return h[s]!=inf;
44 }
45 inline void push(int now)//推送
46 {
47     int d;
48     register int go;
49     for(go=first[now];go;go=next[go])
50         if(w[go]&&h[v[go]]+1==h[now])
51         {
52             d=min(e[now],w[go]);
53             w[go]-=d;w[go^1]+=d;e[now]-=d;e[v[go]]+=d;
54             if(v[go]!=s&&v[go]!=t&&!inq[v[go]])
55                 pQ.push(v[go]),inq[v[go]]=1;
56             if(!e[now])//已经推送完毕可以直接退出
57                 break;
58         }
59     return;
60 }
61 inline void relabel(int now)//重贴标签
62 {
63     register int go;
64     h[now]=inf;
65     for(go=first[now];go;go=next[go])
66         if(w[go]&&h[v[go]]+1<h[now])
67             h[now]=h[v[go]]+1;
68     return;
69 }
70 inline int hlpp()
71 {
72     int now,d;
73     register int i,go;
74     if(!bfs())//s和t不连通
75         return 0;
76     h[s]=n;
77     memset(gap,0,sizeof(int)*(n<<1));
78     for(i=1;i<=n;i++)
79         if(h[i]<inf)
80             ++gap[h[i]];
81     for(go=first[s];go;go=next[go])
82         if(d=w[go])
83         {
84             w[go]-=d;w[go^1]+=d;e[s]-=d;e[v[go]]+=d;
85             if(v[go]!=s&&v[go]!=t&&!inq[v[go]])
86                 pQ.push(v[go]),inq[v[go]]=1;
87         }
88     while(!pQ.empty())
89     {

```

```

90         inq[now=pQ.top()]=0;pQ.pop();push(now);
91         if(e[now])
92         {
93             if(--gap[h[now]])//gap优化因为当前节点是最高的
94                 ↪ 所以修改的节点一定不在优先队列中不必担心修
95                 ↪ 改对优先队列会造成影响
96                 for(i=1;i<=n;i++)
97                     if(i!=s&&i!=t&&h[i]>h[now]&&h[i]<n+1)
98                         h[i]=n+1;
99                     relabel(now);++gap[h[now]];
100                     pQ.push(now);inq[now]=1;
101         }
102     }
103     return e[t];
104 }
105 int m;
106 signed main()
107 {
108     int u,v,w;
109     scanf("%d%d%d%d",&n,&m,&s,&t);
110     while(m--)
111     {
112         scanf("%d%d%d",&u,&v,&w);
113         add_edge(u,v,w);
114     }
115     printf("%d\n",hlpp());
116     return 0;
117 }

```

## 2. 字符串

### 2.1 AC自动机

```

1 // Aho-Corasick Automata AC自动机
2 // By AntiLeaf
3 // 通过题目@bzoj3881 Divljak
4
5
6 // 全局变量与数组定义
7 int ch[maxm][26] = {{0}}, f[maxm][26] = {{0}}, q[maxm] = {0},
8     ↪ sum[maxm] = {0}, cnt = 0;
9
10 // 在字典树中插入一个字符串 O(n)
11 int insert(const char *c) {
12     int x = 0;
13     while (*c) {
14         if (!ch[x][*c - 'a'])
15             ch[x][*c - 'a'] = ++cnt;
16         x = ch[x][*c++ - 'a'];
17     }
18     return x;
19 }
20
21 // 建AC自动机 O(n*sigma)
22 void getfail() {
23     int x, head = 0, tail = 0;
24
25     for (int c = 0; c < 26; c++)
26         if (ch[0][c])
27             q[tail++] = ch[0][c]; // 把根节点的儿子加入队列
28
29

```

```

30 while (head != tail) {
31     x = q[head++];
32
33     G[f[x][0]].push_back(x);
34     fill(f[x] + 1, f[x] + 26, cnt + 1);
35
36     for (int c = 0; c < 26; c++) {
37         if (ch[x][c]) {
38             int y = f[x][0];
39
40             while (y&&!ch[y][c])
41                 y=f[y][0];
42
43             f[ch[x][c]][0] = ch[y][c];
44             q[tail++] = ch[x][c];
45         }
46         else
47             ch[x][c] = ch[f[x][0]][c];
48     }
49 }
50 fill(f[0], f[0] + 26, cnt + 1);
51 }

```

## 2.2 后缀数组

### 2.2.1 SAMSA

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int maxn=100005;
4 void expand(int);
5 void dfs(int);
6 int
7     ↳ root,last,cnt=0,val[maxn<<1]={0},par[maxn<<1]={0},go[maxn<<1][26];
8     ↳ [26]={0};
9 bool vis[maxn<<1]={0};
10 char s[maxn];
11 int n,id[maxn<<1]={0},ch[maxn<<1][26];
12     ↳ [26]={0},height[maxn],tim=0;
13 int main(){
14     root=last=++cnt;
15     scanf("%s",s+1);
16     n=strlen(s+1);
17     for(int i=n;i--){
18         expand(s[i]-'a');
19         id[last]=i;
20     }
21     vis[1]=true;
22     for(int i=1;i<=cnt;i++)if(id[i])for(int
23         ↳ x=i,pos=n;x&&!vis[x];x=par[x]){
24         vis[x]=true;
25         pos-=val[x]-val[par[x]];
26         ch[par[x]][s[pos+1]-'a']=x;
27     }
28     dfs(root);
29     printf("\n");
30     for(int i=1;i<=n;i++)printf("%d ",height[i]);
31     return 0;
32 }
33 void expand(int c){
34     int p=last,np=++cnt;
35     val[np]=val[p]+1;
36     while(p&&!go[p][c]){
37         go[p][c]=np;
38         p=par[p];
39     }
40 }

```

```

35 }
36 if(!p)par[np]=root;
37 else{
38     int q=go[p][c];
39     if(val[q]==val[p]+1)par[np]=q;
40     else{
41         int nq=++cnt;
42         val[nq]=val[p]+1;
43         memcpy(go[nq],go[q],sizeof(go[q]));
44         par[nq]=par[q];
45         par[np]=par[q]=nq;
46         while(p&&go[p][c]==q){
47             go[p][c]=nq;
48             p=par[p];
49         }
50     }
51 }
52 last=np;
53 }
54 void dfs(int x){
55     if(id[x]){
56         printf("%d ",id[x]);
57         height[tim++]=val[last];
58         last=x;
59     }
60     for(int c=0;c<26;c++)if(ch[x][c])dfs(ch[x][c]);
61     last=par[x];
62 }

```

## 2.3 后缀自动机

```

1 //Suffix Automaton 后缀自动机 O(n)
2 //By ysf
3 //通过题目 Bzoj3473 字符串
4
5 //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
6     ↳ 希表替换
7 //注意 结点数要开成串长的两倍
8
9 //全局变量与数组定义
10 int last,val[maxn],par[maxn],go[maxn][26],cnt;
11 int c[maxn],q[maxn]; //用来桶排序
12
13 //在主函数开头加上这句初始化
14 last=cnt=1;
15
16 //以下是按val进行桶排序的代码
17 for(int i=1;i<=cnt;i++)c[val[i]+1]++;
18 for(int i=1;i<=n;i++)c[i]+=c[i-1]; //这里n是串长
19 for(int i=1;i<=cnt;i++)q[++c[val[i]]]=i;
20
21 //加入一个字符 均摊O(1)
22 void extend(int c){
23     int p=last,np=++cnt;
24     val[np]=val[p]+1;
25     while(p&&!go[p][c]){
26         go[p][c]=np;
27         p=par[p];
28     }
29     if(!p)par[np]=1;
30     else{
31         int q=go[p][c];
32         if(val[q]==val[p]+1)par[np]=q;
33         else{

```

```

33     int nq=++cnt;
34     val[nq]=val[p]+1;
35     memcpy(go[nq],go[q],sizeof(go[q]));
36     par[nq]=par[q];
37     par[np]=par[q]=nq;
38     while(p&&go[p][c]==q){
39         go[p][c]=nq;
40         p=par[p];
41     }
42 }
43 }
44 last=np;
45 }

```

## 2.4 回文树

```

1 //Palindromic Tree/EERTREE 回文树 O(n)
2 //By ysf
3 //通过题目 BAPI02014 回文串
4
5 //定理 一个字符串本质不同的回文子串个数是O(n)的
6 //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
7
8 //全局数组定义
9 int val[maxn], par[maxn], go[maxn][26], last, cnt;
10 char s[maxn];
11
12 //重要 在主函数最前面一定要加上以下初始化
13 par[0]=cnt=1;
14 val[1]=-1;
15
16 //extend函数 均摊O(1)
17 //向后扩展一个字符
18 //传入对应下标
19 void extend(int n){
20     int p=last, c=s[n]-'a';
21     while(s[n-val[p]-1]!=s[n]) p=par[p];
22     if(!go[p][c]){
23         int q=++cnt, now=p;
24         val[q]=val[p]+2;
25         do p=par[p]; while(s[n-val[p]-1]!=s[n]);
26         par[q]=go[p][c];
27         last=go[now][c]=q;
28     }
29     else last=go[p][c];
30     a[last]++;
31 }

```

### 2.4.1 广义回文树

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 constexpr int maxn = 1000005, mod = 1000000007;
6
7 int val[maxn], par[maxn], go[maxn][26], fail[maxn][26],
8     pam_last[maxn], pam_cnt;
9 int weight[maxn], pow_26[maxn];
10
11 int trie[maxn][26], trie_cnt, d[maxn], mxd[maxn], son[maxn],
12     top[maxn], len[maxn], sum[maxn];
13 char chr[maxn];
14 int f[25][maxn], log_tbl[maxn];

```

```

13 vector<int> v[maxn];
14
15 vector<int> queries[maxn];
16
17 char str[maxn];
18 int n, m, ans[maxn];
19
20 int add(int x, int c) {
21     if (!trie[x][c]) {
22         trie[x][c] = ++trie_cnt;
23         f[0][trie[x][c]] = x;
24         chr[trie[x][c]] = c + 'a';
25     }
26
27     return trie[x][c];
28 }
29
30 int del(int x) {
31     return f[0][x];
32 }
33
34 void dfs1(int x) {
35     mxd[x] = d[x] = d[f[0][x]] + 1;
36
37     for (int i = 0; i < 26; i++)
38         if (trie[x][i]) {
39             int y = trie[x][i];
40
41             dfs1(y);
42
43             mxd[x] = max(mxd[x], mxd[y]);
44             if (mxd[y] > mxd[son[x]])
45                 son[x] = y;
46         }
47 }
48
49 void dfs2(int x) {
50     if (x == son[f[0][x]])
51         top[x] = top[f[0][x]];
52     else
53         top[x] = x;
54
55     for (int i = 0; i < 26; i++)
56         if (trie[x][i]) {
57             int y = trie[x][i];
58             dfs2(y);
59         }
60
61     if (top[x] == x) {
62         int u = x;
63         while (top[son[u]] == x)
64             u = son[u];
65
66         len[x] = d[u] - d[x];
67
68         for (int i = 0; i < len[x]; i++) {
69             v[x].push_back(u);
70             u = f[0][u];
71         }
72
73         u = x;
74         for (int i = 0; i < len[x]; i++) { // 梯子剖分 要延
75             v[x].push_back(u);

```

```

76         u = f[0][u];
77     }
78 }
79 }
80
81 int get_anc(int x, int k) {
82     if (!k)
83         return x;
84     if (k > d[x])
85         return 0;
86
87     x = f[log_tbl[k]][x];
88     k ^= 1 << log_tbl[k];
89
90     return v[top[x]][d[top[x]] + len[top[x]] - d[x] + k];
91 }
92
93 char get_char(int x, int k) { // 查询x前面k个的字符是哪个
94     return chr[get_anc(x, k)];
95 }
96
97 int getfail(int x, int p) {
98     if (get_char(x, val[p] + 1) == chr[x])
99         return p;
100     return fail[p][chr[x] - 'a'];
101 }
102
103 int extend(int x) {
104
105     int p = pam_last[f[0][x]], c = chr[x] - 'a';
106
107     p = getfail(x, p);
108
109     int new_last;
110
111     if (!go[p][c]) {
112         int q = ++pam_cnt, now = p;
113         val[q] = val[p] + 2;
114
115         p = getfail(x, par[p]);
116
117         par[q] = go[p][c];
118         new_last = go[now][c] = q;
119
120         for (int i = 0; i < 26; i++)
121             fail[q][i] = fail[par[q]][i];
122
123         if (get_char(x, val[par[q]]) >= 'a')
124             fail[q][get_char(x, val[par[q]]) - 'a'] = par[q];
125
126         if (val[q] <= n)
127             weight[q] = (weight[par[q]] + (long long)(n -
128                 ↪ val[q] + 1) * pow_26[n - val[q]]) % mod;
129         else
130             weight[q] = weight[par[q]];
131     }
132     else
133         new_last = go[p][c];
134
135     pam_last[x] = new_last;
136
137     return weight[pam_last[x]];
138 }

```

```

139 void bfs() {
140
141     queue<int> q;
142
143     q.push(1);
144
145     while (!q.empty()) {
146         int x = q.front();
147         q.pop();
148
149         sum[x] = sum[f[0][x]];
150         if (x > 1)
151             sum[x] = (sum[x] + extend(x)) % mod;
152
153         for (int i : queries[x])
154             ans[i] = sum[x];
155
156         for (int i = 0; i < 26; i++)
157             if (trie[x][i])
158                 q.push(trie[x][i]);
159     }
160 }
161
162 int main() {
163
164     pow_26[0] = 1;
165     log_tbl[0] = -1;
166
167     for (int i = 1; i <= 1000000; i++) {
168         pow_26[i] = 26ll * pow_26[i - 1] % mod;
169         log_tbl[i] = log_tbl[i / 2] + 1;
170     }
171
172     int T;
173     scanf("%d", &T);
174
175     while (T--) {
176         scanf("%d%d%s", &n, &m, str);
177
178         trie_cnt = 1;
179         chr[1] = '#';
180
181         int last = 1;
182         for (char *c = str; *c; c++)
183             last = add(last, *c - 'a');
184
185         queries[last].push_back(0);
186
187         for (int i = 1; i <= m; i++) {
188             int op;
189             scanf("%d", &op);
190
191             if (op == 1) {
192                 char c;
193                 scanf("%c", &c);
194
195                 last = add(last, c - 'a');
196             }
197             else
198                 last = del(last);
199
200             queries[last].push_back(i);
201         }
202     }

```

```

203
204     dfs1(1);
205     dfs2(1);
206
207     for (int j = 1; j <= log_tbl[trie_cnt]; j++)
208         for (int i = 1; i <= trie_cnt; i++)
209             f[j][i] = f[j - 1][f[j - 1][i]];
210
211     par[0] = pam_cnt = 1;
212
213
214     for (int i = 0; i < 26; i++)
215         fail[0][i] = fail[1][i] = 1;
216
217     val[1] = -1;
218     pam_last[1] = 1;
219
220     bfs();
221
222     for (int i = 0; i <= m; i++)
223         printf("%d\n", ans[i]);
224
225     for (int j = 0; j <= log_tbl[trie_cnt]; j++)
226         memset(f[j], 0, sizeof(f[j]));
227
228     for (int i = 1; i <= trie_cnt; i++) {
229         chr[i] = 0;
230         d[i] = mxd[i] = son[i] = top[i] = len[i] =
231             ↪ pam_last[i] = sum[i] = 0;
232         v[i].clear();
233         queries[i].clear();
234
235         memset(trie[i], 0, sizeof(trie[i]));
236     }
237     trie_cnt = 0;
238
239     for (int i = 0; i <= pam_cnt; i++) {
240         val[i] = par[i] = weight[i];
241
242         memset(go[i], 0, sizeof(go[i]));
243         memset(fail[i], 0, sizeof(fail[i]));
244     }
245     pam_cnt = 0;
246 }
247
248     return 0;
249 }

```

## 2.5 Manacher马拉车

```

1 //Manacher O(n)
2 //By ysf
3 //通过题目51Nod1089 最长回文子串V2
4
5 //n为串长回文半径输出到p数组中
6 //数组要开串长的两倍
7 void manacher(const char *t, int n) {
8     static char s[maxn * 2];
9
10    for (int i = n; i; i--)
11        s[i * 2] = t[i];
12    for (int i = 0; i <= n; i++)
13        s[i * 2 + 1] = '#';

```

```

14
15    s[0] = '$';
16    s[(n + 1) * 2] = '\0';
17    n = n * 2 + 1;
18
19    int mx = 0, j = 0;
20
21    for (int i = 1; i <= n; i++) {
22        p[i] = (mx > i ? min(p[j * 2 - i], mx - i) : 1);
23        while (s[i - p[i]] == s[i + p[i]])
24            p[i]++;
25
26        if (i + p[i] > mx){
27            mx = i + p[i];
28            j = i;
29        }
30    }
31 }

```

## 2.6 KMP

### 2.6.1 ex-KMP

```

1 //Extended KMP 扩展KMP
2 //By AntiLeaf
3 //通过题目小作业OJ 4182
4
5 //全局变量与数组定义
6 char s[maxn], t[maxn];
7 int n, m, a[maxn];
8
9 //主过程 O(n + m)
10 //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
11 //0-baseds的长度是mt的长度是n
12 void exKMP(const char *s, const char *t, int *a) {
13     static int nx[maxn];
14
15     memset(nx, 0, sizeof(nx));
16
17     int j = 0;
18     while (j + 1 < m && s[j] == s[j + 1])
19         j++;
20     nx[1] = j;
21
22     for (int i = 2, k = 1; i < m; i++) {
23         int pos = k + nx[k], len = nx[i - k];
24
25         if (i + len < pos)
26             nx[i] = len;
27         else {
28             j = max(pos - i, 0);
29             while (i + j < m && s[j] == s[i + j])
30                 j++;
31
32             nx[i] = j;
33             k = i;
34         }
35     }
36
37     j = 0;
38     while (j < n && j < m && s[j] == t[j])
39         j++;
40     a[0] = j;
41
42     for (int i = 1, k = 0; i < n; i++) {

```



```

43     int pos = k + a[k], len = nx[i - k];
44     if (i + len < pos)
45         a[i] = len;
46     else {
47         j = max(pos - i, 0);
48         while(j < m && i + j < n && s[j] == t[i + j])
49             j++;
50
51         a[i] = j;
52         k = i;
53     }
54 }
55 }

```

```

35     for(int k=2;k<=n;k<=1)
36         for(int i=0;i<n;i+=k)
37             for(int j=0;j<(k>>1);j++){
38                 Complex a=A[i+j],b=(tp>0?w:w_inv)
39                     ↳ [n/k*j]*A[i+j+(k>>1)];
40                 A[i+j]=a+b;
41                 A[i+j+(k>>1)]=a-b;
42             }
43     if(tp<0)for(int i=0;i<n;i++)A[i].a/=n;
44 }

```

## 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

```

1 //Fast Fourier Transform 快速傅里叶变换  $O(n\log n)$ 
2 //By ysf
3 //通过题目 BCOGS2294 释迦作为拆系数FFT的组成部分
4 //使用时一定要注意double的精度是否足够极限大概是 $10^{14}$ 
5
6 const double pi=acos((double)-1.0);
7
8 //手写复数类
9 //支持加减乘三种运算
10 //+=运算符如果用的不多可以不重载
11 struct Complex{
12     double a,b;//由于long double精度和double几乎相同通常没
13     ↳ 有必要用long double
14     Complex(double a=0.0,double b=0.0):a(a),b(b){}
15     Complex operator+(const Complex &x)const{return
16     ↳ Complex(a+x.a,b+x.b);}
17     Complex operator-(const Complex &x)const{return
18     ↳ Complex(a-x.a,b-x.b);}
19     Complex operator*(const Complex &x)const{return
20     ↳ Complex(a*x.a-b*x.b,a*x.b+b*x.a);}
21     Complex &operator+=(const Complex &x){return
22     ↳ *this=*this+x;}
23 }w[maxn],w_inv[maxn];
24
25 //FFT初始化  $O(n)$ 
26 //需要调用sin、cos函数
27 void FFT_init(int n){
28     for(int i=0;i<n;i++){//根据单位根的旋转性质可以节省计算
29     ↳ 单位根逆元的时间
30         w[i]=w_inv[n-i-1]=Complex(cos(2*pi/n*i),sin(2*pi/n*i));
31     }
32     //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
33     ↳ 大常数
34 }
35
36 //FFT主过程  $O(n\log n)$ 
37 void FFT(Complex *A,int n,int tp){
38     for(int i=1,j=0,k;i<n-1;i++){
39         k=n;
40         do j^=(k>=1);while(j<k);
41         if(i<j)swap(A[i],A[j]);
42     }
43 }

```

#### 3.2.2 NTT

```

1 // Number Theory Transform 快速数论变换  $O(n\log n)$ 
2 // By AntiLeaf
3 // 通过题目 UOJ#34 多项式乘法
4 // 要求模数为 $10^9$ 以内的NTT模数
5
6 const int p = 998244353, g = 3; // p为模数g为p的任意一个原
7     ↳ 根
8
9 void NTT(int *A, int n, int tp) { // n为变换长度
10     ↳ tp为1或-1表示正/逆变换
11     for (int i = 1, j = 0, k; i < n - 1; i++) { //  $O(n)$ 旋转算
12     ↳ 法原理是模拟二进制加一
13         k = n;
14         do
15             j ^= (k >= 1);
16         while (j < k);
17
18         if(i < j)
19             swap(A[i], A[j]);
20     }
21
22     for (int k = 2; k <= n; k <= 1) {
23         int wn = qpow(g, (tp > 0 ? (p - 1) / k : (p - 1) / k
24     ↳ * (long long)(p - 2) % (p - 1)));
25         for (int i = 0; i < n; i += k) {
26             int w = 1;
27             for (int j = 0; j < (k >> 1); j++, w = (long
28     ↳ long)w * wn % p){
29                 int a = A[i + j], b = (long long)w * A[i + j
30     ↳ + (k >> 1)] % p;
31                 A[i + j] = (a + b) % p;
32                 A[i + j + (k >> 1)] = (a - b + p) % p;
33             } // 更好的写法是预处理单位根的次幂参照FFT的代
34     ↳ 码
35         }
36     }
37
38     if (tp < 0) {
39         int inv = qpow(n, p - 2); // 如果预处理过逆元的话就
40     ↳ 不用快速幂了
41         for (int i = 0; i < n; i++)
42             A[i] = (long long)A[i] * inv % p;
43     }
44 }

```

#### 3.2.3 任意模数卷积 三模数NTT

```

1 //只要求模数在 $2^{30}-1$ 以内,无其他特殊要求
2 //常数很大,慎用
3 //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
4 ↳ 任意模数卷积了

```

```

4 //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
5 //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
6
7 //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
8 //以对23333333(不是质数)取模为例
9 const int
  ↳ maxn=262200,Mod=23333333,g=3,m[]={998244353,1004535809,104540273};
10 m0_inv=669690699,m1_inv=332747959,M_inv=942377029;//这三
  ↳ 个模数最小原根都是3
11 const long long M=(long long)m[0]*m[1];
12
13 //主函数(当然更多时候包装一下比较好)
14 //用来卷积的是A和B
15 //需要调用mul
16 int n,N=1,A[maxn],B[maxn],C[maxn],D[maxn],ans[3][maxn];
17 int main(){
18     scanf("%d",&n);
19     while(N<(n<<1))N<<=1;
20     for(int i=0;i<n;i++)scanf("%d",&A[i]);
21     for(int i=0;i<n;i++)scanf("%d",&B[i]);
22     for(int i=0;i<3;i++)mul(m[i],ans[i]);
23     for(int i=0;i<n;i++)printf("%d ",China(ans[0][i],ans[1]
  ↳ [i],ans[2][i]));
24     return 0;
25 }
26
27 //mul  $O(n \log n)$ 
28 //包装了模NTT模数的卷积
29 //需要调用NTT
30 void mul(int p,int *ans){
31     copy(A,A+N,C);
32     copy(B,B+N,D);
33     NTT(C,N,1,p);
34     NTT(D,N,1,p);
35     for(int i=0;i<N;i++)ans[i]=(long long)C[i]*D[i]%p;
36     NTT(ans,N,-1,p);
37 }
38
39 //中国剩余定理  $O(1)$ 
40 //由于直接合并会爆Long Long,采用神奇的方法合并
41 //需要调用 $O(1)$ 快速乘
42 inline int China(int a0,int a1,int a2){
43     long long A=(mul((long long)a0*m1_inv,m[1],M)
  ↳ +mul((long long)a1*m0_inv,m[0],M))%M;
44     int k=((a2-A)%m[2]+m[2])%m[2]*M_inv%m[2];
45     return (k%Mod*(M%Mod)%Mod+A%Mod)%Mod;
46 }
47
48 //-----分割线-----
49
50 //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
51 //仍然以模23333333为例
52 const int maxn=262200,p=23333333,M=4830;//M取值要使得结果不
  ↳ 超过 $10^{14}$ 
53
54 //需要开的数组
55 struct Complex{//内容略
56 }w[maxn],w_inv[maxn],A[maxn],B[maxn],C[maxn],D[maxn],F[maxn],G[maxn],H[maxn];
57
58 //主函数(当然更多时候包装一下比较好)
59 //需要调用FFT初始化,FFT
60 int main(){

```

```

62     scanf("%d",&n);
63     int N=1;
64     while(N<(n<<1))N<<=1;
65     for(int i=0,x;i<n;i++){
66         scanf("%d",&x);
67         A[i]=x/M;
68         B[i]=x%M;
69     }
70     for(int i=0,x;i<n;i++){
71         scanf("%d",&x);
72         C[i]=x/M;
73         D[i]=x%M;
74     }
75     FFT_init(N);
76     FFT(A,N,1);
77     FFT(B,N,1);
78     FFT(C,N,1);
79     FFT(D,N,1);
80     for(int i=0;i<N;i++){
81         F[i]=A[i]*C[i];
82         G[i]=A[i]*D[i]+B[i]*C[i];
83         H[i]=B[i]*D[i];
84     }
85     FFT(F,N,-1);
86     FFT(G,N,-1);
87     FFT(H,N,-1);
88     for(int i=0;i<n;i++)
89         printf("%d\n",(int)((M*M*((long long)
  ↳ (F[i].a+0.5)%p)%p+
90         M*((long long)(G[i].a+0.5)%p)%p+(long long)
  ↳ (H[i].a+0.5)%p)%p));
91     return 0;
92 }

```

### 3.2.4 多项式操作

```

1 //Polymial Operations 多项式操作
2 //By ysf
3 //通过题目COGS2189 帕秋莉的超级多项式板子题
4
5 const int maxn=262200;//以下所有代码均为NTT版本
6 //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
7
8 //多项式求逆  $O(n \log n)$ 
9 //要求A常数项不为0
10 void getinv(int *A,int *C,int n){
11     static int B[maxn];
12     memset(C,0,sizeof(int)*(n<<1));
13     C[0]=qpow(A[0],p-2);//一般题目直接赋值为1就可以
14     for(int k=2;k<=n;k<<=1){
15         memcpy(B,A,sizeof(int)*k);
16         memset(B+k,0,sizeof(int)*k);
17         NTT(B,k<<1,1);
18         NTT(C,k<<1,1);
19         for(int i=0;i<(k<<1);i++)
20             C[i]=((2-(long long)B[i]*C[i])%p*C[i]%p+p)%p;
21         NTT(C,k<<1,-1);
22         memset(C+k,0,sizeof(int)*k);
23     }
24 }
25
26 //多项式开根  $O(n \log n)$ 
27 //要求A常数项可以开根/存在二次剩余

```

```

28 //需要调用多项式求逆且需要预处理2的逆元
29 void getsqrt(int *A,int *C,int n){
30     static int B[maxn],D[maxn];
31     memset(C,0,sizeof(int)*(n<<1));
32     C[0]=(int)(sqrt(A[0])+1e-7);//一般题目直接赋值为1就可以
33     for(int k=2;k<=n;k<<=1){
34         memcpy(B,A,sizeof(int)*k);
35         memset(B+k,0,sizeof(int)*k);
36         getinv(C,D,k);
37         NTT(B,k<<1,1);
38         NTT(D,k<<1,1);
39         for(int i=0;i<(k<<1);i++)B[i]=(long long)B[i]*D[i]%p;
40         NTT(B,k<<1,-1);
41         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
42     }
43 }
44
45 //求导 O(n)
46 void getderivative(int *A,int *C,int n){
47     for(int i=1;i<n;i++)C[i-1]=(long long)A[i]*i%p;
48     C[n-1]=0;
49 }
50
51 //不定积分 O(n\log n)如果预处理过逆元可以降到O(n)
52 void getintegrate(int *A,int *C,int n){
53     for(int i=1;i<n;i++)C[i]=(long long)A[i-1]*qpow(i,p-2)%p;
54     C[0]=0;//由于是不定积分结果没有常数项
55 }
56
57 //多项式\ln O(n\log n)
58 //要求A常数项不为0/存在离散对数
59 //需要调用多项式求逆求导不定积分
60 void getln(int *A,int *C,int n){//通常情况下A常数项都是1
61     static int B[maxn];
62     getderivative(A,B,n);
63     memset(B+n,0,sizeof(int)*n);
64     getinv(A,C,n);
65     NTT(B,n<<1,1);
66     NTT(C,n<<1,1);
67     for(int i=0;i<(n<<1);i++)B[i]=(long long)B[i]*C[i]%p;
68     NTT(B,n<<1,-1);
69     getintegrate(B,C,n);
70     memset(C+n,0,sizeof(int)*n);
71 }
72
73 //多项式\exp O(n\log n)
74 //要求A没有常数项
75 //需要调用多项式\ln
76 //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
77 //分治FFT依据设G(x)=\exp F(x)则有g_i=\sum_{k=1}^i f_k
    ↪ g_{i-k}
78 void getexp(int *A,int *C,int n){
79     static int B[maxn];
80     memset(C,0,sizeof(int)*(n<<1));
81     C[0]=1;
82     for(int k=2;k<=n;k<<=1){
83         getln(C,B,k);
84         for(int i=0;i<k;i++){
85             B[i]=A[i]-B[i];
86             if(B[i]<0)B[i]+=p;
87         }
88         (+=B[0])%=p;

```

```

89     NTT(B,k<<1,1);
90     NTT(C,k<<1,1);
91     for(int i=0;i<(k<<1);i++)C[i]=(long long)C[i]*B[i]%p;
92     NTT(C,k<<1,-1);
93     memset(C+k,0,sizeof(int)*k);
94 }
95 }
96
97 //多项式k次幂 O(n\log n)
98 //在A常数项不为1时需要转化
99 //需要调用多项式/exp、\ln
100 //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
101 void getpow(int *A,int *C,int n,int k){
102     static int B[maxn];
103     getln(A,B,n);
104     for(int i=0;i<n;i++)B[i]=(long long)B[i]*k%p;
105     getexp(B,C,n);
106 }

```

### 3.2.5 拉格朗日反演

### 3.2.6 半在线卷积

```

1 // Half-OnLine Convolution 半在线卷积
2 // By AntiLeaf
3 // O(n\log^2 n)
4 // 通过题目自己出的题
5
6
7 // 主过程递归调用自身
8 void solve(int l, int r) {
9     if (r <= m)
10         return;
11
12     if (r - l == 1) {
13         if (l == m)
14             f[l] = a[m];
15         else
16             f[l] = (long long)f[l] * inv[l - m] % p;
17
18         for (int i = l, t = (long long)l * f[l] % p; i <= n;
            ↪ i += 1)
19             g[i] = (g[i] + t) % p;
20
21         return;
22     }
23
24     int mid = (l + r) / 2;
25
26     solve(l, mid);
27
28     if (l == 0) {
29         for (int i = 1; i < mid; i++) {
30             A[i] = f[i];
31             B[i] = (c[i] + g[i]) % p;
32         }
33         NTT(A, r, 1);
34         NTT(B, r, 1);
35         for (int i = 0; i < r; i++)
36             A[i] = (long long)A[i] * B[i] % p;
37         NTT(A, r, -1);
38
39         for (int i = mid; i < r; i++)
40             f[i] = (f[i] + A[i]) % p;

```

```

41 }
42 else {
43     for (int i = 0; i < r - 1; i++)
44         A[i] = f[i];
45     for (int i = 1; i < mid; i++)
46         B[i - 1] = (c[i] + g[i]) % p;
47     NTT(A, r - 1, 1);
48     NTT(B, r - 1, 1);
49     for (int i = 0; i < r - 1; i++)
50         A[i] = (long long)A[i] * B[i] % p;
51     NTT(A, r - 1, -1);
52
53     for (int i = mid; i < r; i++)
54         f[i] = (f[i] + A[i - 1]) % p;
55
56     memset(A, 0, sizeof(int) * (r - 1));
57     memset(B, 0, sizeof(int) * (r - 1));
58
59     for (int i = 1; i < mid; i++)
60         A[i - 1] = f[i];
61     for (int i = 0; i < r - 1; i++)
62         B[i] = (c[i] + g[i]) % p;
63     NTT(A, r - 1, 1);
64     NTT(B, r - 1, 1);
65     for (int i = 0; i < r - 1; i++)
66         A[i] = (long long)A[i] * B[i] % p;
67     NTT(A, r - 1, -1);
68
69     for (int i = mid; i < r; i++)
70         f[i] = (f[i] + A[i - 1]) % p;
71 }
72
73 memset(A, 0, sizeof(int) * (r - 1));
74 memset(B, 0, sizeof(int) * (r - 1));
75
76 solve(mid, r);
77 }

```

### 3.3 FWT快速沃尔什变换

```

1 //Fast Walsh-Hadamard Transform 快速沃尔什变换  $O(n \log n)$ 
2 //By ysf
3 //通过题目 PCOGS 上几道板子题
4
5 //注意FWT常数比较小这点与FFT/NTT不同
6 //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
7
8 //按位或版本
9 void FWT_or(int *A, int n, int tp){
10     for(int k=2; k<=n; k<=1)
11         for(int i=0; i<n; i+=k)
12             for(int j=0; j<(k>>1); j++){
13                 if(tp>0)A[i+j+(k>>1)]=(A[i+j+(k>>1)]+A[i+j])%p;
14                 else
15                     A[i+j+(k>>1)]=(A[i+j+(k>>1)]-A[i+j]+p)%p;
16             }
17 }
18
19 //按位与版本
20 void FWT_and(int *A, int n, int tp){
21     for(int k=2; k<=n; k<=1)
22         for(int i=0; i<n; i+=k)
23             for(int j=0; j<(k>>1); j++){

```

```

23         if(tp>0)A[i+j]=(A[i+j]+A[i+j+(k>>1)])%p;
24         else A[i+j]=(A[i+j]-A[i+j+(k>>1)]+p)%p;
25     }
26 }
27
28 //按位异或版本
29 void FWT_xor(int *A, int n, int tp){
30     for(int k=2; k<=n; k<=1)
31         for(int i=0; i<n; i+=k)
32             for(int j=0; j<(k>>1); j++){
33                 int a=A[i+j], b=A[i+j+(k>>1)];
34                 A[i+j]=(a+b)%p;
35                 A[i+j+(k>>1)]=(a-b+p)%p;
36             }
37     if(tp<0){
38         int inv=qpow(n%p, p-2); //n的逆元在不取模时需要用每层
39         //除以2代替
40         for(int i=0; i<n; i++)A[i]=A[i]*inv%p;
41     }
42 }

```

### 3.4 单纯形

```

1 //Simplex Method 单纯形方法求解线性规划
2 //By ysf
3 //通过题目 P00J#179 线性规划 然而被hack了QAQ.....
4
5 //单纯形其实是指数算法但在实践中跑得飞快所以复杂度什么的也
6 //就无所谓了
7
8 const double eps=1e-10;
9
10 double A[maxn][maxn], x[maxn];
11 int n, m, t, id[maxn<<1];
12
13 //方便起见这里附上主函数
14 int main(){
15     scanf("%d%d%d", &n, &m, &t);
16     for(int i=1; i<=n; i++){
17         scanf("%lf", &A[0][i]);
18         id[i]=i;
19     }
20     for(int i=1; i<=m; i++){
21         for(int j=1; j<=n; j++)scanf("%lf", &A[i][j]);
22         scanf("%lf", &A[i][0]);
23     }
24     if(!inititalize())printf("Infeasible");
25     else if(!simplex())printf("Unbounded");
26     else{
27         printf("%.15lf\n", -A[0][0]);
28         if(t){
29             for(int i=1; i<=m; i++)x[id[i+n]]=A[i][0];
30             for(int i=1; i<=n; i++)printf("%.15lf ", x[i]);
31         }
32     }
33     return 0;
34 }
35
36 //初始化
37 //对于初始解可行的问题可以把初始化省略掉
38 bool inititalize(){
39     for(;;){
40         double t=0.0;

```

```

41     int l=0,e=0;
42     for(int i=1;i<=m;i++)if(A[i][0]+eps<t){
43         t=A[i][0];
44         l=i;
45     }
46     if(!l)return true;
47     for(int i=1;i<=n;i++)if(A[l]
        ↪ [i]<-eps&&(!e||id[i]<id[e]))e=i;
48     if(!e)return false;
49     pivot(l,e);
50 }
51 }
52
53 //求解
54 bool simplex(){
55     for(;;){
56         int l=0,e=0;
57         for(int i=1;i<=n;i++)if(A[0]
            ↪ [i]>eps&&(!e||id[i]<id[e]))e=i;
58         if(!e)return true;
59         double t=1e50;
60         for(int i=1;i<=m;i++)if(A[i][e]>eps&&A[i][0]/A[i]
            ↪ [e]<t){
61             l=i;

```

```

62         t=A[l][0]/A[l][e];
63     }
64     if(!l)return false;
65     pivot(l,e);
66 }
67 }
68
69 //转轴操作本质是
70 void pivot(int l,int e){
71     swap(id[e],id[n+1]);
72     double t=A[l][e];
73     A[l][e]=1.0;
74     for(int i=0;i<=n;i++)A[l][i]/=t;
75     for(int i=0;i<=m;i++)if(i!=l){
76         t=A[i][e];
77         A[i][e]=0.0;
78         for(int j=0;j<=n;j++)A[i][j]-=t*A[l][j];
79     }
80 }

```

## 3.5 线性代数

### 3.5.1 线性基