

All-in at the River

Standard Code Library

Shanghai Jiao Tong University

Desprado2

fstqwq

AntiLeaf



“

那一年的区域赛是 ICPC2020 银川

最终我们差 100 分钟出线

当时我看见 fstqwq 趴在魄罗上 泣不成声

这个画面我永生难忘

那一刻我在想 如果能再给我一次机会 我一定要赢回所有
如今沈阳就在眼前 我必须考虑这会不会是我此生仅有的机会

我相信 Talancode 能有过去的霸主地位 fstqwq 功不可没

重铸交大荣光，我辈义不容辞

”

Contents

1 图论

1.1 最小生成树	2
1.1.1 Boruvka算法	2
1.1.2 动态最小生成树	2
1.1.3 Steiner Tree 斯坦纳树	3
1.2 最短路	3
1.2.1 k短路	3
1.3 仙人掌	4
1.3.1 仙人掌DP	4
1.4 二分图	5
1.4.1 KM二分图最大权匹配	5
1.5 一般图匹配	6
1.5.1 高斯消元	6
1.5.2 带花树	7
1.6 最大流	8
1.6.1 Dinic	8
1.6.2 ISAP	9
1.6.3 HLPP最高标号预流推进	10

2 字符串

2.1 AC自动机	10
2.2 后缀数组	11
2.2.1 SA-IS	11
2.3 后缀自动机	12
2.4 回文树	13
2.4.1 广义回文树	13
2.5 Manacher马拉车	15
2.6 KMP	15
2.6.1 ex-KMP	15

3 数学

3.1 插值	15
3.1.1 牛顿插值	15
3.1.2 拉格朗日插值	15
3.2 多项式	16
3.2.1 FFT	16
3.2.2 NTT	16
3.2.3 任意模数卷积(三模数NTT)	16
3.2.4 多项式操作	17
3.2.5 拉格朗日反演	18
3.2.6 半在线卷积	18
3.3 FWT快速沃尔什变换	19
3.4 单纯形	19
3.5 线性代数	20
3.5.1 线性基	20
3.6 常见数列	20
3.6.1 伯努利数	20

4 数论

4.1 $O(n)$ 预处理逆元	20
4.2 杜教筛	20
4.3 线性筛	20
4.4 Miller-Rabin	21
4.5 Pollard's Rho	21

5 数据结构

5.1 线段树	21
5.1.1 主席树	21
5.2 陈丹琦分治	21
5.3 Treap	22
5.4 Splay	23
5.5 树分治	23
5.5.1 动态树分治	23
5.5.2 紫荆花之恋	23

5.6 LCT	25
5.6.1 不换根(弹飞绵羊)	25
5.6.2 换根/维护生成树(GREALD07加强版)	25
5.6.3 维护子树信息	27
5.6.4 模板题:动态QTREE4(询问树上相距最远点)	28
5.7 长链剖分	30
5.8 梯子剖分	30
5.9 左偏树	31
5.10 常见根号思路	31

6 动态规划

6.1 决策单调性 $O(n \log n)$	31
-------------------------	----

7 Miscellaneous

7.1 $O(1)$ 快速乘	32
7.2 $O(n^2)$ 高精度	32
7.3 xorshift	35
7.4 枚举子集	35
7.5 STL	35
7.6 pb_ds	35
7.7 rope	35

8 注意事项

8.1 常见下毒手法	35
8.2 场外相关	36
8.3 做题策略与心态调节	36

1. 图论

1.1 最小生成树

1.1.1 Boruvka算法

思想: 每次选择连接每个连通块的最小边, 把连通块缩起来.

每次连通块个数至少减半, 所以迭代 $O(\log n)$ 次即可得到最小生成树.

一种比较简单的实现方法: 每次迭代遍历所有边, 用并查集维护连通性和每个连通块的最小边权.

应用: 最小异或生成树

1.1.2 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
10 // 全局结构体与数组定义
11 struct edge { // 边的定义
12     int u, v, w, id; // id表示边在原图中的编号
13     bool vis; // 在Kruskal时, 记录这条边是否是树边
14     bool operator < (const edge &e) const { return w <
        e.w; }
15 } e[20][maxn], t[maxn]; // 为了便于回滚, 在每层分治存一个
  // 副本
16
17
18 // 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
19 struct A {
20     int id, u, v;
21 } a[maxn];
22
23
24 int id[20][maxn]; // 每条边在当前图中的编号
25 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
26 int n, m, q; // 点数, 边数, 修改数
27
28 // 方便起见, 附上可能需要用到的预处理代码
29 for (int i = 1; i <= n; i++) { // 并查集初始化
30     p[i] = i;
31     size[i] = 1;
32 }
33
34
35 for (int i = 1; i <= m; i++) { // 读入与预标号
36     scanf("%d%d%d", &e[0][i].u, &e[0][i].v, &e[0][i].w);
37     e[0][i].id = i;
38     id[0][i] = i;
39 }
40
41 for (int i = 1; i <= q; i++) { // 预处理出调用数组
42     scanf("%d%d", &a[i].id, &a[i].v);
43     a[i].u = e[0][a[i].id].w;
44     e[0][a[i].id].w = a[i].v;
```

```
45 }
46
47 for(int i = q; i; i--)
48     e[0][a[i].id].w = a[i].u;
49
50 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 = 1; 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 |     m=cnt;
112 | }
113 |
114 | // 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
115 | // 返回缩掉的边的总权值
116 | // 需要调用Kruskal
117 | long long Contraction(int l, int r, int d, int &m) {
118 |     long long ans = 0;
119 |
120 |     for (int i = l; i <= r; i++)
121 |         e[d][id[d][a[i].id]].w = -INF; // 待修改边标
    |         ↪ 为-INF
122 |
123 |     Kruskal(m, e[d]);
124 |     copy(e[d] + 1, e[d] + m + 1, t + 1);
125 |
126 |     int cnt = 0;
127 |     for (int i = 1; i <= m; i++) {
128 |
129 |         if (t[i].w != -INF && t[i].vis) { // 必须边
130 |             ans += t[i].w;
131 |             mergeset(t[i].u, t[i].v);
132 |         }
133 |         else { // 不确定边
134 |             id[d][t[i].id]++;
135 |             e[d][cnt]=t[i];
136 |         }
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 | // Kruskal算法 O(mlogn)
151 | // 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
152 | long long Kruskal(int m, edge *e) {
153 |     int tmp = top;
154 |     long long ans = 0;
155 |
156 |     sort(e + 1, e + m + 1); // 比较函数在结构体中定义过了
157 |
158 |     for (int i = 1; i <= m; i++) {
159 |         if (findroot(e[i].u) != findroot(e[i].v)) {
160 |             e[i].vis = true;
161 |             ans += e[i].w;
162 |             mergeset(e[i].u, e[i].v);
163 |         }
164 |         else
165 |             e[i].vis = false;
166 |     }
167 | }
168 |
169 | for(int i = top; i > tmp; i--)
170 |     cut(stk[i]); // 撤销所有操作
171 | top = tmp;
172 |
173 | return ans;

```

```

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
185 |     x = findroot(x); // 但是按秩合并要再开一个栈记录合并
    |     ↪ 之前的秩
186 |     y = findroot(y);
187 |
188 |     if (x == y)
189 |         return;
190 |
191 |     if (size[x] > size[y])
192 |         swap(x, y);
193 |
194 |     p[x] = y;
195 |     size[y] += size[x];
196 |     stk[++top] = x;
197 | }
198 |
199 | void cut(int x) { // 并查集撤销
200 |     int y = x;
201 |
202 |     do
203 |         size[y = p[y]] -= size[x];
204 |     while (p[y] != y);
205 |
206 |     p[x] = x;
207 | }
208 |

```

1.1.3 Steiner Tree 斯坦纳树

问题: 一张图上有 k 个关键点, 求让关键点两两连通的最小生成树
做法: 状压DP, $f_{i,S}$ 表示以 i 号为树根, i 与 S 中的点连通的最小边权和
 转移有两种:

1. 枚举子集:

$$f_{i,S} = \min_{T \subset S} \{f_{i,T} + f_{i,S \setminus T}\}$$

2. 新加一条边:

$$f_{i,S} = \min_{(i,j) \in E} \{f_{j,S} + w_{i,j}\}$$

第一种直接枚举子集DP就行了, 第二种可以用SPFA或者Dijkstra松弛(显然负边一开始全选就行了, 所以只需要处理非负边).
 复杂度 $O(n3^k + 2^k m \log n)$.

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[maxn],*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!!!
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短路径用二又堆维护
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 $O(n \log n)$ 
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 struct edge{
2     int to, w, prev;
3 }e[maxn * 2];
4
5 vector<pair<int, int> > v[maxn];
6
7 vector<long long> d[maxn];
8
9 stack<int> stk;
10
11 int p[maxn];
12
13 bool vis[maxn], vise[maxn * 2];
14
15 int last[maxn], cnte;

```

```

16 long long f[maxn], g[maxn], sum[maxn];
17
18 int n, m, cnt;
19
20 void addedge(int x, int y, int w) {
21     v[x].push_back(make_pair(y, w));
22 }
23
24 void dfs(int x) {
25     vis[x] = true;
26
27     for (int i = last[x]; ~i; i = e[i].prev) {
28         if (vis[e[i].to]) continue;
29
30         int y = e[i].to, w = e[i].w;
31
32         vis[e[i].to] = true;
33
34         if (!vis[y]) {
35             stk.push(i);
36             p[y] = x;
37             dfs(y);
38
39             if (!stk.empty() && stk.top() == i) {
40                 stk.pop();
41                 addedge(x, y, w);
42             }
43         }
44
45         else {
46             cnt++;
47
48             long long tmp = w;
49             while (!stk.empty()) {
50                 int i = stk.top();
51                 stk.pop();
52
53                 int yy = e[i].to, ww = e[i].w;
54
55                 addedge(cnt, yy, 0);
56
57                 d[cnt].push_back(tmp);
58
59                 tmp += ww;
60
61                 if (e[i].to == y) break;
62             }
63
64             addedge(y, cnt, 0);
65
66             sum[cnt] = tmp;
67         }
68     }
69 }
70
71 void dp(int x) {
72     for (auto o : v[x]) {
73         int y = o.first, w = o.second;
74         dp(y);
75     }
76
77     if (x <= n) {
78         for (auto o : v[x]) {
79             int y = o.first, w = o.second;
80
81             f[x] += 2 * w + f[y];
82         }
83
84         g[x] = f[x];
85
86         for (auto o : v[x]) {
87             int y = o.first;
88
89             f[x] += f[y];
90
91             g[x] = min(g[x], f[x] - f[y] - 2 * w + g[y] + w);
92         }
93
94         else {
95             f[x] = sum[x];
96             for (auto o : v[x]) {
97                 int y = o.first;
98
99                 f[x] += f[y];
100
101                 g[x] = min(g[x], f[x] - f[y] + g[y] + min(d[x][i], sum[x] - d[x][i]));
102             }
103         }
104     }
105 }
106
107 int main() {
108     int n, m, N, e;
109     // 增广
110     bool check(int y) {
111         visy[y] = true;
112
113         if (boy[y]) {
114             visx[boy[y]] = true;
115             q[tail++] = boy[y];
116             return false;
117         }
118
119         while (y) {
120             boy[y] = p[y];
121             swap(y, girl[p[y]]);
122         }
123
124         return true;
125     }
126
127     // bfs每个点
128     void bfs(int x) {

```

```

86         f[x] += 2 * w + f[y];
87     }
88
89     g[x] = f[x];
90
91     for (auto o : v[x]) {
92         int y = o.first, w = o.second;
93
94         g[x] = min(g[x], f[x] - f[y] - 2 * w + g[y] + w);
95     }
96 }
97
98 else {
99     f[x] = sum[x];
100     for (auto o : v[x]) {
101         int y = o.first;
102
103         f[x] += f[y];
104
105         g[x] = min(g[x], f[x] - f[y] + g[y] + min(d[x][i], sum[x] - d[x][i]));
106     }
107 }
108
109 for (int i = 0; i < (int)v[x].size(); i++) {
110     int y = v[x][i].first;
111
112     g[x] = min(g[x], f[x] - f[y] + g[y] + min(d[x][i], sum[x] - d[x][i]));
113 }
114 }
115 }

```

1.4 二分图

1.4.1 KM二分图最大权匹配

```

1 const long long INF = 0x3f3f3f3f3f3f3f3f;
2
3 long long w[maxn][maxn], lx[maxn], ly[maxn], slack[maxn];
4 // 边权 顶标 slack
5 // 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
6 // 边对0取max
7
8 bool visx[maxn], visy[maxn];
9
10 int boy[maxn], girl[maxn], p[maxn], q[maxn], head, tail;
11 // p : pre
12
13 int n, m, N, e;
14
15 // 增广
16 bool check(int y) {
17     visy[y] = true;
18
19     if (boy[y]) {
20         visx[boy[y]] = true;
21         q[tail++] = boy[y];
22         return false;
23     }
24
25     while (y) {
26         boy[y] = p[y];
27         swap(y, girl[p[y]]);
28     }
29
30     return true;
31 }
32
33 // bfs每个点
34 void bfs(int x) {

```

```

33     memset(q, 0, sizeof(q));
34     head = tail = 0;
35
36     q[tail++] = x;
37     visx[x] = true;
38
39     while (true) {
40         while (head != tail) {
41             int x = q[head++];
42
43             for (int y = 1; y <= N; y++)
44                 if (!visy[y]) {
45                     long long d = lx[x] + ly[y] - w[x]
46                         ↪ [y];
47
48                     if (d < slack[y]) {
49                         p[y] = x;
50                         slack[y] = d;
51
52                         if (!slack[y] && check(y))
53                             return;
54                     }
55                 }
56
57             long long d = INF;
58             for (int i = 1; i <= N; i++)
59                 if (!visy[i])
60                     d = min(d, slack[i]);
61
62             for (int i = 1; i <= N; i++) {
63                 if (visx[i])
64                     lx[i] -= d;
65
66                 if (visy[i])
67                     ly[i] += d;
68                 else
69                     slack[i] -= d;
70             }
71
72             for (int i = 1; i <= N; i++)
73                 if (!visy[i] && !slack[i] && check(i))
74                     return;
75         }
76     }
77
78 // 主过程
79 long long KM() {
80     for (int i = 1; i <= N; i++) {
81         // lx[i] = 0;
82         ly[i] = -INF;
83         // boy[i] = girl[i] = -1;
84
85         for (int j = 1; j <= N; j++)
86             ly[i] = max(ly[i], w[j][i]);
87     }
88
89     for (int i = 1; i <= N; i++) {
90         memset(slack, 0x3f, sizeof(slack));
91         memset(visx, 0, sizeof(visx));
92         memset(visy, 0, sizeof(visy));
93         bfs(i);
94     }
95
96     long long ans = 0;
97     for (int i = 1; i <= N; i++)
98         ans += w[i][girl[i]];
99     return ans;
100 }

```

```

101
102 // 为了方便贴上主函数
103 int main() {
104
105     scanf("%d%d", &n, &m, &e);
106     N = max(n, m);
107
108     while (e--) {
109         int x, y, c;
110         scanf("%d%d", &x, &y, &c);
111         w[x][y] = max(c, 0);
112     }
113
114     printf("%lld\n", KM());
115
116     for (int i = 1; i <= n; i++) {
117         if (i > 1)
118             printf(" ");
119         printf("%d", w[i][girl[i]] > 0 ? girl[i] : 0);
120     }
121     printf("\n");
122
123     return 0;
124 }

```

1.5 一般图匹配

1.5.1 高斯消元

```

1 // 这个算法基于Tutte定理和高斯消元,思维难度相对小一些,也
2   ↪ 更方便进行可行边的判定
3 // 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
4   ↪ 是跑不满的
5 // 以及,根据Tutte定理,如果求最大匹配的大小的话直接输
6   ↪ 出Tutte矩阵的秩/2即可
7 // 需要输出方案时才需要再写后面那些乱七八糟的东西
8
9 // 复杂度和常数所限,1s之内500已经是这个算法的极限了
10 const int maxn = 505, p = 1000000007; // p可以是任
11   ↪ 意10^9以内的质数
12
13 // 全局数组和变量定义
14 int A[maxn][maxn], B[maxn][maxn], t[maxn][maxn],
15   ↪ 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", &n, &m); // 点数和边数
25     while (m--) {
26         int x, y;
27         scanf("%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         for (int i = 1; i <= n; i++)
32             id[i] = i; // 输出方案用的,因为高斯消元的时候会
33             ↪ 交换列
34
35         memcpy(t, A, sizeof(t));
36         Gauss(A, NULL, n);
37
38         m = n;
39         n = 0; // 这里变量复用纯属个人习惯.....

```



```

35
36     for (int i = 1; i <= m; i++)
37         if (A[id[i]][id[i]])
38             a[+n] = i; // 找出一个极大满秩子矩阵
39
40     for (int i = 1; i <= n; i++)
41         for (int j = 1; j <= n; j++)
42             A[i][j] = t[a[i]][a[j]];
43
44     Gauss(A, B, n);
45
46     for (int i = 1; i <= n; i++)
47         if (!girl[a[i]])
48             for (int j = i + 1; j <= n; j++)
49                 if (!girl[a[j]] && t[a[i]][a[j]] && B[j]
50                     ↪ [i]) {
51                     // 注意上面那句if的写法, 现在t是邻接矩
52                     ↪ 阵的备份,
53                     // 逆矩阵j行i列不为0当且仅当这条边可
54                     ↪ 行
55                     girl[a[i]] = a[j];
56                     girl[a[j]] = a[i];
57                     eliminate(i, j);
58                     eliminate(j, i);
59                     break;
60             }
61
62     printf("%d\n", n >> 1);
63     for (int i = 1; i <= m; i++)
64         printf("%d ", girl[i]);
65
66     return 0;
67 }
68
69 // 高斯消元  $O(n^3)$ 
70 // 在传入B时表示计算逆矩阵, 传入NULL则只需计算矩阵的秩
71 void Gauss(int A[][maxn], int B[][maxn], int n){
72     if(B) {
73         memset(B, 0, sizeof(t));
74         for (int i = 1; i <= n; i++)
75             B[i][i] = 1;
76     }
77
78     for (int i = 1; i <= n; i++) {
79         if (!A[i][i]) {
80             for (int j = i + 1; j <= n; j++)
81                 if (A[j][i]) {
82                     swap(id[i], id[j]);
83                     for (int k = i; k <= n; k++)
84                         swap(A[i][k], A[j][k]);
85
86                     if (B)
87                         for (int k = 1; k <= n; k++)
88                             swap(B[i][k], B[j][k]);
89                     break;
90                 }
91             if (!A[i][i])
92                 continue;
93         }
94
95         int inv = qpow(A[i][i], p - 2);
96
97         for (int j = 1; j <= n; j++)
98             if (i != j && A[j][i]){
99                 int t = (long long)A[j][i] * inv % p;
100                 for (int k = i; k <= n; k++)
101                     if (A[i][k])

```

```

101         A[j][k] = (A[j][k] - (long long)t
102             ↪ * A[i][k]) % p;
103     }
104     if (B)
105         for (int k = 1; k <= n; k++)
106             if (B[i][k])
107                 B[j][k] = (B[j][k] - (long
108                     ↪ long)t * B[i][k])%p;
109     }
110 }
111
112 if (B)
113     for (int i = 1; i <= n; i++) {
114         int inv = qpow(A[i][i], p - 2);
115
116         for (int j = 1; j <= n; j++)
117             if (B[i][j])
118                 B[i][j] = (long long)B[i][j] * inv %
119                     ↪ p;
120     }
121 }
122
123 // 消去一行一列  $O(n^2)$ 
124 void eliminate(int r, int c) {
125     row[r] = col[c] = true; // 已经被消掉
126
127     int inv = qpow(B[r][c], p - 2);
128
129     for (int i = 1; i <= n; i++)
130         if (!row[i] && B[i][c]) {
131             int t = (long long)B[i][c] * inv % p;
132
133             for (int j = 1; j <= n; j++)
134                 if (!col[j] && B[r][j])
135                     B[i][j] = (B[i][j] - (long long)t *
136                         ↪ B[r][j]) % p;
137         }
138     }
139 }

```

1.5.2 带花树

```

1 // 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
2 // 时候并没有高斯消元好写
3 // 当然输出方案要方便很多
4
5 // 全局数组与变量定义
6 vector<int> G[maxn];
7 int girl[maxn], f[maxn], t[maxn], p[maxn], vis[maxn],
8     tim, q[maxn], head, tail;
9 int n, m;
10
11 // 封装好的主过程  $O(nm)$ 
12 int blossom() {
13     int ans = 0;
14
15     for (int i = 1; i <= n; i++)
16         if (!girl[i])
17             ans += bfs(i);
18
19     return ans;
20 }
21
22 // bfs找增广路  $O(m)$ 
23 bool bfs(int s) {
24     memset(t, 0, sizeof(t));
25     memset(p, 0, sizeof(p));
26

```



```

27     for (int i = 1; i <= n; i++)
28         f[i] = i; // 并查集
29
30     head = tail = 0;
31     q[tail++] = s;
32     t[s] = 1;
33
34     while (head != tail){
35         int x = q[head++];
36         for (int y : G[x]){
37             if (findroot(y) == findroot(x) || t[y] == 2)
38                 continue;
39
40             if (!t[y]){
41                 t[y] = 2;
42                 p[y] = x;
43
44                 if (!girl[y]){
45                     for (int u = y, t; u = t) {
46                         t = girl[p[u]];
47                         girl[p[u]] = u;
48                         girl[u] = p[u];
49                     }
50                     return true;
51                 }
52                 t[girl[y]] = 1;
53                 q[tail++] = girl[y];
54             }
55             else if (t[y] == 1) {
56                 int z = LCA(x, y);
57                 shrink(x, y, z);
58                 shrink(y, x, z);
59             }
60         }
61     }
62
63     return false;
64 }
65
66 //缩奇环 O(n)
67 void shrink(int x, int y, int z) {
68     while (findroot(x) != z){
69         p[x] = y;
70         y = girl[x];
71
72         if (t[y] == 2) {
73             t[y] = 1;
74             q[tail++] = y;
75         }
76
77         if (findroot(x) == x)
78             f[x] = z;
79         if (findroot(y) == y)
80             f[y] = z;
81
82         x = p[y];
83     }
84 }
85
86 //暴力找LCA O(n)
87 int LCA(int x, int y) {
88     tim++;
89     while (true) {
90         if (x) {
91             x = findroot(x);
92
93             if (vis[x] == tim)
94                 return x;
95             else {

```

```

96         vis[x] = tim;
97         x = p[girl[x]];
98     }
99 }
100 swap(x, y);
101 }
102 }
103
104 //并查集的查找 O(1)
105 int findroot(int x) {
106     return x == f[x] ? x : (f[x] = findroot(f[x]));
107 }

```

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{
4     int to, cap, prev;
5 } e[maxe * 2];
6
7 int last[maxn], len, d[maxn], cur[maxn], q[maxn];
8
9 memset(last, -1, sizeof(last));
10
11 void AddEdge(int x, int y, int z) {
12     e[len].to = y;
13     e[len].cap = z;
14     e[len].prev = last[x];
15     last[x] = len++;
16 }
17
18 int Dinic() {
19     int flow = 0;
20     while (bfs(), ~d[t]) {
21         memcpy(cur, last, sizeof(int) * (t + 5));
22         flow += dfs(s, inf);
23     }
24     return flow;
25 }
26
27 void bfs() {
28     int head = 0, tail = 0;
29     memset(d, -1, sizeof(int) * (t + 5));
30     q[tail++] = s;
31     d[s] = 0;
32
33     while (head != tail){
34         int x = q[head++];
35         for (int i = last[x]; ~i; i = e[i].prev)
36             if (e[i].cap > 0 && d[e[i].to] == -1) {
37                 d[e[i].to] = d[x] + 1;
38                 q[tail++] = e[i].to;
39             }
40     }
41 }
42
43 int dfs(int x, int a) {
44     if (x == t || !a)
45         return a;
46
47     int flow = 0, f;
48     for (int &i = cur[x]; ~i; i = e[i].prev)
49         if (e[i].cap > 0 && d[e[i].to] == d[x] + 1 && (f
50             ↳ = dfs(e[i].to, min(e[i].cap, a)))) {

```

```

51 |         e[i].cap -= f;
52 |         e[i^1].cap += f;
53 |         flow += f;
54 |         a -= f;
55 |
56 |         if (!a)
57 |             break;
58 |     }
59 |
60 |     return flow;
61 | }

```

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
30 // 真·加边函数 O(1)
31 void AddEdge(int x, int y, int z){
32     e[cnte].to = y;
33     e[cnte].cap = z;
34     e[cnte].prev = last[x];
35     last[x] = cnte++;
36 }
37
38
39 // 主过程 O(n^2 m)
40 // 返回最大流的流量
41 // 需要调用bfs, augment
42 // 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
43 // 非递归
44 int ISAP() {
45     bfs();
46
47     memcpy(cur, last, sizeof(cur));
48
49     int x = s, flow = 0;
50
51     while (d[s] < n) {

```

```

52 |         if (x == t) { //如果走到了t就增广一次,并返回s重新
  ↳ 找增广路
53 |             flow += augment();
54 |             x = s;
55 |         }
56 |
57 |         bool ok = false;
58 |         for (int &i = cur[x]; ~i; i = e[i].prev)
59 |             if (e[i].cap && d[x] == d[e[i].to] + 1) {
60 |                 p[e[i].to] = i;
61 |                 x = e[i].to;
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
18         ↪ 改变,所以可以直接比较
19     }
20 };
21 queue<int> Q;
22 priority_queue<int,vector<int>,cmp> pQ;
23 inline void add_edge(int from,int to,int flow)
24 {
25     tot+=2;
26     v[tot+1]=from;v[tot]=to;w[tot]=flow;w[tot+1]=0;
27     next[tot]=first[from];first[from]=tot;
28     next[tot+1]=first[to];first[to]=tot+1;
29     return;
30 }
31 inline bool bfs()
32 {
33     int now;
34     register int go;
35     memset(h+1,0x3f,sizeof(int)*n);
36     h[t]=0;Q.push(t);
37     while(!Q.empty())
38     {
39         now=Q.front();Q.pop();
40         for(go=first[now];go;go=next[go])
41             if(w[go^1]&&h[v[go]]>h[now]+1)
42                 h[v[go]]=h[now]+1,Q.push(v[go]);
43     }
44     return h[s]!=inf;
45 }
46 inline void push(int now)//推送
47 {
48     int d;
49     register int go;
50     for(go=first[now];go;go=next[go])
51         if(w[go]&&h[v[go]]+1==h[now])
52         {
53             d=min(e[now],w[go]);
54             w[go]-=d;w[go^1]+=d;e[now]-=d;e[v[go]]+=d;
55             if(v[go]!=s&&v[go]!=t&&!inq[v[go]])
56                 pQ.push(v[go]),inq[v[go]]=1;
57             if(!e[now])//已经推送完毕可以直接退出
58                 break;
59         }
60     return;
61 }
62 inline void relabel(int now)//重贴标签
63 {
64     register int go;
65     h[now]=inf;
66     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
94             ↪ 高的所以修改的节点一定不在优先队列中,不
95             ↪ 必担心修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",&n,&m,&s,&t);
110     while(m--)
111     {
112         scanf("%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] =
8     ↪ {0}, 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  $O(n \cdot \text{sigma})$ 
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
28                                     ↪ 列
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 SA-IS

```

1 // 注意求完的SA有效位只有1~n, 但它是0-based, 如果其他部
   ↪ 分是1-based记得+1再用
2
3 constexpr int maxn = 100005, l_type = 0, s_type = 1;
4
5 // 判断一个字符是否为LMS字符
6 bool is_lms(int *tp, int x) {
7     return x > 0 && tp[x] == s_type && tp[x - 1] ==
   ↪ l_type;
8 }
9
10 // 判断两个LMS子串是否相同
11 bool equal_substr(int *s, int x, int y, int *tp) {
12     do {
13         if (s[x] != s[y])
14             return false;
15         x++;
16         y++;
17     } while (!is_lms(tp, x) && !is_lms(tp, y));
18
19     return s[x] == s[y];
20 }
21
22 // 诱导排序(从*型诱导到L型,从L型诱导到S型)
23 // 调用之前应将*型按要求放入SA中

```

```

24 void induced_sort(int *s, int *sa, int *tp, int *buc, int
   ↪ *lbuc, int *sbuc, int n, int m) {
25     for (int i = 0; i <= n; i++)
26         if (sa[i] > 0 && tp[sa[i] - 1] == l_type)
27             sa[lbuc[s[sa[i] - 1]]++] = sa[i] - 1;
28
29     for (int i = 1; i <= m; i++)
30         sbuc[i] = buc[i] - 1;
31
32     for (int i = n; ~i; i--)
33         if (sa[i] > 0 && tp[sa[i] - 1] == s_type)
34             sa[sbuc[s[sa[i] - 1]]--] = sa[i] - 1;
35 }
36
37 // s是输入字符串, n是字符串的长度, m是字符集的大小
38 int *sais(int *s, int len, int m) {
39     int n = len - 1;
40
41     int *tp = new int[n + 1];
42     int *pos = new int[n + 1];
43     int *name = new int[n + 1];
44     int *sa = new int[n + 1];
45     int *buc = new int[m + 1];
46     int *lbuc = new int[m + 1];
47     int *sbuc = new int[m + 1];
48
49     memset(buc, 0, sizeof(int) * (m + 1));
50
51     for (int i = 0; i <= n; i++)
52         buc[s[i]]++;
53
54     for (int i = 1; i <= m; i++) {
55         buc[i] += buc[i - 1];
56
57         lbuc[i] = buc[i - 1];
58         sbuc[i] = buc[i] - 1;
59     }
60
61     tp[n] = s_type;
62     for (int i = n - 1; ~i; i--) {
63         if (s[i] < s[i + 1])
64             tp[i] = s_type;
65         else if (s[i] > s[i + 1])
66             tp[i] = l_type;
67         else
68             tp[i] = tp[i + 1];
69     }
70
71     int cnt = 0;
72     for (int i = 1; i <= n; i++)
73         if (tp[i] == s_type && tp[i - 1] == l_type)
74             pos[cnt++] = i;
75
76     memset(sa, -1, sizeof(int) * (n + 1));
77     for (int i = 0; i < cnt; i++)
78         sa[sbuc[s[pos[i]]]--] = pos[i];
79     induced_sort(s, sa, tp, buc, lbuc, sbuc, n, m);
80
81     memset(name, -1, sizeof(int) * (n + 1));
82     int lastx = -1, namecnt = 1;
83     bool flag = false;
84
85     for (int i = 1; i <= n; i++) {
86         int x = sa[i];
87
88         if (is_lms(tp, x)) {
89             if (lastx >= 0 && !equal_substr(s, x, lastx,
   ↪ tp))
90                 namecnt++;

```

```

91         if (lastx >= 0 && namecnt == name[lastx])
92             flag = true;
93     }
94     name[x] = namecnt;
95     lastx = x;
96 }
97 name[n] = 0;
98
99 int *t = new int[cnt];
100 int p = 0;
101 for (int i = 0; i <= n; i++)
102     if (name[i] >= 0)
103         t[p++] = name[i];
104
105 int *tsa;
106 if (!flag) {
107     tsa = new int[cnt];
108
109     for (int i = 0; i < cnt; i++)
110         tsa[t[i]] = i;
111 }
112 else
113     tsa = sais(t, cnt, namecnt);
114
115 lbuc[0] = sbuc[0] = 0;
116 for (int i = 1; i <= m; i++) {
117     lbuc[i] = buc[i - 1];
118     sbuc[i] = buc[i] - 1;
119 }
120
121 memset(sa, -1, sizeof(int) * (n + 1));
122 for (int i = cnt - 1; ~i; i--)
123     sa[sbuc[s[pos[tsa[i]]]]--] = pos[tsa[i]];
124 induced_sort(s, sa, tp, buc, lbuc, sbuc, n, m);
125
126 return sa;
127 }
128
129 // O(n)求height数组, 注意是sa[i]与sa[i - 1]的LCP
130 void get_height(int *s, int *sa, int *rnk, int *height,
131     ↪ int n) {
132     for (int i = 0; i <= n; i++)
133         rnk[sa[i]] = i;
134
135     int k = 0;
136     for (int i = 0; i <= n; i++) {
137         if (!rnk[i])
138             continue;
139
140         if (k)
141             k--;
142
143         while (s[sa[rnk[i]] + k] == s[sa[rnk[i] - 1] +
144             ↪ k])
145             k++;
146
147         height[rnk[i]] = k;
148     }
149 }
150
151 char str[maxn];
152 int n, s[maxn], sa[maxn], rnk[maxn], height[maxn];
153
154 // 方便起见附上主函数
155 int main() {
156     scanf("%s", str);
157     n = strlen(str);
158     str[n] = '$';

```

```

159     for (int i = 0; i <= n; i++)
160         s[i] = str[i];
161
162     memcpy(sa, sais(s, n + 1, 256), sizeof(int) * (n +
163         ↪ 1));
164
165     get_height(s, sa, rnk, height, n);
166
167     return 0;
168 }

```

2.3 后缀自动机

```

1 // 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
2   ↪ 者哈希表替换
3 // 注意!!!结点数要开成串长的两倍
4 // 全局变量与数组定义
5 int last, val[maxn], par[maxn], go[maxn][26], cnt;
6 int c[maxn], q[maxn]; // 用来桶排序
7
8 // 在主函数开头加上这句初始化
9 last = cnt = 1;
10
11 // 以下是按val进行桶排序的代码
12 for (int i = 1; i <= cnt; i++)
13     c[val[i] + 1]++;
14 for (int i = 1; i <= n; i++)
15     c[i] += c[i - 1]; // 这里n是串长
16 for (int i = 1; i <= cnt; i++)
17     q[++c[val[i]]] = i;
18
19 //加入一个字符 均摊O(1)
20 void extend(int c) {
21     int p = last, np = ++cnt;
22     val[np] = val[p] + 1;
23
24     while (p && !go[p][c]) {
25         go[p][c] = np;
26         p = par[p];
27     }
28
29     if (!p)
30         par[np] = 1;
31     else {
32         int q = go[p][c];
33
34         if (val[q] == val[p] + 1)
35             par[np] = q;
36         else {
37             int nq = ++cnt;
38             val[nq] = val[p] + 1;
39             memcpy(go[nq], go[q], sizeof(go[q]));
40
41             par[nq] = par[q];
42             par[np] = par[q] = nq;
43
44             while (p && go[p][c] == q){
45                 go[p][c] = nq;
46                 p = par[p];
47             }
48         }
49     }
50
51     last = np;
52 }

```

2.4 回文树

```

1 //定理:一个字符串本质不同的回文子串个数是 $O(n)$ 的
2 //注意回文树只需要开一倍结点,另外结点编号也是一个可用
  ↳ 的bfs序
3
4 //全局数组定义
5 int val[maxn], par[maxn], go[maxn][26], last, cnt;
6 char s[maxn];
7
8 //重要!在主函数最前面一定要加上以下初始化
9 par[0]=cnt=1;
10 val[1]=-1;
11 //这个初始化和广义回文树不一样,写普通题可以用,广义回文树
  ↳ 就不要乱搞了
12
13 //extend函数 均摊 $O(1)$ 
14 //向后扩展一个字符
15 //传入对应下标
16 void extend(int n){
17     int p=last, c=s[n]-'a';
18     while(s[n-val[p]-1]!=s[n]) p=par[p];
19     if(!go[p][c]){
20         int q=++cnt, now=p;
21         val[q]=val[p]+2;
22         do p=par[p]; while(s[n-val[p]-1]!=s[n]);
23         par[q]=go[p][c];
24         last=go[now][c]=q;
25     }
26     else last=go[p][c];
27     a[last]++;
28 }

```

2.4.1 广义回文树

(代码是梯子剖分的版本,压力不大的题目换成直接倍增就好了,常数只差不到一倍)

```

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],
  ↳ pam_last[maxn], pam_cnt;
8 int weight[maxn], pow_26[maxn];
9
10 int trie[maxn][26], trie_cnt, d[maxn], mxd[maxn],
  ↳ son[maxn], top[maxn], len[maxn], sum[maxn];
11 char chr[maxn];
12 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
75             ↳ 延长一倍
76             v[x].push_back(u);
77             u = f[0][u];
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'] =
125                 par[q];
126
127         if (val[q] <= n)
128             weight[q] = (weight[par[q]] + (long long)(n -
129                 val[q] + 1) * pow_26[n - val[q]]) % mod;
130         else
131             weight[q] = weight[par[q]];
132     }
133     else
134         new_last = go[p][c];
135
136     pam_last[x] = new_last;
137
138     return weight[pam_last[x]];
139 }
140
141 void bfs() {
142
143     queue<int> q;
144
145     q.push(1);
146
147     while (!q.empty()) {
148         int x = q.front();
149         q.pop();
150
151         sum[x] = sum[f[0][x]];
152         if (x > 1)
153             sum[x] = (sum[x] + extend(x)) % mod;
154
155         for (int i : queries[x])
156             ans[i] = sum[x];
157
158         for (int i = 0; i < 26; i++)
159             if (trie[x][i])
160                 q.push(trie[x][i]);
161     }
162 }
163
164 int main() {
165
166     pow_26[0] = 1;
167     log_tbl[0] = -1;
168
169     for (int i = 1; i <= 1000000; i++) {
170         pow_26[i] = 26ll * pow_26[i - 1] % mod;
171         log_tbl[i] = log_tbl[i / 2] + 1;
172     }
173
174     int T;
175     scanf("%d", &T);
176
177     while (T--) {
178
179         trie_cnt = 1;
180         chr[1] = '#';
181
182         int last = 1;
183         for (char *c = str; *c; c++)
184             last = add(last, *c - 'a');
185
186         queries[last].push_back(0);
187
188         for (int i = 1; i <= m; i++) {
189             int op;
190             scanf("%d", &op);
191
192             if (op == 1) {
193                 char c;
194                 scanf(" %c", &c);
195
196                 last = add(last, c - 'a');
197             }
198             else
199                 last = del(last);
200
201             queries[last].push_back(i);
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         }

```



```

244     pam_cnt = 0;
245 }
246 }
247
248     return 0;
249 }

```

2.5 Manacher马拉车

```

1 //n为串长,回文半径输出到p数组中
2 //数组要开串长的两倍
3 void manacher(const char *t, int n) {
4     static char s[maxn * 2];
5
6     for (int i = n; i; i--)
7         s[i * 2] = t[i];
8     for (int i = 0; i <= n; i++)
9         s[i * 2 + 1] = '#';
10
11     s[0] = '$';
12     s[(n + 1) * 2] = '\0';
13     n = n * 2 + 1;
14
15     int mx = 0, j = 0;
16
17     for (int i = 1; i <= n; i++) {
18         p[i] = (mx > i ? min(p[j * 2 - i], mx - i) : 1);
19         while (s[i - p[i]] == s[i + p[i]])
20             p[i]++;
21
22         if (i + p[i] > mx) {
23             mx = i + p[i];
24             j = i;
25         }
26     }
27 }

```

2.6 KMP

2.6.1 ex-KMP

```

1 //全局变量与数组定义
2 char s[maxn], t[maxn];
3 int n, m, a[maxn];
4
5 //主过程 O(n + m)
6 //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
7 //在nx中
8 //0-based,s的长度是m,t的长度是n
9 void exKMP(const char *s, const char *t, int *a) {
10     static int nx[maxn];
11
12     memset(nx, 0, sizeof(nx));
13
14     int j = 0;
15     while (j + 1 < m && s[j] == s[j + 1])
16         j++;
17     nx[1] = j;
18
19     for (int i = 2, k = 1; i < m; i++) {
20         int pos = k + nx[k], len = nx[i - k];
21
22         if (i + len < pos)
23             nx[i] = len;
24         else {
25             j = max(pos - i, 0);
26             while (i + j < m && s[j] == s[i + j])
27                 j++;
28         }
29     }
30 }

```

```

27     nx[i] = j;
28     k = i;
29 }
30 }
31 }
32
33 j = 0;
34 while (j < n && j < m && s[j] == t[j])
35     j++;
36 a[0] = j;
37
38 for (int i = 1, k = 0; i < n; i++) {
39     int pos = k + a[k], len = nx[i - k];
40     if (i + len < pos)
41         a[i] = len;
42     else {
43         j = max(pos - i, 0);
44         while (j < m && i + j < n && s[j] == t[i + j])
45             j++;
46
47         a[i] = j;
48         k = i;
49     }
50 }
51 }

```

3. 数学

3.1 插值

3.1.1 牛顿插值

牛顿插值的原理是二项式反演.

二项式反演:

$$f(n) = \sum_{k=0}^n \binom{n}{k} g(k) \Leftrightarrow g(n) = \sum_{k=0}^n (-1)^{n-k} \binom{n}{k} f(k)$$

可以用 e^x 和 e^{-x} 的麦克劳林展开式证明.

套用二项式反演的结论即可得到牛顿插值:

$$f(n) = \sum_{i=0}^k \binom{n}{i} r_i$$

$$r_i = \sum_{j=0}^i (-1)^{i-j} \binom{i}{j} f(j)$$

其中 k 表示 $f(n)$ 的最高次项系数.

实现时可以用 k 次差分替代右边的式子:

```

1 for (int i = 0; i <= k; i++)
2     r[i] = f(i);
3 for (int j = 0; j < k; j++)
4     for (int i = k; i > j; i--)
5         r[i] -= r[i - 1];

```

注意到预处理 r_i 的式子满足卷积形式,必要时可以用FFT优化至 $O(k \log k)$ 预处理.

3.1.2 拉格朗日插值

$$f(x) = \sum_i f(x_i) \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

3.2 多项式

3.2.1 FFT

```

1 //使用时一定要注意double的精度是否足够(极限大概是10^14)
2
3 const double pi=acos((double)-1.0);
4
5 //手写复数类
6 //支持加减乘三种运算
7 //+=运算符如果用的不多可以不重载
8 struct Complex{
9     double a,b;//由于long double精度和double几乎相同,通常
10     ↪ 没有必要用long double
11     Complex(double a=0.0,double b=0.0):a(a),b(b){}
12     Complex operator+(const Complex &x)const{return
13     ↪ Complex(a+x.a,b+x.b);}
14     Complex operator-(const Complex &x)const{return
15     ↪ Complex(a-x.a,b-x.b);}
16     Complex operator*(const Complex &x)const{return
17     ↪ Complex(a*x.a-b*x.b,a*x.b+b*x.a);}
18     Complex &operator+=(const Complex &x){return
19     ↪ *this=*this+x;}
20 }w[maxn],w_inv[maxn];
21
22 //FFT初始化 O(n)
23 //需要调用sin, cos函数
24 void FFT_init(int n){
25     for(int i=0;i<n;i++)//根据单位根的旋转性质可以节省计
26     ↪ 算单位根逆元的时间
27     w[i]=w_inv[n-i-1]=Complex(cos(2*pi/n*i),sin(2*pi/n*i));
28     //当然不存单位根也可以,只不过在FFT次数较多时很可能会
29     ↪ 增大常数
30 }
31
32 //FFT主过程 O(n\log n)
33 void FFT(Complex *A,int n,int tp){
34     for(int i=1,j=0,k;i<n-1;i++){
35         k=n;
36         do j^=(k>>=1);while(j<k);
37         if(i<j)swap(A[i],A[j]);
38     }
39     for(int k=2;k<=n;k<=&=1)
40         for(int i=0;i<n;i+=k)
41             for(int j=0;j<(k>>1);j++){
42                 Complex a=A[i+j], b = (tp>0?w:w_inv)[n /
43                 ↪ k * j] * A[i + j + (k / 2)];
44                 A[i+j]=a+b;
45                 A[i+j+(k>>1)]=a-b;
46             }
47     if(tp<0)for(int i=0;i<n;i++)A[i].a/=n;
48 }

```

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
7     ↪ 原根
8 void NTT(int *A, int n, int tp) { // n为变换长度
9     ↪ tp为1或-1表示正/逆变换
10     for (int i = 1, j = 0, k; i < n - 1; i++) { // O(n)旋
11     ↪ 转算法原理是模拟二进制加一
12         k = n;
13         do
14             j ^= (k >>= 1);
15         while(j < k);
16         if(i < j) swap(A[i], A[j]);
17     }
18     for(int k=2;k<=n;k<=&=1)
19         for(int i=0;i<n;i+=k)
20             for(int j=0;j<(k>>1);j++){
21                 Complex a=A[i+j], b = (tp>0?w:w_inv)[n /
22                 ↪ k * j] * A[i + j + (k / 2)];
23                 A[i+j]=a+b;
24                 A[i+j+(k>>1)]=a-b;
25             }
26     if(tp<0)for(int i=0;i<n;i++)A[i].a/=n;
27 }

```

```

13 while (j < k);
14
15 if(i < j)
16     swap(A[i], A[j]);
17 }
18
19 for (int k = 2; k <= n; k <=&= 1) {
20     int wn = qpow(g, (tp > 0 ? (p - 1) / k : (p - 1)
21     ↪ / k * (long long)(p - 2) % (p - 1)));
22     for (int i = 0; i < n; i += k) {
23         int w = 1;
24         for (int j = 0; j < (k >> 1); j++, w = (long
25         ↪ long)w * wn % p){
26             int a = A[i + j], b = (long long)w * A[i
27             ↪ + j + (k >> 1)] % p;
28             A[i + j] = (a + b) % p;
29             A[i + j + (k >> 1)] = (a - b + p) % p;
30         } // 更好的写法是预处理单位根的次幂参
31         ↪ 照FFT的代码
32     }
33 }
34
35 if (tp < 0) {
36     int inv = qpow(n, p - 2); // 如果预处理过逆元的话
37     ↪ 就不用快速幂了
38     for (int i = 0; i < n; i++)
39         A[i] = (long long)A[i] * inv % p;
40 }
41 }

```

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
4     ↪ 写任意模数卷积了
5 //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
6     ↪ 些
7 //但在答案不超过10^18时可以改成双模数NTT,这时就比拆系
8     ↪ 数FFT快一些了
9
10 //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
11     ↪ 后中国剩余定理合并
12 //以对23333333(不是质数)取模为例
13 const int
14     ↪ maxn=262200,Mod=23333333,g=3,m[]={998244353,1004535809,10454
15     ↪ m0_inv=669690699,m1_inv=332747959,M_inv=942377029;//这
16     ↪ 三个模数最小原根都是3
17 const long long M=(long long)m[0]*m[1];
18
19 //主函数(当然更多时候包装一下比较好)
20 //用来卷积的是A和B
21 //需要调用mul
22 int n,N=1,A[maxn],B[maxn],C[maxn],D[maxn],ans[3][maxn];
23 int main(){
24     scanf("%d",&n);
25     while(N<(n<<1))N<=&=1;
26     for(int i=0;i<n;i++)scanf("%d",&A[i]);
27     for(int i=0;i<n;i++)scanf("%d",&B[i]);
28     for(int i=0;i<3;i++)mul(m[i],ans[i]);
29     for(int i=0;i<n;i++)printf("%d ",China(ans[0]
30     ↪ [i],ans[1][i],ans[2][i]));
31     return 0;
32 }
33
34 //mul O(n\log n)
35 //包装了模NTT模数的卷积
36 //需要调用NTT
37 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)
44         +mul((long long)a1*m0_inv,m[0],M))%M;
45     int k=((a2-A)%m[2]+m[2])%m[2]*M_inv%m[2];
46     return (k%M*(M%M)%Mod+A%M)%Mod;
47 }
48
49 //-----分割
50 //线-----
51 //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
52 //仍然以模233333333为例
53 const int maxn=262200,p=23333333,M=4830;//M取值要使得结果
54 //不超过 $10^{14}$ 
55 //需要开的数组
56 struct Complex{//内容略
57 }w[maxn],w_inv[maxn],A[maxn],B[maxn],C[maxn],D[maxn],F[maxn],G[maxn],H[maxn];
58
59 //主函数(当然更多时候包装一下比较好)
60 //需要调用FFT初始化,FFT
61 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)
90             (F[i].a+0.5)%p)%p+
91             M*((long long)(G[i].a+0.5)%p)%p+(long long)
92             (H[i].a+0.5)%p)%p));
93     return 0;
94 }

```

3.2.4 多项式操作

```

1 //以下所有代码均为NTT版本
2 //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
3 //长度
4 //多项式求逆  $O(n\log n)$ 
5 //要求A常数项不为0
6 void getinv(int *A, int *C, int n) {
7     static int B[maxn];
8
9     memset(C, 0, sizeof(int) * (n * 2));
10    C[0] = qpow(A[0],p - 2); // 一般常数项都是1,直接赋值
11    // 为1就可以
12
13    for (int k = 2; k <= n; k <= 1) {
14        memcpy(B, A, sizeof(int) * k);
15        memset(B + k, 0, sizeof(int) * k);
16
17        NTT(B, k * 2, 1);
18        NTT(C, k * 2, 1);
19
20        for (int i = 0; i < k * 2; i++) {
21            C[i] = (2 - (long long)B[i] * C[i]) % p *
22                // C[i] % p;
23            if (C[i] < 0)
24                C[i] += p;
25        }
26        NTT(C, k * 2, -1);
27        memset(C + k, 0, sizeof(int) * k);
28    }
29 }
30
31 //多项式开根  $O(n\log n)$ 
32 //要求A常数项可以开根/存在二次剩余
33 //需要调用多项式求逆,且需要预处理2的逆元
34 void getsqrt(int *A,int *C,int n){
35     static int B[maxn],D[maxn];
36     memset(C,0,sizeof(int)*(n<<1));
37     C[0]=1; // 如果不是1就要考虑二次剩余
38     for(int k=2;k<=n;k<=1){
39         memcpy(B,A,sizeof(int)*k);
40         memset(B+k,0,sizeof(int)*k);
41         getinv(C,D,k);
42         NTT(B,k<<1,1);
43         NTT(D,k<<1,1);
44         for(int i=0;i<(k<<1);i++)B[i]=(long
45             long)B[i]*D[i]%p;
46         NTT(B,k<<1,-1);
47         for(int i=0;i<k;i++)C[i]=(long long)
48             (C[i]+B[i])*inv_2%p;//inv_2是2的逆元
49     }
50 }
51
52 //求导  $O(n)$ 
53 void getderivative(int *A,int *C,int n){
54     for(int i=1;i<n;i++)C[i-1]=(long long)A[i]*i%p;
55     C[n-1]=0;
56 }
57
58 //不定积分  $O(n\log n)$ ,如果预处理过逆元可以降到 $O(n)$ 
59 void getintegrate(int *A,int *C,int n){
60     for(int i=1;i<n;i++)C[i]=(long
61         long)A[i-1]*qpow(i,p-2)%p;
62     C[0]=0;//由于是不定积分,结果没有常数项
63 }
64
65 //多项式ln  $O(n\log n)$ 

```

```

63 //要求A常数项不为0/存在离散对数
64 //需要调用多项式求逆/求导/不定积分
65 void getln(int *A,int *C,int n){//通常情况下A常数项都是1
66     static int B[maxn];
67     getderivative(A,B,n);
68     memset(B+n,0,sizeof(int)*n);
69     getinv(A,C,n);
70     NTT(B,n<<1,1);
71     NTT(C,n<<1,1);
72     for(int i=0;i<(n<<1);i++)B[i]=(long long)B[i]*C[i]%p;
73     NTT(B,n<<1,-1);
74     getintegrate(B,C,n);
75     memset(C+n,0,sizeof(int)*n);
76 }
77
78 //多项式\exp O(n\log n)
79 //要求A没有常数项
80 //需要调用多项式\ln
81 //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
82 //分治FFT依据:设G(x)=\exp F(x),则有g_i=\sum_{k=1}^i f_k
    ↳ g_{i-k}
83 void getexp(int *A,int *C,int n){
84     static int B[maxn];
85     memset(C,0,sizeof(int)*(n<<1));
86     C[0]=1;
87     for(int k=2;k<=n;k<=1){
88         getln(C,B,k);
89         for(int i=0;i<k;i++){
90             B[i]=A[i]-B[i];
91             if(B[i]<0)B[i]+=p;
92         }
93         (++B[0])%=p;
94         NTT(B,k<<1,1);
95         NTT(C,k<<1,1);
96         for(int i=0;i<(k<<1);i++)C[i]=(long
            ↳ long)C[i]*B[i]%p;
97         NTT(C,k<<1,-1);
98         memset(C+k,0,sizeof(int)*k);
99     }
100 }
101
102 //多项式k次幂 O(n\log n)
103 //在A常数项不为1时需要转化
104 //需要调用多项式/exp、\ln
105 //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
106 void getpow(int *A,int *C,int n,int k){
107     static int B[maxn];
108     getln(A,B,n);
109     for(int i=0;i<n;i++)B[i]=(long long)B[i]*k%p;
110     getexp(B,C,n);
111 }

```

3.2.5 拉格朗日反演

用于求复合逆.

如果 $f(x)$ 与 $g(x)$ 互为复合逆 则有

$$[x^n]g(x) = \frac{1}{n}[x^{n-1}] \left(\frac{x}{f(x)} \right)^n$$

$$[x^n]h(g(x)) = \frac{1}{n}[x^{n-1}]h'(x) \left(\frac{x}{f(x)} \right)^n$$

3.2.6 半在线卷积

```

1 void solve(int l, int r) {
2     if (r <= m)
3         return;
4

```

```

5     if (r - l == 1) {
6         if (l == m)
7             f[l] = a[m];
8         else
9             f[l] = (long long)f[l] * inv[l - m] % p;
10
11         for (int i = l, t = (long long)l * f[l] % p; i <=
            ↳ n; i += 1)
12             g[i] = (g[i] + t) % p;
13
14         return;
15     }
16
17     int mid = (l + r) / 2;
18
19     solve(l, mid);
20
21     if (l == 0) {
22         for (int i = 1; i < mid; i++) {
23             A[i] = f[i];
24             B[i] = (c[i] + g[i]) % p;
25         }
26         NTT(A, r, 1);
27         NTT(B, r, 1);
28         for (int i = 0; i < r; i++)
29             A[i] = (long long)A[i] * B[i] % p;
30         NTT(A, r, -1);
31
32         for (int i = mid; i < r; i++)
33             f[i] = (f[i] + A[i]) % p;
34     }
35     else {
36         for (int i = 0; i < r - l; i++)
37             A[i] = f[i];
38         for (int i = l; i < mid; i++)
39             B[i - l] = (c[i] + g[i]) % p;
40         NTT(A, r - l, 1);
41         NTT(B, r - l, 1);
42         for (int i = 0; i < r - l; i++)
43             A[i] = (long long)A[i] * B[i] % p;
44         NTT(A, r - l, -1);
45
46         for (int i = mid; i < r; i++)
47             f[i] = (f[i] + A[i - l]) % p;
48
49         memset(A, 0, sizeof(int) * (r - l));
50         memset(B, 0, sizeof(int) * (r - l));
51
52         for (int i = l; i < mid; i++)
53             A[i - l] = f[i];
54         for (int i = 0; i < r - l; i++)
55             B[i] = (c[i] + g[i]) % p;
56         NTT(A, r - l, 1);
57         NTT(B, r - l, 1);
58         for (int i = 0; i < r - l; i++)
59             A[i] = (long long)A[i] * B[i] % p;
60         NTT(A, r - l, -1);
61
62         for (int i = mid; i < r; i++)
63             f[i] = (f[i] + A[i - l]) % p;
64     }
65
66     memset(A, 0, sizeof(int) * (r - l));
67     memset(B, 0, sizeof(int) * (r - l));
68
69     solve(mid, r);
70 }

```

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 BCOGS 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++){
24                 if(tp>0) A[i+j] = (A[i+j] + A[i+j+(k>>1)])%p;
25                 else A[i+j] = (A[i+j] - A[i+j+(k>>1)]+p)%p;
26             }
27 }
28
29 //按位异或版本
30 void FWT_xor(int *A, int n, int tp){
31     for(int k=2; k<=n; k<<=1)
32         for(int i=0; i<n; i+=k)
33             for(int j=0; j<(k>>1); j++){
34                 int a=A[i+j], b=A[i+j+(k>>1)];
35                 A[i+j] = (a+b)%p;
36                 A[i+j+(k>>1)] = (a-b+p)%p;
37             }
38     if(tp<0){
39         int inv=qpow(n%p, p-2); //n的逆元,在不取模时需要用
40         ↪ 每层除以2代替
41         for(int i=0; i<n; i++) A[i] = A[i]*inv%p;
42     }
43 }

```

3.4 单纯形

```

1 const double eps = 1e-10;
2
3 double A[maxn][maxn], x[maxn];
4 int n, m, t, id[maxn * 2];
5
6 // 方便起见,这里附上主函数
7 int main() {
8     scanf("%d%d%d", &n, &m, &t);
9
10     for (int i = 1; i <= n; i++) {
11         scanf("%lf", &A[0][i]);
12         id[i] = i;
13     }
14
15     for (int i = 1; i <= m; i++) {
16         for (int j = 1; j <= n; j++)
17             scanf("%lf", &A[i][j]);
18
19         scanf("%lf", &A[i][0]);
20     }
21 }

```

```

21
22 if (!inititalize())
23     printf("Infeasible"); // 无解
24 else if (!simplex())
25     printf("Unbounded"); // 最优解无限大
26
27 else {
28     printf("%.15lf\n", -A[0][0]);
29     if (t) {
30         for (int i = 1; i <= m; i++)
31             x[id[i + n]] = A[i][0];
32         for (int i = 1; i <= n; i++)
33             printf("%.15lf ", x[i]);
34     }
35 }
36 return 0;
37
38 //初始化
39 //对于初始解可行的问题,可以把初始化省略掉
40 bool inititalize() {
41     while (true) {
42         double t = 0.0;
43         int l = 0, e = 0;
44
45         for (int i = 1; i <= m; i++)
46             if (A[i][0] + eps < t) {
47                 t = A[i][0];
48                 l = i;
49             }
50
51         if (!l)
52             return true;
53
54         for (int i = 1; i <= n; i++)
55             if (A[l][i] < -eps && (!e || id[i] < id[e]))
56                 e = i;
57
58         if (!e)
59             return false;
60
61         pivot(l, e);
62     }
63 }
64
65 //求解
66 bool simplex(){
67     while (true) {
68         int l = 0, e = 0;
69         for (int i = 1; i <= n; i++)
70             if (A[0][i] > eps && (!e || id[i] < id[e]))
71                 e = i;
72
73         if (!e)
74             return true;
75
76         double t = 1e50;
77         for (int i = 1; i <= m; i++)
78             if (A[i][e] > eps && A[i][0] / A[i][e] < t) {
79                 l = i;
80                 t = A[i][0] / A[i][e];
81             }
82
83         if (!l)
84             return false;
85
86         pivot(l, e);
87     }
88 }
89 }

```

```

90 //转轴操作,本质是在凸包上沿着一条棱移动
91 void pivot(int l, int e) {
92     swap(id[e], id[n + 1]);
93     double t = A[l][e];
94     A[l][e] = 1.0;
95
96     for (int i = 0; i <= n; i++)
97         A[l][i] /= t;
98
99     for (int i = 0; i <= m; i++)
100         if (i != l) {
101             t = A[i][e];
102             A[i][e] = 0.0;
103             for (int j = 0; j <= n; j++)
104                 A[i][j] -= t * A[l][j];
105         }
106 }
107

```

3.5 线性代数

3.5.1 线性基

3.6 常见数列

3.6.1 伯努利数

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

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

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

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

4. 数论

4.1 $O(n)$ 预处理逆元

```

1 //要求p为质数
2 inv[0]=inv[1]=1;
3 for(int i=2;i<=n;i++)
4     inv[i]=(long long)(p-(p/i))*inv[p%i]%p;//p为模数
5 //i^{-1} \equiv -\lfloor \frac{p}{i} \rfloor \times (p \bmod i)^{-1} \pmod{p}
6 //i^{-1} = -(p/i) * (p%i)^{-1}
7

```

4.2 杜教筛

```

1 //用于求可以用狄利克雷卷积构造出好求和的东西的函数的前缀
2 //和(有点绕)
3 //有些题只要求n<=10^9,这时就没必要开Long Long了,但记得乘
4 //时强转
5 //常量/全局变量/数组定义
6 const int
7     maxn=5000005, table_size=5000000, p=1000000007, inv_2=(p+1)/2;
8 bool notp[maxn];
9 int prime[maxn/20], phi[maxn], tbl[100005];
10 //tbl用来顶替哈希表,其实开到n^{1/3}就够了,不过保险起见开
11 //成sqrt n比较好
12 long long N;
13

```

```

11 //主函数前面加上这么一句
12 memset(tbl, -1, sizeof(tbl));
13
14 //线性筛预处理部分略去
15
16 //杜教筛主过程 总计O(n^{2/3})
17 //递归调用自身
18 //递推式还需具体情况具体分析,这里以求欧拉函数前缀和(mod
19 //10^9+7)为例
20 int S(long long n){
21     if(n<=table_size)return phi[n];
22     else if(~tbl[N/n])return tbl[N/n];
23     //原理:n除以所有可能的数的结果一定互不相同
24     int ans=0;
25     for(long long i=2,last;i<=n;i=last+1){
26         last=n/(n/i);
27         ans=(ans+(last-i+1)%p*S(n/i))%p;//如果n是int范围
28         //的话记得强转
29     }
30     ans=(n%p*((n+1)%p)%p*inv_2-ans+p)%p;//同上
31     return tbl[N/n]=ans;
32 }
33

```

4.3 线性筛

```

1 //此代码以计算约数之和函数\sigma_1(对10^9+7取模)为例
2 //适用于任何f(p^k)便于计算的积性函数
3 const int p=1000000007;
4
5 int prime[maxn/10], sigma_one[maxn], f[maxn], g[maxn];
6 //f:除掉最小质因子后剩下的部分
7 //g:最小质因子的幂次,在f(p^k)比较复杂时很有用,但f(p^k)可
8 //以递推时就可以省了
9 //这里没有记录最小质因子,但根据线性筛的性质,每个合数只会
10 //被它最小的质因子筛掉
11 bool notp[maxn]; //顾名思义
12
13 void get_table(int n){
14     sigma_one[1]=1; //积性函数必有f(1)=1
15     for(int i=2;i<=n;i++){
16         if(!notp[i]){ //质数情况
17             prime[++prime[0]]=i;
18             sigma_one[i]=i+1;
19             f[i]=g[i]=1;
20         }
21         for(int j=1;j<=prime[0]&&i*prime[j]<=n;j++){
22             notp[i*prime[j]]=true;
23             if(i%prime[j]){ //加入一个新的质因子,这种情况
24                 //很简单
25                 sigma_one[i*prime[j]]=(long
26                     long)sigma_one[i]*(prime[j]+1)%p;
27                 f[i*prime[j]]=i;
28                 g[i*prime[j]]=1;
29             }
30             else{ //再加入一次最小质因子,需要再进行分类讨
31                 //论
32                 f[i*prime[j]]=f[i];
33                 g[i*prime[j]]=g[i]+1;
34                 //对于f(p^k)可以直接递推的函数,这里的判
35                 //断可以改成
36                 //i/prime[j]%prime[j]!=0,这样可以省
37                 //下f[]的空间
38                 //但常数很可能会稍大一些
39                 if(f[i]==1) //质数的幂次,这里\sigma_1可以
40                     //递推
41                     sigma_one[i*prime[j]]=(sigma_one[i]+i*prime[
42                         j])%p;
43                 //对于更一般的情况,可以借助g[]计
44                 //算f(p^k)
45             }
46         }
47     }
48 }
49

```



```

35         else sigma_one[i*prime[j]]//=否则直接利用
           ↳ 积性, 两半乘起来
36         (long
           ↳ long)sigma_one[i*prime[j]/f[i]]*sigma_one[f[i]]%p;
37         break;
38     }
39 }
40 }
41 }

```

```

21     while(a%2==0){
22         a>>=1;
23         k++;
24         sigma_one[f[i]]%=p;
25     long long t=qpow(b,a,n);//这里的快速幂函数需要
           ↳ 写O(1)快速乘
26     if(t==1||t==n-1)return true;
27     while(k--){
28         t=mul(t,t,n);//mul是O(1)快速乘函数
29         if(t==n-1)return true;
30     }
31     return false;
32 }

```

4.4 Miller-Rabin

```

1 //复杂度可以认为是常数
2 //封装好的函数体
3 //需要调用check
4 bool Miller_Rabin(long long n){
5     if(n==1)return false;
6     if(n==2)return true;
7     if(n%2==0)return false;
8     for(int i:{2,3,5,7,11,13,17,19,23,29,31,37}){
9         if(i>n)break;
10        if(!check(n,i))return false;
11    }
12    return true;
13 }
14 }
15 //用一个数检测
16 //需要调用Long Long快速幂和O(1)快速乘
17 bool check(long long n,long long b){//b是base
18     long long a=n-1;
19     int k=0;
20     while(a%2==0){
21         a>>=1;
22         k++;
23     }
24     long long t=qpow(b,a,n);//这里的快速幂函数需要
           ↳ 写O(1)快速乘
25     if(t==1||t==n-1)return true;
26     while(k--){
27         t=mul(t,t,n);//mul是O(1)快速乘函数
28         if(t==n-1)return true;
29     }
30     return false;
31 }
32 }

```

4.5 Pollard's Rho

```

1 //复杂度可以认为是常数
2 //封装好的函数体
3 //需要调用check
4 bool Miller_Rabin(long long n){
5     if(n==1)return false;
6     if(n==2)return true;
7     if(n%2==0)return false;
8     for(int i:{2,3,5,7,11,13,17,19,23,29,31,37}){
9         if(i>n)break;
10        if(!check(n,i))return false;
11    }
12    return true;
13 }
14 }
15 //用一个数检测
16 //需要调用Long Long快速幂和O(1)快速乘
17 bool check(long long n,long long b){//b是base
18     long long a=n-1;
19     int k=0;

```

5. 数据结构

5.1 线段树

5.1.1 主席树

参见GREAD07加强版

5.2 陈丹琦分治

```

1 // Division of Danqi Chen CDQ分治
2 // By AntiLeaf
3 // 通过题目B四维偏序
4 void CDQ1(int l,int r){
5     if(l>=r)return;
6     int mid=(l+r)>>1;
7     CDQ1(l,mid);CDQ1(mid+1,r);
8     int i=l,j=mid+1,k=l;
9     while(i<=mid&&j<=r){
10        if(a[i].x<a[j].x){
11            a[i].ins=true;
12            b[k++]=a[i++];
13        }
14        else{
15            a[j].ins=false;
16            b[k++]=a[j++];
17        }
18    }
19    while(i<=mid){
20        a[i].ins=true;
21        b[k++]=a[i++];
22    }
23    while(j<=r){
24        a[j].ins=false;
25        b[k++]=a[j++];
26    }
27    copy(b+l,b+r+1,a+l);
28    CDQ2(l,r);
29 }
30 void CDQ2(int l,int r){
31     if(l>=r)return;
32     int mid=(l+r)>>1;
33     CDQ2(l,mid);CDQ2(mid+1,r);
34     int i=l,j=mid+1,k=l;
35     while(i<=mid&&j<=r){
36        if(b[i].y<b[j].y){
37            if(b[i].ins)add(b[i].z,1);
38            t[k++]=b[i++];
39        }
40        else{
41            if(!b[j].ins)ans+=query(b[j].z-1);
42            t[k++]=b[j++];
43        }
44    }

```



```

45     }
46     while(i<=mid){
47         if(b[i].ins)add(b[i].z,1);
48         t[k++]=b[i++];
49     }
50     while(j<=r){
51         if(!b[j].ins)ans+=query(b[j].z-1);
52         t[k++]=b[j++];
53     }
54     for(i=1;i<=mid;i++)if(b[i].ins)add(b[i].z,-1);
55     copy(t+1,t+r+1,b+1);
56 }

```

5.3 Treap

```

1 //Treap Minimum Heap Version 小根堆版本
2 //By ysf
3 //通过题目普通平衡树
4
5 //注意相同键值可以共存
6
7 struct node{//结点类定义
8     int key,size,p;//分别为键值子树大小优先度
9     node *ch[2];//0表示左儿子1表示右儿子
10     node(int key=0):key(key),size(1),p(rand()){}
11     void refresh(){size=ch[0]->size+ch[1]->size+1;}//更新
12     ↳ 子树大小和附加信息
13 }null[maxn],*root=null,*ptr=null;//数组名叫做null是为了方便
14 ↳ 便开哨兵节点
15 //如果需要删除而空间不能直接开下所有结点则需要再写一个
16 ↳ 垃圾回收
17 //注意数组里的元素一定不能delete否则会导致RE
18
19 //重要
20 //在主函数最开始一定要加上以下预处理
21 null->ch[0]=null->ch[1]=null;
22 null->size=0;
23
24 //伪构造函数 O(1)
25 //为了方便在结点类外面再定义一个伪构造函数
26 node *newnode(int x){//键值为x
27     *++ptr=node(x);
28     ptr->ch[0]=ptr->ch[1]=null;
29     return ptr;
30 }
31
32 //插入键值 期望O(\log n)
33 //需要调用旋转
34 void insert(int x,node *&rt){//rt为当前结点建议调用时传
35     ↳ 入root下同
36     if(rt==null){
37         rt=newnode(x);
38         return;
39     }
40     int d=x>rt->key;
41     insert(x,rt->ch[d]);
42     rt->refresh();
43     if(rt->ch[d]->p<rt->p)rot(rt,d^1);
44 }
45
46 //删除一个键值 期望O(\log n)
47 //要求键值必须存在至少一个否则会导致RE
48 //需要调用旋转
49 void erase(int x,node *&rt){
50     if(x==rt->key){
51         if(rt->ch[0]!=null&&rt->ch[1]!=null){
52             int d=rt->ch[0]->p<rt->ch[1]->p;
53             rot(rt,d);
54             erase(x,rt->ch[d]);
55         }
56     }
57 }

```

```

52     else rt=rt->ch[rt->ch[0]==null];
53 }
54 else erase(x,rt->ch[x>rt->key]);
55 if(rt!=null)rt->refresh();
56 }
57
58 //求元素的排名严格小于键值的个数+1 期望O(\log n)
59 //非递归
60 int rank(int x,node *rt){
61     int ans=1,d;
62     while(rt!=null){
63         if((d=x>rt->key))ans+=rt->ch[0]->size+1;
64         rt=rt->ch[d];
65     }
66     return ans;
67 }
68
69 //返回排名第k从1开始的键值对应的指针 期望O(\log n)
70 //非递归
71 node *kth(int x,node *rt){
72     int d;
73     while(rt!=null){
74         if(x==rt->ch[0]->size+1)return rt;
75         if((d=x>rt->ch[0]->size))x-=rt->ch[0]->size+1;
76         rt=rt->ch[d];
77     }
78     return rt;
79 }
80
81 //返回前驱最大的比给定键值小的键值对应的指针 期望O(\log n)
82 //非递归
83 node *pred(int x,node *rt){
84     node *y=null;
85     int d;
86     while(rt!=null){
87         if((d=x>rt->key))y=rt;
88         rt=rt->ch[d];
89     }
90     return y;
91 }
92
93 //返回后继最小的比给定键值大的键值对应的指针 期望O(\log n)
94 //非递归
95 node *succ(int x,node *rt){
96     node *y=null;
97     int d;
98     while(rt!=null){
99         if((d=x<rt->key))y=rt;
100         rt=rt->ch[d^1];
101     }
102     return y;
103 }
104
105 //旋转Treap版本 O(1)
106 //平衡树基础操作
107 //要求对应儿子必须存在否则会导致后续各种莫名其妙的问题
108 void rot(node *&x,int d){//x为被转下去的结点会被修改以维护
109     ↳ 护树结构
110     node *y=x->ch[d^1];
111     x->ch[d^1]=y->ch[d];
112     y->ch[d]=x;
113     x->refresh();
114     (x=y)->refresh();
115 }

```

5.4 Splay

(参见LCT,除了splay()需要传一个点表示最终它的父亲,其他写法都和LCT相同)

5.5 树分治

5.5.1 动态树分治

```

1 //Dynamic Divide and Couquer on Tree 动态树分治  $O(n\log n)$ 
   $\hookrightarrow n)-O(\log n)$ 
2 //By ysf
3 //通过题目 PCOGS2278 树黑白
4
5 //为了减小常数这里采用bfs写法实测预处理比dfs快将近一半
6 //以下以维护一个点到每个黑点的距离之和为例
7
8 //全局数组定义
9 vector<int>G[maxn],W[maxn];
10 int size[maxn],son[maxn],q[maxn];
11 int p[maxn],depth[maxn],id[maxn][20],d[maxn][20]; //id是对
   $\hookrightarrow$  应层所在子树的根
12 int a[maxn],ca[maxn],b[maxn][20],cb[maxn][20]; //维护距离
   $\hookrightarrow$  和用的
13 bool vis[maxn]={false},col[maxn]={false};
14
15 //建树 总计 $O(n\log n)$ 
16 //需要调用找重心预处理距离同时递归调用自身
17 void build(int x,int k,int s,int pr){ //结点深度连通块大
   $\hookrightarrow$  小点分树上的父亲
18     x=getcenter(x,s);
19     vis[x]=true;
20     depth[x]=k;
21     p[x]=pr;
22     for(int i=0;i<(int)G[x].size();i++){
23         if(!vis[G[x][i]]){
24             d[G[x][i]][k]=W[x][i];
25             p[G[x][i]]=x;
26             getdis(G[x][i],k,G[x][i]);
27         }
28     }
29     for(int i=0;i<(int)G[x].size();i++){
30         if(!vis[G[x][i]])build(G[x][i],k+1,size[G[x]
   $\hookrightarrow$  [i]],x);
31     }
32 //找重心  $O(n)$ 
33 int getcenter(int x,int s){
34     int head=0,tail=0;
35     q[tail++]=x;
36     while(head!=tail){
37         x=q[head++];
38         size[x]=1;
39         son[x]=0;
40         for(int i=0;i<(int)G[x].size();i++){
41             if(!vis[G[x][i]]&&G[x][i]!=p[x]){
42                 p[G[x][i]]=x;
43                 q[tail++]=G[x][i];
44             }
45         }
46         for(int i=tail-1;i--){
47             x=q[i];
48             size[p[x]]+=size[x];
49             if(size[x]>size[son[p[x]]])son[p[x]]=x;
50         }
51         x=q[0];
52         while(son[x]&&(size[son[x]]<<1)>=s)x=son[x];
53         return x;
54     }
55 //预处理距离  $O(n)$ 

```

```

57 //方便起见这里直接用了笨一点的方法 $O(n\log n)$ 全存下来
58 void getdis(int x,int k,int rt){
59     int head=0,tail=0;
60     q[tail++]=x;
61     while(head!=tail){
62         x=q[head++];
63         size[x]=1;
64         id[x][k]=rt;
65         for(int i=0;i<(int)G[x].size();i++){
66             if(!vis[G[x][i]]&&G[x][i]!=p[x]){
67                 p[G[x][i]]=x;
68                 d[G[x][i]][k]=d[x][k]+W[x][i];
69                 q[tail++]=G[x][i];
70             }
71         }
72         for(int i=tail-1;i--){
73             size[p[q[i]]]+=size[q[i]];
74         }
75     }
76 //修改  $O(\log n)$ 
77 void modify(int x){
78     if(col[x])ca[x]--;
79     else ca[x]++; //记得先特判自己作为重心的那层
80     for(int u=p[x],k=depth[x]-1;u=p[u],k--){
81         if(col[x]){
82             a[u]-=d[x][k];
83             ca[u]--;
84             b[id[x][k]][k]-=d[x][k];
85             cb[id[x][k]][k]--;
86         }
87         else{
88             a[u]+=d[x][k];
89             ca[u]++;
90             b[id[x][k]][k]+=d[x][k];
91             cb[id[x][k]][k]++;
92         }
93     }
94     col[x]^=true;
95 }
96
97 //询问  $O(\log n)$ 
98 int query(int x){
99     int ans=a[x]; //特判自己是重心的那层
100     for(int u=p[x],k=depth[x]-1;u=p[u],k--){
101         ans+=a[u]-b[id[x][k]][k]+d[x][k]*(ca[u]-cb[id[x]
   $\hookrightarrow$  [k]][k]);
102     }
103     return ans;
104 }

```

5.5.2 紫荆花之恋

```

1 #include<cstdio>
2 #include<cstring>
3 #include<algorithm>
4 #include<vector>
5 using namespace std;
6 const int maxn=100010;
7 const double alpha=0.7;
8 struct node{
9     static int randint(){
10         static int
   $\hookrightarrow$  a=1213,b=97818217,p=998244353,x=751815431;
11         x=a*x+b;x%=p;
12         return x<0?(x+=p):x;
13     }
14     int data,size,p;
15     node *ch[2];
16     node(int d):data(d),size(1),p(randint()){

```

```

17 inline void refresh()
18     ↪ {size=ch[0]->size+ch[1]->size+1;}
19 }*null=new node(0),*root[maxn],*root1[maxn][50];
20 void addnode(int,int);
21 void rebuild(int,int,int,int);
22 void dfs_getcenter(int,int,int&);
23 void dfs_getdis(int,int,int,int);
24 void dfs_destroy(int,int);
25 void insert(int,node*&);
26 int order(int,node*);
27 void destroy(node*&);
28 void rot(node*&,int);
29 vector<int>G[maxn],W[maxn];
30 int size[maxn]={0},siz[maxn][50]={0},son[maxn];
31 bool vis[maxn];
32 int depth[maxn],p[maxn],d[maxn][50],id[maxn][50];
33 int n,m,w[maxn],tmp;
34 long long ans=0;
35 int main(){
36     freopen("flowera.in","r",stdin);
37     freopen("flowera.out","w",stdout);
38     null->size=0;
39     null->ch[0]=null->ch[1]=null;
40     scanf("%d%d",&n);
41     fill(vis,vis+n+1,true);
42     fill(root,root+n+1,null);
43     for(int i=0;i<n;i++)fill(root1[i],root1[i]+50,null);
44     scanf("%d%d%d",&w[1]);
45     insert(-w[1],root[1]);
46     size[1]=1;
47     printf("0\n");
48     for(int i=2;i<n;i++){
49         scanf("%d%d%d",&p[i],&tmp,&w[i]);
50         p[i]^=(ans%(int)1e9);
51         G[i].push_back(p[i]);
52         W[i].push_back(tmp);
53         G[p[i]].push_back(i);
54         W[p[i]].push_back(tmp);
55         addnode(i,tmp);
56         printf("%lld\n",ans);
57     }
58     return 0;
59 }
60 void addnode(int x,int z){//wj-dj>=di-wi
61     depth[x]=depth[p[x]]+1;
62     size[x]=1;
63     insert(-w[x],root[x]);
64     int rt=0;
65     for(int u=p[x],k=depth[p[x]];u=p[u],k--){
66         if(u==p[x]){
67             id[x][k]=x;
68             d[x][k]=z;
69         }
70         else{
71             id[x][k]=id[p[x]][k];
72             d[x][k]=d[p[x]][k]+z;
73         }
74         ans+=order(w[x]-d[x][k],root[u])-order(w[x]-d[x]
75             ↪ [k],root1[id[x][k]][k]);
76         insert(d[x][k]-w[x],root[u]);
77         insert(d[x][k]-w[x],root1[id[x][k]][k]);
78         size[u]++;
79         siz[id[x][k]][k]++;
80         if(siz[id[x][k]][k]>size[u]*alpha+5)rt=u;
81     }
82     id[x][depth[x]]=0;
83     d[x][depth[x]]=0;
84     if(rt){
85         dfs_destroy(rt,depth[rt]);
86     }
87     rebuild(rt,depth[rt],size[rt],p[rt]);
88 }
89 void rebuild(int x,int k,int s,int pr){
90     int u=0;
91     dfs_getcenter(x,s,u);
92     vis[x]=true;
93     p[x]=pr;
94     depth[x]=k;
95     size[x]=s;
96     d[x][k]=id[x][k]=0;
97     destroy(root[x]);
98     insert(-w[x],root[x]);
99     if(s<=1)return;
100     for(int i=0;i<(int)G[x].size();i++){if(!vis[G[x][i]]){
101         p[G[x][i]]=0;
102         d[G[x][i]][k]=W[x][i];
103         siz[G[x][i]][k]=p[G[x][i]][i]=0;
104         destroy(root1[G[x][i]][k]);
105         dfs_getdis(G[x][i],x,G[x][i],k);
106     }
107     for(int i=0;i<(int)G[x].size();i++){if(!vis[G[x]
108         ↪ [i]])rebuild(G[x][i],k+1,size[G[x][i]],x);
109 }
110 void dfs_getcenter(int x,int s,int &u){
111     size[x]=1;
112     son[x]=0;
113     for(int i=0;i<(int)G[x].size();i++){if(!vis[G[x]
114         ↪ [i]]&&G[x][i]!=p[x]){
115         p[G[x][i]]=x;
116         dfs_getcenter(G[x][i],s,u);
117         size[x]+=size[G[x][i]];
118         if(size[G[x][i]]>size[son[x]])son[x]=G[x][i];
119     }
120     if(!u||max(s-size[x],size[son[x]])<max(s-size[u],size[son[u]]))u=son[x];
121 }
122 void dfs_getdis(int x,int u,int rt,int k){
123     insert(d[x][k]-w[x],root[u]);
124     insert(d[x][k]-w[x],root1[rt][k]);
125     id[x][k]=rt;
126     siz[rt][k]++;
127     size[x]=1;
128     for(int i=0;i<(int)G[x].size();i++){if(!vis[G[x]
129         ↪ [i]]&&G[x][i]!=p[x]){
130         p[G[x][i]]=x;
131         d[G[x][i]][k]=d[x][k]+W[x][i];
132         dfs_getdis(G[x][i],u,rt,k);
133         size[x]+=size[G[x][i]];
134     }
135 }
136 void dfs_destroy(int x,int k){
137     vis[x]=false;
138     for(int i=0;i<(int)G[x].size();i++){if(depth[G[x]
139         ↪ [i]]>=k&&G[x][i]!=p[x]){
140         p[G[x][i]]=x;
141         dfs_destroy(G[x][i],k);
142     }
143 }
144 void insert(int x,node *&rt){
145     if(rt==null){
146         rt=new node(x);
147         rt->ch[0]=rt->ch[1]=null;
148         return;
149     }
150     int d=x>rt->data;
151     insert(x,rt->ch[d]);
152     rt->refresh();
153     if(rt->ch[d]->p<rt->p)rot(rt,d^1);
154 }

```

```

149 int order(int x,node *rt){
150     int ans=0,d;
151     x++;
152     while(rt!=null){
153         if((d=x>rt->data))ans+=rt->ch[0]->size+1;
154         rt=rt->ch[d];
155     }
156     return ans;
157 }
158 void destroy(node *&x){
159     if(x==null)return;
160     destroy(x->ch[0]);
161     destroy(x->ch[1]);
162     delete x;
163     x=null;
164 }
165 void rot(node *&x,int d){
166     node *y=x->ch[d^1];
167     x->ch[d^1]=y->ch[d];
168     y->ch[d]=x;
169     x->refresh();
170     (x=y)->refresh();
171 }

```

```

38 //把x的父亲设为y
39 //要求x必须为所在树的根节点,否则会导致后续各种莫名其妙的
    ↳ 问题
40 //需要调用splay
41 void link(node *x,node *y){
42     splay(x);
43     x->p=y;
44 }
45
46 //Cut 均摊O(\log n)
47 //把x与其父亲的连边断掉
48 //x可以是所在树的根节点,这时此操作没有任何实质效果
49 //需要调用access和splay
50 void cut(node *x){
51     access(x);
52     splay(x);
53     x->ch[0]->p=null;
54     x->ch[0]=null;
55     x->refresh();
56 }
57
58 //Splay 均摊O(\log n)
59 //需要调用旋转
60 void splay(node *x){
61     while(!isroot(x)){
62         if(isroot(x->p)){
63             rot(x->p,dir(x)^1);
64             break;
65         }
66         if(dir(x)==dir(x->p))rot(x->p->p,dir(x->p)^1);
67         else rot(x->p,dir(x)^1);
68         rot(x->p,dir(x)^1);
69     }
70 }
71
72 //旋转LCT版本O(1)
73 //平衡树基本操作
74 //要求对应儿子必须存在,否则会导致后续各种莫名其妙的的问题
75 void rot(node *x,int d){
76     node *y=x->ch[d^1];
77     y->p=x->p;
78     if(!isroot(x))x->p->ch[dir(x)]=y;
79     if((x->ch[d^1]=y->ch[d])!=null)y->ch[d]->p=x;
80     (y->ch[d]=x)->p=y;
81     x->refresh();
82     y->refresh();
83 }

```

5.6 LCT

5.6.1 不换根(弹飞绵羊)

```

1 //Link-Cut Trees without Changing Root LCT不换根版本
    ↳ O((n+m)\log n)
2 //By ysf
3 //通过题目,弹飞绵羊
4
5 //常数较大,请根据数据范围谨慎使用
6
7 #define isroot(x) ((x)!=(x)->p->ch[0]&&(x)!=(x)->p-
    ↳ >ch[1])//判断是不是Splay的根
8 #define dir(x) ((x)==(x)->p->ch[1])//判断它是它父亲的
    ↳ 左/右儿子
9
10 struct node{//结点类定义
11     int size;//Splay的子树大小
12     node *ch[2],*p;
13     node():size(1){}
14     void refresh(){size=ch[0]->size+ch[1]->size+1;}//附加
        ↳ 信息维护
15 }null[maxn];
16
17 //在主函数开头加上这句初始化
18 null->size=0;
19
20 //初始化结点
21 void initailize(node *x){x->ch[0]=x->ch[1]=x->p=null;}//
22
23 //Access 均摊O(\log n)
24 //LCT核心操作,把结点到根的路径打通,顺便把与重儿子的连边
    ↳ 变成轻边
25 //需要调用splay
26 node *access(node *x){
27     node *y=null;
28     while(x!=null){
29         splay(x);
30         x->ch[1]=y;
31         (y=x)->refresh();
32         x=x->p;
33     }
34     return y;
35 }
36
37 //Link 均摊O(\log n)

```

5.6.2 换根/维护生成树(GREALD07加强版)

```

1 #include<cstdio>
2 #include<cstring>
3 #include<algorithm>
4 #include<map>
5 #define isroot(x) ((x)->p==null||((x)->p->ch[0]!
    ↳ =(x)&&(x)->p->ch[1]!=(x)))
6 #define dir(x) ((x)==(x)->p->ch[1])
7 using namespace std;
8 const int maxn=200010;
9 struct node{
10     int key,mn,pos;
11     bool rev;
12     node *ch[2],*p;
13     node(int
        ↳ key=(~0u)>>1:key(key),mn(key),pos(-1),rev(false)
        ↳ {}
14     inline void pushdown(){
15         if(!rev)return;
16         ch[0]->rev^=true;

```

```

17     ch[1]->rev^=true;
18     swap(ch[0],ch[1]);
19     if(pos!=-1)pos^=1;
20     rev=false;
21 }
22 inline void refresh(){
23     mn=key;
24     pos=-1;
25     if(ch[0]->mn<mn){
26         mn=ch[0]->mn;
27         pos=0;
28     }
29     if(ch[1]->mn<mn){
30         mn=ch[1]->mn;
31         pos=1;
32     }
33 }
34 }null[maxn<<1],*ptr=null;
35 node *newnode(int);
36 node *access(node*);
37 void makeroot(node*);
38 void link(node*,node*);
39 void cut(node*,node*);
40 node *getroot(node*);
41 node *getmin(node*,node*);
42 void splay(node*);
43 void rot(node*,int);
44 void build(int,int,int&,int);
45 void query(int,int,int,int);
46 int
47 ↪ sm[maxn<<5]={0},lc[maxn<<5]={0},rc[maxn<<5]={0},root[maxn<<5]={0},yenth[0]=0;
48 map<node*,pair<node*,node*> >mp;
49 node *tmp;
50 int n,m,q,tp,x,y,k,l,r,t,ans=0;
51 int main(){
52     null->ch[0]=null->ch[1]=null->p=null;
53     scanf("%d%d%d",&n,&m,&q,&tp);
54     for(int i=1;i<=n;i++)newnode((~0u)>>1);
55     for(int i=1;i<=m;i++){
56         scanf("%d",&x,&y);
57         if(x==y){
58             root[i]=root[i-1];
59             continue;
60         }
61         if(getroot(null+x)!=getroot(null+y)){
62             tmp=newnode(i);
63             k=0;
64         }
65         else{
66             tmp=getmin(null+x,null+y);
67             cut(tmp,mp[tmp].first);
68             cut(tmp,mp[tmp].second);
69             k=tmp->key;
70             tmp->key=i;
71             tmp->refresh();
72         }
73         link(tmp,null+x);
74         link(tmp,null+y);
75         mp[tmp]=make_pair(null+x,null+y);
76         build(0,m-1,root[i],root[i-1]);
77     }
78     while(q--){
79         scanf("%d",&l,&r);
80         if(tp){
81             l^=ans;
82             r^=ans;
83         }
84         ans=n;
85         t=-1;
86         query(0,m-1,root[r],root[l]);
87         printf("%d\n",ans);
88     }
89 }
90 node *newnode(int x){
91     *++ptr=newnode(x);
92     ptr->ch[0]=ptr->ch[1]=ptr->p=null;
93     return ptr;
94 }
95 node *access(node *x){
96     node *y=null;
97     while(x!=null){
98         splay(x);
99         x->ch[1]=y;
100         (y=x)->refresh();
101         x=x->p;
102     }
103     return y;
104 }
105 void makeroot(node *x){
106     access(x);
107     splay(x);
108     x->rev^=true;
109 }
110 void link(node *x,node *y){
111     makeroot(x);
112     x->p=y;
113 }
114 void cut(node *x,node *y){
115     makeroot(x);
116     access(y);
117     splay(y);
118     y->ch[0]->p=null;
119     y->ch[0]=null;
120     y->refresh();
121 }
122 node *getroot(node *x){
123     x=access(x);
124     while(x->pushdown(),x->ch[0]!=null)x=x->ch[0];
125     splay(x);
126     return x;
127 }
128 node *getmin(node *x,node *y){
129     makeroot(x);
130     x=access(y);
131     while(x->pushdown(),x->pos!=-1)x=x->ch[x->pos];
132     splay(x);
133     return x;
134 }
135 void splay(node *x){
136     x->pushdown();
137     while(!isroot(x)){
138         if(!isroot(x->p))x->p->p->pushdown();
139         x->p->pushdown();
140         x->pushdown();
141         if(isroot(x->p)){
142             rot(x->p,dir(x)^1);
143             break;
144         }
145         if(dir(x)==dir(x->p))rot(x->p->p,dir(x->p)^1);
146         else rot(x->p,dir(x)^1);
147         rot(x->p,dir(x)^1);
148     }
149 }
150 void rot(node *x,int d){
151     node *y=x->ch[d^1];
152     if((x->ch[d^1]=y->ch[d])!=null)y->ch[d]->p=x;
153     y->p=x->p;
154     if(!isroot(x))x->p->ch[dir(x)]=y;
155     (y->ch[d]=x)->p=y;
156     x->refresh();
157     y->refresh();
158 }
159 void build(int l,int r,int &rt,int pr){

```

```

160     sm[rt]=++cnt;sm[pr]+1;
161     if(l==r)return;
162     lc[rt]=lc[pr];
163     rc[rt]=rc[pr];
164     int mid=(l+r)>>1;
165     if(k<=mid)build(l,mid,lc[rt],lc[pr]);
166     else build(mid+1,r,rc[rt],rc[pr]);
167 }
168 void query(int l,int r,int rt,int pr){
169     if(!rt&&!pr)return;
170     if(t>=r){
171         ans-=sm[rt]-sm[pr];
172         return;
173     }
174     int mid=(l+r)>>1;
175     query(l,mid,lc[rt],lc[pr]);
176     if(t>mid)query(mid+1,r,rc[rt],rc[pr]);
177 }

```

5.6.3 维护子树信息

```

1 //Link-Cut Trees with subtree values LCT维护子树信息
2   ↳  $O((n+m)\log n)$ 
3 //By ysf
4 //通过题目LOJ#558 我们的CPU遭到攻击维护黑点到根距离和
5 //这个东西虽然只需要抄板子但还是极其难写常数极其巨大没
6   ↳ 必要的时候就不要用
7 //如果维护子树最小值就需要套一个可删除的堆来维护复杂度
8   ↳ 会变成 $O(n\log^2 n)$ 
9 //注意由于这道题与边权有关需要边权拆点变点权
10 //宏定义
11 #define isroot(x) ((x)->p==NULL||((x)!=(x)->p-
12   ↳ >ch[0]&&(x)!=(x)->p->ch[1]))
13 #define dir(x) ((x)==(x)->p->ch[1])
14 //节点类定义
15 struct node{//以维护子树中黑点到根距离和为例
16     int w,chain_cnt,tree_cnt;
17     long long sum,suml,sumr,tree_sum;//由于换根需要子树反
18   ↳ 转需要维护两个方向的信息
19     bool rev,col;
20     node *ch[2],*p;
21     node():w(0),chain_cnt(0),tree_cnt(0),sum(0),suml(0),sumr(0){}
22     ↳ {}
23     inline void pushdown(){
24         if(!rev)return;
25         ch[0]->rev^=true;
26         ch[1]->rev^=true;
27         swap(ch[0],ch[1]);
28         swap(suml,sumr);
29         rev=false;
30     }
31     inline void refresh(){//不多解释了.....这毒瘤题恶心的要
32   ↳ 死我骂我自己.png
33         sum=ch[0]->sum+ch[1]->sum+w;
34         suml=(ch[0]->rev?ch[0]->sumr:ch[0]->suml)+(ch[1]->rev?ch[1]->sumr:ch[1]->suml);
35         sumr=(ch[0]->rev?ch[0]->suml:ch[0]->sumr)+(ch[1]->rev?ch[1]->suml:ch[1]->sumr);
36         chain_cnt=ch[0]->chain_cnt+ch[1]->chain_cnt+tree_cnt;
37     }
38 }null[maxn<<1];//如果没有边权变点权就不用乘2了
39 //封装构造函数
40 node *newnode(int w){
41     node *x=nodes.front();
42     nodes.pop();
43     initialize(x);

```

```

43     x->w=w;
44     x->refresh();
45     return x;
46 }
47 //封装初始化函数
48 //记得在进行操作之前对所有结点调用一遍
49 inline void initialize(node *x){
50     *x=node();
51     x->ch[0]=x->ch[1]=x->p=NULL;
52 }
53 //Access函数
54 //注意一下在Access的同时更新子树信息的方法
55 node *access(node *x){
56     node *y=NULL;
57     while(x!=NULL){
58         splay(x);
59         x->tree_cnt+=x->ch[1]->chain_cnt-y->chain_cnt;
60         x->tree_sum+=(x->ch[1]->rev?x->ch[1]->sumr:x->ch[1]->suml);
61         x->ch[1]=y;
62         (y=x)->refresh();
63         x=x->p;
64     }
65     return y;
66 }
67 //找到一个点所在连通块的根
68 //对比原版没有变化
69 node *getroot(node *x){
70     x=access(x);
71     while(x->pushdown(),x->ch[0]!=NULL)x=x->ch[0];
72     splay(x);
73     return x;
74 }
75 //换根同样没有变化
76 void makeroot(node *x){
77     access(x);
78     splay(x);
79     x->rev^=true;
80     x->pushdown();
81 }
82 //连接两个点
83 void link(node *x,node *y){
84     makeroot(x);
85     makeroot(y);
86     x->p=y;
87     y->tree_cnt+=x->chain_cnt;
88     y->tree_sum+=x->suml;
89     y->refresh();
90 }
91 //删除一条边
92 //对比原版没有变化
93 void cut(node *x,node *y){
94     rev[ch[1]->sumr:ch[1]->suml)
95     makeroot(x);
96     access(y);
97     rev[ch[1]->suml:ch[1]->sumr)
98     splay(y);
99     y->ch[0]->p=NULL;
100     y->ch[0]=NULL;
101     y->refresh();
102 }
103 //修改/询问一个点这里以询问为例
104 //如果是修改就在换根之后搞一些操作
105 long long query(node *x){
106     makeroot(x);

```



```

113     return x->sum1;
114 }
115 //Splay函数
116 //对比原版没有变化
117 void splay(node *x){
118     x->pushdown();
119     while(!isroot(x)){
120         if(!isroot(x->p))x->p->p->pushdown();
121         x->p->pushdown();
122         x->pushdown();
123         if(isroot(x->p)){
124             rot(x->p,dir(x)^1);
125             break;
126         }
127         if(dir(x)==dir(x->p))rot(x->p->p,dir(x->p)^1);
128         else rot(x->p,dir(x)^1);
129         rot(x->p,dir(x)^1);
130     }
131 }
132 }
133 //旋转函数
134 //对比原版没有变化
135 void rot(node *x,int d){
136     node *y=x->ch[d^1];
137     if((x->ch[d^1]=y->ch[d])!=null)y->ch[d]->p=x;
138     y->p=x->p;
139     if(!isroot(x))x->p->ch[dir(x)]=y;
140     (y->ch[d]=x)->p=y;
141     x->refresh();
142     y->refresh();
143 }
144 }

```

```

34     return a+b;
35 }
36 int size(){return q1.size()-q2.size();}
37 bool empty(){return q1.size()==q2.size();}
38 }heap;//全局堆维护每条链的最大子段和
39 struct node{
40     long long sum,maxsum,prefix,suffix;
41     int key;
42     binary_heap heap;//每个点的堆存的是它的子树中到它的
43     ↳ 最远距离,如果它是黑点的话还会包括自己
44     node *ch[2],*p;
45     bool rev;
46     node(int k=0):sum(k),maxsum(-INF),prefix(-INF),
47         suffix(-INF),key(k),rev(false){}
48     inline void pushdown(){
49         if(!rev)return;
50         ch[0]->rev^=true;
51         ch[1]->rev^=true;
52         swap(ch[0],ch[1]);
53         swap(prefix,suffix);
54         rev=false;
55     }
56     inline void refresh(){
57         pushdown();
58         ch[0]->pushdown();
59         ch[1]->pushdown();
60         sum=ch[0]->sum+ch[1]->sum+key;
61         prefix=max(ch[0]->prefix,
62             ch[0]->sum+key+ch[1]->prefix);
63         suffix=max(ch[1]->suffix,
64             ch[1]->sum+key+ch[0]->suffix);
65         maxsum=max(max(ch[0]->maxsum,ch[1]->maxsum),
66             ch[0]->suffix+key+ch[1]->prefix);
67         if(!heap.empty()){
68             prefix=max(prefix,
69                 ch[0]->sum+key+heap.top());
70             suffix=max(suffix,
71                 ch[1]->sum+key+heap.top());
72             maxsum=max(maxsum,max(ch[0]->suffix,
73                 ch[1]->prefix)+key+heap.top());
74             if(heap.size()>1){
75                 maxsum=max(maxsum,heap.top2()+key);
76             }
77         }
78     }null[maxn<<1],*ptr=null;
79     void addedge(int,int,int);
80     void deledge(int,int);
81     void modify(int,int,int);
82     void modify_color(int);
83     node *newnode(int);
84     node *access(node*);
85     void makeroot(node*);
86     void link(node*,node*);
87     void cut(node*,node*);
88     void splay(node*);
89     void rot(node*,int);
90     queue<node*>freenodes;
91     tree<pair<int,int>,node*>mp;
92     bool col[maxn]={false};
93     char c;
94     int n,m,k,x,y,z;
95     int main(){
96         null->ch[0]=null->ch[1]=null->p=null;
97         scanf("%d%d%d",&n,&m,&k);
98         for(int i=1;i<=n;i++){
99             newnode(0);
100         }
101         heap.push(0);
102         while(k--){
103             scanf("%d",&x);
104             col[x]=true;

```

5.6.4 模板题:动态QTREE4(询问树上相距最远点)

```

1 #include<bits/stdc++.h>
2 #include<ext/pb_ds/assoc_container.hpp>
3 #include<ext/pb_ds/tree_policy.hpp>
4 #include<ext/pb_ds/priority_queue.hpp>
5
6 #define isroot(x) ((x)->p==null||((x)!=(x)->p->
7 ↳ >ch[0]&&(x)!=(x)->p->ch[1]))
8 #define dir(x) ((x)==(x)->p->ch[1])
9
10 using namespace std;
11 using namespace __gnu_pbds;
12
13 const int maxn=100010;
14 const long long INF=1000000000000000000ll;
15
16 struct binary_heap{
17     __gnu_pbds::priority_queue<long long,less<long
18 ↳ long>,binary_heap_tag>q1,q2;
19     binary_heap(){
20         void push(long long x){if(x>(-INF)>>2)q1.push(x);}
21         void erase(long long x){if(x>(-INF)>>2)q2.push(x);}
22         long long top(){
23             if(empty())return -INF;
24             while(!q2.empty()&&q1.top()==q2.top()){
25                 q1.pop();
26                 q2.pop();
27             }
28             return q1.top();
29         }
30         long long top2(){
31             if(size()<2)return -INF;
32             long long a=top();
33             erase(a);
34             long long b=top();
35             push(a);

```



```

105     null[x].heap.push(0);
106 }
107 for(int i=1;i<n;i++){
108     scanf("%d%d%d",&x,&y,&z);
109     if(x>y)swap(x,y);
110     addedge(x,y,z);
111 }
112 while(m--){
113     scanf("%c",&c,&x);
114     if(c=='A'){
115         scanf("%d",&y);
116         if(x>y)swap(x,y);
117         deledge(x,y);
118     }
119     else if(c=='B'){
120         scanf("%d%d",&y,&z);
121         if(x>y)swap(x,y);
122         addedge(x,y,z);
123     }
124     else if(c=='C'){
125         scanf("%d%d",&y,&z);
126         if(x>y)swap(x,y);
127         modify(x,y,z);
128     }
129     else modify_color(x);
130     printf("%lld\n",(heap.top()>0?heap.top():-1));
131 }
132 return 0;
133 }
134 void addedge(int x,int y,int z){
135     node *tmp;
136     if(freenodes.empty())tmp=newnode(z);
137     else{
138         tmp=freenodes.front();
139         freenodes.pop();
140         *tmp=node(z);
141     }
142     tmp->ch[0]=tmp->ch[1]=tmp->p=null;
143     heap.push(tmp->maxsum);
144     link(tmp,null+x);
145     link(tmp,null+y);
146     mp[make_pair(x,y)]=tmp;
147 }
148 void deledge(int x,int y){
149     node *tmp=mp[make_pair(x,y)];
150     cut(tmp,null+x);
151     cut(tmp,null+y);
152     freenodes.push(tmp);
153     heap.erase(tmp->maxsum);
154     mp.erase(make_pair(x,y));
155 }
156 void modify(int x,int y,int z){
157     node *tmp=mp[make_pair(x,y)];
158     makeroot(tmp);
159     tmp->pushdown();
160     heap.erase(tmp->maxsum);
161     tmp->key=z;
162     tmp->refresh();
163     heap.push(tmp->maxsum);
164 }
165 void modify_color(int x){
166     makeroot(null+x);
167     col[x]^=true;
168     if(col[x])null[x].heap.push(0);
169     else null[x].heap.erase(0);
170     heap.erase(null[x].maxsum);
171     null[x].refresh();
172     heap.push(null[x].maxsum);
173 }
174 node *newnode(int k){
175     *(++ptr)=node(k);
176     ptr->ch[0]=ptr->ch[1]=ptr->p=null;
177     return ptr;
178 }
179 node *access(node *x){
180     splay(x);
181     heap.erase(x->maxsum);
182     x->refresh();
183     if(x->ch[1]!=null){
184         x->ch[1]->pushdown();
185         x->heap.push(x->ch[1]->prefix);
186         x->refresh();
187         heap.push(x->ch[1]->maxsum);
188     }
189     x->ch[1]=null;
190     x->refresh();
191     node *y=x;
192     x=x->p;
193     while(x!=null){
194         splay(x);
195         heap.erase(x->maxsum);
196         if(x->ch[1]!=null){
197             x->ch[1]->pushdown();
198             x->heap.push(x->ch[1]->prefix);
199             heap.push(x->ch[1]->maxsum);
200         }
201         x->heap.erase(y->prefix);
202         x->ch[1]=y;
203         (y=x)->refresh();
204         x=x->p;
205     }
206     heap.push(y->maxsum);
207     return y;
208 }
209 void makeroot(node *x){
210     access(x);
211     splay(x);
212     x->rev^=true;
213 }
214 void link(node *x,node *y){//新添一条虚边维护y对应的堆
215     makeroot(x);
216     makeroot(y);
217     x->pushdown();
218     x->p=y;
219     heap.erase(y->maxsum);
220     y->heap.push(x->prefix);
221     y->refresh();
222     heap.push(y->maxsum);
223 }
224 void cut(node *x,node *y){//断开一条实边一条链变成两条
    ⇨ 链需要维护全局堆
225     makeroot(x);
226     access(y);
227     splay(y);
228     heap.erase(y->maxsum);
229     heap.push(y->ch[0]->maxsum);
230     y->ch[0]->p=null;
231     y->ch[0]=null;
232     y->refresh();
233     heap.push(y->maxsum);
234 }
235 void splay(node *x){
236     x->pushdown();
237     while(!isroot(x)){
238         if(!isroot(x->p))
239             x->p->p->pushdown();
240         x->p->pushdown();
241         x->pushdown();
242         if(isroot(x->p)){
243             rot(x->p,dir(x)^1);
244             break;
245         }
246         if(dir(x)==dir(x->p))
247             rot(x->p->p,dir(x->p)^1);

```

```

248     else rot(x->p,dir(x)^1);
249     rot(x->p,dir(x)^1);
250 }
251 }
252 void rot(node *x,int d){
253     node *y=x->ch[d^1];
254     if((x->ch[d^1]=y->ch[d])!=null)
255         y->ch[d]->p=x;
256     y->p=x->p;
257     if(!isroot(x))
258         x->p->ch[dir(x)]=y;
259     (y->ch[d]=x)->p=y;
260     x->refresh();
261     y->refresh();
262 }

```

```

49
50     v[y].clear();
51 }
52 }

```

5.8 梯子剖分

```

1 // 在线求一个点的第k祖先  $O(n \log n) - O(1)$ 
2 // 理论基础: 任意一个点x的k级祖先y所在长链长度一定  $\geq k$ 
3
4 // 全局数组定义
5 vector<int> G[maxn], v[maxn];
6 int d[maxn], mxd[maxn], son[maxn], top[maxn], len[maxn];
7 int f[19][maxn], log_tbl[maxn];
8
9 // 在主函数中两遍dfs之后加上如下预处理
10 log_tbl[0] = -1;
11 for (int i = 1; i <= n; i++)
12     log_tbl[i] = log_tbl[i / 2] + 1;
13 for (int j = 1; (1 << j) < n; j++)
14     for (int i = 1; i <= n; i++)
15         f[j][i] = f[j - 1][f[j - 1][i]];
16
17 // 第一遍dfs, 用于计算深度和找出重儿子
18 void dfs1(int x) {
19     mxd[x] = d[x];
20
21     for (int y : G[x])
22         if (y != f[0][x]){
23             f[0][y] = x;
24             d[y] = d[x] + 1;
25
26             dfs1(y);
27
28             mxd[x] = max(mxd[x], mxd[y]);
29             if (mxd[y] > mxd[son[x]])
30                 son[x] = y;
31         }
32 }
33
34 // 第二遍dfs, 用于进行剖分和预处理梯子剖分(每条链向上延
35 // 伸一倍)数组
36 void dfs2(int x) {
37     top[x] = (x == son[f[0][x]] ? top[f[0][x]] : x);
38
39     for (int y : G[x])
40         if (y != f[0][x])
41             dfs2(y);
42
43     if (top[x] == x) {
44         int u = x;
45         while (top[son[u]] == x)
46             u = son[u];
47
48         len[x] = d[u] - d[x];
49         for (int i = 0; i < len[x]; i++, u = f[0][u])
50             v[x].push_back(u);
51
52         u = x;
53         for (int i = 0; i < len[x] && u; i++, u = f[0][u])
54             v[x].push_back(u);
55     }
56 }
57
58 // 在线询问x的第k级祖先  $O(1)$ 
59 // 不存在时返回0
60 int query(int x, int k) {
61     if (!k)

```

5.7 长链剖分

```

1 // 顾名思义, 长链剖分是取最深的儿子作为重儿子
2
3 //  $O(n)$ 维护以深度为下标的子树信息
4 vector<int> G[maxn], v[maxn];
5 int n, p[maxn], h[maxn], son[maxn], ans[maxn];
6
7 // 原题题意: 求每个点的子树中与它距离是几的点最多, 相同的
8 // 取最大深度
9 // 由于vector只能在后面加入元素, 为了写代码方便, 这里反
10 // 过来存
11 void dfs(int x) {
12     h[x] = 1;
13
14     for (int y : G[x])
15         if (y != p[x]){
16             p[y] = x;
17             dfs(y);
18
19             if (h[y] > h[son[x]])
20                 son[x] = y;
21         }
22
23     if (!son[x]) {
24         v[x].push_back(1);
25         ans[x] = 0;
26         return;
27     }
28
29     h[x] = h[son[x]] + 1;
30     swap(v[x], v[son[x]]);
31
32     if (v[x][ans[son[x]]] == 1)
33         ans[x] = h[x] - 1;
34     else
35         ans[x] = ans[son[x]];
36
37     v[x].push_back(1);
38
39     int mx = v[x][ans[x]];
40     for (int y : G[x])
41         if (y != p[x] && y != son[x]) {
42             for (int j = 1; j <= h[y]; j++) {
43                 v[x][h[x] - j - 1] += v[y][h[y] - j];
44
45                 int t = v[x][h[x] - j - 1];
46                 if (t > mx || (t == mx && h[x] - j - 1 >
47                     // 取ans[x])) {
48                     mx = t;
49                     ans[x] = h[x] - j - 1;
50                 }
51             }
52         }
53 }

```

```

61     return x;
62     if (k > d[x])
63         return 0;
64
65     x = f[log_tbl[k]][x];
66     k ^= 1 << log_tbl[k];
67     return v[top[x]][d[top[x]] + len[top[x]] - d[x] + k];
68 }

```

5.9 左偏树

(参见k短路)

5.10 常见根号思路

通用

- 出现次数大于 \sqrt{n} 的数不会超过 \sqrt{n} 个
- 对于带修改问题, 如果不方便分治或者二进制分组, 可以考虑对操作分块, 每次查询时暴力最后的 \sqrt{n} 个修改并更正答案
- 根号分治: 如果分治时每个子问题需要 $O(N)$ (N 是全局问题的大小) 的时间, 而规模较小的子问题可以 $O(n^2)$ 解决, 则可以使用根号分治
 - 规模大于 \sqrt{n} 的子问题用 $O(N)$ 的方法解决, 规模小于 \sqrt{n} 的子问题用 $O(n^2)$ 暴力
 - 规模大于 \sqrt{n} 的子问题最多只有 \sqrt{n} 个
 - 规模不大于 \sqrt{n} 的子问题大小的平方和也必定不会超过 $n\sqrt{n}$
- 如果输入规模之和不大于 n (例如给定多个小字符串与大字符串进行询问), 那么规模超过 \sqrt{n} 的问题最多只有 \sqrt{n} 个

序列

- 某些维护序列的问题可以用分块/块状链表维护
- 对于静态区间询问问题, 如果可以快速将左/右端点移动一位, 可以考虑莫队
 - 如果强制在线可以分块预处理, 但是一般空间需要 $n\sqrt{n}$
 - * 例题: 询问区间中有几种数出现次数恰好为 k , 强制在线
 - 如果带修改可以试着想一想带修莫队, 但是复杂度高达 $n^{\frac{5}{3}}$
- 线段树可以解决的问题也可以用分块来做到 $O(1)$ 询问或是 $O(1)$ 修改, 具体要看哪种操作更多

树

- 与序列类似, 树上也有树分块和树上莫队
 - 树上带修莫队很麻烦, 常数也大, 最好不要先考虑
 - 树分块不要想当然
- 树分治也可以套根号分治, 道理是一样的

字符串

- 循环节长度大于 \sqrt{n} 的子串最多只有 $O(n)$ 个, 如果是极长子串则只有 $O(\sqrt{n})$ 个

6. 动态规划

6.1 决策单调性 $O(n \log n)$

```

1  int a[maxn], q[maxn], p[maxn], g[maxn]; // 存左端点, 右端
    ↪ 点就是下一个左端点 - 1
2
3  long long f[maxn], s[maxn];
4
5  int n, m;
6
7  long long calc(int l, int r) {
8      if (r < l)
9          return 0;
10
11     int mid = (l + r) / 2;
12     if ((r - l + 1) % 2 == 0)
13         return (s[r] - s[mid]) - (s[mid] - s[l - 1]);
14     else
15         return (s[r] - s[mid]) - (s[mid - 1] - s[l - 1]);
16 }
17
18 int solve(long long tmp) {
19     memset(f, 63, sizeof(f));
20     f[0] = 0;
21
22     int head = 1, tail = 0;
23
24     for (int i = 1; i <= n; i++) {
25         f[i] = calc(1, i);
26         g[i] = 1;
27
28         while (head < tail && p[head + 1] <= i)
29             head++;
30         if (head <= tail) {
31             if (f[q[head]] + calc(q[head] + 1, i) < f[i])
32                 ↪ {
33                     f[i] = f[q[head]] + calc(q[head] + 1, i);
34                     g[i] = g[q[head]] + 1;
35                 }
36             while (head < tail && p[head + 1] <= i + 1)
37                 head++;
38             if (head <= tail)
39                 p[head] = i + 1;
40         }
41         f[i] += tmp;
42
43         int r = n;
44
45         while (head <= tail) {
46             if (f[q[tail]] + calc(q[tail] + 1, p[tail]) >
47                 ↪ f[i] + calc(i + 1, p[tail])) {
48                 r = p[tail] - 1;
49                 tail--;
50             }
51             else if (f[q[tail]] + calc(q[tail] + 1, r) <=
52                 ↪ f[i] + calc(i + 1, r)) {
53                 if (r < n) {
54                     q[++tail] = i;
55                     p[tail] = r + 1;
56                 }
57                 break;
58             }
59             else {
60                 int L = p[tail], R = r;
61                 while (L < R) {
62                     int M = (L + R) / 2;
63
64                     if (f[q[tail]] + calc(q[tail] + 1, M)
65                         ↪ <= f[i] + calc(i + 1, M))
66                         L = M + 1;
67                     else
68                         R = M;
69                 }
70                 q[++tail] = i;
71                 p[tail] = R + 1;
72             }
73         }
74     }
75 }

```

```

63         else
64             R = M;
65     }
66     q[++tail] = i;
67     p[tail] = L;
68
69     break;
70 }
71 }
72 if (head > tail) {
73     q[++tail] = i;
74     p[tail] = i + 1;
75 }
76 }
77 }
78
79 return g[n];
80 }

```

```

24     }
25
26     while (x) {
27         a[++a[0]] = x % 10;
28         x /= 10;
29     }
30 }
31
32 big_decimal(string s) {
33     memset(a, 0, sizeof(a));
34     negative = false;
35
36     if (s == "")
37         return;
38
39     if (s[0] == '-') {
40         negative = true;
41         s = s.substr(1);
42     }
43     a[0] = s.size();
44     for (int i = 1; i <= a[0]; i++)
45         a[i] = s[a[0] - i] - '0';
46
47     while (a[0] && !a[a[0]])
48         a[0]--;
49 }
50
51 void input() {
52     string s;
53     cin >> s;
54     *this = s;
55 }
56
57 string str() const {
58     if (!a[0])
59         return "0";
60
61     string s;
62     if (negative)
63         s = "-";
64
65     for (int i = a[0]; i; i--)
66         s.push_back('0' + a[i]);
67
68     return s;
69 }
70
71 operator string () const {
72     return str();
73 }
74
75 big_decimal operator - () const {
76     big_decimal o = *this;
77     if (a[0])
78         o.negative ^= true;
79
80     return o;
81 }
82
83 friend big_decimal abs(const big_decimal &u) {
84     big_decimal o = u;
85     o.negative = false;
86     return o;
87 }
88
89 big_decimal &operator <= (int k) {
90     a[0] += k;
91
92     for (int i = a[0]; i > k; i--)

```

7. Miscellaneous

7.1 $O(1)$ 快速乘

```

1 // Long double 快速乘
2 // 在两数直接相乘会爆Long Long时才有必要使用
3 // 常数比直接Long Long乘法+取模大很多,非必要时不建议使用
4 long long mul(long long a,long long b,long long p){
5     a%=p;b%=p;
6     return ((a*b-p*(long long)((long
7     ↪ double)a/p*b+0.5))%p+p)%p;
8 }
9 // 指令集快速乘
10 // 试机记得测试能不能过编译
11 inline long long mul(const long long a, const long long
12     ↪ b, const long long p) {
13     long long ans;
14     __asm__ __volatile__ ("\tmulq %%rbx\n\tdivq %%rcx\n"
15     ↪ : "=d"(ans) : "a"(a), "b"(b), "c"(p));
16     return ans;
17 }

```

7.2 $O(n^2)$ 高精度

```

1 // 注意如果只需要正数运算的话
2 // 可以只抄英文名的运算函数
3 // 按需自取
4 // 乘法 $O(n^2)$ 除法 $O(10 * n^2)$ 
5
6 const int maxn = 1005;
7
8 struct big_decimal {
9     int a[maxn];
10     bool negative;
11
12     big_decimal() {
13         memset(a, 0, sizeof(a));
14         negative = false;
15     }
16
17     big_decimal(long long x) {
18         memset(a, 0, sizeof(a));
19         negative = false;
20
21         if (x < 0) {
22             negative = true;
23             x = -x;

```

```

93     a[i] = a[i - k];
94
95     for(int i = k; i; i--)
96         a[i] = 0;
97
98     return *this;
99 }
100
101 friend big_decimal operator << (const big_decimal &u,
102     ↪ int k) {
103     big_decimal o = u;
104     return o <= k;
105 }
106
107 big_decimal &operator >>= (int k) {
108     if (a[0] < k)
109         return *this = big_decimal(0);
110
111     a[0] -= k;
112     for (int i = 1; i <= a[0]; i++)
113         a[i] = a[i + k];
114
115     for (int i = a[0] + 1; i <= a[0] + k; i++)
116         a[i] = 0;
117
118     return *this;
119 }
120
121 friend big_decimal operator >> (const big_decimal &u,
122     ↪ int k) {
123     big_decimal o = u;
124     return o >>= k;
125 }
126
127 friend int cmp(const big_decimal &u, const
128     ↪ big_decimal &v) {
129     if (u.negative || v.negative) {
130         if (u.negative && v.negative)
131             return -cmp(-u, -v);
132
133         if (u.negative)
134             return -1;
135
136         if (v.negative)
137             return 1;
138     }
139
140     if (u.a[0] != v.a[0])
141         return u.a[0] < v.a[0] ? -1 : 1;
142
143     for (int i = u.a[0]; i; i--)
144         if (u.a[i] != v.a[i])
145             return u.a[i] < v.a[i] ? -1 : 1;
146
147     return 0;
148 }
149
150 friend bool operator < (const big_decimal &u, const
151     ↪ big_decimal &v) {
152     return cmp(u, v) == -1;
153 }
154
155 friend bool operator > (const big_decimal &u, const
156     ↪ big_decimal &v) {
157     return cmp(u, v) == 1;
158 }
159
160 friend bool operator <= (const big_decimal &u, const
161     ↪ big_decimal &v) {
162     return cmp(u, v) <= 0;
163 }
164
165 friend bool operator >= (const big_decimal &u, const
166     ↪ big_decimal &v) {
167     return cmp(u, v) >= 0;
168 }
169
170 friend big_decimal decimal_plus(const big_decimal &u,
171     ↪ const big_decimal &v) { // 保证u, v均为正数的话可
172     ↪ 以直接调用
173     big_decimal o;
174
175     o.a[0] = max(u.a[0], v.a[0]);
176
177     for (int i = 1; i <= u.a[0] || i <= v.a[0]; i++)
178         ↪ {
179             o.a[i] += u.a[i] + v.a[i];
180
181             if (o.a[i] >= 10) {
182                 o.a[i + 1]++;
183                 o.a[i] -= 10;
184             }
185         }
186
187     if (o.a[o.a[0] + 1])
188         o.a[o.a[0] + 1]++;
189
190     return o;
191 }
192
193 friend big_decimal decimal_minus(const big_decimal
194     ↪ &u, const big_decimal &v) { // 保证u, v均为正数的
195     ↪ 话可以直接调用
196     int k = cmp(u, v);
197
198     if (k == -1)
199         return -decimal_minus(v, u);
200     else if (k == 0)
201         return big_decimal(0);
202
203     big_decimal o;
204
205     o.a[0] = u.a[0];
206
207     for (int i = 1; i <= u.a[0]; i++) {
208         o.a[i] += u.a[i] - v.a[i];
209
210         if (o.a[i] < 0) {
211             o.a[i] += 10;
212             o.a[i + 1]--;
213         }
214     }
215
216     while (o.a[o.a[0]] && !o.a[o.a[0] + 1])
217         o.a[o.a[0]]--;
218
219     return o;
220 }
221
222 friend big_decimal decimal_multi(const big_decimal
223     ↪ &u, const big_decimal &v) {
224     big_decimal o;
225
226     o.a[0] = u.a[0] + v.a[0] - 1;

```

```

218         for (int i = 1; i <= u.a[0]; i++)
219             for (int j = 1; j <= v.a[0]; j++)
220                 o.a[i + j - 1] += u.a[i] * v.a[j];
221
222         for (int i = 1; i <= o.a[0]; i++)
223             if (o.a[i] >= 10) {
224                 o.a[i + 1] += o.a[i] / 10;
225                 o.a[i] %= 10;
226             }
227
228         if (o.a[o.a[0] + 1])
229             o.a[0]++;
230
231         return o;
232     }
233 }
234
235 friend pair<big_decimal, big_decimal>
236     ↪ decimal_divide(big_decimal u, big_decimal v) { //
237     ↪ 整除
238         if (v > u)
239             return make_pair(big_decimal(0), u);
240
241         big_decimal o;
242         o.a[0] = u.a[0] - v.a[0] + 1;
243
244         int m = v.a[0];
245         v <= u.a[0] - m;
246
247         for (int i = u.a[0]; i >= m; i--) {
248             while (u >= v) {
249                 u = u - v;
250                 o.a[i - m + 1]++;
251             }
252             v >>= 1;
253         }
254
255         while (o.a[0] && !o.a[o.a[0]])
256             o.a[0]--;
257
258         return make_pair(o, u);
259     }
260
261 friend big_decimal operator + (const big_decimal &u,
262     ↪ const big_decimal &v) {
263     if (u.negative || v.negative) {
264         if (u.negative && v.negative)
265             return -decimal_plus(-u, -v);
266
267         if (u.negative)
268             return v - (-u);
269
270         if (v.negative)
271             return u - (-v);
272     }
273
274     return decimal_plus(u, v);
275 }
276
277 friend big_decimal operator - (const big_decimal &u,
278     ↪ const big_decimal &v) {
279     if (u.negative || v.negative) {
280         if (u.negative && v.negative)
281             return -decimal_minus(-u, -v);
282
283         if (u.negative)
284             return -decimal_plus(-u, v);
285
286         if (v.negative)
287             return decimal_plus(u, -v);
288     }
289
290     return decimal_minus(u, v);
291 }
292
293 friend big_decimal operator * (const big_decimal &u,
294     ↪ const big_decimal &v) {
295     if (u.negative || v.negative) {
296         big_decimal o = decimal_multi(abs(u),
297             ↪ abs(v));
298
299         if (u.negative ^ v.negative)
300             return -o;
301         return o;
302     }
303
304     return decimal_multi(u, v);
305 }
306
307 big_decimal operator * (long long x) const {
308     if (x >= 10)
309         return *this * big_decimal(x);
310
311     if (negative)
312         return -(*this * x);
313
314     big_decimal o;
315
316     o.a[0] = a[0];
317
318     for (int i = 1; i <= a[0]; i++) {
319         o.a[i] += a[i] * x;
320
321         if (o.a[i] >= 10) {
322             o.a[i + 1] += o.a[i] / 10;
323             o.a[i] %= 10;
324         }
325     }
326
327     if (o.a[o.a[0] + 1])
328         o.a[0]++;
329
330     return o;
331 }
332
333 friend pair<big_decimal, big_decimal>
334     ↪ decimal_div(const big_decimal &u, const
335     ↪ big_decimal &v) {
336     if (u.negative || v.negative) {
337         pair<big_decimal, big_decimal> o =
338             ↪ decimal_div(abs(u), abs(v));
339
340         if (u.negative ^ v.negative)
341             return make_pair(-o.first, -o.second);
342         return o;
343     }
344
345     return decimal_divide(u, v);
346 }
347
348 friend big_decimal operator / (const big_decimal &u,
349     ↪ const big_decimal &v) { // v不能是0
350     if (u.negative || v.negative) {
351         big_decimal o = abs(u) / abs(v);
352
353         if (u.negative ^ v.negative)
354             return -o;
355     }
356
357     return decimal_divide(u, v);
358 }

```

```

346         return o;
347     }
348
349     return decimal_divide(u, v).first;
350 }
351
352 friend big_decimal operator % (const big_decimal &u,
353     ↪ const big_decimal &v) {
354     if (u.negative || v.negative) {
355         big_decimal o = abs(u) % abs(v);
356
357         if (u.negative ^ v.negative)
358             return -o;
359         return o;
360     }
361     return decimal_divide(u, v).second;
362 }
363 };

```

7.3 xorshift

```

1  ull k1, k2;
2  const int mod = 10000000;
3  ull xorShift128Plus() {
4      ull k3 = k1, k4 = k2;
5      k1 = k4;
6      k3 ^= (k3 << 23);
7      k2 = k3 ^ k4 ^ (k3 >> 17) ^ (k4 >> 26);
8      return k2 + k4;
9  }
10 void gen(ull _k1, ull _k2) {
11     k1 = _k1, k2 = _k2;
12     int x = xorShift128Plus() % threshold + 1;
13     // do sth
14 }
15
16
17 uint32_t xor128(void) {
18     static uint32_t x = 123456789;
19     static uint32_t y = 362436069;
20     static uint32_t z = 521288629;
21     static uint32_t w = 88675123;
22     uint32_t t;
23
24     t = x ^ (x << 11);
25     x = y; y = z; z = w;
26     return w = w ^ (w >> 19) ^ (t ^ (t >> 8));
27 }

```

7.4 枚举子集

(注意这是 $t \neq 0$ 的写法, 如果可以等于0需要在循环里手动break)

```

1  for (int t = s; t; (--t) &= s) {
2      // do something
3  }

```

7.5 STL

1. vector

- `vector<int nSize>`:创建一个vector,元素个数为nSize
- `vector<int nSize, const t& t>`:创建一个vector 元素个数为nSize,且值均为t
- `vector(begin, end)`:复制[begin, end)区间内另一个数组的元素到vector中

- `void assign(int n, const T& x)`:设置向量中前n个元素的值为x
- `void assign(const_iterator first, const_iterator last)`:向量中[first, last)中元素设置成当前向量元素

2. list

- `assign()` 给list赋值
- `back()` 返回最后一个元素
- `begin()` 返回指向第一个元素的迭代器
- `clear()` 删除所有元素
- `empty()` 如果list是空的则返回true
- `end()` 返回末尾的迭代器
- `erase()` 删除一个元素
- `front()` 返回第一个元素
- `insert()` 插入一个元素到list中
- `max_size()` 返回list能容纳的最大元素数量
- `merge()` 合并两个list
- `pop_back()` 删除最后一个元素
- `pop_front()` 删除第一个元素
- `push_back()` 在list的末尾添加一个元素
- `push_front()` 在list的头部添加一个元素
- `rbegin()` 返回指向第一个元素的逆向迭代器
- `remove()` 从list删除元素
- `remove_if()` 按指定条件删除元素
- `rend()` 指向list末尾的逆向迭代器
- `resize()` 改变list的大小
- `reverse()` 把list的元素倒转
- `size()` 返回list中的元素个数
- `sort()` 给list排序
- `splice()` 合并两个list
- `swap()` 交换两个list
- `unique()` 删除list中重复的元素

7.6 pb_ds

7.7 rope

8. 注意事项

8.1 常见下毒手法

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

8.2 场外相关

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

8.3 做题策略与心态调节

- 拿到题后立刻按照商量好的顺序读题,前半小时最好跳过题意太复杂的题(除非被过穿了)
- 签到题写完不要激动,稍微检查一下最可能的下毒点再交,避免无谓的罚时
 - 一两行的那种傻逼题就算了
- 读完题及时输出题意,一方面避免重复读题,一方面也可以让队友有一个初步印象,方便之后决定开题顺序
- 一个题如果卡了很久又有其他题可以写,那不妨先放掉写更容易的题,不要在一棵树上吊死

— 不要被一两道题搞得心态爆炸,一方面急也没有意义,一方面你很可能真的离AC就差一步

- 榜是不会骗人的,一个题如果被不少人过了就说明这个题很可能并没有那么难;如果不是有十足的把握就不要轻易开没什么人交的题;另外不要忘记最后一小时会封榜
- 想不出题/找不出毒自然容易犯困,一定不要放任自己昏昏欲睡,最好去洗手间冷静一下,没有条件就站起来踱步
- 思考的时候不要挂机,一定要在草稿纸上画一画,最好说出声来最容易不掉思路
- 出完算法一定要check一下样例和一些trivial的情况,不然容易写了半天发现写了个假算法
- 上机前有时间就提前给需要思考怎么写的地方打草稿,不要浪费机时
- 查毒时如果最难的地方反复check也没有问题,就从头到脚仔仔细细查一遍,不要放过任何细节,即使是并查集和sort这种东西也不能想当然
- 后半场如果时间不充裕就不要冒险开难题,除非真的无事可做
 - 如果是没写过的东西也不要轻举妄动,在有其他好写的题的时候就等一会再说
- 大多数时候都要听队长安排,虽然不一定最正确但可以保持组织性
- 最好注意一下影响,就算忍不住嘴臭也不要太大声
- 任何时候都不要着急,着急不能解决问题,不要当喆国王
- 输了游戏,还有人生;赢了游戏,还有人生.