

# ICPC Templates For ZJUT12

EIPsilly

October 25, 2019

## Contents

<b>1</b>	<b>树</b>	<b>2</b>
1.1	树上倍增求 LCA	2
1.2	树链剖分 (重链剖分)	3
1.3	点分治	7
1.4	长链剖分	9
<b>2</b>	<b>线段树</b>	<b>11</b>
2.1	zkw 线段树	11
2.2	树状数组维护区间和	12
2.3	树状数组维护区间最大值	13
2.4	主席树	14
<b>3</b>	<b>图论</b>	<b>16</b>
3.1	最短路	16
3.1.1	dijkstra 求最短路	16
3.1.2	spfa 求最短路	17
3.2	网络流	18
3.2.1	dinic	18
3.3	二分图	20
3.3.1	二分图 KM 板子	20
3.3.2	二分图最大匹配	22
3.4	tarjan	23
3.4.1	tarjan 求环的个数和点的染色	23
3.4.2	tarjan 求无向图割边和割点	24
<b>4</b>	<b>字符串</b>	<b>26</b>
4.1	Manacher	26

# 1 树

## 1.1 树上倍增求 LCA

```
1  #include<vector>
2  #include<cstring>
3  #include<stdio.h>
4  using namespace std;
5  #define mem(a) memset(a,0,sizeof(a))
6
7  const int Vertex_MAXN = 1e4 + 10;
8  const int DEEP = 15;
9
10 vector<int> g[Vertex_MAXN];
11
12 int up[Vertex_MAXN][DEEP] = {},d[Vertex_MAXN] = {};
13
14 void dfs(int x)
15 {
16     d[x] = d[up[x][0]] + 1;
17     for (int i = 1;i<=14;i++) up[x][i] = up[up[x][i-1]][i-1];
18     for (int i = 0;i<g[x].size();i++)
19         if (g[x][i] != up[x][0])
20         {
21             up[g[x][i]][0] = x;
22             dfs(g[x][i]);
23         }
24 }
25
26 int LCA(int x,int y)
27 {
28     if (d[x] < d[y]) swap(x,y);
29     int dis = d[x] - d[y];
30     for (int i = 14;i>=0;i--)
31         if (dis & (1<<i)) x = up[x][i];
32     if (x == y) return x;
33     for (int i = 14;i>=0;i--)
34     {
35         if (up[x][i] != up[y][i])
36         {
37             x = up[x][i],y = up[y][i];
38         }
39     }
40     return up[x][0];
41 }
42
43 int main()
44 {
45     int n,in[11000] = {};
46     scanf("%d",&n);
47     for (int i = 1,u,v;i<=n-1;i++)
48     {
49         scanf("%d%d",&u,&v);//规定前者为后者的父节点
```

```

50     g[u].push_back(v);
51     in[v]++;
52 }
53 int u,v;
54 for (int i = 1;i<=n-1;i++)
55 {
56     if (in[i] == 0)
57     {
58         dfs(i);
59         break;
60     }
61 }
62 scanf("%d%d",&u,&v); //询问u和v的LCA
63 printf("%d",LCA(u,v));
64 mem(up); mem(d);
65 for (int i = 1;i<=n-1;i++) g[i].clear();
66 return 0;
67 }

```

## 1.2 树链剖分 (重链剖分)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define mem(a) memset(a,0,sizeof(a))
4
5  const int MAXN = 40100;
6  vector<int> g[MAXN];
7  int n,size1[MAXN],dfn[MAXN],top[MAXN],son[MAXN],fa[MAXN],d[MAXN],num = 0,aa[MAXN];
8  map<pair<int,int>,int> w;
9
10 //树链剖分
11 void dfs1(int x)
12 {
13     d[x] = d[fa[x]] + 1;
14     size1[x] = 1;
15     for (int i = 0;i<g[x].size();i++)
16     {
17         int v = g[x][i];
18         if (v == fa[x]) continue;
19         fa[v] = x;
20         dfs1(v);
21         size1[x] += size1[v];
22         if (size1[v] > size1[son[x]]) son[x] = v;
23     }
24 }
25
26 void dfs2(int x,int tp)
27 {
28     dfn[x] = ++num;
29     top[x] = tp;
30     aa[num] = w[make_pair(x,fa[x])];
31     if (son[x]) dfs2(son[x],tp);

```

```

32     for (int i = 0; i < g[x].size(); i++)
33         if (g[x][i] != fa[x] && g[x][i] != son[x]) dfs2(g[x][i], g[x][i]);
34 }
35
36 struct node{
37     int l, r, lazy, num, nl, nr;
38     int mid(){ return (r - l) / 2 + 1; }
39     void merge(node a, node b) // 合并两个区间
40     {
41         num = a.num + b.num;
42         if (a.nr == b.nl && a.nr != -1 && b.nl != -1) num--;
43         nl = a.nl; nr = b.nr;
44     }
45     void overturn() { swap(nr, nl); } // 将区间顺序倒置
46     // void write() // 检验时用
47     // {
48     //     printf("%d %d %d %d %d %d\n", l, r, lazy, num, nl, nr);
49     // }
50 };
51
52 struct seg_tree{
53     node tree[MAXN * 4];
54     void build(int st, int ed, int x)
55     // 建树, lazy表示当前结点下的子节点需要更改的值, nl表示左端点的数字, nr表示右端点的数
56     // 字, num表示有多少段颜色相同的
57     {
58         tree[x].l = st;
59         tree[x].r = ed;
60         tree[x].lazy = -1;
61         if (st == ed)
62         {
63             tree[x].nl = tree[x].nr = aa[st];
64             tree[x].num = 1;
65             if (st == 1) // 建树的时候, 一条边上两点, 深度较大的那个点的dfn作为这条线段的dfn
66             // , 所以根节点1没有对应的边
67             {
68                 tree[x].nl = tree[x].nr = -1;
69                 tree[x].num = 0;
70             }
71             return;
72         }
73         int mid = tree[x].mid();
74         build(st, mid, x * 2);
75         build(mid + 1, ed, x * 2 + 1);
76         tree[x].merge(tree[x * 2], tree[x * 2 + 1]);
77     }
78     void modify(int st, int ed, int x, int c) // 区间修改值
79     {
80         if (tree[x].l >= st && tree[x].r <= ed)
81         {
82             tree[x].nl = tree[x].nr = c;
83             tree[x].num = 1;
84             tree[x].lazy = c;

```

```

83         return;
84     }
85     if (tree[x].lazy != -1)
86     {
87         tree[x*2].lazy=tree[x*2+1].lazy=tree[x*2].nl=tree[x*2].nr=tree[x*2+1].nl=
            tree[x*2+1].nr=tree[x].lazy;
88         tree[x * 2].num = tree[x * 2 + 1].num = 1;
89         tree[x].lazy = -1;
90     }
91     int mid = tree[x].mid();
92     if (mid >= st) modify(st,ed,x * 2,c);
93     if (ed > mid) modify(st,ed,x * 2 + 1,c);
94     tree[x].merge(tree[x * 2],tree[x * 2 + 1]);
95 }
96 node query(int st,int ed,int x,char c)//查询区间
97 {
98     if (tree[x].l >= st && tree[x].r <= ed)
99     {
100         node temp = tree[x];
101         if (c == 'L') temp.overtorn();//如果是从起始点开始合并
102         return temp;
103     }
104     if (tree[x].lazy != -1)
105     {
106         tree[x*2].lazy=tree[x*2+1].lazy=tree[x*2].nl=tree[x*2].nr=tree[x*2+1].nl=
            tree[x*2+1].nr=tree[x].lazy;
107         tree[x * 2].num = tree[x * 2 + 1].num = 1;
108         tree[x].lazy = -1;
109     }
110     int mid = tree[x].mid();
111     node ans;
112     ans.nl = -1; ans.nr = -1; ans.num = 0;
113     if (mid >= st) ans = query(st,ed,x * 2,c);
114     if (ed > mid)
115     {
116         if (ans.num == 0) ans = query(st,ed,x * 2 + 1,c);
117         else if (c == 'L') ans.merge(query(st,ed,x * 2 + 1,c),ans);
118         else if (c == 'R') ans.merge(ans,query(st,ed,x * 2 + 1,c));
119     }
120     tree[x].merge(tree[x * 2],tree[x * 2 + 1]);
121     return ans;
122 }
123 }seg;
124
125
126 int mapping(int x,int y,int c)//c表示当前是修改还是查询
127 {
128     int fx = top[x],fy = top[y];
129     node ans1,ans2;
130     ans1.nl = ans1.nr = ans2.nl = ans2.nr = -1;//ans1表示从起始点开始合并, ans2表示从终
        点开始合并
131     ans1.num = ans2.num = 0;
132     while (fx != fy)

```

```

133 {
134     if (d[fx] > d[fy])
135     {
136         if (c != -1) seg.modify(dfn[fx],dfn[x],1,c);
137         else
138         {
139             if (ans1.num == 0) ans1 = seg.query(dfn[fx],dfn[x],1,'L'); //如果是从起
                点开始合并
140             else ans1.merge(ans1,seg.query(dfn[fx],dfn[x],1,'L'));
141         }
142         x = fa[fx];
143         fx = top[x];
144     }
145     else
146     {
147         if (c != -1) seg.modify(dfn[fy],dfn[y],1,c);
148         else
149         {
150             if (ans2.num == 0) ans2 = seg.query(dfn[fy],dfn[y],1,'R'); //如果是从终点
                开始合并
151             else ans2.merge(seg.query(dfn[fy],dfn[y],1,'R'),ans2);
152         }
153         y = fa[fy];
154         fy = top[y];
155     }
156 }
157 if (x == y) //如果两条路径相交在同一个点
158 {
159     if (c != -1) return 0;
160     if (ans1.num == 0) return ans2.num;
161     else if (ans2.num == 0) return ans1.num;
162     ans1.merge(ans1,ans2);
163     return ans1.num;
164 }
165 if (d[x] < d[y]) //y在下面
166 {
167     if (c != -1)
168     {
169         seg.modify(dfn[x] + 1,dfn[y],1,c);
170         return 0;
171     }
172     if (ans1.num == 0) ans1 = seg.query(dfn[x] + 1,dfn[y],1,'R');
173     else ans1.merge(ans1,seg.query(dfn[x] + 1,dfn[y],1,'R'));
174     if (ans2.num == 0) return ans1.num;
175     else ans1.merge(ans1,ans2);
176 }
177 else //x在下面
178 {
179     if (c != -1)
180     {
181         seg.modify(dfn[y] + 1,dfn[x],1,c);
182         return 0;
183     }

```

```

184     if (ans1.num == 0) ans1 = seg.query(dfn[y] + 1, dfn[x], 1, 'L');
185     else ans1.merge(ans1, seg.query(dfn[y] + 1, dfn[x], 1, 'L'));
186     if (ans2.num != 0) ans1.merge(ans1, ans2);
187 }
188 return ans1.num;
189 }
190
191 int main()
192 {
193     int n, p;
194     while (scanf("%d%d", &n, &p) != EOF)
195     {
196         for (int i = 1, u, v, t; i <= n - 1; i++)
197         {
198             scanf("%d%d%d", &u, &v, &t);
199             w[make_pair(u, v)] = t;
200             w[make_pair(v, u)] = t;
201             g[u].push_back(v);
202             g[v].push_back(u);
203         }
204         dfs1(1);
205         dfs2(1, 1);
206         char c[10];
207         seg.build(1, n, 1);
208         while (p--)
209         {
210             scanf("%s", &c);
211             if (c[0] == 'Q')
212             {
213                 int u, v;
214                 scanf("%d%d", &u, &v);
215                 if (u == v) printf("0\n");
216                 else printf("%d\n", mapping(u, v, -1));
217             }
218             else if (c[0] == 'C')
219             {
220                 int u, v, t;
221                 scanf("%d%d%d", &u, &v, &t);
222                 int temptemp = mapping(u, v, t);
223             }
224         }
225         for (int i = 1; i <= n; i++) g[i].clear();
226         mem(son); mem(size1); mem(d); mem(fa); mem(dfn); mem(top); mem(aa);
227         num = 0;
228         w.clear();
229     }
230 }

```

### 1.3 点分治

```

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

```

```
3  #define ll long long
4  const int MAXN = 2e4 + 10;
5
6  struct node{
7      int v,w,nx;
8  }edge[MAXN<<1];
9
10 int tot,head[MAXN];
11
12 inline void add(int u,int v,int w)
13 {
14     edge[++tot].v = v;
15     edge[tot].w = w;
16     edge[tot].nx = head[u];
17     head[u] = tot;
18 }
19
20 int sz[MAXN],maxsubtree,root,size,vis[MAXN]; //vis标记当前这个点有没有被作为分治点
21
22 void getroot(int x,int fa) //找重心，把它作为当前树的根，是根据定义来求的
23 {
24     int maxn = 0;
25     sz[x] = 1;
26     for (int i = head[x]; i; i = edge[i].nx)
27     {
28         int v = edge[i].v;
29         if (v == fa || vis[v]) continue;
30         getroot(v,x);
31         sz[x] += sz[v];
32         maxn = max(maxn,sz[v]); //记录以x为根的最大子树大小
33     }
34     maxn = max(maxn,size - sz[x]); //当以x为根时，其祖先结点也变成了x的子树
35     if (maxn < maxsubtree) //寻找最大子树最小
36     {
37         maxsubtree = maxn;
38         root = x;
39     }
40 }
41
42 ll ans = 0,sum[4];
43
44 void dfs1(int x,int fa,int st) //统计子树中到分治点（重心）对3取模的路径有几条
45 {
46     for (int i = head[x]; i; i = edge[i].nx)
47     {
48         int v = edge[i].v;
49         if (v == fa || vis[v]) continue;
50         sum[(st + edge[i].w) % 3]++;
51         dfs1(v,x,(st + edge[i].w) % 3);
52     }
53 }
54
55 ll cal(int x,int st) //计算路径数量
```



```

56 {
57     sum[0] = sum[1] = sum[2] = 0;
58     sum[st]++;
59     ll res = 0;
60     dfs1(x,0,st); //以重心为起始点，跑其子树的dfs，得到到重心的路径权值%3为0,1,2的数量
61     res = sum[1] * sum[2] * 2 + sum[0] * sum[0]; //统计答案
62     return res;
63 }
64
65 void dfs(int x)
66 {
67     ans += cal(x,0); //计算经过当前点的路径的数量
68     vis[x] = 1;
69     for (int i = head[x]; i; i = edge[i].nx)
70     {
71         int v = edge[i].v;
72         if (vis[v]) continue;
73         ans -= cal(v,edge[i].w); //容斥，去掉统计答案时子树中互相组成路径没有经过重心（分治点）的路径数
74         //之所以要加上edge[i].w是因为以重心为根，计算子树到重心的路径权值的时候，加上了这条边
75         //所以以其儿子点为根计算时也要加上这条边
76         maxsubtree = 2147483647; size = sz[v];
77         getroot(v,0); //寻找子树的重心
78         dfs(root); //以子树的重心为根分治子树
79     }
80 }
81
82 int main()
83 {
84     int n;
85     scanf("%d",&n);
86     for (int i = 1,u,v,w;i<n;i++)
87     {
88         scanf("%d%d%d",&u,&v,&w);
89         add(u,v,w % 3);
90         add(v,u,w % 3);
91     }
92     maxsubtree = 2147483647; size = n;
93     getroot(1,0); //先以1为根，寻找树的重心
94     dfs(root); //依照重心分治
95     ll t = (ll)n * (ll)n;
96     ll g = __gcd(ans,t);
97     printf("%lld/%lld",ans / g,t / g);
98     return 0;
99 }

```

## 1.4 长链剖分

```

1 #include<bits/stdc++.h>
2 using namespace std;
3

```

```
4  const int MAXN = 1e6 + 10;
5
6  struct node{
7      int u,v,nx;
8  }edge[MAXN<<1];
9
10 int tot,head[MAXN];
11
12 inline void add(int u,int v)
13 {
14     edge[++tot].v = v;
15     edge[tot].nx = head[u];
16     head[u] = tot;
17 }
18
19 int len[MAXN],son[MAXN],fa[MAXN],num;
20 //找长链
21 void dfs1(int x)
22 {
23     len[x] = 1;
24     for (int i = head[x];i;i = edge[i].nx)
25     {
26         int v = edge[i].v;
27         if (v == fa[x]) continue;
28         fa[v] = x;
29         dfs1(v);
30         len[x] = max(len[x],len[v] + 1);
31         if (len[v] > len[son[x]]) son[x] = v;
32     }
33 }
34
35 int *dp[MAXN],tmp[MAXN],*id = tmp;
36 int ans[MAXN];
37
38 void dfs2(int x)
39 {
40     dp[x][0] = 1;
41     if (son[x])//沿长链dp
42     {
43         dp[son[x]] = dp[x] + 1;
44         dfs2(son[x]);
45         ans[x] = ans[son[x]] + 1;
46     }
47     for (int i = head[x];i;i = edge[i].nx)
48     {
49         int v = edge[i].v;
50         if (v == fa[x] || v == son[x]) continue;
51         dp[v] = id;
52         id += len[v];
53         dfs2(v);
54         //合并两条链
55         for (int j = 0;j<len[v];j++)
56         {
```

```

57         dp[x][j + 1] += dp[v][j];
58         if (dp[x][j+1] > dp[x][ans[x]] || (dp[x][j+1] == dp[x][ans[x]] && j + 1 <
            ans[x]))
59             ans[x] = j + 1;
60     }
61 }
62 if (dp[x][ans[x]] == 1) ans[x] = 0;
63 }
64
65 int main()
66 {
67     int n;
68     cin>>n;
69     for (int i = 1,u,v;i<n;i++)
70     {
71         scanf("%d%d",&u,&v);
72         add(u,v);
73         add(v,u);
74     }
75     dfs1(1);
76     dp[1] = id; id += len[1];
77     dfs2(1);
78     for (int i = 1;i<=n;i++) printf("%d\n",ans[i]);
79 }

```

## 2 线段树

### 2.1 zkw 线段树

```

1  //zkw单点修改，区间查询
2  #include<iostream>
3  #include<stdio.h>
4  #include<cstring>
5  #include<math.h>
6  using namespace std;
7  const int NUM = 50000;
8
9  int m,tree[NUM * 4];
10
11 void modify(int n,int v)
12 {
13     for (tree[n += m] += v,n>>=1;n>>=1) tree[n] = tree[n * 2] + tree[n * 2 + 1];
14 }
15
16 int query(int st,int ed)
17 {
18     int ans = 0;
19     for (int l = m + st - 1,r = m + ed + 1;l ^ r ^ 1;l>>=1,r>>=1)
20     {
21         if (~l & 1) ans += tree[l ^ 1];
22         if (r & 1) ans += tree[r ^ 1];
23     }

```

```
24     return ans;
25 }
26
27 int main()
28 {
29     memset(tree,0,sizeof(tree));
30     int n;
31     cin>>n;
32     m = log(n) / log(2);
33     if (pow(2,m) < n) m++;
34     m = pow(2,m);
35     for (int i = m + 1;i<=m + n;i++) scanf("%d",&tree[i]);
36     for (int i = m - 1;i;i--) tree[i] = tree[i * 2] + tree[i * 2 + 1];
37     char c[10];
38     while (1)
39     {
40         //E结束 Query询问x~y的区间和 Add在x处+y Sub在x处-y
41         int x,y;
42         scanf("%s",c);
43         if (c[0] == 'E') break;
44         scanf("%d%d",&x,&y);
45         if (c[0] == 'Q') printf("%d\n",query(x,y));
46         else if (c[0] == 'A') modify(x,y);
47         else modify(x,-y);
48     }
49     return 0;
50 }
```

## 2.2 树状数组维护区间和

```
1 //树状数组维护区间和
2 #include<iostream>
3 #define ll long long
4 using namespace std;
5
6 int lowbit(int x)
7 {
8     return x&-x;
9 }
10
11 int n;
12 ll a[50001] = {},b[50001] = {};
13
14 void change(ll x,ll w)
15 {
16     while (x <= n)
17     {
18         b[x] += w;
19         x +=lowbit(x);
20     }
21 }
22
```

```

23 ll sum(int x)
24 {
25     ll num = 0;
26     while (x > 0)
27     {
28         num+=b[x];
29         x -=lowbit(x);
30     }
31     return num;
32 }
33
34 int main()
35 {
36     scanf("%d",&n);
37     for (int i = 1;i<=n;i++)
38     {
39         scanf("%d",&a[i]);
40         a[i] += a[i-1];
41         b[i] = a[i] - a[i - lowbit(i)];
42     }
43     char c[10] = {};
44     scanf("%s",c);
45     while (c[0] != 'E')
46     {
47         //E结束 Query询问x~y的区间和 Add在x处+y Sub在x处-y
48         int i,j;
49         scanf("%d%d",&i,&j);
50         if (c[0] == 'Q')
51         {
52             ll t = sum(j) - sum(i-1) ;
53             printf("%lld\n",t);
54         }
55         else if (c[0] == 'A') change(i,j);
56         else change(i,-j);
57         scanf("%s",c);
58     }
59 }

```

### 2.3 树状数组维护区间最大值

```

1 //树状数组维护区间最大值（不可有修改操作）
2 #include<bits/stdc++.h>
3 #define ll long long
4 using namespace std;
5 const int MAXN = 3e6 + 10;
6 #define ll long long
7
8 ll treemax[MAXN],a[MAXN];
9 int n;
10
11 inline int lowbit(int x)
12 {

```

```
13     return x & -x;
14 }
15
16 void insert(ll v,int x)
17 {
18     while (x <= n)
19     {
20         treemax[x] = max(v,treemax[x]);
21         x += lowbit(x);
22     }
23 }
24
25 ll query(int l,int r)
26 {
27     ll res = -9e18;
28     while (r >= 1)
29     {
30         if (r - lowbit(r) < 1)
31         {
32             res = max(res,a[r]);
33             r--;
34         }
35         else
36         {
37             res = max(res,treemax[r]);
38             r -=lowbit(r);
39         }
40     }
41     return res;
42 }
43
44 int main()
45 {
46     int m;
47     scanf("%d",&n,&m);
48     fill(treemax,treemax+MAXN,-9e18);
49     for (int i = 1;i<=n;i++)
50     {
51         scanf("%lld",&a[i]);
52         insert(a[i],i);
53     }
54     int l,r;
55     while (m--)
56     {
57         scanf("%d",&l,&r);
58         printf("%lld\n",query(l,r));
59     }
60     return 0;
61 }
```

## 2.4 主席树

```

1 //询问区间第k大
2 #include<bits/stdc++.h>
3 using namespace std;
4
5 const int MAXN = 2e5 + 10;//数组大小
6
7 int mapping[MAXN],root[MAXN];
8
9 struct node{
10     int val,index,num;
11 }a[MAXN];
12
13 bool cmp1(node i,node j){return i.val < j.val;}
14 bool cmp2(node i,node j){return i.index < j.index;}
15
16 struct tree{
17     int l[MAXN<<5],r[MAXN<<5],num[MAXN<<5],numNode;
18     int update(int pre,int L,int R,int x)//新增一条从根节点到叶子的链
19     {
20         int cur = ++numNode;
21         l[cur] = l[pre],r[cur] = r[pre],num[cur] = num[pre] + 1;
22         //延用上一个结点的左右子节点
23         //num存从序列开始到现在插入的这个值的位置,该节点下面由多少个值
24         if (L < R)
25         {
26             if (x <= (L + R) / 2) l[cur] = update(l[pre],L,(L+R)/2,x);
27             //如果x的值在左边,则新建一个左结点连接在该节点上
28             else r[cur] = update(r[pre],(L + R) / 2 + 1,R,x);
29             //如果x的值在右边,则新建一个右结点连接在该节点上
30         }
31         return cur;
32     }
33     int query(int st,int ed,int L,int R,int k)
34     {
35         int sum = num[l[ed]] - num[l[st]];//区间内的数值在左边的个数
36         if (L == R) return L;
37         if (sum >= k) return query(l[st],l[ed],L,(R + L) / 2,k);
38         //左边的值的个数比k大,则第k大的值在左边
39         else return query(r[st],r[ed],(R + L) / 2 + 1,R,k - sum);
40         //否则第k大的值在右边
41     }
42 }zxs;
43
44 int main()
45 {
46     int n,m;
47     scanf("%d%d",&n,&m);
48     for (int i = 1;i<=n;i++) scanf("%d",&a[i].val),a[i].index = i;
49     sort(a+1,a+n+1,cmp1);
50     int count = 0;
51     for (int i = 1;i<=n;i++)
52     {
53         if (a[i].val != a[i-1].val || i == 1) mapping[a[i].num = ++count] = a[i].val;

```

```

54     else a[i].num = count;
55 }
56 sort(a+1,a+n+1,cmp2);
57 //离散化, mapping表示离散化后的值对应原来的值是哪个
58 for (int i = 1;i<=n;i++)
59     root[i] = zxs.update(root[i-1],1,count,a[i].num);//建树
60 while (m--)
61 {
62     int l,r,k;
63     scanf("%d%d%d",&l,&r,&k);
64     printf("%d\n",mapping[zxs.query(root[l-1],root[r],1,count,k)]);
65     //返回[l,r]第k大的值
66 }
67 return 0;
68 }

```

## 3 图论

### 3.1 最短路

#### 3.1.1 dijkstra 求最短路

```

1 //有向图
2 #include<bits/stdc++.h>
3 using namespace std;
4 typedef long long ll;
5 #define PII pair<ll,ll>
6
7 const int Vertex_MAXN = 1e4 + 10;
8 const int Edge_MAXN = 5e5 + 10;
9 const ll inf = 9e18;
10
11 struct node{
12     int v,nx;
13     ll w;
14 }edge[Edge_MAXN];
15
16 int head[Vertex_MAXN],tot,n,m,s;
17 ll dis[Vertex_MAXN];
18
19 inline void add(int u,int v,ll w)
20 {
21     edge[++tot].v = v;
22     edge[tot].w = w;
23     edge[tot].nx = head[u];
24     head[u] = tot;
25 }
26
27 void dij(int s) {
28     priority_queue<PII,vector<PII>,greater<PII>> q;
29     fill(dis,dis+n+1,inf);
30     dis[s] = 0;

```



```

31     q.push(PII(0,s));
32     while (!q.empty())
33     {
34         ll d = q.top().first;
35         int u = q.top().second;
36         q.pop();
37         if (d != dis[u]) continue;
38         for (int i = head[u]; i; i = edge[i].nx)
39         {
40             PII y(d+edge[i].w,edge[i].v);
41             if (dis[y.second]>y.first)
42             {
43                 dis[y.second] = y.first;
44                 q.push(y);
45             }
46         }
47     }
48 }
49
50 int main()
51 {
52     scanf("%d%d%d",&n,&m,&s);
53     for (int i = 1,u,v,w;i<=m;i++)
54     {
55         scanf("%d%d%d",&u,&v,&w);
56         add(u,v,w);
57     }
58     dij(s);
59     for (int i = 1;i<=n;i++) printf("%lld ",dis[i]);
60     return 0;
61 }

```

### 3.1.2 spfa 求最短路

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int MAXN = 1e4 + 10;
5
6  struct node{
7      int v,nx,w;
8  }edge[500010];
9
10 int head[MAXN],tot;
11
12 inline void add(int u,int v,int w)
13 {
14     edge[++tot].v = v;
15     edge[tot].nx = head[u];
16     edge[tot].w = w;
17     head[u] = tot;
18 }

```

```
19
20 int dis[MAXN],vis[MAXN],s;
21
22 void spfa()
23 {
24     queue<int> q;
25     fill(dis,dis+MAXN,2147483647);
26     q.push(s);
27     vis[s] = 1;
28     dis[s] = 0;
29     while (!q.empty())
30     {
31         int u = q.front();
32         q.pop();
33         vis[u] = 0;
34         for (int i = head[u];i;i = edge[i].nx)
35         {
36             int v = edge[i].v;
37             if (dis[v] > dis[u] + edge[i].w)
38             {
39                 dis[v] = dis[u] + edge[i].w;
40                 if (!vis[v]) vis[v] = 1,q.push(v);
41             }
42         }
43     }
44 }
45
46 int main()
47 {
48     int n,m;
49     scanf("%d%d",&n,&m,&s);
50     for (int i = 1,u,v,w;i<=m;i++)
51     {
52         scanf("%d%d%d",&u,&v,&w);
53         add(u,v,w);
54     }
55     spfa();
56     for (int i = 1;i<=n;i++) printf("%d ",dis[i]);
57     return 0;
58 }
```

## 3.2 网络流

### 3.2.1 dinic

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 const int MAXN = 2e3;
5 const int inf = 2147483647;
6
7 int s,t;
```

```
8
9 struct node{
10     int v,w,nx;
11 }edge[MAXN];
12
13 int tot,head[MAXN];
14
15 inline void add(int u,int v,int w)
16 {
17     edge[tot].v = v;
18     edge[tot].nx = head[u];
19     edge[tot].w = w;
20     head[u] = tot++;
21 }
22
23 int dis[MAXN];
24 queue<int> q;
25
26 bool bfs()
27 {
28     memset(dis,-1,sizeof(dis));
29     dis[s] = 0;
30     q.push(s);
31     while (!q.empty())
32     {
33         int u = q.front();
34         q.pop();
35         for (int i = head[u];i != -1;i = edge[i].nx)
36         {
37             int v = edge[i].v;
38             if (dis[v] == -1 && edge[i].w > 0)
39             {
40                 dis[v] = dis[u] + 1;
41                 q.push(v);
42             }
43         }
44     }
45     return dis[t] != -1;
46 }
47
48 int dfs(int u,int exp)
49 {
50     if (u == t) return exp;
51     int flow = 0,tmp = 0;
52     for (int i = head[u];i != -1;i = edge[i].nx)
53     {
54         int v = edge[i].v;
55         if (dis[v] == dis[u] + 1 && edge[i].w > 0)
56         {
57             tmp = dfs(v,min(exp,edge[i].w));
58             if (!tmp) continue;
59             exp -= tmp;
60             flow += tmp;
```

```

61         edge[i].w -= tmp;
62         edge[i^1].w += tmp;
63         if (!exp) break;
64     }
65 }
66 return flow;
67 }
68
69 void dinic()
70 {
71     int ans = 0;
72     while (bfs()) ans += dfs(s,inf);
73     printf("%d",ans);
74 }
75
76 int main()
77 {
78     int n;
79     scanf("%d",&n);
80     memset(head,-1,sizeof(head));
81     for (int i = 1,u,v,w;i<=n;i++)
82     {
83         scanf("%d%d%d",&u,&v,&w);
84         add(u,v,w);
85         add(v,u,0);
86     }
87     s = 65,t = 90;
88     dinic();
89     return 0;
90 }

```

### 3.3 二分图

#### 3.3.1 二分图 KM 板子

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int MAXN = 3e2 + 10;
5
6  int a[MAXN][MAXN],d,n;
7  int c_girl[MAXN],c_boy[MAXN];//记录匹配对象
8  int ex_girl[MAXN],ex_boy[MAXN];//记录男生和女生的期望
9  bool vis_girl[MAXN],vis_boy[MAXN];//记录每一轮匹配匹配过的女生和男生
10 int slack[MAXN]; // 记录每个汉子如果能被妹子倾心最少还需要多少期望值
11
12 bool dfs(int u)//匈牙利算法找增广路径
13 {
14     vis_girl[u] = 1;
15     for (int i = 1;i<=n;i++)
16     {
17         if (vis_boy[i] == 0)//每一轮匹配 每个男生只尝试一次

```

```

18     {
19         int num = ex_girl[u] + ex_boy[i] - a[u][i];
20         if (num == 0) //如果符合要求
21         {
22             vis_boy[i] = 1;
23             if (c_boy[i] == 0 || dfs(c_boy[i])) //找到一个没有匹配的男生 或者该男生的
                妹子可以找到其他人
24             {
25                 c_girl[u] = i;
26                 c_boy[i] = u;
27                 return 1;
28             }
29         }
30         else slack[i] = min(slack[i], num); //slack可以理解为该男生要得到女生的倾心 还
            需多少期望值 取最小值
31     }
32 }
33 return 0;
34 }
35
36 int KM()
37 {
38     memset(ex_girl, 0, sizeof(ex_girl)); // 每个女生的初始期望值是与她相连的男生最大的好感度
39     memset(ex_boy, 0, sizeof(ex_boy)); // 初始每个男生的期望值为0
40     memset(c_girl, 0, sizeof(c_girl)); // 初始每个男生都没有匹配的女生
41     memset(c_boy, 0, sizeof(c_boy)); // 初始每个女生都没有匹配的男生
42     for (int i = 1; i <= n; i++)
43     for (int j = 1; j <= n; j++)
44         ex_girl[i] = max(ex_girl[i], a[i][j]); // 每个女生的初始期望值是与她相连的男生最大的好
            感度
45
46     for (int i = 1; i <= n; i++) // 尝试为每一个女生解决归宿问题
47     {
48         fill(slack, slack + MAXN, 2147483647); // 因为要取最小值 初始化为无穷大
49         while (true) // 为每个女生解决归宿问题的方法是：如果找不到就降低期望值，直到找到为
            止
50         {
51             memset(vis_girl, 0, sizeof(vis_girl));
52             memset(vis_boy, 0, sizeof(vis_boy)); // 记录每轮匹配中男生女生是否被尝试匹配过
53             if (dfs(i)) break; // 找到归宿 退出
54
55             // 如果不能找到 就降低期望值
56             // 最小可降低的期望值
57             d = 2147483647;
58             for (int j = 1; j <= n; j++)
59             {
60                 if (!vis_boy[j]) d = min(d, slack[j]);
61             }
62             for (int j = 1; j <= n; j++)
63             {
64                 if (vis_girl[j] == 1) ex_girl[j] -= d; //所有访问过的女生降低期望值
65                 if (vis_boy[j] == 1) ex_boy[j] += d; //所有访问过的男生增加期望值
66                 else slack[j] -= d; //没有访问过的boy 因为girl们的期望值降低，距离得到女生

```

```

        }
    }
}
// 匹配完成 求出所有配对的好感度的和
int res = 0;
for (int i = 1; i <= n; i++) res += a[i][c_girl[i]];
return res;
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            scanf("%d", &a[i][j]);
    cout << KM() << '\n';
}

```

### 3.3.2 二分图最大匹配

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int MAXN = 1e3;
5
6  int vis[MAXN][MAXN], cx[MAXN], cy[MAXN], n, m;
7  bool check[MAXN];
8
9  bool dfs(int u)
10 {
11     for (int v = 1; v <= n; v++)
12     {
13         if (vis[u][v] && !check[v]) // 邻接矩阵存储, 如果 u-v 之间有一条路 而且 v 没有被访问过
14         {
15             check[v] = 1; // 标记 v 已经访问
16             if (cy[v] == -1 || dfs(cy[v]))
17             {
18                 cx[u] = v;
19                 cy[v] = u;
20                 return 1;
21             }
22         }
23     }
24     return 0;
25 }
26
27 int maxmatch()
28 {
29     int ans = 0;
30     memset(cx, -1, sizeof(cx));
31     memset(cy, -1, sizeof(cy));

```

```

32     for (int i = 1; i <= n; i++) //字典序从大到小
33     {
34         if (cx[i] == -1)
35         {
36             memset(check, 0, sizeof(check));
37             ans += dfs(i);
38         }
39     }
40     return ans;
41 }
42
43 int main()
44 {
45     cin >> n;
46     for (int i = 1; i <= n; i++)
47     for (int j = 1; j <= n; j++)
48         cin >> vis[i][j];
49     int ans = maxmatch();
50     cout << ans;
51 }

```

### 3.4 tarjan

#### 3.4.1 tarjan 求环的个数和点的染色

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int MAXN = 2e2 + 10;
4
5  vector<int> g[MAXN];
6  int dfn[MAXN], low[MAXN], s[MAXN], vis[MAXN], num, slen, scnt, col[MAXN];
7
8  void tarjan(int u)
9  {
10     low[u] = dfn[u] = ++num;
11     //low存u的子树里所能到达的dfn最小的点
12     s[++slen] = u; //s为栈
13     vis[u] = 1; //标记u已经放到了栈里
14     for (int i = 0; i < g[u].size(); i++)
15     {
16         int v = g[u][i];
17         if (!dfn[v]) //如果v没有访问过
18         {
19             tarjan(v);
20             low[u] = min(low[u], low[v]);
21         }
22         else if (vis[v]) low[u] = min(low[u], dfn[v]);
23         //一旦遇到已入栈的点，就将该点作为连通量的根
24         //这里用dfn[e[i].v]更新的原因是：这个点可能
25         //已经在另一个强连通分量中了但暂时尚未出栈
26         //所以now不一定能到达low[e[i].v]但一定能到达
27         //dfn[e[i].v].

```

```

28     }
29     if (dfn[u] == low[u])
30     {
31         scnt++; //环的数量
32         do
33         {
34             vis[s[slen]] = 0; //出栈
35             col[s[slen]] = scnt; //染色
36         } while (s[slen--] != u);
37     }
38 }
39
40 int main()
41 {
42     int p;
43     cin >> p;
44     while (p--)
45     {
46         int n, m;
47         scanf("%d%d", &n, &m);
48         for (int i = 1, u, v; i <= m; i++)
49         {
50             scanf("%d%d", &u, &v);
51             g[u].push_back(v);
52         }
53         for (int i = 0; i < n; i++)
54             if (dfn[i] == 0) tarjan(i);
55         for (int i = 0; i < n; i++) g[i].clear();
56         printf("%d\n", scnt);
57         memset(dfn, 0, sizeof(dfn));
58         memset(low, 0, sizeof(low));
59         memset(s, 0, sizeof(s));
60         memset(vis, 0, sizeof(vis));
61         scnt = num = 0;
62     }
63 }

```

### 3.4.2 tarjan 求无向图割边和割点

```

1 //tarjan求无向图割边和割点
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 const int MAXN = 2e4 + 10;
6
7 vector<int> g[MAXN];
8
9 int dfn[MAXN], low[MAXN], num, fa[MAXN];
10
11 struct EDGE{
12     EDGE(int a = 0, int b = 0):u(a),v(b){}
13     int u,v;

```



```
14 };
15
16 vector<EDGE> cutedge;
17 vector<int> cutnode;
18
19 bool cmp(EDGE a,EDGE b) {return a.u == b.u?a.v<b.v:a.u<b.u;}
20
21 void tarjan(int u)
22 {
23     dfn[u] = low[u] = ++num;
24     bool flag = false;
25     int son = 0;
26     for (int i = 0;i<g[u].size();i++)
27     {
28         int v = g[u][i];
29         if (v == fa[u]) continue;
30         if (!dfn[v])
31         {
32             son++;
33             fa[v] = u;
34             tarjan(v);
35             low[u] = min(low[u],low[v]);
36             if (low[v] >= dfn[u]) flag = true;
37             //判断是否存在子节点只能通过u访问到u的祖先
38             if (low[v] > dfn[u]) cutedge.push_back(EDGE(min(u,v),max(v,u)));
39             //判断割边
40         }
41         else low[u] = min(low[u],dfn[v]);
42     }
43     if ((fa[u] == 0 && son >= 2) || (fa[u] != 0 && flag)) cutnode.push_back(u);
44     //判断割点
45     //u是根节点且u有两个连通分量则u是割点
46     //u不是根节点，u存在一个子节点只能通过u访问到u的祖先
47 }
48
49 int main()
50 {
51     int n,m;
52     scanf("%d%d",&n,&m);
53     for (int i = 1,u,v;i<=m;i++)
54     {
55         scanf("%d%d",&u,&v);
56         g[u].push_back(v);
57         g[v].push_back(u);
58     }
59     for (int i = 1;i<=n;i++)
60     if (dfn[i] == 0) tarjan(i);
61     sort(cutedge.begin(),cutedge.end(),cmp);
62     sort(cutnode.begin(),cutnode.end());
63     //割点从小到大输出
64     //割边(u,v), u<v, 按照u为第一关键字, v为第二关键字排序
65     if (cutnode.size() == 0) printf("Null\n");
66     else
```

```

67     {
68         printf("%d",cutnode[0]);
69         for (int i = 1;i<cutnode.size();i++) printf(" %d",cutnode[i]);
70     }
71     puts("");
72     for (int i = 0;i<cutedge.size();i++)
73     {
74         printf("%d %d\n",cutedge[i].u,cutedge[i].v);
75     }
76     return 0;
77 }

```

## 4 字符串

### 4.1 Manacher

```

1  #include <bits/stdc++.h>
2  #define maxn 2000005
3  using namespace std;
4  int mp[maxn];
5  string str;
6  char c[maxn];
7  void Manacher(string s,int len){
8      int l=0,R=0,C=0;;
9      c[l++]='$', c[l++]='#';
10     for(int i=0;i<len;i++){
11         c[l++]=s[i], c[l++]='#';
12     }
13     for(int i=0;i<l;i++){
14         mp[i]=R>i?min(mp[2*C-i],R-i):1;
15         while(i+mp[i]<l&&i-mp[i]>0){
16             if(c[i+mp[i]]==c[i-mp[i]]) mp[i]++;
17             else break;
18         }
19         if(i+mp[i]>R){
20             R=i+mp[i], C=i;
21         }
22     }
23 }
24 int main()
25 {
26     int cnt=0;
27     while(cin>>str){
28         if(str=="END") break;
29         int len=str.length();
30         Manacher(str,len);
31         int ans=0;
32         for(int i=0;i<2*len+4;i++){
33             ans=max(ans,mp[i]-1);
34         }
35         printf("Case %d: %d\n",++cnt,ans);
36     }

```

```
37 |   return 0;  
38 | }
```