

华南师范大学
ACM-ICPC 集训队
DeDeRong



2019 年 10 月 17 日

目录

| | | | |
|--------------------------------|-----------|------------------------------------|-----------|
| 1 图论 | 3 | 2.5 主席树 | 25 |
| 1.1 最短路径 | 3 | 2.5.1 区间第 K 大 | 25 |
| 1.1.1 Dijkstra | 3 | 2.5.2 动态区间第 K 大 (主席树套树状数组) ZOJ2112 | 27 |
| 1.1.2 Dijkstra 优化 | 3 | | |
| 1.1.3 Floyd | 3 | 3 DP | 29 |
| 1.1.4 Bellman-Ford | 4 | 3.1 背包 | 29 |
| 1.1.5 SPFA | 4 | 3.2 最长公共递增子序列 | 30 |
| 1.2 次短路 | 5 | 3.3 最长公共子序列 | 30 |
| 1.3 LCA | 6 | 4 字符串 | 31 |
| 1.3.1 倍增 | 6 | 5 数学 | 31 |
| 1.3.2 RMQ | 7 | 6 STL | 31 |
| 1.4 强连通分量 | 8 | 7 计算几何 | 31 |
| 1.4.1 Tarjan | 8 | 8 其它 | 31 |
| 1.5 割点 | 8 | | |
| 1.6 桥 | 9 | | |
| 1.7 最大流 | 10 | | |
| 1.7.1 Dinic | 10 | | |
| 1.7.2 Dinic 优化 | 11 | | |
| 1.8 二分图匹配 | 12 | | |
| 1.8.1 匈牙利算法 | 12 | | |
| 1.9 最小生成树 | 13 | | |
| 1.9.1 Prim | 13 | | |
| 1.9.2 kruskal | 13 | | |
| 1.10 次小生成树-POJ1679 | 14 | | |
| 1.11 拓扑排序 | 15 | | |
| 1.12 Floyd 找最小环 | 16 | | |
| 2 数据结构 | 17 | | |
| 2.1 并查集 | 17 | | |
| 2.2 ST 表 | 18 | | |
| 2.3 树状数组 | 18 | | |
| 2.4 线段树 | 18 | | |
| 2.4.1 单点修改 + 区间求和 HDU1166 | 18 | | |
| 2.4.2 区间修改 + 区间求和 POJ3465 | 19 | | |
| 2.4.3 单点修改 + 区间最值 HDU1754 | 20 | | |
| 2.4.4 区间染色 + 统计 + 离散化 POJ2528 | 21 | | |
| 2.4.5 线段树 + 扫描线求矩阵覆盖周长 POJ1177 | 23 | | |
| 2.4.6 线段树 + 扫描线求矩阵面积并 HDU1542 | 24 | | |

1 图论

1.1 最短路径

1.1.1 Dijkstra

```

1 const int maxn = 1e4;
2 const int inf = 0x3f3f3f3f;
3
4 //d数组用来记录源点s到顶点i的最短距离
5 //v表示该顶点是否在顶点集S中
6 //g邻接矩阵存图, g[i][j]表示i到j的边的权值, 无边时为inf
7 //n为顶点数量
8 int d[maxn], v[maxn];
9 int g[maxn][maxn];
10 int n;
11 void dij(int s)
12 {
13     memset(v, 0, sizeof(v));
14     for(int i=1; i<=n; i++)
15         d[i] = g[s][i];
16     v[s] = 1;
17     for(int i=1; i<=n; i++)
18     {
19         int u = 0;
20         for(int j=1; j<=n; j++)
21         {
22             if(!v[j] && (u==0 || d[j] < d[u]))
23                 u = j;
24         }
25         if(u==0) return ;
26         v[u] = 1;
27         for(int j=1; j<=n; j++)
28         {
29             d[j] = min(d[j], d[u]+g[u][j]);
30         }
31     }
32 }

```

1.1.2 Dijkstra 优化

```

1 const int maxn = 1e4;

```

```

2 const int inf = 0x3f3f3f3f;
3 typedef pair<int, int> P; //first表示最短距离, second表示顶点编号
4 //边: to表示这条边指向的顶点, 权值为w
5 struct Edge
6 {
7     int to, w;
8 };
9 //用vector实现邻接表
10 vector<Edge> g[maxn];
11 int d[maxn]; //记录源点到顶点i的最短距离
12 int n;
13
14 void dij(int s)
15 {
16     priority_queue<P, vector<P>, greater<P> > q;
17     memset(d, inf, sizeof(d));
18     d[s] = 0;
19     q.push(P(0, s));
20     while(!q.empty())
21     {
22         P p = q.top();
23         q.pop();
24         int u = p.second;
25         if(d[u] < p.first) continue;
26         for(int i=0; i<g[u].size(); i++)
27         {
28             Edge e = g[u][i];
29             if(d[e.to] > d[u] + e.w)
30             {
31                 d[e.to] = d[u] + e.w;
32                 q.push(P(d[e.to], e.to));
33             }
34         }
35     }
36 }

```

1.1.3 Floyd

```

1 int g[maxn][maxn];
2 int n;
3 void floyd()
4 {

```

```

5   for(int k=1;k<=n;k++)
6       for(int i=1;i<=n;i++)
7           for(int j=1;j<=n;j++)
8               g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
9   }

```

1.1.4 Bellman-Ford

```

1   const int maxn = 1e4;
2   const int inf = 0x3f3f3f3f; //常用于表示无穷大
3
4   //边结构体, 记录u→v的边, 权值为w
5   struct Edge
6   {
7       int u, v, w;
8       Edge(int uu, int vv, int ww) { u=uu; v=vv; w=ww; }
9       Edge(){}
10  }e[maxn];
11  int edgecnt; // 边的数量
12  //加边操作
13  void addEdge(int u, int v, int w)
14  {
15      e[edgecnt++] = Edge(u, v, w);
16  }
17
18  int n; //顶点总数
19  int d[maxn]; //记录最短距离的数组
20
21  //存在负权回路则返回true, 否则返回false
22  bool bellman_ford(int s)
23  {
24      memset(d, inf, sizeof(d));
25      d[s] = 0;
26      //进行n-1次松弛操作, 第n次检查是否含有负权回路
27      for(int i=1;i<=n;i++)
28      {
29          int flag = 0;
30          for(int j=0; j<edgecnt; j++)
31          {
32              Edge t = e[j];
33              int u, v, w;
34              u = t.u; v = t.v; w = t.w;

```

```

35          if(d[v] > d[u] + w)
36          {
37              d[v] = d[u] + w;
38              flag = 1;
39          }
40      }
41      if(!flag) return false;
42      if(i==n && flag) return true;
43  }
44  return false;
45  }

```

1.1.5 SPFA

```

1   const int maxn = 1e4;
2   const int inf = 0x3f3f3f3f; //常用于表示无穷大
3
4   //边结构体, to表示边指向的顶点编号, 权值为w
5   struct Edge
6   {
7       int to, w;
8       Edge(int tt, int ww) { to = tt; w = ww; }
9       Edge(){}
10  };
11  //vector实现的邻接表
12  vector<Edge> g[maxn];
13  int n; //顶点数
14  //d表示最短距离, inq[i]表示结点是否在队列中, 为1则在, cnt[i]记录i入队的次数
15  int d[maxn], inq[maxn], cnt[maxn];
16  //初始化
17  void init()
18  {
19      memset(d, inf, sizeof(d));
20      memset(inq, 0, sizeof(inq));
21      memset(cnt, 0, sizeof(cnt));
22  }
23  //返回true表示存在负权回路
24  bool spfa(int s)
25  {
26      init();
27      d[s] = 0;
28      inq[s] = 1;

```

```

29 cnt[s] = 1;
30 queue<int> q;
31 q.push(s);
32 while(!q.empty())
33 {
34     int u = q.front();
35     inq[u] = 0;
36     q.pop();
37     for(int i=0;i < g[u].size(); i++)
38     {
39         Edge e = g[u][i];
40         if(d[e.to] > d[u] + e.w)
41         {
42             d[e.to] = d[u] + e.w;
43             if(inq[e.to] == 0)
44             {
45                 inq[e.to] = 1;
46                 q.push(e.to);
47                 cnt[e.to]++;
48                 if(cnt[e.to] > n) return true;
49             }
50         }
51     }
52 }
53 return true;
54 }

```

1.2 次短路

```

1 #include <bits/stdc++.h>
2 #define INF 1e16+100
3 #define ms(x,y) memset(x,y,sizeof(x))
4 using namespace std;
5
6 typedef long long ll;
7 typedef pair<ll,ll> P;
8
9 const double pi = acos(-1.0);
10 const int mod = 1e9 + 7;
11 const int maxn = 1e5 + 5;
12
13 struct Edge{

```

```

14     ll to, cost;
15 };
16
17 ll n,m;
18 vector<Edge> a[maxn];
19 ll dist[maxn],dist2[maxn];
20
21 void addedge(ll u,ll v,ll w)
22 {
23     a[u].push_back(Edge{v,w});
24     a[v].push_back(Edge{u,w});
25 }
26
27 void solve()
28 {
29     priority_queue<P, vector<P>, greater<P> >que;
30     //ms(dist,INF);
31     //ms(dist2,INF);
32     fill(dist,dist+n,INF);
33     fill(dist2,dist2+n,INF);
34     dist[0]=0;
35     que.push(P(0,0));
36     while(que.size())
37     {
38         P u=que.top();que.pop();
39         int v=u.second;
40         ll d=u.first;
41         if(dist2[v]<d) continue; //不是次短距离则抛弃
42         for(int i=0;i<a[v].size();i++)
43         {
44             Edge e=a[v][i];
45             ll d2=d+e.cost;
46             if(dist[e.to]>d2) //更新最短
47             {
48                 swap(dist[e.to],d2);
49                 que.push(P(dist[e.to],e.to));
50             }
51             if(dist2[e.to]>d2&&dist[e.to]<d2) //更新次短
52             {
53                 dist2[e.to]=d2;
54                 que.push(P(dist2[e.to],e.to));
55             }
56         }
57     }

```

```

58 printf("%lld\n",dist2[n-1]);
59 }
60
61 int main()
62 {
63     //freopen("in.txt","r",stdin);
64     //freopen("out.txt","w",stdout);
65     int t;
66     scanf("%d",&t);
67     while(t--)
68     {
69         scanf("%lld%lld",&n,&m);
70         for(int i=0;i<n;i++) a[i].clear();
71         for(int i=0;i<m;i++)
72         {
73             ll p,q,w;
74             scanf("%lld%lld%lld",&p,&q,&w);
75             addedge(p-1,q-1,w);
76         }
77         solve();
78     }
79     return 0;
80 }

```

1.3 LCA

1.3.1 倍增

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 #define inf 0x3f3f3f3f
5 typedef pair<int, int> P;
6 const int maxn = 5e5+5;
7 const ll mod = 1e9+7;
8
9 vector<int> son[maxn]; // 存储儿子顶点
10 // dep[i]表示顶点i的深度, n个顶点, m个询问, rt为树根, fa数组用来预处理顶点i
   向上跳 $2^j$ 步之后的顶点
11 int dep[maxn], n, m, rt, fa[maxn][20];
12 int v[maxn]={0}; // 是否访问标记
13
14 // pre是父顶点, rt是当前顶点

```

```

15 void dfs(int pre, int rt)
16 {
17     dep[rt] = dep[pre]+1; // 当前顶点的深度为父顶点加一
18     fa[rt][0] = pre; // 当前顶点向上跳一步为父顶点
19     v[rt] = 1; // 访问
20     // dp预处理
21     for(int i=1;i<=19;i++)
22         fa[rt][i] = fa[fa[rt][i-1]][i-1];
23     // 继续dfs
24     for(int i=0;i<son[rt].size();i++)
25         if(v[son[rt][i]]==0)
26             dfs(rt, son[rt][i]);
27 }
28
29 // 求解LCA(a, b)
30 int lca(int a, int b)
31 {
32     if(dep[a] < dep[b])
33         swap(a, b);
34     for(int i=19;i>=0;i--)
35     {
36         if(dep[a]-dep[b] >= (1<<i))
37         {
38             a = fa[a][i];
39         }
40     }
41     if(a==b)return a;
42     for(int i=19;i>=0;i--)
43     {
44         if(fa[a][i] != fa[b][i])
45         {
46             a = fa[a][i];
47             b = fa[b][i];
48         }
49     }
50     return fa[a][0];
51 }
52
53 int main()
54 {
55     scanf("%d%d%d", &n, &m, &rt);
56     for(int i=1;i<n;i++)
57     {
58         int a, b;

```

```

59     scanf("%d%d", &a, &b);
60     son[a].push_back(b);
61     son[b].push_back(a);
62 }
63 memset(fa, 0, sizeof(fa));
64 memset(dep, inf, sizeof(dep));
65 v[0]=1;
66 dep[0] = 0;
67 dfs(0, rt);
68 for(int i=1;i<=m;i++)
69 {
70     int a, b;
71     scanf("%d%d", &a, &b);
72     printf("%d\n", lca(a, b));
73 }
74 return 0;
75 }

```

1.3.2 RMQ

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 #define inf 0x3f3f3f3f
5 typedef pair<int, int> P;
6 const int maxn = 5e5+5;
7 const ll mod = 1e9+7;
8
9 vector<int> g[maxn]; // 存图
10 // dep记录DFS序中每一个顶点的深度, vis记录DFS序, id记录顶点i第一次在DFS序
   中的位置, st表
11 int dep[maxn<<1]={0}, vis[maxn<<1]={0}, id[maxn]={0}, st[maxn<<1][25];
12 // dfs序计数用, 看代码能理解
13 int dfs_c=1;
14
15 // 父顶点为pre, 当前顶点为now, 当前深度为d
16 void dfs(int pre, int now, int d)
17 {
18     id[now] = dfs_c; // now顶点在DFS序中第一次出现的位置是dfs_c
19     dep[dfs_c] = d; // 记录now的深度
20     vis[dfs_c++] = now; // DFS序中第dfs_c个顶点是now, 同时将dfs_c加一
21     for(int i=0;i<g[now].size();i++)

```

```

22 {
23     if(g[now][i]!=pre)
24     {
25         dfs(now, g[now][i], d+1);
26         vis[dfs_c] = now;
27         dep[dfs_c++] = d;
28     }
29 }
30 }
31
32 // 预处理st表
33 void getSt(int n)
34 {
35     for(int i=1;i<=n;i++)
36     st[i][0] = i;
37     for(int j=1; (1<<j)<=n; j++)
38     {
39         for(int i=1;i+(1<<j)<=n; i++)
40         {
41             int a = st[i][j-1], b = st[i+(1<<(j-1))][j-1];
42             if(dep[a] < dep[b])
43                 st[i][j] = a;
44             else st[i][j] = b;
45         }
46     }
47 }
48
49 // 查询DFS序中区间[l, r]深度最小的顶点在DFS序中的位置
50 int query(int l, int r)
51 {
52     int k = log2(r-l+1);
53     int a = st[l][k];
54     int b = st[r-(1<<k)+1][k];
55     // 返回深度较小的那一个顶点在DFS序中的位置
56     if(dep[a]<dep[b])return a;
57     else return b;
58 }
59
60 // 求LCA(a, b)
61 int lca(int a, int b)
62 {
63     int x, y;
64     x = id[a], y = id[b];
65     if(x>y)return vis[query(y, x)];

```

```

66 else return vis[query(x, y)];
67 }
68
69 // 检查用的
70 void check(int n)
71 {
72     for(int i=1;i<=dfs_c;i++)cout<<dep[i]<<" ";cout<<"\n\n";
73     for(int i=1;i<=dfs_c;i++)cout<<vis[i]<<" ";cout<<"\n\n";
74     for(int i=1;i<=n;i++)cout<<id[i]<<" ";cout<<"\n\n";
75 }
76
77 int main()
78 {
79     int n, m, rt;
80     scanf("%d%d%d", &n, &m, &rt);
81     for(int i=1;i<n;i++)
82     {
83         int a, b;
84         scanf("%d%d", &a, &b);
85         g[a].push_back(b);
86         g[b].push_back(a);
87     }
88     dfs(0, rt, 1);
89     getSt(dfs_c);
90     //check(n);
91     for(int i=1;i<=m;i++)
92     {
93         int a, b;
94         scanf("%d%d", &a, &b);
95         printf("%d\n", lca(a, b));
96     }
97     return 0;
98 }

```

1.4 强连通分量

1.4.1 Tarjan

```

1 vector<int> g[maxn];
2 int low[maxn], dfn[maxn], sta[maxn], ins[maxn], belong[maxn];
3 int cnt, ind, tot; //cnt: 强连通分量的数量, ind: 时间戳, tot: sta的top
4
5 void init()

```

```

6 {
7     memset(ins, 0, sizeof(ins));
8     memset(belong, 0, sizeof(belong));
9     memset(dfn, 0, sizeof(dfn));
10    cnt = ind = tot = 0;
11 }
12
13 void Tarjan(int u)
14 {
15     low[u] = dfn[u] = ++ind;
16     ins[u] = 1;
17     sta[++tot] = u;
18     for(int i=0;i<g[u].size();i++)
19     {
20         int v = g[u][i];
21         if(!dfn[v])
22         {
23             Tarjan(v);
24             low[u] = min(low[u], low[v]);
25         }
26         else if(ins[v])
27             low[u] = min(low[u], dfn[v]);
28     }
29     int p;
30     if(low[u] == dfn[u])
31     {
32         ++cnt;
33         do
34         {
35             p = sta[tot--];
36             belong[p] = cnt;
37             ins[p] = 0;
38         }while(p != u);
39     }
40 }

```

1.5 割点

```

1 vector<int> g[maxn];
2 // iscut[i]: 若顶点i是割点, 则为1, 反之为0
3 int low[maxn], dfn[maxn], iscut[maxn];
4 int ind;

```



```

5
6 void init()
7 {
8     memset(dfn, 0, sizeof(dfn));
9     memset(iscut, 0, sizeof(iscut));
10    ind = 0;
11 }
12
13 // pa为u的父节点, 初始时Tarjan(i, i)
14 void Tarjan(int u, int pa)
15 {
16     int cnt = 0; //用来记录子树的数量
17     low[u] = dfn[u] = ++ind;
18     for(int i=0; i<g[u].size(); i++)
19     {
20         int v = g[u][i];
21         if(!dfn[v])
22         {
23             Tarjan(v, u);
24             low[u] = min(low[u], low[v]);
25             // 若low[v]>=dfn[u], 并且u不是根节点, 则u是割点
26             if(low[v] >= dfn[u] && pa!=u)
27                 iscut[u] = 1;
28             // 若u是根节点, 则cnt++
29             if(u == pa)
30                 cnt++;
31         }
32         else if(v != pa) //若v不等于父节点
33             low[u] = min(low[u], dfn[v]);
34     }
35     if(cnt>=2 && u==pa) //根节点子树数量大于等于2, 则为割点
36         iscut[u] = 1;
37 }

```

1.6 桥

```

1 // 用链式前向星来存储边
2 struct Edge
3 {
4     // iscut表示是否为桥
5     int to, next, iscut;
6 }e[maxn*maxn*2];

```

```

7
8 int head[maxn], low[maxn], dfn[maxn];
9 int ind, tot; // tot是边的数量
10
11 void init()
12 {
13     memset(head, -1, sizeof(head));
14     memset(dfn, 0, sizeof(dfn));
15     ind = tot = 0;
16 }
17
18 void addedge(int u, int v)
19 {
20     e[tot].to = v;
21     e[tot].next = head[u];
22     e[tot].iscut = 0;
23     head[u] = tot++;
24 }
25
26 void Tarjan(int u, int pa)
27 {
28     low[u] = dfn[u] = ++ind;
29     for(int i=head[u]; ~i; i = e[i].next)
30     {
31         int v = e[i].to;
32         if(v == pa) continue;
33         if(!dfn[v])
34         {
35             Tarjan(v, u);
36             low[u] = min(low[u], low[v]);
37             // 是桥
38             if(low[v] > dfn[u])
39             {
40                 e[i].iscut = e[i^1].iscut = 1;
41             }
42         }
43         else
44         {
45             low[u] = min(low[u], dfn[v]);
46         }
47     }
48 }

```

1.7 最大流

1.7.1 Dinic

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 typedef pair<int, int> P;
5 const int maxn = 1e6+5;
6 const int inf = 0x3f3f3f3f;
7 const int mod = 1e9+7;
8
9 // 用链式前向星来存储图
10 struct ed
11 {
12     int to, val, ne;
13 }edge[maxn<<1];
14 int head[maxn], dep[maxn];
15 // 顶点数n, 边数m, 源点s, 汇点e, 加边时的指针tot
16 int n, m, s, e, tot;
17
18 void init()
19 {
20     tot = -1;
21     memset(head, -1, sizeof(head));
22 }
23
24 void addEdge(int u, int v, int val)
25 {
26     edge[++tot].to = v;
27     edge[tot].val = val;
28     edge[tot].ne = head[u];
29     head[u] = tot;
30 }
31
32 // 就是最普通的bfs
33 int bfs()
34 {
35     memset(dep, -1, sizeof(dep));
36     dep[s] = 0;
37     queue<int> q;
38     q.push(s);
39     while(!q.empty())
40     {

```

```

41         int u = q.front();
42         q.pop();
43         for(int i=head[u]; ~i; i=edge[i].ne)
44         {
45             int v = edge[i].to;
46             if(dep[v]==-1 && edge[i].val>0)
47             {
48                 dep[v] = dep[u]+1;
49                 q.push(v);
50             }
51         }
52     }
53     return (dep[e] != -1); //若dep[e]==-1则表示没有可以到达e的增广路了，算法结束。
54 }
55
56 // 当前顶点u, 当前流量flow
57 // 初始时dfs(s, inf)
58 int dfs(int u, int flow)
59 {
60     if(u == e)return flow;
61     for(int i=head[u]; ~i; i=edge[i].ne)
62     {
63         int v = edge[i].to;
64         if(dep[v]==dep[u]+1 && edge[i].val)
65         {
66             int a = dfs(v, min(flow, edge[i].val));
67             if(a>0) //若找到增广路
68             {
69                 edge[i].val -= a;
70                 edge[i^1].val += a;
71                 return a;
72             }
73         }
74     }
75     return 0;
76 }
77
78 ll dinic()
79 {
80     ll ans = 0;
81     while(bfs())
82     {
83         int a = dfs(s, (1<<30));

```

```

84     ans += a;
85 }
86 return ans;
87 }
88
89 int main()
90 {
91     scanf("%d%d%d", &n, &m, &s, &e);
92     init();
93     for(int i=1;i<=m;i++)
94     {
95         int u, v, w;
96         scanf("%d%d", &u, &v, &w);
97         addEdge(u, v, w);
98         addEdge(v, u, 0); //反边
99     }
100     printf("%lld\n", dinic());
101     return 0;
102 }

```

1.7.2 Dinic 优化

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 typedef pair<int, int> P;
5 const int maxn = 1e6+5;
6 const int inf = 0x3f3f3f3f;
7 const int mod = 1e9+7;
8
9 struct ed
10 {
11     int to, val, ne;
12 }edge[maxn<<1];
13 int head[maxn],dep[maxn], cur[maxn];
14 int n, m, s, e, tot;
15
16 void init()
17 {
18     tot = -1;
19     memset(head, -1, sizeof(head));
20 }

```

```

21
22 void addEdge(int u, int v, int val)
23 {
24     edge[++tot].to = v;
25     edge[tot].val = val;
26     edge[tot].ne = head[u];
27     head[u] = tot;
28 }
29
30 int bfs()
31 {
32     memset(dep, -1, sizeof(dep));
33     dep[s] = 0;
34     queue<int> q;
35     q.push(s);
36     while(!q.empty())
37     {
38         int u = q.front();
39         q.pop();
40         for(int i=head[u]; ~i; i=edge[i].ne)
41         {
42             int v = edge[i].to;
43             if(dep[v]==-1 && edge[i].val>0)
44             {
45                 dep[v] = dep[u]+1;
46                 q.push(v);
47             }
48         }
49     }
50     return (dep[e] != -1);
51 }
52
53 int dfs(int u, int flow)
54 {
55     if(u == e)return flow;
56     // rflow用于多路增广，表示流入到顶点u的剩余未流出的流量
57     int rflow = flow;
58     // 当前弧优化，通过引用，可以改变cur[i]的值，使得下次遍历到顶点u时，会直接
    // 从上次增广的边开始遍历
59     for(int& i=cur[u]; ~i; i=edge[i].ne)
60     {
61         int v = edge[i].to;
62         if(dep[v]==dep[u]+1 && edge[i].val)
63         {

```

```

64     int a = dfs(v, min(rflow, edge[i].val));
65     edge[i].val -= a;
66     edge[i^1].val += a;
67     rflow -= a; // 剩余流量要减少
68     if(rflow<=0)break; // 若没有剩余流量了, 就break
69 }
70 }
71 // 若没有一丝流量流出, 则表示通过顶点u已经无法增广了, 于是炸点, dep可以设置
    为任何无意义值
72 if(rflow == flow)
73     dep[u] = -2;
74     return flow - rflow; // 返回流出的流量
75 }
76
77 ll dinic()
78 {
79     ll ans = 0;
80     while(bfs())
81     {
82         // 新一轮dfs之前要对cur进行初始化
83         for(int i=1;i<=n;i++)cur[i] = head[i];
84         int a = dfs(s, (1<<30));
85         ans += a;
86     }
87     return ans;
88 }
89
90 int main()
91 {
92     scanf("%d%d%d", &n, &m, &s, &e);
93     init();
94     for(int i=1;i<=m;i++)
95     {
96         int u, v, w;
97         scanf("%d%d", &u, &v, &w);
98         addEdge(u, v, w);
99         addEdge(v, u, 0);
100     }
101     printf("%lld\n", dinic());
102     return 0;
103 }

```

1.8 二分图匹配

1.8.1 匈牙利算法

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 typedef pair<int, int> P;
5 const int maxn = 1e3+5;
6 const int inf = 0x3f3f3f3f;
7 const int mod = 1e9+7;
8
9
10 int mp[maxn][maxn];
11 int use[maxn], link[maxn];
12 int n, m, e, ans;
13
14 int found(int u)
15 {
16     for(int i=1;i<=m;i++)
17     {
18         if(!use[i] && mp[u][i])
19         {
20             use[i] = 1;
21             if(!link[i] || found(link[i]))
22             {
23                 link[i] = u;
24                 return 1;
25             }
26         }
27     }
28     return 0;
29 }
30
31 int main()
32 {
33     scanf("%d%d", &n, &m, &e);
34     for(int i=1;i<=e;i++)
35     {
36         int u, v;
37         scanf("%d", &u, &v);
38         if(u<=n && v<=m)
39         {
40             mp[u][v] = 1;

```

```

41     }
42 }
43 for(int i=1;i<=n;i++)
44 {
45     memset(use, 0, sizeof(use));
46     if(found(i))ans++;
47 }
48 printf("%d\n", ans);
49 return 0;
50 }

```

1.9 最小生成树

1.9.1 Prim

```

1 const int MAX=10000007;
2 int dis[5002],map[5002][5002],mark[5002];
3 int prim(int n)
4 {
5     for(int i=1;i<=n;i++) //初始化每个点到生成树中点的距离
6     {
7         dis[i]=map[1][i];
8         mark[i]=0;
9     }
10    dis[1]=0;
11    mark[1]=1; //1这个点加入生成树中。
12    int sum=0;
13    for(int i=1;i<n;i++) //枚举n-1条边
14    {
15        int sta=-1,Min=MAX;
16        for(int j=1;j<=n;j++) //找不在生成树中的点中距离生成树中的点长度最小的
17        {
18            if(!mark[j]&&dis[j]<Min)
19            {
20                Min=dis[j];
21                sta=j;
22            }
23        }
24        if(sta==-1) return -1; //没找到可以可以联通的路
25        mark[sta]=1; //新找到的点加入生成树
26        sum+=Min;
27        for(int j=1;j<=n;j++) //更新树外的点到树中的点的距离
28        {

```

```

29            if(!mark[j]&&dis[j]>map[sta][j])
30                dis[j]=map[sta][j];
31        }
32    }
33    return sum;
34 }
35
36 int main()
37 {
38     int n,m;
39     cin>>n>>m;
40     for(int i=1;i<=n;i++)
41     {
42         for(int j=1;j<=n;j++)
43         {
44             map[i][j]=MAX;
45         }
46     }
47     for(int i=1;i<=m;i++)
48     {
49         int a,b,c;
50         cin>>a>>b>>c;
51         if(c<map[a][b])
52         {
53             map[a][b]=c;
54             map[b][a]=c;
55         }
56     }
57     int ans = prim(n);
58     if(ans==-1)
59         cout<<"orz"<<endl;
60     else
61         cout<<ans<<endl;
62     return 0;
63 }

```

1.9.2 kruskal

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define N 5005
4 int father[N];

```

```

5
6 int find(int x)
7 {
8     int k = x;
9     while(father[k]!=k)
10    {
11        k = father[k];
12    }
13    while(father[x]!=x)
14    {
15        int temp = x;
16        x = father[x];
17        father[temp] = k;
18    }
19    return k;
20 }
21
22 void join(int a, int b)
23 {
24     int f1, f2;
25     f1 = find(a);
26     f2 = find(b);
27     father[f1] = f2;
28 }
29
30 struct edge
31 {
32     int node1, node2;
33     int cost;
34 };
35
36 vector<edge> edges;
37
38 bool cmp(edge a, edge b)
39 {
40     return a.cost > b.cost;
41 }
42
43 int kruskal(int n)
44 {
45     sort(edges.begin(), edges.end(), cmp);
46     for(int i=1;i<=n;i++)
47         father[i] = i;
48     int sum=0;

```

```

49     while(n!=1 && !edges.empty())
50     {
51         edge temp = edges[edges.size()-1];
52         edges.pop_back();
53         if(find(temp.node1)!=find(temp.node2))
54         {
55             sum += temp.cost;
56             n--;
57             join(temp.node1, temp.node2);
58         }
59     }
60     if(n!=1 && edges.empty())
61         sum = -1;
62     return sum;
63 }
64
65 int main()
66 {
67     int n,m;
68     int result;
69     cin>>n>>m;
70     for(int i=1;i<=m;i++)
71     {
72         int a,b,c;
73         cin>>a>>b>>c;
74         edge t;
75         t.node1=a;t.node2=b;t.cost=c;
76         edges.push_back(t);
77     }
78     result = kruskal(n);
79     if(result == -1)
80         cout<<"orz"<<endl;
81     else
82         cout<<result<<endl;
83     return 0;
84 }

```

1.10 次小生成树-POJ1679

```

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

```

```

4 #define inf 0x3f3f3f3f
5 typedef pair<int, int> P;
6 const int maxn = 110;
7 const ll mod = 1e9+7;
8
9 int n, m;
10 int g[maxn][maxn];
11 int d[maxn], v[maxn], maxd[maxn][maxn], pre[maxn], mst[maxn][maxn];
12 int ans = 0;
13
14 void prim()
15 {
16     for(int i=1;i<=n;i++)
17     {
18         v[i] = 0; d[i] = inf; pre[i] = 1;
19     }
20     memset(maxd, 0, sizeof(maxd));
21     memset(mst, 0, sizeof(mst));
22     ans = 0;
23     priority_queue<P, vector<P>, greater<P> > q;
24     d[1] = 0; q.push(P(0, 1));
25     while(!q.empty())
26     {
27         P p = q.top(); q.pop();
28         int u = p.second;
29         if(v[u]) continue;
30         v[u] = 1; ans += d[u];
31         mst[pre[u]][u] = mst[u][pre[u]] = 1;
32         for(int i=1; i<=n;i++)
33         {
34             if(v[i] && g[u][i] < inf)
35                 maxd[u][i] = maxd[i][u] = max(maxd[pre[u]][u], d[u]);
36             if(d[i] > g[u][i])
37             {
38                 d[i] = g[u][i];
39                 pre[i] = u;
40                 q.push(P(d[i], i));
41             }
42         }
43     }
44 }
45
46 int main()
47 {

```

```

48     int t;
49     cin>>t;
50     while(t--)
51     {
52         memset(g, inf, sizeof(g));
53         cin>>n>>m;
54         while(m--)
55         {
56             int a, b, c;
57             cin>>a>>b>>c;
58             g[a][b] = g[b][a] = c;
59         }
60         prim();
61         int flag = 0;
62         for(int i=1;i<=n&&!flag;i++)
63         {
64             for(int j=1;j<=n;j++)
65             {
66                 if(mst[i][j] || g[i][j]==inf)continue;
67                 if(g[i][j] == maxd[i][j])
68                 {
69                     flag = 1;
70                     break;
71                 }
72             }
73         }
74         if(flag) cout<<"Not Unique!"<<endl;
75         else cout<<ans<<endl;
76     }
77     return 0;
78 }

```

1.11 拓扑排序

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int maxn=30;
4
5 int head[maxn],ip,indegree[maxn];
6 int n,m,seq[maxn];
7
8 struct note

```

```

9 {
10     int v,next;
11 }edge[maxn*maxn];
12
13 void init()
14 {
15     memset(head,-1,sizeof(head));
16     ip=0;
17 }
18
19 void addedge(int u,int v)
20 {
21     edge[ip].v=v,edge[ip].next=head[u],head[u]=ip++;
22 }
23
24 int topo()
25 {
26     queue<int>q;
27     int indeg[maxn];
28     for(int i=0; i<n; i++)
29     {
30         indeg[i]=indegree[i];
31         if(indeg[i]==0)
32             q.push(i);
33     }
34     int k=0;
35     bool res=false;
36     while(!q.empty())
37     {
38         if(q.size()!=1)res=true;
39         int u=q.front();
40         q.pop();
41         seq[k++]=u;
42         for(int i=head[u]; i!=-1; i=edge[i].next)
43         {
44             int v=edge[i].v;
45             indeg[v]--;
46             if(indeg[v]==0)
47                 q.push(v);
48         }
49     }
50     if(k<n)return -1;// no
51     if(res)return 0;// more
52     return 1; // only

```

```

53 }

```

1.12 Floyd 找最小环

```

1 const int INF = 0x3f3f3f3f;
2 const int MAXN = 110;
3
4 int n, m;           // n: 节点个数, m: 边的个数
5 int g[MAXN][MAXN];  // 无向图
6 int dist[MAXN][MAXN]; // 最短路径
7 int r[MAXN][MAXN];   // r[i][j]: i到j的最短路径的第一步
8 int out[MAXN], ct;   // 记录最小环
9
10 int solve(int i, int j, int k)
11 { // 记录最小环
12     ct = 0;
13     while (j != i)
14     {
15         out[ct++] = j;
16         j = r[i][j];
17     }
18     out[ct++] = i;
19     out[ct++] = k;
20     return 0;
21 }
22
23 int main()
24 {
25     while (scanf("%d%d", &n, &m) != EOF)
26     {
27         int i, j, k;
28         for (i = 0; i < n; i++)
29         {
30             for (j = 0; j < n; j++)
31             {
32                 g[i][j] = INF;
33                 r[i][j] = i;
34             }
35         }
36         for (i = 0; i < m; i++)
37         {
38             int x, y, l;

```



```

39     scanf("%d%d%d", &x, &y, &l);
40     —x;
41     —y;
42     if (l < g[x][y])
43     {
44         g[x][y] = g[y][x] = l;
45     }
46 }
47 memmove(dist, g, sizeof(dist));
48 int Min = INF;           // 最小环
49 for (k = 0; k < n; k++)
50 {                         // Floyd
51     for (i = 0; i < k; i++) // 一个环中的最大结点为k(编号最大)
52     {
53         if (g[k][i] < INF)
54         {
55             for (j = i + 1; j < k; j++)
56             {
57                 if (dist[i][j] < INF && g[k][j] < INF && Min > dist[i][j] + g[k]
58                     [i] + g[k][j])
59                 {
60                     Min = dist[i][j] + g[k][i] + g[k][j];
61                     solve(i, j, k); // 记录最小环
62                 }
63             }
64         }
65     }
66     for (i = 0; i < n; i++)
67     {
68         if (dist[i][k] < INF)
69         {
70             for (j = 0; j < n; j++)
71             {
72                 if (dist[k][j] < INF && dist[i][j] > dist[i][k] + dist[k][j])
73                 {
74                     dist[i][j] = dist[i][k] + dist[k][j];
75                     r[i][j] = r[k][j];
76                 }
77             }
78         }
79     }
80     if (Min < INF)
81     {

```

```

82         for (ct—; ct >= 0; ct—)
83         {
84             printf("%d", out[ct] + 1);
85             if (ct)
86             {
87                 printf(" ");
88             }
89         }
90     }
91     else
92     {
93         printf("No solution.");
94     }
95     printf("\n");
96 }
97 return 0;
98 }

```

2 数据结构

2.1 并查集

```

1 int pre[maxn];
2
3 int Find(int x)
4 {
5     int p,tmp;
6     p=x;
7     while(x!=pre[x])
8         x=pre[x];
9     while(p!=x)
10    {
11        tmp=pre[x];
12        pre[x]=x;
13        p=tmp;
14    }
15    return x;
16 }
17
18 void join(int x,int y)
19 {
20     int fx=Find(x);

```

```

21 int fy=Find(y);
22 if(fx!=fy)
23     pre[fx]=fy;
24 }

```

2.2 ST 表

```

1 int st[maxn][20];
2 void st_init()
3 {
4     for(int i=1;i<=n;i++) st[i][0]=a[i]; // 长度为1的区间最小值党委就为自身
5     // 预处理从i开始, 长度为2^j的区间
6     for(int j=1;(1<<j)<=n;j++)
7         for(int i=1;i+(1<<j)-1<=n;i++)
8             st[i][j]=max(st[i][j-1],st[i+(1<<(j-1))][j-1]);
9 }
10
11 int query(int l,int r)
12 {
13     int k=log2(r-l+1);
14     return min(st[l][k],st[r-(1<<k)+1][k]);
15 }

```

2.3 树状数组

```

1 /*
2 * INIT: ar[]置为0;
3 * CALL: add(i, v): 将i点的值加v; sum(i): 求[1, i]的和;
4 */
5 #define typev int // type of res
6 const int N = 1010;
7 typev ar[N]; // index: 1 ~ N
8 int lowb(int t)
9 {
10     return t & (-t);
11 }
12
13 void add(int i, typev v)
14 {
15     for (; i < N; ar[i] += v, i += lowb(i));
16     return ;

```

```

17 }
18
19 typev sum(int i)
20 {
21     typev s = 0;
22     for (; i > 0; s += ar[i], i -= lowb(i));
23     return s;
24 }

```

2.4 线段树

2.4.1 单点修改 + 区间求和 HDU1166

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5 int num[50005];
6 ll tree[4 * 50000 + 5];
7
8 void build(int p, int l, int r)
9 {
10     if(l == r)
11     {
12         tree[p] = num[l];
13         return ;
14     }
15     else
16     {
17         int mid = (r+l) >> 1;
18         build(p<<1, l, mid);
19         build(p<<1|1, mid+1, r);
20         tree[p] = tree[p<<1] + tree[p<<1|1];
21     }
22 }
23
24 void add(int p, int l, int r, int ind, int v)
25 {
26     if(l == r)
27     {
28         tree[p] += v;
29         return ;
30     }

```

```

31 else
32 {
33     int mid = (r+l) >> 1;
34     if(ind <= mid) add(p<<1, l, mid, ind, v);
35     else add(p<<1|1, mid+1, r, ind, v);
36     tree[p] = tree[p<<1] + tree[p<<1|1];
37 }
38 }
39
40 ll query(int p, int l, int r, int x, int y)
41 {
42     if(x <= l && r <= y)
43     {
44         return tree[p];
45     }
46     else
47     {
48         int mid = (l+r) >> 1;
49         ll ans = 0;
50         if(x <= mid)
51             ans += query(p<<1, l, mid, x, y);
52         if(mid < y)
53             ans += query(p<<1|1, mid+1, r, x, y);
54         return ans;
55     }
56 }
57
58 int main()
59 {
60     int t;
61     scanf("%d", &t);
62     for(int i=1;i<=t;i++)
63     {
64         int n;
65         scanf("%d", &n);
66         for(int j=1;j<=n;j++)
67             scanf("%d", &num[j]);
68         build(1, 1, n);
69         string s;
70         printf("Case %d:\n", i);
71         while(cin>>s && s[0] != 'E')
72         {
73             if(s[0] == 'Q')
74             {

```

```

75         int x, y;
76         scanf("%d%d", &x, &y);
77         printf("%lld\n", query(1, 1, n, x, y));
78     }
79     else if(s[0] == 'A')
80     {
81         int x, y;
82         scanf("%d%d", &x, &y);
83         add(1, 1, n, x, y);
84     }
85     else
86     {
87         int x, y;
88         scanf("%d%d", &x, &y);
89         add(1, 1, n, x, -y);
90     }
91 }
92 }
93 return 0;
94 }

```

2.4.2 区间修改 + 区间求和 POJ3465

```

1 #include<iostream>
2 using namespace std;
3 typedef long long ll;
4 ll num[100005];
5 ll tree[4 * 100000 + 5];
6 ll lazy[4 * 100000 + 5]={0};
7 int n,m;
8 void build(int p, int l, int r)
9 {
10     if(l == r)
11         tree[p] = num[l];
12     else
13     {
14         int mid = (l+r) >> 1;
15         build(p<<1, l, mid);
16         build(p<<1|1, mid+1, r);
17         tree[p] = tree[p<<1] + tree[p<<1|1];
18     }
19 }

```

```

20
21 void pushdown(int p, int l, int r)
22 {
23     if(lazy[p])
24     {
25         lazy[p<<1] += lazy[p];
26         lazy[p<<1|1] += lazy[p];
27         tree[p<<1] += lazy[p] * (((l+r)>>1) - l + 1);
28         tree[p<<1|1] += lazy[p] * (r - ((l+r)>>1));
29         lazy[p] = 0;
30     }
31 }
32
33 void add(int p, int l, int r, int x, int y, ll v)
34 {
35     if(x <= l && r <= y)
36     {
37         lazy[p] += v;
38         tree[p] += v * (r-l+1);
39         return ;
40     }
41     else
42     {
43         int mid = (l+r) >> 1;
44         pushdown(p, l, r);
45         if(x <= mid)
46             add(p<<1, l, mid, x, y, v);
47         if(y > mid)
48             add(p<<1|1, mid+1, r, x, y, v);
49         tree[p] = tree[p<<1] + tree[p<<1|1];
50     }
51 }
52
53 ll query(int p, int l, int r, int x, int y)
54 {
55     if(x <= l && r <= y)
56     {
57         return tree[p];
58     }
59     else
60     {
61         int mid = (l+r) >> 1;
62         ll ans = 0;
63         pushdown(p, l, r);

```

```

64         if(x <= mid)
65             ans += query(p<<1, l, mid, x, y);
66         if(y > mid)
67             ans += query(p<<1|1, mid+1, r, x, y);
68         return ans;
69     }
70 }
71
72 int main()
73 {
74     cin>>n>>m;
75     for(int i=1;i<=n;i++)
76         cin>>num[i];
77     build(1, 1, n);
78     for(int i=1;i<=m;i++)
79     {
80         char a;
81         cin>>a;
82         if(a == 'Q')
83         {
84             int x, y;
85             cin>>x>>y;
86             cout<<query(1, 1, n, x, y)<<endl;
87         }
88         else
89         {
90             int x, y;
91             ll c;
92             cin>>x>>y>>c;
93             add(1, 1, n, x, y, c);
94         }
95     }
96     return 0;
97 }

```

2.4.3 单点修改 + 区间最值 HDU1754

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 int n,m;
5

```

```

6 int num[200005];
7 int tree[200000 * 4 + 5];
8
9 void build(int p, int l, int r)
10 {
11     if(l == r)
12     {
13         tree[p] = num[l];
14         return ;
15     }
16     else
17     {
18         int mid = (l+r) >> 1;
19         build(p<<1, l, mid);
20         build(p<<1|1, mid+1, r);
21         tree[p] = max(tree[p<<1], tree[p<<1|1]);
22         return ;
23     }
24 }
25
26 void update(int p, int l, int r, int ind, int v)
27 {
28     if(l == r)
29     {
30         tree[p] = v;
31         return ;
32     }
33     else
34     {
35         int mid = (l+r) >> 1;
36         if(ind <= mid)
37             update(p<<1, l, mid, ind, v);
38         else
39             update(p<<1|1, mid+1, r, ind, v);
40         tree[p] = max(tree[p<<1], tree[p<<1|1]);
41     }
42 }
43
44 ll query(int p, int l, int r, int x, int y)
45 {
46     if(x <= l && r <= y)
47     {
48         return tree[p];
49     }

```

```

50     else
51     {
52         int mid = (l+r) >> 1;
53         ll ans = -1;
54         if(x <= mid)
55             ans = max(ans, query(p<<1, l, mid, x, y));
56         if(y > mid)
57             ans = max(ans, query(p<<1|1, mid+1, r, x, y));
58         return ans;
59     }
60 }
61
62 int main()
63 {
64     while(scanf("%d %d", &n, &m) != EOF)
65     {
66         for(int i=1;i<=n;i++)
67         {
68             scanf("%d", &num[i]);
69         }
70         build(1, 1, n);
71         for(int i=1;i<=m;i++)
72         {
73             char a;
74             int x, y;
75             scanf("%c%d%d", &a, &x, &y);
76             if(a == 'Q')
77             {
78                 printf("%lld\n", query(1, 1, n, x, y));
79             }
80             else
81             {
82                 update(1, 1, n, x, y);
83             }
84         }
85     }
86     return 0;
87 }

```

2.4.4 区间染色 + 统计 + 离散化 POJ2528

```
1 #include<iostream>
```

```

2 #include<algorithm>
3 #include<string.h>
4 using namespace std;
5 typedef long long ll;
6
7 const int maxn = 20000 + 100;
8 int tree[maxn<<4];
9 int li[maxn],ri[maxn];
10 bool vis[maxn];
11 int lisan[maxn*3];
12 int ans = 0;
13
14 void init()
15 {
16     memset(tree, -1, sizeof(tree));
17     memset(vis, 0, sizeof(vis));
18     ans = 0;
19 }
20
21 void pushdown(int p)
22 {
23     tree[p<<1] = tree[p<<1|1] = tree[p];
24     tree[p] = -1;
25 }
26
27 void update(int p, int l, int r, int x, int y, int v)
28 {
29     if(x <= l && r <= y)
30     {
31         tree[p] = v;
32         return ;
33     }
34     if(tree[p]!=-1)
35         pushdown(p);
36     int mid = (l+r)>>1;
37     if(x <= mid)
38         update(p<<1, l, mid, x, y, v);
39     if(y > mid)
40         update(p<<1|1, mid+1, r, x, y, v);
41     tree[p] = -1;
42 }
43
44 void query(int p, int l, int r)
45 {

```

```

46     if(tree[p]!=-1)
47     {
48         if(vis[tree[p]]==0)
49         {
50             vis[tree[p]] = 1;
51             ans++;
52         }
53         return;
54     }
55     if(l==r)return;
56     int mid = (l+r)>>1;
57     query(p<<1, l, mid);
58     query(p<<1|1, mid+1, r);
59 }
60
61 int main()
62 {
63     int t;
64     cin>>t;
65     while(t—)
66     {
67         init();
68         int n;
69         cin>>n;
70         int tot = 0;
71         for(int i=0;i<n;i++)
72         {
73             cin>>li[i]>>ri[i];
74             lisan[tot++] = li[i];
75             lisan[tot++] = ri[i];
76         }
77         sort(lisan, lisan+tot);
78         int m = unique(lisan, lisan+tot) - lisan;
79         int t = m;
80         for(int i=1;i<t;i++)
81         {
82             if(lisan[i]-lisan[i-1]>1)
83             {
84                 lisan[m++] = lisan[i-1]+1;
85             }
86         }
87         sort(lisan, lisan+m);
88         for(int i=0;i<n;i++)
89         {

```

```

90     int x,y;
91     x = lower_bound(lisan, lisan+m, li[i]) - lisan;
92     y = lower_bound(lisan, lisan+m, ri[i]) - lisan;
93     update(1, 0, m-1, x, y, i);
94 }
95 query(1, 0, m-1);
96 cout<<ans<<endl;
97 }
98 return 0;
99 }

```

2.4.5 线段树 + 扫描线求矩阵覆盖周长 POJ1177

```

1 #include<iostream>
2 #include<vector>
3 #include<algorithm>
4 #include<cmath>
5 using namespace std;
6 typedef long long ll;
7
8 const int maxn = 10005;
9 vector<int> x;
10 int getID(int v)
11 {
12     return lower_bound(x.begin(), x.end(), v) - x.begin();
13 }
14
15 struct Segment
16 {
17     int l, r;
18     int h;
19     int flag;
20 }segment[maxn];
21 bool cmp(Segment a, Segment b)
22 {
23     return a.h < b.h;
24 }
25
26 struct Node
27 {
28     int l,r;
29     int lr, rr;

```

```

30     int len;
31     int line;
32     int s;
33 }node[maxn<<2];
34
35 void build(int p, int l, int r)
36 {
37     node[p].l = l; node[p].r = r;
38     node[p].line = node[p].len = node[p].s = 0;
39     node[p].lr = node[p].rr = 0;
40     if(l==r)return;
41     int mid = (l+r)>>1;
42     build(p<<1, l, mid);
43     build(p<<1|1, mid+1, r);
44 }
45
46 void pushup(int p)
47 {
48     if(node[p].s)
49     {
50         node[p].line = 1;
51         node[p].rr = node[p].lr = 1;
52         node[p].len = x[node[p].r+1] - x[node[p].l];
53         return;
54     }
55     else if(node[p].l == node[p].r)
56     {
57         node[p].lr = node[p].rr = node[p].line = node[p].len = 0;
58     }
59     else
60     {
61         node[p].lr = node[p<<1].lr;
62         node[p].rr = node[p<<1|1].rr;
63         node[p].len = node[p<<1].len + node[p<<1|1].len;
64         node[p].line = node[p<<1].line + node[p<<1|1].line - (node[p<<1].rr&&
65             node[p<<1|1].lr);
66     }
67 }
68
69 void update(int p, int l, int r, int v)
70 {
71     if(node[p].r < l || node[p].l > r)return;
72     if(l <= node[p].l && node[p].r <= r)
73     {

```

```

73     node[p].s += v;
74     pushup(p);
75     return;
76 }
77 update(p<<1, l, r, v);
78 update(p<<1|1, l, r, v);
79 pushup(p);
80 }
81
82 int main()
83 {
84     int n;
85     cin>>n;
86     for(int i=1;i<=n;i++)
87     {
88         int x1,x2,y1,y2;
89         cin>>x1>>y1>>x2>>y2;
90         Segment &s1 = segment[2*i-1];
91         Segment &s2 = segment[i<<1];
92         s1.l = s2.l = x1;
93         s1.r = s2.r = x2;
94         s1.h = y1; s2.h = y2;
95         s1.flag = 1; s2.flag = -1;
96         x.push_back(x1);
97         x.push_back(x2);
98     }
99     sort(segment+1, segment+2*n+1, cmp);
100
101     sort(x.begin(), x.end());
102     x.erase(unique(x.begin(), x.end()), x.end());
103
104     build(1, 0, x.size()-1);
105
106     ll ans = 0;
107     int last = 0;
108     for(int i=1;i<=2*n;i++)
109     {
110         int l = getID(segment[i].l);
111         int r = getID(segment[i].r);
112         update(1, l, r-1, segment[i].flag);
113         ans += abs(node[1].len - last);
114         if(i!=2*n)
115             ans += node[1].len * 2 * (segment[i+1].h - segment[i].h);
116         last = node[1].len;

```

```

117     }
118     cout<<ans<<endl;
119     return 0;
120 }

```

2.4.6 线段树 + 扫描线求矩阵面积并 HDU1542

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5 const int maxn = 210;
6 int n;
7 vector<double> x;
8 inline int getID(double v)
9 {
10     return lower_bound(x.begin(), x.end(), v) - x.begin();
11 }
12
13 struct Segment
14 {
15     double l, r;
16     double h;
17     int flag;
18 }segment[maxn];
19 bool cmp(Segment a, Segment b)
20 {
21     return a.h < b.h;
22 }
23
24 struct Node
25 {
26     int l, r;
27     int s;
28     double len;
29 }node[maxn<<2];
30
31 void pushup(int p)
32 {
33     if(node[p].s)
34         node[p].len = x[node[p].r+1] - x[node[p].l];
35     else if(node[p].l == node[p].r)

```



```

36     node[p].len = 0;
37     else
38         node[p].len = node[p<<1].len + node[p<<1|1].len;
39 }
40
41 void build(int p, int l, int r)
42 {
43     if(l>r)return;
44     node[p].l = l; node[p].r = r;
45     node[p].s = 0; node[p].len = 0;
46     if(l==r) return;
47     int mid = (l+r)>>1;
48     build(p<<1, l, mid);
49     build(p<<1|1, mid+1, r);
50     pushup(p);
51 }
52
53 void update(int p, int l, int r, int v)
54 {
55     if(l>node[p].r || r<node[p].l) return;
56     if(l <= node[p].l && node[p].r <= r)
57     {
58         node[p].s += v;
59         pushup(p);
60         return;
61     }
62     update(p<<1, l, r, v);
63     update(p<<1|1, l, r, v);
64     pushup(p);
65 }
66
67 int main()
68 {
69     int cas = 0;
70     while(scanf("%d", &n) && n)
71     {
72         x.clear();
73         for(int i=1;i<=n;i++)
74         {
75             double x1,x2,y1,y2;
76             scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
77             Segment &s1 = segment[2*i-1];
78             Segment &s2 = segment[i<<1];
79             s1.l=s2.l=x1;

```

```

80         s1.r=s2.r=x2;
81         s1.h=y1;
82         s2.h=y2;
83         s1.flag=1;
84         s2.flag=-1;
85         x.push_back(x1);
86         x.push_back(x2);
87     }
88     sort(segment+1, segment+2*n+1, cmp);
89
90     sort(x.begin(), x.end());
91     x.erase(unique(x.begin(), x.end()), x.end());
92
93     build(1, 0, x.size()-1);
94     double ans = 0;
95     for(int i=1;i<=2*n;i++)
96     {
97         int l=getID(segment[i].l);
98         int r=getID(segment[i].r);
99         update(1, l, r-1, segment[i].flag);
100         ans+=node[1].len*(segment[i+1].h - segment[i].h);
101     }
102     printf("Test case #%d\n", ++cas);
103     printf("Total explored area: %.2f\n\n", ans);
104 }
105 return 0;
106 }

```

2.5 主席树

2.5.1 区间第 K 大

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 #define inf 0x3f3f3f3f
5 typedef pair<int, int> P;
6 const int maxn = 1e6+5;
7 const ll mod = 1e9+7;
8
9 // 顶点, 代表区间[l, r]中有v个数字
10 struct node
11 {

```

```

12  int l, r, v;
13 }tree[maxn*20]; // 空间开大一点, 因为要动态开点
14
15 // edit[i]存的是第i颗权值线段树的根节点在tree数组中的位置
16 // a是存放原数据的数组, b是离散化后的数组, tot表示顶点的个数
17 int edit[maxn], a[maxn], b[maxn], tot=0;
18
19 // 建树
20 int build(int l, int r)
21 {
22     // 这里就是动态开辟新的结点, 就是将tot加一
23     int pos = ++tot;
24     tree[pos].v = 0; // 初始化为0
25     if(l==r) return pos; // 到根节点了, 返回
26     // 二分建树没什么好说的
27     int mid = (l+r)>>1;
28     tree[pos].l = build(l, mid);
29     tree[pos].r = build(mid+1, r);
30     // 要返回pos的位置, 因为edit数组要存新根的位置
31     return pos;
32 }
33
34 // 就是插入操作, 插入新的元素
35 // ed是前一版本的结点在tree的位置, 更新区间[l, r], 位置为v
36 int update(int ed, int l, int r, int v)
37 {
38     // 动态开点
39     int pos = ++tot;
40     // 先将新一版本的当前结点复制为上一个版本的对应结点
41     tree[pos] = tree[ed]; tree[pos].v++; // 新一版本的结点的v要加一, 因为对应区
    间插入了一个数
42     if(l==r) return pos; // 到叶子节点了, 返回
43     // 二分
44     int mid = (l+r)>>1;
45     // 如果更新位置v在左子树中, 递归更新即可, 在右子树中同理, 最后要返回pos
46     if(v<=mid) tree[pos].l = update(tree[ed].l, l, mid, v);
47     else tree[pos].r = update(tree[ed].r, mid+1, r, v);
48     return pos;
49 }
50
51 // 查询区间[l, r]第k大的数字, pre对应l-1版本的权值线段树的节点位置, ed代表r
    版本的权值线段树的节点位置
52 int query(int pre, int ed, int l, int r, int k)
53 {

```

```

54 // 到叶子节点就返回
55 if(l==r) return l;
56 // 二分
57 int mid = (l+r)>>1;
58 // 先计算左子树的数字个数
59 int x = tree[tree[ed].l].v - tree[tree[pre].l].v;
60 // 若左子树的数字个数大于等于k, 说明我们要找的数字在左子树中, 递归走到左子
    树继续寻找
61 if(x>=k) return query(tree[pre].l, tree[ed].l, l, mid, k);
62 // 否则就在右子树中, 我们要在右子树中寻找k-x大的数字, 递归寻找就好
63 else return query(tree[pre].r, tree[ed].r, mid+1, r, k-x);
64 }
65
66 int main()
67 {
68     int n, q; // n个数字, q次询问
69     scanf("%d%d", &n, &q); // 这一题cin/cout会被卡
70     // 输入数据, 并copy到b数组中
71     for(int i=1; i<=n; i++)
72     {
73         scanf("%d", &a[i]);
74         b[i] = a[i];
75     }
76     // 离散化
77     sort(b+1, b+1+n);
78     int m = unique(b+1, b+1+n) - b - 1;
79     // 构建一颗空的权值线段树, edit[0]存放的就是这颗空树的根节点的位置
80     edit[0] = build(1, m);
81     // 插入n个数据
82     for(int i=1; i<=m; i++)
83     {
84         // 找到a[i]离散化后对应的位置
85         a[i] = lower_bound(b+1, b+m+1, a[i]) - b;
86         // edit[i]存放第i版本的权值线段树的根节点位置
87         edit[i] = update(edit[i-1], 1, m, a[i]);
88     }
89     // 处理q次询问
90     while(q--)
91     {
92         int x, y, k;
93         scanf("%d%d%d", &x, &y, &k);
94         // pos对应的是离散化后的位置, 所以最后输出b[pos]即可
95         int pos = query(edit[x-1], edit[y], 1, m, k);
96         printf("%d\n", b[pos]);

```

```

97 }
98 return 0;
99 }

```

2.5.2 动态区间第 K 大 (主席树套树状数组) ZOJ2112

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 #define inf 0x3f3f3f3f
5 typedef pair<int, int> P;
6 const int maxn = 60010;
7 const ll mod = 1e9+7;
8 const int M = 2500010;
9
10 int n, m, q, tot;
11 struct node
12 {
13     int l, r, v;
14 }tree[M];
15
16 // T是主席树，与上面代码的edit作用一样，S数组就是树状数组，use数组是树状数
17 // 组求和时用的，记录的是树状数组中哪些权值线段树要被用来求和
18 int T[maxn], S[maxn], use[maxn], a[maxn], b[maxn];
19
20 // 记录询问，因为要将修改后的值一起构建主席树，所以将在线转为离线
21 struct Q
22 {
23     // 对于查询区间[l, r]第k大的询问来说，flag为1
24     //若是修改操作，l记录修改的位置，r记录新值，flag为0
25     int l, r, k, flag;
26 }query[10010];
27
28 // 快速找出x在离散化后的位置
29 int HASH(int x)
30 {
31     return lower_bound(b+1, b+m+1, x) - b;
32 }
33
34 // 建静态主席树，和之前的一样
35 int build(int l, int r)
36 {

```

```

36 int pos = ++tot;
37 tree[pos].v = 0;
38 if(l==r) return pos;
39 int mid = (l+r)>>1;
40 tree[pos].l = build(l, mid);
41 tree[pos].r = build(mid+1, r);
42 return pos;
43 }
44
45 // 和之前的静态主席树update差不多，只不过不是直接将tree[pos].v+1,而是加参数v
46 //消除影响v就为-1，添加影响v就为1，其他没什么不同
47 int update(int ed, int l, int r, int p, int v)
48 {
49     int pos = ++tot;
50     tree[pos] = tree[ed];
51     tree[pos].v += v;
52     if(l==r) return pos;
53     int mid = (l+r)>>1;
54     if(p<=mid) tree[pos].l = update(tree[ed].l, l, mid, p, v);
55     else tree[pos].r = update(tree[ed].r, mid+1, r, p, v);
56     return pos;
57 }
58
59 // 树状数组的lowbit
60 int lowbit(int x) { return x&(-x); }
61
62 // 修改操作，修改位置x的影响
63 int add(int x, int v)
64 {
65     // 找出a[x]在离散化后的位置p
66     int p = HASH(a[x]);
67     while(x<=n)
68     {
69         // 修改操作，对树状数组中相应的权值线段树进行修改，消除影响：v=-1，添加
70         // 影响：v=1
71         // 因为树状数组中的权值线段树不需要可持久化，所以直接在原版本上修改就可
72         // 以了
73         S[x] = update(S[x], 1, m, p, v);
74         x+=lowbit(x);
75     }
76 }
77
78 // 树状数组求和，求左子树包含的数字个数，和静态主席树一样的思想，都是先求左
79 // 子树

```

```

77 int sum(int x)
78 {
79     int ret = 0;
80     while(x)
81     {
82         // use[i]记录的就是树状数组中相应的权值线段树的结点位置
83         // 似乎这一句有一点点难以理解, 结合整体代码多看几遍吧
84         ret += tree[tree[use[x]].l].v;
85         x -= lowbit(x);
86     }
87     return ret;
88 }
89
90 // 询问操作。树状数组求[pre, ed]的和, tpre和ted是静态主席树的区间左右顶点的位置, 区间[l, r]第k大
91 int Query(int pre, int ed, int tpre, int ted, int l, int r, int k)
92 {
93     if(l==r)return l;
94     int mid = (l+r)>>1;
95     // sum就是树状数组的求和, 相对于静态主席树, 多了sum求修改操作的影响
96     // tmp为当前左子树的数字个数
97     int tmp = sum(ed) - sum(pre) + tree[tree[tet].l].v - tree[tree[tpre].l].v;
98     // 若左子树的数字个数大于等于k就往左子树继续走, 这里和主席树没什么区别
99     if(tmp >= k)
100     {
101         // 两个for循环是更新左子树需要用到树状数组中的权值线段树的位置, 有那么一丢丢难以理解, 多看几遍?
102         for(int i=ed; i; i-=lowbit(i)) use[i] = tree[use[i]].l;
103         for(int i=pre; i; i-=lowbit(i)) use[i] = tree[use[i]].l;
104         return Query(pre, ed, tree[tpre].l, tree[tet].l, l, mid, k);
105     }
106     else
107     {
108         // 走右子树同理
109         for(int i=ed; i; i-=lowbit(i)) use[i] = tree[use[i]].r;
110         for(int i=pre; i; i-=lowbit(i)) use[i] = tree[use[i]].r;
111         return Query(pre, ed, tree[tpre].r, tree[tet].r, mid+1, r, k-tmp);
112     }
113 }
114
115 int main()
116 {
117     int t; // t个case
118     scanf("%d", &t);

```

```

119 while(t--)
120 {
121     scanf("%d%d", &n, &q);
122     m = tot = 0; // 记得初始化
123     for(int i=1; i<=n; i++)
124     {
125         scanf("%d", &a[i]);
126         b[++m] = a[i];
127     }
128     char op[5];
129     for(int i=1; i<=q; i++)
130     {
131         scanf("%s", op);
132         // 查询操作
133         if(op[0]=='Q')
134         {
135             scanf("%d%d%d", &query[i].l, &query[i].r, &query[i].k);
136             query[i].flag = 1;
137         }
138         else
139         {
140             scanf("%d%d", &query[i].l, &query[i].r);
141             b[++m] = query[i].r; // 注意要将修改后的新值加入到待离散化的b数组
142             query[i].flag = 0;
143         }
144     }
145     // 离散化
146     sort(b+1, b+m+1);
147     m = unique(b+1, b+m+1) - b - 1;
148     // 构建主席树
149     T[0] = build(1, m);
150     for(int i=1; i<=n; i++)
151         T[i] = update(T[i-1], 1, m, HASH(a[i]), 1);
152     // 构建树状数组, 每一个节点都是一颗空的权值线段树
153     for(int i=1; i<=n; i++)
154         S[i] = T[0];
155     // 离线处理q个询问
156     for(int i=1; i<=q; i++)
157     {
158         if(query[i].flag) // 查询
159         {
160             // 两个for循环标记区间[l, r]要使用的树状数组中的权值线段树的位置
161             for(int j=query[i].r; j; j-=lowbit(j)) use[j] = S[j];
162             for(int j=query[i].l-1; j; j-=lowbit(j)) use[j] = S[j];

```

```

163     printf("%d\n", b[Query(query[i].l-1, query[i].r, T[query[i].l-1], T[
        query[i].r], 1, m, query[i].k)]);
164 }
165 else
166 {
167     // 先消除影响
168     add(query[i].l, -1);
169     // 在原数组中更新值
170     a[query[i].l] = query[i].r;
171     // 添加新值的影响
172     add(query[i].l, 1);
173 }
174 }
175 }
176 return 0;
177 }

```

3 DP

3.1 背包

```

1 const int MAXN = 10000;
2 const int SIZE = 100000;
3
4 int dp[SIZE];
5 int volume[MAXN], value[MAXN], c[MAXN];
6 int n, v;          // 总物品数, 背包容量
7
8 // 01背包
9 void ZeroOnepark(int val, int vol)
10 {
11     for (int j = v ; j >= vol; j--)
12     {
13         dp[j] = max(dp[j], dp[j - vol] + val);
14     }
15 }
16
17 // 完全背包
18 void Completopark(int val, int vol)
19 {
20     for (int j = vol; j <= v; j++)
21     {

```

```

22         dp[j] = max(dp[j], dp[j - vol] + val);
23     }
24 }
25
26 // 多重背包
27 void Multiplepark(int val, int vol, int amount)
28 {
29     if (vol * amount >= v)
30     {
31         Completopark(val, vol);
32     }
33     else
34     {
35         int k = 1;
36         while (k < amount)
37         {
38             ZeroOnepark(k * val, k * vol);
39             amount -= k;
40             k <<= 1;
41         }
42         if (amount > 0)
43         {
44             ZeroOnepark(amount * val, amount * vol);
45         }
46     }
47 }
48
49 int main()
50 {
51     while (cin >> n >> v)
52     {
53         for (int i = 1 ; i <= n ; i++)
54         {
55             cin >> volume[i] >> value[i] >> c[i];          // 费用, 价值, 数量
56         }
57         memset(dp, 0, sizeof(dp));
58         for (int i = 1; i <= n; i++)
59         {
60             Multiplepark(value[i], volume[i], c[i]);
61         }
62         cout << dp[v] << endl;
63     }
64     return 0;
65 }

```

3.2 最长公共递增子序列

```

1  /*
2  * 最长公共递增子序列  $O(n^2)$ 
3  * f记录路径,DP记录长度,用a对b扫描,逐步最优化。
4  */
5  const int N = 1010;
6
7  int f[N][N], dp[N];
8
9  int gcis(int a[], int la, int b[], int lb, int ans[])
10 { // a[1...la], b[1...lb]
11     int i, j, k, mx;
12     memset(f, 0, sizeof(f));
13     memset(dp, 0, sizeof(dp));
14     for (i = 1; i <= la; i++)
15     {
16         memcpy(f[i], f[i-1], sizeof(f[0]));
17         for (k = 0, j = 1; j <= lb; j++)
18         {
19             if (b[j - 1] < a[i - 1] && dp[j] > dp[k])
20             {
21                 k = j;
22             }
23             if (b[j - 1] == a[i - 1] && dp[k] + 1 > dp[j])
24             {
25                 dp[j] = dp[k] + 1,
26                 f[i][j] = i * (lb + 1) + k;
27             }
28         }
29     }
30     for (mx = 0, i = 1; i <= lb; i++)
31     {
32         if (dp[i] > dp[mx])
33         {
34             mx = i;
35         }
36     }
37     for (i = la * lb + la + mx, j = dp[mx]; j; i = f[i / (lb + 1)][i % (lb + 1)], j--)
38     {

```

```

39         ans[j - 1] = b[i % (lb + 1) - 1];
40     }
41     return dp[mx];
42 }

```

3.3 最长公共子序列

```

1  const int N = 1010;
2
3  int a[N][N];
4
5  int LCS(const char *s1, const char *s2)
6 { // s1:0...m, s2:0...n
7     int m = (int)strlen(s1), n = (int)strlen(s2);
8     int i, j;
9     a[0][0] = 0;
10    for (i = 1; i <= m; ++i)
11    {
12        a[i][0] = 0;
13    }
14    for (i = 1; i <= n; ++i)
15    {
16        a[0][i] = 0;
17    }
18    for (i = 1; i <= m; ++i)
19    {
20        for (j = 1; j <= n; ++j)
21        {
22            if (s1[i - 1] == s2[j - 1])
23            {
24                a[i][j] = a[i - 1][j - 1] + 1;
25            }
26            else if (a[i - 1][j] > a[i][j - 1])
27            {
28                a[i][j] = a[i - 1][j];
29            }
30            else
31            {
32                a[i][j] = a[i][j - 1];
33            }
34        }
35    }

```

```
36 | return a[m][n];  
37 | }
```

4 字符串

5 数学

6 STL

7 计算几何

8 其它