

华南师范大学
ACM-ICPC 集训队
DeDeRong



2019 年 10 月 16 日

目录

1	图论	3
1.1	最短路径	3
1.1.1	Dijkstra	3
1.1.2	Dijkstra 优化	3
1.1.3	Floyd	3
1.1.4	Bellman-Ford	4
1.1.5	SPFA	4
1.2	LCA	5
1.2.1	倍增	5
1.2.2	RMQ	6
1.3	强连通分量	7
1.3.1	Tarjan	7
1.4	割点	7
1.4.1	Tarjan	7
1.5	桥	8
1.5.1	Tarjan	8
1.6	最大流	9
1.6.1	Dinic	9
1.6.2	Dinic 优化	10
2	数据结构	11
3	DP	11
4	字符串	11
5	数学	11
6	STL	11
7	计算几何	11
8	其它	11

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 LCA

1.2.1 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 }

```

1.2.2 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 }

```

1.3 强连通分量

1.3.1 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 }

```

1.4 割点

1.4.1 Tarjan

```

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 }

```

1.5 桥

1.5.1 Tarjan

```

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 }

```

1.6 最大流

1.6.1 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%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 }

```

1.6.2 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 }

```

2 数据结构

3 DP

4 字符串

5 数学

6 STL

7 计算几何

8 其它