华南师范大学
ACM-ICPC 集训队
DeDeRong

2019 年 10 月 16 日

目录

1 图论

1.1 最短路径

1.1.1 Dijkstra

```
const int maxn = 1e4;
const int inf = 0x3f3f3f3f;

//d数组用来记录源点s到顶点i的最短距离
//v表示该顶点是否在顶点集S中
//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
//n为顶点数量
int d[maxn], v[maxn];
int g[maxn][maxn];
int n;
void dij(int s)
{
  memset(v, 0, sizeof(v));
  for(int i=1;i<=n;i++)
  d[i] = g[s][i];
  v[s] = 1;
  for(int i=1;i<=n;i++)
  {
    int u = 0;
    for(int j=1;j<=n;j++)
    {
      if(!v[j] && (u==0 || d[j] < d[u]))
        u = j;
    }
    if(u==0)return ;
    v[u] = 1;
    for(int j=1;j<=n;j++)
    {
      d[j] = min(d[j], d[u]+g[u][j]);
    }
  }
}
```

1.1.2 Dijkstra 优化

```
const int maxn = 1e4;
const int inf = 0x3f3f3f3f;
typedef pair<int, int> P; //first表示最短距离，second表示顶点编号
//边：to表示这条边指向的顶点，权值为w
struct Edge
{
  int to, w;
};
//用vector实现邻接表
vector<Edge> g[maxn];
int d[maxn]; //记录源点到顶点i的最短距离
int n;

void dij(int s)
{
  priority_queue<P, vector<P>, greater<P> > q;
  memset(d, inf, sizeof(d));
  d[s] = 0;
  q.push(P(0, s));
  while(!q.empty())
  {
    P p = q.top();
    q.pop();
    int u = p.second;
    if(d[u] < p.first) continue;
    for(int i=0; i<g[u].size(); i++)
    {
      Edge e = g[u][i];
      if(d[e.to] > d[u] + e.w)
      {
        d[e.to] = d[u] + e.w;
        q.push(P(d[e.to], e.to));
      }
    }
  }
}
```

1.1.3 Floyd

```
int g[maxn][maxn];
int n;
void floyd()
{
```

```
  for(int k=1;k<=n;k++)
    for(int i=1;i<=n;i++)
      for(int j=1;j<=n;j++)
        g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
}
```

1.1.4 Bellman-Ford

```
const int maxn = 1e4;
const int inf = 0x3f3f3f3f; //常用于表示无穷大

//边结构体，记录u—>v的边，权值为w
struct Edge
{
  int u, v, w;
  Edge(int uu, int vv, int ww) { u=uu; v=vv; w=ww; }
  Edge(){}
}e[maxn];
int edgecnt; // 边的数量
//加边操作
void addEdge(int u, int v, int w)
{
  e[edgecnt++] = Edge(u, v, w);
}

int n; //顶点总数
int d[maxn]; //记录最短距离的数组

//存在负权回路则返回true，否则返回false
bool bellman_ford(int s)
{
  memset(d, inf, sizeof(d));
  d[s] = 0;
  //进行n—1次松弛操作,第n次检查是否含有负权回路
  for(int i=1;i<=n;i++)
  {
    int flag = 0;
    for(int j=0; j<edgecnt; j++)
    {
      Edge t = e[j];
      int u, v, w;
      u = t.u; v = t.v; w = t.w;
      if(d[v] > d[u] + w)
      {
        d[v] = d[u] + w;
        flag = 1;
      }
    }
    if(!flag) return false;
    if(i==n && flag) return true;
  }
  return false;
}
```

1.1.5 SPFA

```
const int maxn = 1e4;
const int inf = 0x3f3f3f3f; //常用于表示无穷大

//边结构体，to表示边指向的顶点编号，权值为w
struct Edge
{
  int to, w;
  Edge(int tt, int ww) { to = tt; w = ww; }
  Edge(){}
};
//vector实现的邻接表
vector<Edge> g[maxn];
int n;//顶点数
//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
int d[maxn], inq[maxn], cnt[maxn];
//初始化
void init()
{
  memset(d, inf, sizeof(d));
  memset(inq, 0, sizeof(inq));
  memset(cnt, 0, sizeof(cnt));
}
//返回true表示存在负权回路
bool spfa(int s)
{
  init();
  d[s] = 0;
  inq[s] = 1;
```

```
  cnt[s] = 1;
  queue<int> q;
  q.push(s);
  while(!q.empty())
  {
    int u = q.front();
    inq[u] = 0;
    q.pop();
    for(int i=0;i < g[u].size(); i++)
    {
      Edge e = g[u][i];
      if(d[e.to] > d[u] + e.w)
      {
        d[e.to] = d[u] + e.w;
        if(inq[e.to] == 0)
        {
          inq[e.to] = 1;
          q.push(e.to);
          cnt[e.to]++;
          if(cnt[e.to] > n) return true;
        }
      }
    }
  }
  return true;
}
```

1.2 LCA

1.2.1 倍增

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define inf 0x3f3f3f3f
typedef pair<int, int> P;
const int maxn = 5e5+5;
const ll mod = 1e9+7;

vector<int> son[maxn]; // 存储儿子顶点
// 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
int dep[maxn], n, m, rt, fa[maxn][20];
int v[maxn]={0}; // 是否访问标记

// pre是父顶点，rt是当前顶点
void dfs(int pre, int rt)
{
  dep[rt] = dep[pre]+1; // 当前顶点的深度为父顶点加一
  fa[rt][0] = pre; // 当前顶点向上跳一步为父顶点
  v[rt] = 1; // 访问
  // dp预处理
  for(int i=1;i<=19;i++)
  fa[rt][i] = fa[fa[rt][i-1]][i-1];
  // 继续dfs
  for(int i=0;i<son[rt].size();i++)
  if(v[son[rt][i]]==0)
  dfs(rt, son[rt][i]);
}

// 求解LCA(a, b)
int lca(int a, int b)
{
  if(dep[a] < dep[b])
    swap(a, b);
  for(int i=19;i>=0;i--)
  {
    if(dep[a]-dep[b] >= (1<<i))
    {
      a = fa[a][i];
    }
  }
  if(a==b)return a;
  for(int i=19;i>=0;i--)
  {
    if(fa[a][i] != fa[b][i])
    {
      a = fa[a][i];
      b = fa[b][i];
    }
  }
  return fa[a][0];
}

int main()
{
  scanf("%d%d%d", &n, &m, &rt);
```

```
  for(int i=1;i<n;i++)
  {
    int a, b;
    scanf("%d%d", &a, &b);
    son[a].push_back(b);
    son[b].push_back(a);
  }
  memset(fa, 0, sizeof(fa));
  memset(dep, inf, sizeof(dep));
  v[0]=1;
  dep[0] = 0;
  dfs(0, rt);
  for(int i=1;i<=m;i++)
  {
    int a, b;
    scanf("%d%d", &a, &b);
    printf("%d\n", lca(a, b));
  }
  return 0;
}
```

1.2.2 RMQ

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define inf 0x3f3f3f3f
typedef pair<int, int> P;
const int maxn = 5e5+5;
const ll mod = 1e9+7;

vector<int> g[maxn]; // 存图
// 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
int dep[maxn<<1]={0}, vis[maxn<<1]={0}, id[maxn]={0}, st[maxn<<1][25];
// dfs序计数用，看代码能理解
int dfs_c=1;

// 父顶点为pre， 当前顶点为now， 当前深度为d
void dfs(int pre, int now, int d)
{
  id[now] =  dfs_c; // now顶点在DFS序中第一次出现的位置是dfs_c
  dep[dfs_c] = d; // 记录now的深度
  vis[dfs_c++] = now; // DFS序中第dfs_c个顶点是now，同时将dfs_c加一
  for(int i=0;i<g[now].size();i++)
  {
    if(g[now][i]!=pre)
    {
      dfs(now, g[now][i], d+1);
      vis[dfs_c] = now;
      dep[dfs_c++] = d;
    }
  }
}

// 预处理st表
void getSt(int n)
{
  for(int i=1;i<=n;i++)
  st[i][0] = i;
  for(int j=1; (1<<j)<=n; j++)
  {
    for(int i=1;i+(1<<j)<=n; i++)
    {
      int a = st[i][j-1], b = st[i+(1<<(j-1))][j-1];
      if(dep[a] < dep[b])
        st[i][j] = a;
      else st[i][j] = b;
    }
  }
}

// 查询DFS序中区间[l, r]深度最小的顶点在DFS序中的位置
int query(int l, int r)
{
  int k = log2(r-l+1);
  int a = st[l][k];
  int b = st[r-(1<<k)+1][k];
  // 返回深度较小的那一个顶点在DFS序中的位置
  if(dep[a]<dep[b])return a;
  else return b;
}

// 求LCA(a, b)
int lca(int a, int b)
{
```

```
  int x, y;
  x = id[a], y = id[b];
  if(x>y)return vis[query(y, x)];
  else return vis[query(x, y)];
}

// 检查用的
void check(int n)
{
  for(int i=1;i<=dfs_c;i++)cout<<dep[i]<<" ";cout<<"\n\n";
  for(int i=1;i<=dfs_c;i++)cout<<vis[i]<<" ";cout<<"\n\n";
  for(int i=1;i<=n;i++)cout<<id[i]<<" ";cout<<"\n\n";
}

int main()
{
  int n, m, rt;
  scanf("%d%d%d", &n, &m, &rt);
  for(int i=1;i<n;i++)
  {
    int a, b;
    scanf("%d%d", &a, &b);
    g[a].push_back(b);
    g[b].push_back(a);
  }
  dfs(0, rt, 1);
  getSt(dfs_c);
  //check(n);
  for(int i=1;i<=m;i++)
  {
    int a, b;
    scanf("%d%d", &a, &b);
    printf("%d\n", lca(a, b));
  }
  return 0;
}
```

1.3 强连通分量

1.3.1 Tarjan

```
vector<int> g[maxn];
int low[maxn], dfn[maxn], sta[maxn], ins[maxn], belong[maxn];
int cnt, ind, tot; //cnt: 强连通分量的数量， ind: 时间戳， tot: sta的top

void init()
{
  memset(ins, 0, sizeof(ins));
  memset(belong, 0, sizeof(belong));
  memset(dfn, 0, sizeof(dfn));
  cnt = ind = tot = 0;
}

void Tarjan(int u)
{
  low[u] = dfn[u] = ++ind;
  ins[u] = 1;
  sta[++tot] = u;
  for(int i=0;i<g[u].size();i++)
  {
    int v = g[u][i];
    if(!dfn[v])
    {
      Tarjan(v);
      low[u] = min(low[u], low[v]);
    }
    else if(ins[v])
    low[u] = min(low[u], dfn[v]);
  }
  int p;
  if(low[u] == dfn[u])
  {
    ++cnt;
    do
    {
      p = sta[tot--];
      belong[p] = cnt;
      ins[p] = 0;
    }while(p != u);
  }
}
```

1.4 割点

1.4.1 Tarjan

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

void init()
{
  memset(dfn, 0, sizeof(dfn));
  memset(iscut, 0, sizeof(iscut));
  ind = 0;
}

// pa为u的父节点，初始时Tarjan(i, i)
void Tarjan(int u, int pa)
{
  int cnt = 0; //用来记录子树的数量
  low[u] = dfn[u] = ++ind;
  for(int i=0;i<g[u].size();i++)
  {
    int v = g[u][i];
    if(!dfn[v])
    {
      Tarjan(v, u);
      low[u] = min(low[u], low[v]);
      // 若low[v]>=dfn[u]，并且u不是根节点，则u是割点
      if(low[v] >= dfn[u] && pa!=u)
      iscut[u] = 1;
      // 若u是根节点，则cnt++
      if(u == pa)
        cnt++;
    }
    else if(v != pa) //若v不等于父节点
    low[u] = min(low[u], dfn[v]);
  }
  if(cnt>=2 && u==pa) //根节点子树数量大于等于2，则为割点
    iscut[u] = 1;
}
```

1.5 桥

1.5.1 Tarjan

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

int head[maxn], low[maxn], dfn[maxn];
int ind, tot; // tot是边的数量

void init()
{
  memset(head, -1, sizeof(head));
  memset(dfn, 0, sizeof(dfn));
  ind = tot = 0;
}

void addedge(int u, int v)
{
  e[tot].to = v;
  e[tot].next = head[u];
  e[tot].iscut = 0;
  head[u] = tot++;
}

void Tarjan(int u, int pa)
{
  low[u] = dfn[u] = ++ind;
  for(int i=head[u]; ~i; i = e[i].next)
  {
    int v = e[i].to;
    if(v == pa) continue;
    if(!dfn[v])
    {
      Tarjan(v, u);
      low[u] = min(low[u], low[v]);
      // 是桥
      if(low[v] > dfn[u])
      {
        e[i].iscut = e[i^1].iscut = 1;
      }
    }
    else
```

```
    {
      low[u] = min(low[u], dfn[v]);
    }
  }
}
```

1.6 最大流

1.6.1 Dinic

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
const int maxn = 1e6+5;
const int inf = 0x3f3f3f3f;
const int mod = 1e9+7;

// 用链式前向星来存储图
struct ed
{
  int to, val, ne;
}edge[maxn<<1];
int head[maxn], dep[maxn];
// 顶点数n，边数m，源点s，汇点e，加边时的指针tot
int n, m, s, e, tot;

void init()
{
  tot = -1;
  memset(head, -1, sizeof(head));
}

void addEdge(int u, int v, int val)
{
  edge[++tot].to = v;
  edge[tot].val = val;
  edge[tot].ne = head[u];
  head[u] = tot;
}

// 就是最普通的bfs
int bfs()
{
  memset(dep, -1, sizeof(dep));
  dep[s] = 0;
  queue<int> q;
  q.push(s);
  while(!q.empty())
  {
    int u = q.front();
    q.pop();
    for(int i=head[u]; ~i; i=edge[i].ne)
    {
      int v = edge[i].to;
      if(dep[v]==-1 && edge[i].val>0)
      {
        dep[v] = dep[u]+1;
        q.push(v);
      }
    }
  }
  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。
}

// 当前顶点u，当前流量flow
// 初始时dfs(s, inf)
int dfs(int u, int flow)
{
  if(u == e)return flow;
  for(int i=head[u]; ~i; i=edge[i].ne)
  {
    int v = edge[i].to;
    if(dep[v]==dep[u]+1 && edge[i].val)
    {
      int a = dfs(v, min(flow, edge[i].val));
      if(a>0) //若找到增广路
      {
        edge[i].val -= a;
        edge[i^1].val += a;
        return a;
      }
    }
  }
  return 0;
}
```

```

ll dinic()
{
  ll ans = 0;
  while(bfs())
  {
    int a = dfs(s, (1<<30));
    ans += a;
  }
  return ans;
}

int main()
{
  scanf("%d%d%d%d", &n, &m, &s, &e);
  init();
  for(int i=1;i<=m;i++)
  {
    int u, v, w;
    scanf("%d%d%d", &u, &v, &w);
    addEdge(u, v, w);
    addEdge(v, u, 0); //反边
  }
  printf("%lld\n", dinic());
  return 0;
}
```

1.6.2 Dinic 优化

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
const int maxn = 1e6+5;
const int inf = 0x3f3f3f3f;
const int mod = 1e9+7;

struct ed
{
  int to, val, ne;
}edge[maxn<<1];
int head[maxn],dep[maxn], cur[maxn];
int n, m, s, e, tot;

void init()
{
  tot = -1;
  memset(head, -1, sizeof(head));
}

void addEdge(int u, int v, int val)
{
  edge[++tot].to = v;
  edge[tot].val = val;
  edge[tot].ne = head[u];
  head[u] = tot;
}

int bfs()
{
  memset(dep, -1, sizeof(dep));
  dep[s] = 0;
  queue<int> q;
  q.push(s);
  while(!q.empty())
  {
    int u = q.front();
    q.pop();
    for(int i=head[u]; ~i; i=edge[i].ne)
    {
      int v = edge[i].to;
      if(dep[v]==-1 && edge[i].val>0)
      {
        dep[v] = dep[u]+1;
        q.push(v);
      }
    }
  }
  return (dep[e] != -1);
}

int dfs(int u, int flow)
{
  if(u == e)return flow;
  // rflow用于多路增广，表示流入到顶点u的剩余未流出的流量
  int rflow = flow;
```

```
  // 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
  for(int& i=cur[u]; ~i; i=edge[i].ne)
  {
    int v = edge[i].to;
    if(dep[v]==dep[u]+1 && edge[i].val)
    {
      int a = dfs(v, min(rflow, edge[i].val));
      edge[i].val -= a;
      edge[i^1].val += a;
      rflow -= a; // 剩余流量要减少
      if(rflow<=0)break; // 若没有剩余流量了，就break
    }
  }
  // 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
  if(rflow == flow)
  dep[u] = -2;
  return flow - rflow; // 返回流出的流量
}

ll dinic()
{
  ll ans = 0;
  while(bfs())
  {
    // 新一轮dfs之前要对cur进行初始化
    for(int i=1;i<=n;i++)cur[i] = head[i];
    int a = dfs(s, (1<<30));
    ans += a;
  }
  return ans;
}

int main()
{
  scanf("%d%d%d%d", &n, &m, &s, &e);
  init();
  for(int i=1;i<=m;i++)
  {
    int u, v, w;
    scanf("%d%d%d", &u, &v, &w);
    addEdge(u, v, w);
    addEdge(v, u, 0);
  }
  printf("%lld\n", dinic());
  return 0;
}
```