目录

1 前向星

```
struct edge{
    int v,next,w;
}e[MAXM*2];
void add(int a,int b,int c)
{
    e[idx].v = b;e[idx].w = c;
    e[idx].next = first[a];
    first[a] = idx++;
}
void init()
{
    memset(first,-1,sizeof(first));
    idx = 1;
}
```

2 最短路

2.1 Dijkstra+ 堆优化

```
struct node{
    int id,cost;
    node(int a,int b):id(a),cost(b){}
    bool operator < (const node &t) const
    {
        return t.cost < cost;
    }
};
void dijkstra(int x)
{
    priority_queue<node> q;
    for(int i = 1;i <= n;i++){
        vis[i] = 0;
        dist[i] = INF;
    }
    dist[x] = 0;
    q.push(node(x,0));
    while(!q.empty()){
        node cur = q.top();
        q.pop();
        if(vis[cur.id]) continue;
        vis[cur.id] = 1;
        for(int i = first[cur.id];i!=-1;i=e[i].next){
```

```
            if(dist[e[i].v]>dist[cur.id]+e[i].w){
                dist[e[i].v] = dist[cur.id] + e[i].w;
                q.push(node(e[i].v,dist[e[i].v]));
            }
        }
    }
}
```

2.2 SPFA（判环）

```
bool inq[MAXN];//是否在队列中
int cnt[MAXN];//入队列次数
int dist[MAXN];
bool spfa(int start)
{
    queue<int> q;
    for(int i = 1;i <= n;i++){
        dist[i] = INF;
        inq[i] = 0;
        cnt[i] = 0;
    }
    dist[start] = 0;
    cnt[start] = 1;
    q.push(start);
    while(!q.empty()){
        int cur = q.front();
        q.pop();
        inq[cur] = 0;
        for(int i = first[cur];i != -1;i = e[i].next){
            int v = e[i].v;
            int w = e[i].w;
            if(dist[v] > dist[cur] + w){
                dist[v] = dist[cur] + w;
                if(!inq[v]){
                    inq[v] = 1;
                    q.push(v);
                    if(++cnt[v] > n) return 0;
                    //若入队列次数大于n，说明存在环
                }
            }
        }
    }
    return 1;
}
```

2.3 Floyd

```
void floyd()
{
    for(int k = 1;k <= n;k++){
        for(int i = 1;i <= n;i++){
            for(int j = 1;j <= n;j++){
                if(cost[i][j] > cost[i][k] + cost[k][j]){
                    cost[i][j] = cost[i][k] + cost[k][j];
                    path[j] = k;//path[]记录最短路径
                }
            }
        }
    }
}
```

3 第 K 短路

```
int dist[1010];
int first[1010];//正向图
int rfirst[1010];//反向图
int vis[1010];
int times[1010];//点的访问次数
int idx,ridx;
struct node{
    int p,g,h;//p表示点的编号，g为点到终点的距离（估价），h为点到起点的距离（实际）
    bool operator < (const node &t)const
    {
        return t.g+t.h<g+h;
    }
};
struct qnode{
    int id;
    int cost;
    qnode(int a,int b):id(a),cost(b){}
    bool operator < (const qnode &t) const
    {
        return t.cost < cost;
    }
};
struct edge{
    int v,next,w;
}e[100100],re[100100];
void add(int a,int b,int c)
{
    e[idx].v = b;e[idx].w = c;
    e[idx].next = first[a];
    first[a]=idx++;
}
void radd(int a,int b,int c)
```

```
{
    re[ridx].v=b;re[ridx].w=c;
    re[ridx].next = rfirst[a];
    rfirst[a]=ridx++;
}
void dijkstra(int x)
{
    priority_queue<qnode> q;
    for(int i = 1;i <= n;i++){
        vis[i]=0;
        dist[i]=INF;
    }
    dist[x]=0;
    q.push(qnode(x,0));
    while(!q.empty()){
        qnode cur = q.top();
        q.pop();
        if(vis[cur.id]) continue;
        vis[cur.id] = 1;
        for(int i = rfirst[cur.id];i!=-1;i=re[i].next){
            if(dist[re[i].v]>dist[cur.id]+re[i].w){
                dist[re[i].v] = dist[cur.id]+re[i].w;
                q.push(qnode(re[i].v,dist[re[i].v]));
            }
        }
    }
}

int A_star(int start,int end,int k)
{
    memset(times,0,sizeof(times));
    priority_queue<node> q;
    node t1;
    t1.g = t1.h = 0;
    t1.p = start;
    q.push(t1);
    while(!q.empty()){
        node t = q.top();
        q.pop();
        times[t.p]++;
        if(times[t.p]==k&&t.p==end) return t.h+t.g;
        if(times[t.p]>k) continue;
        for(int i = first[t.p];i!=-1;i=e[i].next){
            node tmp;
            tmp.p = e[i].v;
            tmp.g = dist[e[i].v];
            tmp.h = e[i].w + t.h;
            q.push(tmp);
        }
    }
```

```
    return -1;
}
void init()
{
    memset(first,-1,sizeof(first));
    memset(rfirst,-1,sizeof(rfirst));
    idx = 1;
    ridx = 1;
}
int main()
{
    scanf("%d%d",&n,&m);
    init();
    for(int i = 1;i <= m;i++){
        int u,v,w;
        scanf("%d%d%d",&u,&v,&w);
        add(u,v,w);
        radd(v,u,w);
    }
    scanf("%d%d%d",&start,&end,&k);
    if(start==end) k++;//若题目要求必须走动时加上
    dijkstra(end);
    int ans = A_star(start,end,k);
    printf("%d\n",ans);
    return 0;
}
```

4 最小环

4.1 Floyd

```
int pos[MAXN][MAXN];//pos[i][j]:i到j的最短路的路径
vector<int> path;//最小环路径
int ans;//最小环
//ans == INF  说明无环
void getpath(int x,int y)
{
    if(!pos[x][y]) return;
    getpath(x,pos[x][y]);
    path.push_back(pos[x][y]);
    getpath(pos[x][y],y);
}
void floyd()
{
    ans = INF;
    memcpy(dist,g,sizeof(dist));
    for(int k = 1;k <= n;k++){
        for(int i = 1;i < k;i++){
```

```
            for(int j = i+1;j < k;j++){
                if(dist[i][j]+g[j][k]+g[k][i]<ans){
                    ans = dist[i][j]+g[j][k]+g[k][i];
                    ans.clear();
                    ans.push_back(i);
                    getpath(i,j);
                    ans.push_back(j);
                    ans.push_back(k);
                }
            }
        }
        for(int i = 1;i <= n;i++){
            for(int j = 1;j <= n;j++){
                if(dist[i][k]+dist[k][j]<dist[i][j]){
                    dist[i][j] = dist[i][k]+dist[k][j];
                    pos[i][j] = k;
                }
            }
        }
    }
}
```

4.2 Dijkstra+ 剪枝

```
//边的计数idx从2开始，idx = 2;
ll dijkstra(int x,int y,int k)
{
    priority_queue<node> q;
    for(int i = 1;i <= cnt;i++){
        vis[i] = 0;
        dist[i] = INF;
    }
    dist[x] = 0;
    q.push(node(x,0));
    while(!q.empty()){
        node cur = q.top();
        q.pop();
        if(cur.cost>ans-e[k].w) break;  //剪枝
        if(vis[cur.id]) continue;
        vis[cur.id] = 1;
        for(int i = first[cur.id];i!=-1;i=e[i].next){
            if(i == k||i == (k^1)) continue;
            if(dist[e[i].v]>dist[cur.id]+e[i].w){
                dist[e[i].v] = dist[cur.id]+e[i].w;
                q.push(node(e[i].v,dist[e[i].v]));
            }
        }
    }
    return dist[y];
```

```
}
for(int i = 2;i <= m*2+1;i+=2){
    ans = min(ans,dijkstra(e[i].u,e[i].v,i)+e[i].w);
}
```

5 网络流

5.1 二分图匹配

```
int dfs(int u)
{
    for(int i = first[u];i!=-1;i = e[i].next){
        int v = e[i].to;
        if(!vis[v]){
            vis[v] = 1;
            if(linker[v]==-1||dfs(linker[v])){
                linker[v] = u;
                return 1;
            }
        }
    }
    return 0;
}
int hungary()
{
    int res =  0;
    memset(linker,-1,sizeof(linker));
    for(int  u = 1; u <= n;u++){
        memset(vis,0,sizeof(vis));
        if(dfs(u)){
            res++;
        }
    }
    return res;
}
```

5.2 最大流

5.2.1 Dinic

```
//1.用BFS建立分层图
//2.用DFS的方法寻找一条由源点到汇点的路径,获得这条路径的流量x.
//根据这条路径修改整个图,将所经之处正向边流量减少x,反向边流量增加x
//重复步骤2,直到DFS找不到新的路径时,重复步骤1
//时间复杂度O(n^2*m)
int bfs()
{
```

```
    memset(dis,-1,sizeof(dis));
    queue<node> q;
    q.push(node(1,0));
    dis[1] = 0;
    while(!q.empty()){
        node cur = q.front();
        q.pop();
        for(int i = first[cur.id];i!=-1;i = e[i].next){
            if(e[i].w == 0) continue;
            if(dis[e[i].v] == -1){
                dis[e[i].v] = cur.cost+1;
                q.push(node(e[i].v,dis[e[i].v]));
            }
        }
    }
    if(dis[n] == -1) return 0;
    return 1;
}
int dfs(int x,int low)
{
    if(x == n) return low;
    for(int i = first[x];i!=-1;i = e[i].next){
        int a = 0;
        if(e[i].w > 0&&dis[e[i].v] == dis[x]+1&&(a = dfs(e[i].v,min(low,e[i].w)))){
            e[i].w -= a;
            add(e[i].v,x,a);
            return a;
        }
    }
    return 0;
}
int main()
{
    init();
    scanf("%d%d",&n,&m);
    for(int i = 1;i <= m;i++){
      int u,v,w;
      scanf("%d%d%d",&u,&v,&w);
      add(u,v,w);
    }
    ll ans = 0;
    int sum;
    while(bfs()){
      while(sum = dfs(1,INF)){
        ans += sum;
      }
    }
    printf("Max flow: %lld\n",ans);
    return 0;
}
```