

图论

目录

1 前向星

```
1 struct edge{
2     int v,next,w;
3 }e[MAXM*2];
4 void add(int a,int b,int c)
5 {
6     e[idx].v = b;e[idx].w = c;
7     e[idx].next = first[a];
8     first[a] = idx++;
9 }
10 void init()
11 {
12     memset(first,-1,sizeof(first));
13     idx = 1;
14 }
```

2 最短路

2.1 Dijkstra+ 堆优化

```
1 struct node{
2     int id,cost;
3     node(int a,int b):id(a),cost(b){}
4     bool operator < (const node &t) const
5     {
6         return t.cost < cost;
7     }
8 };
9 void dijkstra(int x)
10 {
11     priority_queue<node> q;
12     for(int i = 1;i <= n;i++){
13         vis[i] = 0;
14         dist[i] = INF;
15     }
16     dist[x] = 0;
17     q.push(node(x,0));
18     while(!q.empty()){
19         node cur = q.top();
20         q.pop();
21         if(vis[cur.id]) continue;
22         vis[cur.id] = 1;
23         for(int i = first[cur.id];i!=-1;i=e[i].next){
```

```

24         if(dist[e[i].v]>dist[cur.id]+e[i].w){
25             dist[e[i].v] = dist[cur.id] + e[i].w;
26             q.push(node(e[i].v,dist[e[i].v]));
27         }
28     }
29 }
30 }

```

2.2 SPFA (判环)

```

1  bool inq[MAXN]; //是否在队列中
2  int cnt[MAXN]; //入队列次数
3  int dist[MAXN];
4  bool spfa(int start)
5  {
6      queue<int> q;
7      for(int i = 1; i <= n; i++){
8          dist[i] = INF;
9          inq[i] = 0;
10         cnt[i] = 0;
11     }
12     dist[start] = 0;
13     cnt[start] = 1;
14     q.push(start);
15     while(!q.empty()){
16         int cur = q.front();
17         q.pop();
18         inq[cur] = 0;
19         for(int i = first[cur]; i != -1; i = e[i].next){
20             int v = e[i].v;
21             int w = e[i].w;
22             if(dist[v] > dist[cur] + w){
23                 dist[v] = dist[cur] + w;
24                 if(!inq[v]){
25                     inq[v] = 1;
26                     q.push(v);
27                     if(++cnt[v] > n) return 0;
28                     //若入队列次数大于n, 说明存在环
29                 }
30             }
31         }
32     }
33     return 1;
34 }

```

2.3 Floyd

```

1 void floyd()
2 {
3     for(int k = 1;k <= n;k++){
4         for(int i = 1;i <= n;i++){
5             for(int j = 1;j <= n;j++){
6                 if(cost[i][j] > cost[i][k] + cost[k][j]){
7                     cost[i][j] = cost[i][k] + cost[k][j];
8                     path[j] = k;//path[] 记录最短路径
9                 }
10            }
11        }
12    }
13 }

```

3 第 K 短路

```

1 int dist[1010];
2 int first[1010];//正向图
3 int rfirst[1010];//反向图
4 int vis[1010];
5 int times[1010];//点的访问次数
6 int idx,ridx;
7 struct node{
8     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）
9     bool operator < (const node &t)const
10     {
11         return t.g+t.h<g+h;
12     }
13 };
14 struct qnode{
15     int id;
16     int cost;
17     qnode(int a,int b):id(a),cost(b){}
18     bool operator < (const qnode &t) const
19     {
20         return t.cost < cost;
21     }
22 };
23 struct edge{
24     int v,next,w;
25 }e[100100],re[100100];
26 void add(int a,int b,int c)
27 {
28     e[idx].v = b;e[idx].w = c;
29     e[idx].next = first[a];
30     first[a]=idx++;
31 }
32 void radd(int a,int b,int c)

```

```

33 {
34     re[ridx].v=b;re[ridx].w=c;
35     re[ridx].next = rfirst[a];
36     rfirst[a]=ridx++;
37 }
38 void dijkstra(int x)
39 {
40     priority_queue<qnode> q;
41     for(int i = 1;i <= n;i++){
42         vis[i]=0;
43         dist[i]=INF;
44     }
45     dist[x]=0;
46     q.push(qnode(x,0));
47     while(!q.empty()){
48         qnode cur = q.top();
49         q.pop();
50         if(vis[cur.id]) continue;
51         vis[cur.id] = 1;
52         for(int i = rfirst[cur.id];i!=-1;i=re[i].next){
53             if(dist[re[i].v]>dist[cur.id]+re[i].w){
54                 dist[re[i].v] = dist[cur.id]+re[i].w;
55                 q.push(qnode(re[i].v,dist[re[i].v]));
56             }
57         }
58     }
59 }
60
61 int A_star(int start,int end,int k)
62 {
63     memset(times,0,sizeof(times));
64     priority_queue<node> q;
65     node t1;
66     t1.g = t1.h = 0;
67     t1.p = start;
68     q.push(t1);
69     while(!q.empty()){
70         node t = q.top();
71         q.pop();
72         times[t.p]++;
73         if(times[t.p]==k&&t.p==end) return t.h+t.g;
74         if(times[t.p]>k) continue;
75         for(int i = first[t.p];i!=-1;i=e[i].next){
76             node tmp;
77             tmp.p = e[i].v;
78             tmp.g = dist[e[i].v];
79             tmp.h = e[i].w + t.h;
80             q.push(tmp);
81         }
82     }

```

```

83     return -1;
84 }
85 void init()
86 {
87     memset(first,-1,sizeof(first));
88     memset(rfirst,-1,sizeof(rfirst));
89     idx = 1;
90     ridx = 1;
91 }
92 int main()
93 {
94     scanf("%d%d",&n,&m);
95     init();
96     for(int i = 1;i <= m;i++){
97         int u,v,w;
98         scanf("%d%d%d",&u,&v,&w);
99         add(u,v,w);
100        radd(v,u,w);
101    }
102    scanf("%d%d%d",&start,&end,&k);
103    if(start==end) k++; //若题目要求必须走动时加上
104    dijkstra(end);
105    int ans = A_star(start,end,k);
106    printf("%d\n",ans);
107    return 0;
108 }

```

4 最小环

4.1 Floyd

```

1  int pos[MAXN][MAXN]; //pos[i][j]: i到j的最短路的路径
2  vector<int> path; //最小环路径
3  int ans; //最小环
4  //ans == INF 说明无环
5  void getpath(int x,int y)
6  {
7      if(!pos[x][y]) return;
8      getpath(x,pos[x][y]);
9      path.push_back(pos[x][y]);
10     getpath(pos[x][y],y);
11 }
12 void floyd()
13 {
14     ans = INF;
15     memcpy(dist,g,sizeof(dist));
16     for(int k = 1;k <= n;k++){
17         for(int i = 1;i < k;i++){

```

```

18         for(int j = i+1; j < k; j++){
19             if(dist[i][j]+g[j][k]+g[k][i]<ans){
20                 ans = dist[i][j]+g[j][k]+g[k][i];
21                 ans.clear();
22                 ans.push_back(i);
23                 getpath(i,j);
24                 ans.push_back(j);
25                 ans.push_back(k);
26             }
27         }
28     }
29     for(int i = 1; i <= n; i++){
30         for(int j = 1; j <= n; j++){
31             if(dist[i][k]+dist[k][j]<dist[i][j]){
32                 dist[i][j] = dist[i][k]+dist[k][j];
33                 pos[i][j] = k;
34             }
35         }
36     }
37 }
38 }

```

4.2 Dijkstra+ 剪枝

```

1 //边的计数idx从2开始, idx = 2;
2 ll dijkstra(int x,int y,int k)
3 {
4     priority_queue<node> q;
5     for(int i = 1; i <= cnt; i++){
6         vis[i] = 0;
7         dist[i] = INF;
8     }
9     dist[x] = 0;
10    q.push(node(x,0));
11    while(!q.empty()){
12        node cur = q.top();
13        q.pop();
14        if(cur.cost>ans-e[k].w) break; //剪枝
15        if(vis[cur.id]) continue;
16        vis[cur.id] = 1;
17        for(int i = first[cur.id]; i!=-1; i=e[i].next){
18            if(i == k || i == (k^1)) continue;
19            if(dist[e[i].v]>dist[cur.id]+e[i].w){
20                dist[e[i].v] = dist[cur.id]+e[i].w;
21                q.push(node(e[i].v,dist[e[i].v]));
22            }
23        }
24    }
25    return dist[y];

```

```

26 }
27 for(int i = 2; i <= m*2+1; i+=2){
28     ans = min(ans, dijkstra(e[i].u, e[i].v, i) + e[i].w);
29 }

```

5 网络流

5.1 二分图匹配

```

1  int dfs(int u)
2  {
3      for(int i = first[u]; i != -1; i = e[i].next){
4          int v = e[i].to;
5          if(!vis[v]){
6              vis[v] = 1;
7              if(linker[v] == -1 || dfs(linker[v])){
8                  linker[v] = u;
9                  return 1;
10             }
11         }
12     }
13     return 0;
14 }
15 int hungary()
16 {
17     int res = 0;
18     memset(linker, -1, sizeof(linker));
19     for(int u = 1; u <= n; u++){
20         memset(vis, 0, sizeof(vis));
21         if(dfs(u)){
22             res++;
23         }
24     }
25     return res;
26 }

```

5.2 最大流

5.2.1 Dinic

```

1  //1. 用BFS建立分层图
2  //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.
3  //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
4  //重复步骤2, 直到DFS找不到新的路径时, 重复步骤1
5  //时间复杂度O(n^2*m)
6  int bfs()
7  {

```



```

8     memset(dis,-1,sizeof(dis));
9     queue<node> q;
10    q.push(node(1,0));
11    dis[1] = 0;
12    while(!q.empty()){
13        node cur = q.front();
14        q.pop();
15        for(int i = first[cur.id];i!=-1;i = e[i].next){
16            if(e[i].w == 0) continue;
17            if(dis[e[i].v] == -1){
18                dis[e[i].v] = cur.cost+1;
19                q.push(node(e[i].v,dis[e[i].v]));
20            }
21        }
22    }
23    if(dis[n] == -1) return 0;
24    return 1;
25 }
26 int dfs(int x,int low)
27 {
28     if(x == n) return low;
29     for(int i = first[x];i!=-1;i = e[i].next){
30         int a = 0;
31         if(e[i].w > 0&&dis[e[i].v] == dis[x]+1&&(a = dfs(e[i].v,min(low,e[i].w)))){
32             e[i].w -= a;
33             add(e[i].v,x,a);
34             return a;
35         }
36     }
37     return 0;
38 }
39 int main()
40 {
41     init();
42     scanf("%d%d",&n,&m);
43     for(int i = 1;i <= m;i++){
44         int u,v,w;
45         scanf("%d%d%d",&u,&v,&w);
46         add(u,v,w);
47     }
48     ll ans = 0;
49     int sum;
50     while(bfs()){
51         while(sum = dfs(1,INF)){
52             ans += sum;
53         }
54     }
55     printf("Max flow: %lld\n",ans);
56     return 0;
57 }

```