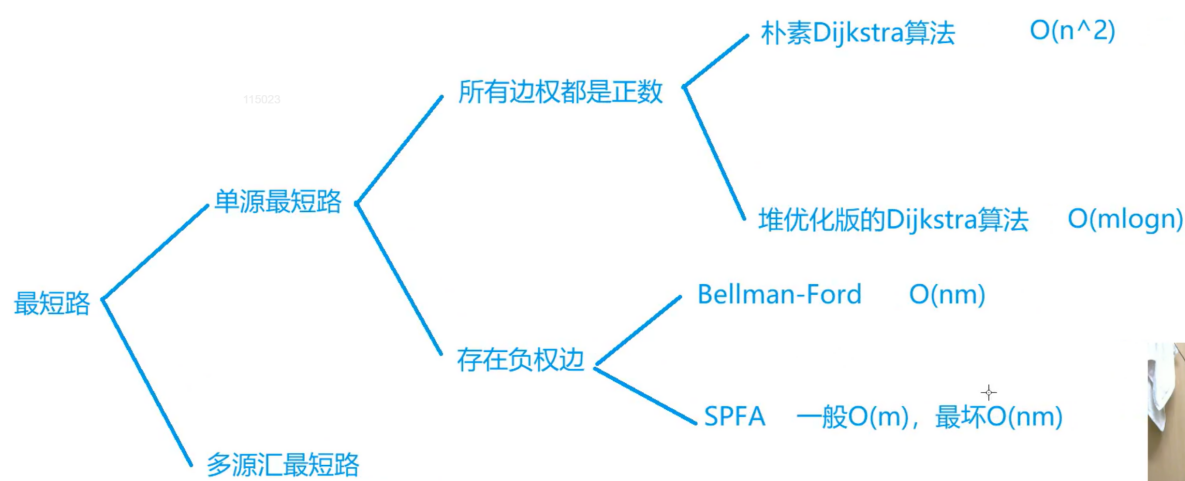


图论最短路



Dijkstra(朴素版)

算法作用

求稠密图的单源最短路

使用邻接矩阵存图

算法思路

1. 初始化距离为正无穷， $dis[1] = 0$;
2. 循环迭代n次，每次可以确定一个点
 1. 遍历该点的每一个节点，找到不在close_set中距离集合最近的点
 2. 标记选中该点
 3. 用该点更新其他点的距离 $dis[j] = \min(dis[j], dis[t] + g[t][j])$;

代码

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  const int maxn = 500 + 10;
4
5  int g[maxn][maxn];
6  int dis[maxn];
7  int n, m;
8  bool st[maxn];
9
10 int dijkstra(){
11     memset(dis, 0x3f, sizeof dis);
12     dis[1] = 0;
13     for(int i = 0; i < n; i++){
14         int t = -1;
15         for(int j = 1; j <= n; j++){
16             if(!st[j] && (t == -1 || dis[j] < dis[t])){
17                 t = j;
18             }
19         }
20     }
```

```

19     }
20     st[t] = true;
21     for(int j = 1; j <= n; j++){
22         dis[j] = min(dis[j], dis[t] + g[t][j]);
23     }
24 }
25
26 if(dis[n] == 0x3f3f3f3f) return -1;
27 return dis[n];
28 }
29
30 int main(){
31     cin >> n >> m;
32     memset(g, 0x3f, sizeof g);
33     for(int i = 0; i < m; i++){
34         int x, y, z; cin >> x >> y >> z;
35         g[x][y] = min(g[x][y], z);
36     }
37
38     cout << dijkstra() << endl;
39
40     return 0;
41 }

```

Dijkstra(堆优化版)

无法求最长路

算法作用

不含负权边的单源最短路

稀疏图使用邻接表存储

算法思路

将枚举过程替换为优先队列

代码

```

1  #include<iostream>
2  #include<cstring>
3  #include<algorithm>
4  #include<queue>
5
6  using namespace std;
7  typedef pair<int, int> PII; //<离起点的距离, 节点编号>
8
9  const int N = 150010;
10 int h[N], e[N], ne[N], idx, w[N];
11 int dist[N];
12 bool st[N];
13 int n, m;
14
15 //在a节点之后插入一个b节点, 权重为c
16 void add(int a, int b, int c) {
17     e[idx] = b;

```

```

18     w[idx] = c;
19     ne[idx] = h[a];
20     h[a] = idx++;
21 }
22
23 int dijkstra() {
24     // 所有距离初始化为无穷大
25     memset(dist, 0x3f, sizeof dist);
26     // 1号节点距离为0
27     dist[1] = 0;
28     // 建立一个小根堆
29     priority_queue<PII, vector<PII>, greater<PII>> heap;
30     // 1号节点插入堆
31     heap.push({0, 1});
32     while (heap.size()) {
33         // 取出堆顶顶点
34         auto t = heap.top();
35         // 并删除
36         heap.pop();
37         // 取出节点编号和节点距离
38         int ver = t.second, distance = t.first;
39         // 如果节点被访问过，则跳过
40         if (st[ver]) continue;
41         st[ver] = true;
42         for (int i = h[ver]; i != -1; i = ne[i]) {
43             // 取出节点编号
44             int j = e[i];
45             // dist[j] 大于从t过来的距离
46             if (dist[j] > distance + w[i]) {
47
48                 dist[j] = distance + w[i];
49                 heap.push({dist[j], j});
50             }
51         }
52     }
53     if (dist[n] == 0x3f3f3f3f) return -1;
54     return dist[n];
55 }
56
57
58 int main() {
59     memset(h, -1, sizeof h);
60     cin >> n >> m;
61     while (m--) {
62         int a, b, c;
63         cin >> a >> b >> c;
64         add(a, b, c);
65     }
66     cout << dijkstra() << endl;
67     return 0;
68 }
69

```

算法作用

- 含有负权边图的最短路
- 一定步数以内的最短路

算法思路

`dis[]`, `bk[]`

1. 初始化距离为无穷
2. 将起点的距离设置为1
3. 循环k-1次（最多点的个数）
 1. 将距离数组进行备份，防止串联
 2. 遍历所有的边
 1. 遍历到边目前的最短距离= $\min(\text{本来到该点的距离}, \text{前置点到该点的距离} + \text{边权})$
4. 如果到目标点的距离小于 $\text{INF} >> 1$, 没有路径。否则有路径。

代码

SPFA

算法作用

- 类地杰斯特拉，优化版Ford（使用邻接表存图）
- 求负环

算法思路（最短路）

`st[]`, `dis[]`, `queue`, 邻接表

1. 初始化距离为正无穷，**起点距离为1**；
2. 新建队列保存要遍历的点。起点入队。
3. 将起点st标记，已经在队列中。
4. 如果队列不空
 1. 取出队头，弹出队头，取消标记
 2. 遍历所有临边
 1. 如果目标点距离 $>$ 起始点+边权：更新距离
 1. 如果该点不在队列中则入队，标记
5. 如果到目标点的距离等于原始距离，则无路径。

代码

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 const int maxn = 1e6 + 10;
4
5 int h[maxn], e[maxn], ne[maxn], w[maxn], idx;
6 int dis[maxn];
```

```

7  bool st[maxn];
8  int n, m;
9
10 void add(int a, int b, int c){
11     e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++;
12 }
13
14 int spfa(){
15     memset(dis, 0x3f, sizeof dis);
16     dis[1] = 0;
17
18     queue<int> q;
19     q.push(1);
20     st[1] = true;
21     while(!q.empty()){
22         int t = q.front(); q.pop(); st[t] = false;
23         for(int i = h[t]; i != -1; i = ne[i]){
24             int j = e[i];
25             if(dis[j] > dis[t] + w[i]){ //注意w的idx为i!!!
26                 dis[j] = dis[t] + w[i];
27                 if(!st[j]){
28                     st[j] = true;
29                     q.push(j);
30                 }
31             }
32         }
33     }
34
35     return dis[n];
36 }
37
38 int main(){
39     cin >> n >> m;
40     memset(h, -1, sizeof h);
41     for(int i = 0; i < m; i++){
42         int x, y, z; cin >> x >> y >> z; add(x, y, z);
43
44         //一定要先复制再判断!! 因为最短路径可能是负数!!
45         int ans = spfa();
46         if(ans == 0x3f3f3f3f) cout << "impossible";
47         else cout << ans;
48         return 0;
49     }

```

算法思路（求负环）

cnt[]//记录路径长度

维护cnt数组，只要大于节点数，必有负环。

开始时要把所有的点全部放在队列中。

不必初始化dis[] 的距离为正无穷

!!! 使用stack可以优化效率效率

代码

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int maxn = 1e4 + 10, N = 2010;
5  int e[maxn], ne[maxn], h[N], w[maxn], idx;
6  int n, m;
7  int dis[N], cnt[N];
8  bool st[N];
9
10 void add(int a, int b, int c){
11     e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++;
12 }
13
14 bool spfa(){
15     queue<int> q;
16     for(int i = 1; i <= n; i ++){
17         q.push(i); st[i] = true;
18     }
19
20     while(q.size()){
21         int t = q.front(); q.pop(); st[t] = false;
22         for(int i = h[t]; i != -1; i = ne[i]){
23             int j = e[i];
24
25             if(dis[j] > dis[t] + w[i]){
26                 dis[j] = dis[t] + w[i];
27                 cnt[j] = cnt[t] + 1; //抽屉原理
28                 if(cnt[j] >= n) return true;
29                 if(!st[j])
30                     q.push(j), st[j] = true;
31             }
32         }
33     }
34     return false;
35 }
36
37 int main(){
38     cin >> n >> m;
39     memset(h, -1, sizeof h);
40     for(int i = 0; i < m; i ++){
41         int a, b, c; cin >> a >> b >> c;
42         add(a, b, c);
43     }
44
45     bool ans = spfa();
46     if(ans) puts("Yes");
47     else puts("No");
48
49     return 0;
50 }
```

Floyd

算法作用

用邻接矩阵存储图，求**多元汇**最短路。复杂度： $O(N^3)$

求任意两点之间的最短路径

传递闭包，判断两点

找最小环（SPFA找负环）

恰好经过k条边的最短路 倍增思想

算法思路

先打表，DP思想。

代码

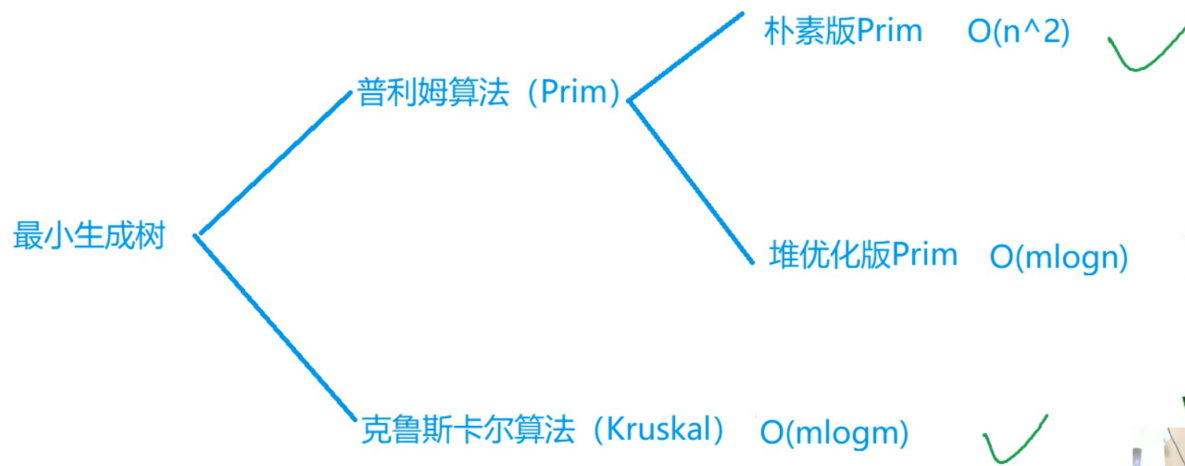
```
1  #include <cstring>
2  #include <iostream>
3  #include <algorithm>
4
5  using namespace std;
6
7  const int N = 210, INF = 1e9;
8
9  int n, m, Q;
10 int d[N][N];
11
12 void floyd(){
13     for(int k = 1; k <= n; k++){
14         for(int i = 1; i <= n; i++){
15             for(int j = 1; j <= n; j++){
16                 d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
17             }
18         }
19     }
20 }
21 int main()
22 {
23     ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
24     cin >> n >> m >> Q;
25
26     for (int i = 1; i <= n; i++) //邻接表初始化，最短路问题故无视自环
27         for (int j = 1; j <= n; j++)
28             if (i == j) d[i][j] = 0;
29             else d[i][j] = INF;
30
31     while(m--){
32         int a, b, c; cin >> a >> b >> c;
33         d[a][b] = min(d[a][b], c);
34     }
35
36     floyd();
37
38     while(Q--){
39         int a, b;
40         cin >> a >> b;
```

```

41
42     int t = d[a][b];
43     if(t > INF / 2) cout << "impossible" << endl;
44     else cout << t << endl;
45 }
46
47 return 0;
48 }

```

最小生成树



Prim

算法作用

稠密图的最小生成树：找到总代价最小的树，使图中的任意两点在同一树中。 $O(n^2)$

算法思路

1. 将所有距离初始化为正无穷
2. 循环迭代n次，每次可以确定一个点
 1. 遍历每个节点，找到不在生成树的剩下点中，到树距离最小的点
 2. 标记选中该点
 3. 用该点更新其他点到生成树的距离（如果小于，则更新）

代码

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int maxn = 510, INF = 0x3f3f3f3f;
5  int g[maxn][maxn];
6  int n, m;
7  int dis[maxn];
8  bool st[maxn];
9
10 int prim(){
11     memset(dis, 0x3f, sizeof dis);
12     dis[1] = 0;
13     int res = 0;
14     for(int i = 0; i < n; i++){

```



```

15     int t = -1;
16     for(int j = 1; j <= n; j ++){
17         if(!st[j] && (t == -1 || dis[t] > dis[j])) t = j;
18
19         if(dis[t] == INF) return INF;
20
21         st[t] = true;
22         res += dis[t];
23         for(int j = 1; j <= n; j ++){
24             dis[j] = min(dis[j], g[t][j]);
25         }
26     }
27     return res;
28 }
29
30 int main(){
31     cin >> n >> m;
32     memset(g, 0x3f, sizeof g); //重边和自环不影响最小生成树
33     for(int i = 0; i < m; i ++){
34         int u, v, w; cin >> u >> v >> w;
35         g[u][v] = g[v][u] = min(g[u][v], w);
36     }
37
38     int t = prim();
39
40     if(t == 0x3f3f3f3f) cout << "impossible" << endl;
41     else cout << t << endl;
42
43     return 0;
44 }
45

```

Kruskal

算法作用

用于稀疏图的最小生成树算法。 $O(m\log m)$

不需要使用邻接表或邻接矩阵存图

求**边权和**最小的最小生成树&&**最大边权最小**的最小生成树

算法思路

1. 将所有的边按边权从小到大排序
2. 枚举所有的边，对 u, v 做并查集，如不属于一个集合则合并。

代码

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  int n, m;
5  const int maxn = 1e5 + 10;
6  int p[maxn];
7  struct Edge{
8      int u, v, w;

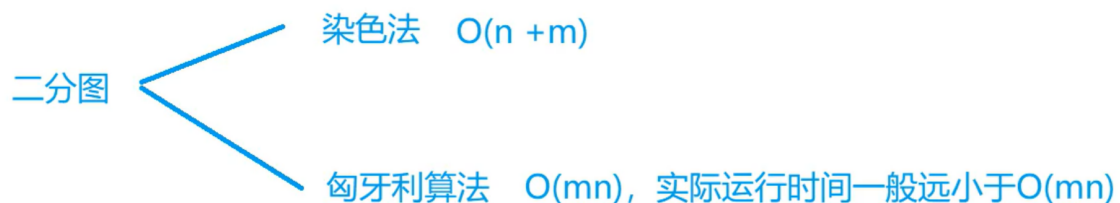
```

```

9     bool operator<(const Edge&e) const{
10         return w < e.w;
11     }
12 } edge[maxn];
13
14 int find(int x){
15     if(x != p[x]) p[x] = find(p[x]);
16     return p[x];
17 }
18
19 int main(){
20     cin >> n >> m;
21     for(int i = 1; i <= n; i++) p[i] = i;
22
23     for(int i = 0; i < m; i++)
24         cin >> edge[i].u >> edge[i].v >> edge[i].w;
25
26     sort(edge, edge + m);
27
28     int res = 0, cnt = 0;
29     for(int i = 0; i < m; i++){
30         int u = edge[i].u, v = edge[i].v, w = edge[i].w;
31         int a = find(u), b = find(v);
32         if(a != b){
33             res += w; cnt++;
34             p[a] = b;
35         }
36     }
37     if(cnt < n - 1) cout << "impossible" << endl;
38     else cout << res;
39
40     return 0;
41 }

```

二分图



染色法

算法作用

判断一个图是不是二分图

一个图是二分图，当且仅当图中不含有奇数环（边的数量为奇数个的环）。

算法思路

遍历每一个点，如果没有被染色，则对它进行染色

- pii存储，第一个存储标号，第二个存储颜色
- 对于每个点，搜索与其相邻的点。如未染色，则染色入队。否则，判断其颜色状态是否合法，不合法返回false。

代码

```
1  #include <bits/stdc++.h>
2  #define pii pair<int, int>
3  using namespace std;
4
5  int n, m;
6  const int maxn = 2e5 + 10;
7  int e[maxn], ne[maxn], h[maxn], idx;
8  int st[maxn];
9
10 void add(int a, int b){
11     e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
12 }
13
14 bool bfs(int x){
15     queue<pii> q;
16     q.push({x, 1}); //第一个是编号，第二个是颜色
17     st[x] = 1;
18
19     while(q.size()){
20         int ver = q.front().first, color = q.front().second; q.pop();
21         for(int i = h[ver]; i != -1; i = ne[i]){
22             int j = e[i];
23
24             if(!st[j]){ //未被染色，则染色
25                 st[j] = 3 - color;
26                 q.push({j, 3 - color});
27             }
28             else{ //已被染色则判断
29                 if(st[j] == color) return false;
30             }
31         }
32     }
33     return true;
34 }
35
36 int main(){
37     ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
38
39     cin >> n >> m;
40     memset(h, -1, sizeof h);
41     for(int i = 1; i <= m; i++){
42         int a, b; cin >> a >> b;
43         add(a, b), add(b, a);
44     }
45
46     for(int i = 1; i <= m; i++){
47         if(!st[i])
```

```

48         if(!bfs(i)){
49             cout << "No" << endl;
50             return 0;
51         }
52     }
53
54     cout << "Yes" << endl;
55     return 0;
56 }

```

匈牙利算法

算法作用

求二分图的最大匹配

算法思路

match[女生] = 男生 //表示女生对应的男生

st[女生] = true // 表示当前女生是否可以被选择（判重和回溯）

find(x) //找st状态下x是否有匹配的女生

1. 枚举每个男生，遍历对应的女生
2. 如果该女生已经被预定，跳过。否则，预定。
3. 如果女生没有对象 || 在st状态下可以换对象。则换对象，return true。

代码

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  int m, n1, n2;
5  const int M = 1e5 + 10, N = 510;
6  int e[M], ne[M], h[N], idx;
7  bool st[N]; // st[a] = true 说明女生 a 目前被一个男生预定了
8  int match[N]; // match[a] = b: 女生 a 目前匹配了男生 b
9
10 void add(int a, int b){
11     e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
12 }
13
14 bool find(int x){ // 为单身狗 x 找一个对象，（或）x的女朋友被别人预定，给x换一个对象
    如果成功，返回true
15     for(int i = h[x]; i != -1; i = ne[i]){ // j 是可以与男生 x 匹配的女生之一
16         int j = e[i];
17         if(st[j]) continue; // 女生 j 目前被一个男生预定了，跳过它
18         st[j] = true; // 将女生 j 预定给男生 x
19
20         // 如果女生 j 没有对象， 或者
21         // 女生 j 在前几轮深搜中已预定有对象，但我们成功给她的对象换了个新对象
22         if(match[j] == 0 || find(match[j])){
23             match[j] = x;
24             return true;
25         }
26     }

```

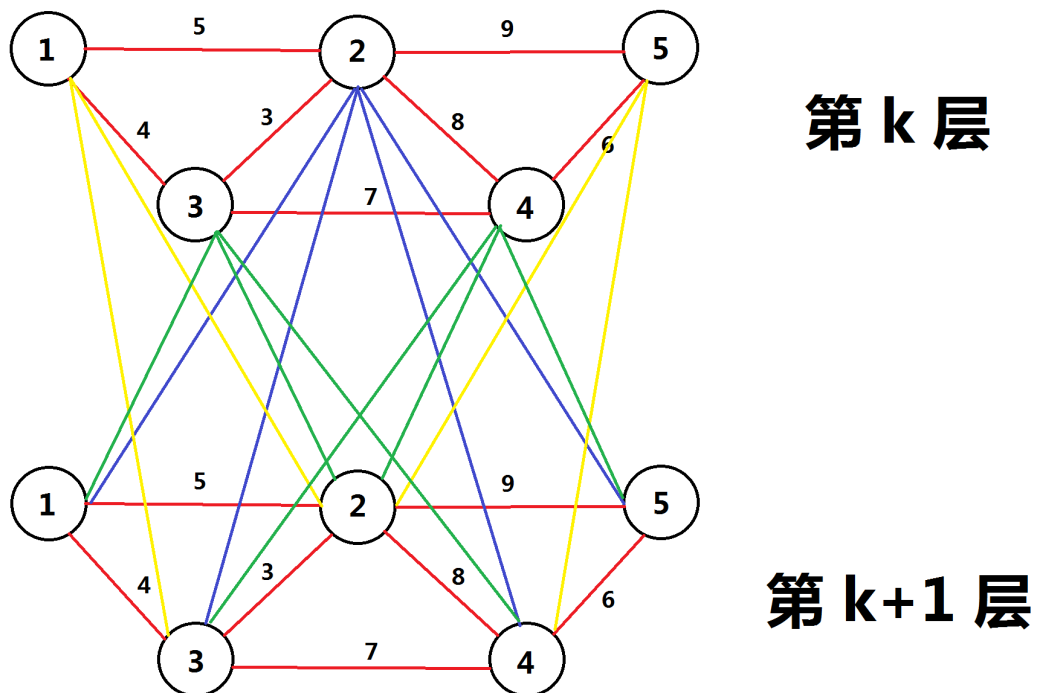
```

27     return false;
28 }
29
30 int main(){
31     cin >> n1 >> n2 >> m;
32     memset(h, -1, sizeof h);
33     for(int i = 0; i < m; i++){
34         int u, v; cin >> u >> v;
35         add(u, v);
36     }
37
38     int cnt = 0;
39     for(int i = 1; i <= n1; i++){
40         memset(st, false, sizeof st);
41         if(find(i)) cnt++;
42     }
43
44     cout << cnt;
45
46     return 0;
47 }

```

分层图

在图上，有 k 次机会可以直接通过一条边，问起点与终点之间的最短路径。



分层图的作用

经过分层后，我们得到了新图

我们可以发现，原本题目中选 k 条边免费的操作被我们等价了：

在从一个点到另一个点时，如果选择免费，就进入下一层，相当于进行一次免费操作

因为可以免费 k 次，所以我们要建 $k+1$ 层图

在 $k+1$ 层图上我们已经不能再往下了，即免费操作已用完

k个免费操作即建k层

建图

```
1  for(int i=1;i<=p;i++){
2      int a,b,l; cin>>a>>b>>l;
3      add(a,b,l), add(b,a,l);
4      for(int j=1;j<=k;j++){
5          add(a+(j-1)*n,b+j*n,0), add(b+(j-1)*n,a+j*n,0); //上层到下层的双向边
6          add(a+j*n,b+j*n,l), add(b+j*n,a+j*n,l); //下层的双向边
7      }
8  }
9  for(int i=1;i<=k;i++){
10     add(i*n,(i+1)*n,0);
11 }
```

模板-dij

```
1  #include<bits/stdc++.h>
2  #define pii pair<int,int>
3  using namespace std;
4
5  const int N=100100*42+10,M=500500*42+10; //容易爆空间
6  int e[M],ne[M],h[N],w[M],idx;
7  bool st[N];
8  int dis[N];
9  int n,p,k;
10
11 void add(int a,int b,int c){
12     e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
13 }
14
15 void dij(){
16     memset(dis,0x3f,sizeof dis);
17     dis[1]=0;
18     priority_queue<pii,vector<pii>,greater<pii>> pq;
19     pq.push({0,1});
20
21     while(pq.size()){
22         auto t=pq.top(); pq.pop();
23         int ver=t.second, d=t.first;
24         if(st[ver]) continue;
25         st[ver]=true;
26         for(int i=h[ver];~i;i=ne[i]){
27             int j=e[i];
28             if(dis[j]>dis[ver]+w[i]){
29                 dis[j]=dis[ver]+w[i];
30                 pq.push({dis[j],j});
31             }
32         }
33     }
34 }
35
36 int main(){
```

```

37     cin>>n>>p>>k;
38     memset(h,-1,sizeof h);
39     for(int i=1;i<=p;i++){
40         int a,b,l; cin>>a>>b>>l;
41         add(a,b,l), add(b,a,l);
42         for(int j=1;j<=k;j++){
43             add(a+(j-1)*n,b+j*n,l/2), add(b+(j-1)*n,a+j*n,l/2); //上层到下层的
双向边
44             add(a+j*n,b+j*n,l), add(b+j*n,a+j*n,l); //下层的双向边
45         }
46     }
47     for(int i=1;i<=k;i++){
48         add(i*n,(i+1)*n,0);
49     }
50
51     dij();
52
53     // cout<<dis[(k+1)*n]<<endl;
54     int ans=dis[n];
55     for(int i=1;i<=k;i++){
56         // cout<<"### dis["<<i*n+n<<"]:"<<dis[i*n+n]<<endl;
57         ans=min(dis[i*n+n],ans);
58     }
59     cout<<ans;
60
61     return 0;
62 }

```

模板-SPFA

```

1 //地杰斯特拉无法求最长路
2 //分层图
3 #include<bits/stdc++.h>
4 #define pii pair<int,int>
5 using namespace std;
6
7 const int N=100000*3+10, M=100000*2*10;
8 int h[N],e[M],ne[M],w[M],idx;
9 int dis[N];
10 bool st[N];
11 int n,m;
12
13 void add(int a,int b,int c=0){
14     e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
15 }
16
17 void spfa(){
18     memset(dis,-0x3f,sizeof dis);
19     queue<int>q;
20     q.push(1);
21     st[1]=true;
22     dis[1]=0;
23
24     while(q.size()){
25         int t=q.front(); q.pop(); st[t]=false;
26         for(int i=h[t];~i;i=ne[i]){
27             int j=e[i];

```

```

28         if(dis[j]<dis[t]+w[i]){
29             dis[j]=dis[t]+w[i];
30             if(!st[j]){
31                 st[j]=true;
32                 q.push(j);
33             }
34         }
35     }
36 }
37 }
38
39 int main(){
40     cin>>n>>m;
41     memset(h,-1,sizeof h);
42     for(int i=1;i<=n;i++){
43         int c; cin>>c;
44         add(i,i+n,-c), add(i+n,i+2*n,c);
45     }
46     for(int i=1;i<=m;i++){
47         int x,y,z; cin>>x>>y>>z;
48         if(z==1){
49             add(x,y), add(x+n,y+n), add(x+2*n,y+2*n);
50         }
51         else{
52             add(x,y), add(x+n,y+n), add(x+2*n,y+2*n);
53             add(y,x), add(y+n,x+n), add(y+2*n,x+2*n);
54         }
55     }
56
57     spfa();
58
59     cout<<dis[3*n]<<endl;
60
61     return 0;
62 }

```