

最短路

Shortest Path

Floyd

Idea: 动态规划思想与三角不等式 $dis[i][j] \geq dis[i][k] + dis[k][j]$ 的应用。枚举 k 作为中转点, i 到 j 的距离用 i 到 k 的距离加上 k 到 j 的距离来 relax。换句话说, 第 k 次中转得到的是 i 号点到 j 号点只经过前 k 个点的最短距离。

Complexity: $O(V^3)$

Code:

```
1  int dis[N][N];
2  void floyd(){
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                  dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
7  }
```

Dijkstra

Idea: 维护集合 S , 凡在集合 S 中的点都已经得到了最小值并不再更改 (即 $S: \{x \mid dis[x] = \delta(x)\}$)。每次选取距离源点最近的不在 S 中的点加入 S , 并更新与之相连的所有点的距离。

Complexity:

- $O(V^2 + E)$ 【朴素实现】
- $O((V + E) \lg V)$ 【小根堆实现 (随时删除旧节点)】
- $O((V + E) \lg E)$ 【优先队列实现】
- $O(E + V \lg V)$ 【斐波那契堆实现】

Code:

```
1  LL dis[N];
```

```

2 void dijkstra(int s){
3     vector<bool> vis(n+5, false);
4     for(int i = 1; i <= n; i++) dis[i] = INF;
5     priority_queue< pair<LL, int>, vector<pair<LL, int>>,
greater<pair<LL, int>> > q;
6     dis[s] = 0;
7     q.push(make_pair(dis[s], s));
8     while(!q.empty()){
9         auto cur = q.top(); q.pop();
10        if(vis[cur.second]) continue;
11        vis[cur.second] = true;
12        for(int i = head[cur.second]; i; i = edge[i].nxt){
13            if(dis[edge[i].to] > dis[cur.second] + edge[i].dis){
14                dis[edge[i].to] = dis[cur.second] + edge[i].dis;
15                q.push(make_pair(dis[edge[i].to], edge[i].to));
16            }
17        }
18    }
19 }

```

SPFA

Idea: 是 **Bellman-Ford** 算法的队列实现，但最坏情况复杂度并没有改变。

Complexity: Worst-Case: $O(VE)$

Code:

```

1 LL dis[N];
2 bool inq[N];
3 void SPFA(int s){
4     for(int i = 1; i <= n; i++)
5         dis[i] = INF;
6     queue<int> q;
7     dis[s] = 0;
8     q.push(s);
9     inq[s] = 1;
10    while(!q.empty()){
11        int cur = q.front();
12        q.pop();
13        inq[cur] = 0;
14        for(int i = head[cur]; i; i = edge[i].nxt){
15            if(dis[edge[i].to] > dis[cur] + edge[i].dis){
16                dis[edge[i].to] = dis[cur] + edge[i].dis;

```

```
17         if(!inq[edge[i].to]){
18             q.push(edge[i].to);
19             inq[edge[i].to] = 1;
20         }
21     }
22 }
23 }
24 }
```