最短路

Shortest Path

Floyd

Idea：动态规划思想与三角不等式 $dis[i][j] \geqslant dis[i][k] + dis[k][j]$ 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：

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

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：

```
LL dis[N];
void dijkstra(int s){
    vector<bool> vis(n+5, false);
    for(int i = 1; i <= n; i++) dis[i] = INF;
    priority_queue< pair<LL, int>, vector<pair<LL, int>>, greater<pair<LL, int>> > q;
    dis[s] = 0;
    q.push(make_pair(dis[s], s));
    while(!q.empty()){
        auto cur = q.top(); q.pop();
        if(vis[cur.second]) continue;
        vis[cur.second] = true;
        for(int i = head[cur.second]; i; i = edge[i].nxt){
            if(dis[edge[i].to] > dis[cur.second] + edge[i].dis){
                dis[edge[i].to] = dis[cur.second] + edge[i].dis;
                q.push(make_pair(dis[edge[i].to], edge[i].to));
            }
        }
    }
}
```

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：

```
LL dis[N];
bool inq[N];
void SPFA(int s){
    for(int i = 1; i <= n; i++)
        dis[i] = INF;
    queue<int> q;
    dis[s] = 0;
    q.push(s);
    inq[s] = 1;
    while(!q.empty()){
        int cur = q.front();
        q.pop();
        inq[cur] = 0;
        for(int i = head[cur]; i; i = edge[i].nxt){
            if(dis[edge[i].to] > dis[cur] + edge[i].dis){
                dis[edge[i].to] = dis[cur] + edge[i].dis;
                if(!inq[edge[i].to]){
                    q.push(edge[i].to);
                    inq[edge[i].to] = 1;
                }
            }
        }
    }
}
```