最短路径

广度优先搜索查找最短路径只是对**无权图**而言的。

当图是带权图时，把从-个顶点v_0到图中其余任意一个顶点v_i的一条路径（可能不止一条）所经过边上的权值之和，定义为该路径的**带权路径长度**，

把带权路径长度最短的那条路径称为**最短路径**

求解最短路径的算法通常都依赖于一种性质，即**两点之间的最短路径也包含了路径上其他顶点间的最短路径。**

带权有向图G的最短路径问题一般可分为**两类**：

一是单源最短路径，即求图中某一顶点到其他各顶点的最短路径，可通过经典的Dijkstra （迪杰斯特拉）算法求解；

二是求每 对顶点间的最短路径，可通过Floyd （弗洛伊德）算法来求解。

1. Dijkstra算法求单源最短路径问题

Dijkstra算法设置一个集合*S* 记录已求得的最短路径的顶点，初试时把源点v_0放入S，集合S 每并入一个新顶点V_i都要修改源点源点v_0到集合V-S顶点当前的最短路径长度值。

在构造的过程中还设置了两个辅助数组:

dist[]：记录从源点v_0到集合V-S顶点当前的最短路径长度,它的初试状态为：若v_0到V_i有孤，则dist[i]为弧上的权值；否则置dist[i]为∞;

Path []：path[i]表示从源点到顶点i之间的最短路径的前驱结点。在算法结束时，可假设从顶点0出发，即v_0=0,集合s最初只包含顶点0，邻接矩阵arcs表示带权有向图，arcs[i][j]表示有向边<i,j> ,则arcs[i][j]为∞.

Dijkstra算法的步骤如下（不考虑对path[ ]的操作）

1）初始化：集合S初始为{0}，dist[ ]的初始值dist[i]=acs[0J[i]，i=1，2，…，n-1。

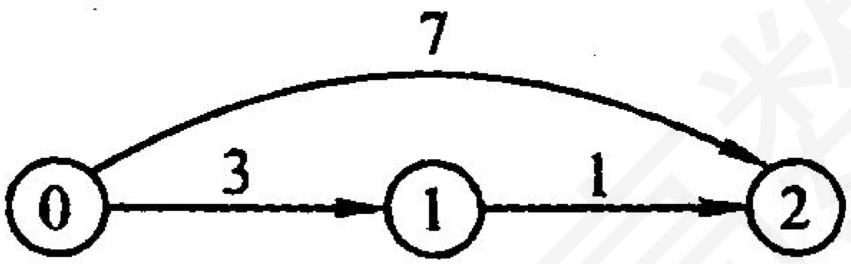
2）从顶点集合V-S中选出v_j，满足dist[j]=Min{dist[i]|v_i∈V-S}，v_j就是当前求得的一条从v_0出发的最短路径的终点，令S=S∪{j}。

3）修改从v_0出发到集合V-S上任一顶点v_k可达的最短路径长度：若Dist[j]+arcs[j][k]<dist[k]，则更新Dist[j]+arcs[j][k]。

4）重复2）~3）操作共n-1次，直到所有的顶点都包含在S中。

步骤3）也就是开头留下的疑问，每当一个顶点加入S后，可能需要修改源点v_0到集合V-S中可达顶点当前的最短路径长度，

举例证明。如下图所示，源点为v_0，初始时$S=\{v_0\}$，dist[1]=3，dist[2]=7，当将v_1并入集合S后，dist[2]需要更新为4。



【题】Dijkstra算法与Prim算法有何相似之处？

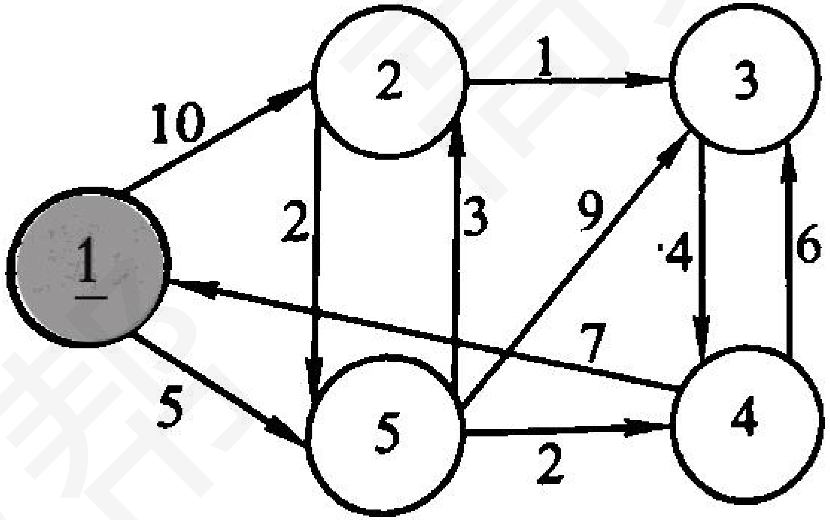


图1 应用Dijkstra算法图

每轮得到的最短路径如下：

第1轮：1→5，路径距离为5

第2轮：1→5→4，路径距离为

第3轮：1→5→2，路径距离为8

第4轮：1→5→2→3，路径距离为9

顶点	第1轮	第2轮	第3轮	第4轮
2	10$v_1 \to v_2$	8$v_1 \to v_5 \to v_2$	8$v_1 \to v_5 \to v_2$	
3	∞	14$v_1 \to v_5 \to v_3$	13$v_1 \to v_5 \to v_4 \to v_3$	12$v_1 \to v_5 \to v_2 \to v_3$
4	∞	7$v_1 \to v_5 \to v_4$		
5	5$v_1 \to v_5$			
集合S	{1，5}	{1，5，4}	{1，5，4，2}	{1，5，4，2，3}

表1 从v_1到各终点的dist值和最短路径的求解过程

例如，对图1中的图应用Dijkstra算法求从顶点1出发至其余顶点的最短路径的过程，如表1所示。算法执行过程如下：

初始化：集合S初始为$\{v_1\}$，v_1可达v_2和v_5，v_1不可达v_3和v_4，因此dist[ ]数组各元素的初值依次设置为dist[2]=10，dist[3]=∞，dist[4]=∞，dist[5]=5。

第一轮：选出最小值dist[5]，将顶点v_5并入集合S，即此时已找到v_1到v_3的最短路径。当v_5加入S后，从v_1到集合V-S中可达顶点的最短路径长度可能会产生变化。因此需要更新dist[ ]数组。v_5可达v_2，因$v_1 \to v_5 \to v_2$的距离8比dist[2]=10小，更新dist[2]=8；v_5可达v_3，$v_1 \to v_5 \to v_3$的距离14，更新dist[3]=14；v_5可达v_4，$v_1 \to v_5 \to v_4$的距离7，更新dist[4]=7。

第二轮：选出最小值dist[4]，将顶点v_4并入集合S。继续更新dist[ ]数组。v_4不可达v_2，dist[2]不变；v_4可达v_3，$v_1 \to v_5 \to v_4 \to v_3$的距离13比dist[3]小，故更新dist[3]=13。

第三轮：选出最小值dist[2]，将顶点v_2并入集合S。继续更新dist[ ]数组。V_2可达v_3， $v_1 \to v_5 \to v_2 \to v_3$的距离9比dist[3]小，更新dist[3]=9。

第四轮：选出唯一最小值dist[3]，将顶点v_3并入集合S，此时全部顶点都已包含在S中。

显然，Dijkstra算法也是基于贪心策略的。

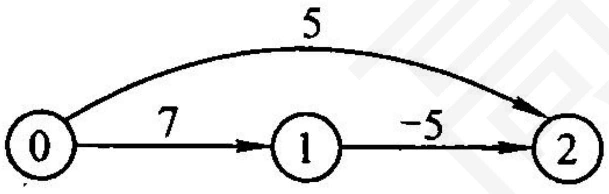
使用邻接矩阵表示时，时间复杂度为O（|V|2）。使用带权的邻接表表示时，虽然修改dist[ ]的时间可以减少，但由于在dist[ ]中选择最小分量的时间不变，时间复杂度仍为O（|V|2）。

“找到从源点到某个特定顶点的最短路径”，与“求解源点到其他所有顶点的最短路径”一样复杂，时间复杂度也为O（|V|2）。

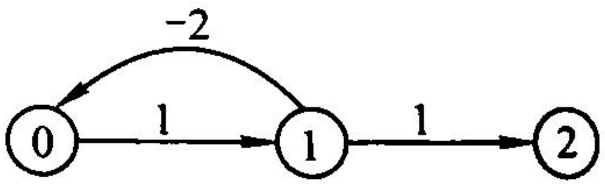
边上带有负权值时，Dijkstra算法并不适用。

若允许边上带有负权值，则在与S（已求得最短路径的顶点集，归入S内的结点的最短路径不再变更）内某点（记为a）以负边相连的点（记为b）确定其最短路径时，其最短路径长度加上这条负边的权值结果可能小于a原先确定的最短路径长度，而此时a在Dijkstra算法下是无法更新的。

例如，对于下图所示的带权有向图，利用Dijkstra算法不一定能得到正确的结果。



(a)



(b)