Dijkstra算法

摘要：算法解决了从图中某一点分别到图中其余各点的所有最短路值。算法按照非递减顺序每次收录一个距离最小的点，然后看他的所有邻接点是否受到影响，如果该邻接点没被收录且受到了影响（有更短的距离值），那么更新它即可。当所有点都被收录时算法结束。

一、算法介绍

     迪杰斯特拉算法为了解决有权图的单源最短路问题，即从图中任意一点出发，到达图中其余各点的最短距离。这里的最短距离，此距离常指欧氏距离。也可以理解为最小代价。即从某一点出发，到达其余各点的最小代价。

二、算法思想

     设G=(V,E)是一个带权有向图，把图中顶点集合V分成两组，第一组为已求出最短路径的顶点集合（用S表示，初始时S中只有一个源点，以后每求得一条最短路径 , 就将加入到集合S中，直到全部顶点都加入到S中，算法就结束了），第二组为其余未确定最短路径的顶点集合（用U表示），按最短路径长度的递增次序依次把第二组的顶点加入S中。在加入的过程中，总保持从源点v到S中各顶点的最短路径长度不大于从源点v到U中任何顶点的最短路径长度。此外，每个顶点对应一个距离，S中的顶点的距离就是从v到此顶点的最短路径长度，U中的顶点的距离，是从v到此顶点只包括S中的顶点为中间顶点的当前最短路径长度。

三、算法步骤

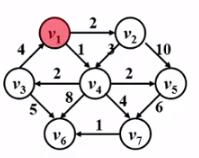
1、初始时，S只包含源点，即S＝{v}，v的距离为0。U包含除v外的其他顶点，即:U={其余顶点}，若v与U中顶点u有边，则<u,v>正常有权值，若u不是v的出边邻接点，则<u,v>权值为∞。

2、从U中选取一个距离v最小的顶点k，把k，加入S中（该选定的距离就是v到k的最短路径长度）。

3、以k为新考虑的中间点，修改U中各顶点的距离；若从源点v到顶点u的距离（经过顶点k）比原来距离（不经过顶点k）短，则修改顶点u的距离值，修改后的距离值的顶点k的距离加上边上的权。

4、重复步骤b和c直到所有顶点都包含在S中。

四、示例



问题叙述：如图所示，求以v1为起始点，分别达到各点的最短距离和路径

问题解决：

1. 设置数组dist[v]表示从起点到达v点的最短距离，设置数组path[v]表示到达v点的上一个顶点。初始化设置所有dist=，path=-1。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | v1 | v2 | v3 | v4 | v5 | v6 | v7 |
| dist |  |  |  |  |  |  |  |
| path | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

1. 从起点v1开始，因为v1到自己的距离为0，所以dist[v1]=0。v1到v2和v4的距离分别为2和1，此时，dist[v2]=2, dist[v4]=1, path[v2]=v1, path[v4]=v1。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | v1 | v2 | v3 | v4 | v5 | v6 | v7 |
| dist | 0 | 2 |  | 1 |  |  |  |
| path | -1 | V1 | -1 | V1 | -1 | -1 | -1 |

1. 下面进入dijkstra算法，因为v1已经在S中，所以下面选择未收录顶点中dist最小的v4进入S，v4有v3,v5,v6,v7四个邻接点，对于每个临界点进行如下判断：首先判断该点是否已经在S中，若已经在S中则跳过该点继续判断下个点；若该点vi（i属于3、5、6、7）不在S中，则判断经由v4到达该点的距离+dist[v4]是否比dist[vi]更小，取其中更小的数值更新dist[vi]，并将path[vi]更新为v4。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | v1 | v2 | v3 | v4 | v5 | v6 | v7 |
| dist | 0 | 2 | 3 | 1 | 3 | 9 | 5 |
| path | -1 | V1 | V4 | V1 | V4 | V4 | V4 |

1. 当v4的所有邻接点均判断完成后，再次选择新的顶点进入S。观察发现v2是未收录的顶点中dist最小的，所以将v2加入S。v2有两个邻接点v4和v5，其中v4已在S中，不与讨论，只看v5,此时计算原点经过v2到v5的距离是v2到v5的距离+dist[v2]=12， 比v5现在的dist[v5]还要大，因此dist[v5]不变，path[v5]也不变。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | v1 | v2 | v3 | v4 | v5 | v6 | v7 |
| dist | 0 | 2 | 3 | 1 | 3 | 9 | 5 |
| path | -1 | V1 | V4 | V1 | V4 | V4 | V4 |

1. 当v2的所有邻接点判断完后，从剩余的未被收录到S中的顶点里选择dist最小的v3（或者v5）加入S。v3的邻接点有v1和v6，其中v1已经在S中，只需要判断v6，从原点经过v3到v6的距离等于从v3到v6的距离+dist[v3]=8。比v6现在的dist[v6]的值要小，所以更新dist[v6]=8，并且更新path[6]=v3。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | v1 | v2 | v3 | v4 | v5 | v6 | v7 |
| dist | 0 | 2 | 3 | 1 | 3 | 8 | 5 |
| path | -1 | V1 | V4 | V1 | V4 | V3 | V4 |

1. 当v3的所有邻接点判断完后，从剩余的未被收录到S中的顶点里选择dist最小的v5加入到S。v5的邻接点只有v7，从原点经过v5到v7的距离等于v5到v7的距离+dist[v5]=9。比现在v7的距离dist[v7]还大，所以dist[v7]不变，path[v7]不变。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | v1 | v2 | v3 | v4 | v5 | v6 | v7 |
| dist | 0 | 2 | 3 | 1 | 3 | 8 | 5 |
| path | -1 | V1 | V4 | V1 | V4 | V3 | V4 |

1. 当v5的所有邻接点判断完后，从剩余的未被收录到S中的顶点里选择dist最小的v7加入到S，v7的邻接点只有v6，从原点经过v7到v6的距离等于v7到v6的距离+dist[v7]=6，比现在的dist[6]要小，所以更新dist[v6]=6，path[v6]=v7。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | v1 | v2 | v3 | v4 | v5 | v6 | v7 |
| dist | 0 | 2 | 3 | 1 | 3 | 6 | 5 |
| path | -1 | V1 | V4 | V1 | V4 | V7 | V4 |

1. 当v7的所有邻接点判断完后，只剩下v6了，将v6加入S中。此时v6已经没有邻接点了。故此时算法结束。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | v1 | v2 | v3 | v4 | v5 | v6 | v7 |
| dist | 0 | 2 | 3 | 1 | 3 | 6 | 5 |
| path | -1 | V1 | V4 | V1 | V4 | V7 | V4 |

1. 代码（C语言）

|  |
| --- |
| 1. /\* 邻接矩阵存储 - 有权图的单源最短路算法 \*/ 2. Vertex FindMinDist( MGraph Graph, **int** dist[], **int** collected[] ) 3. { /\* 返回未被收录顶点中dist最小者 \*/ 4. Vertex MinV, V; 5. **int** MinDist = INFINITY; 7. **for** (V=0; V<Graph->Nv; V++) { 8. **if** ( collected[V]==**false** && dist[V]<MinDist) { 9. /\* 若V未被收录，且dist[V]更小 \*/ 10. MinDist = dist[V]; /\* 更新最小距离 \*/ 11. MinV = V; /\* 更新对应顶点 \*/ 12. } 13. } 14. **if** (MinDist < INFINITY) /\* 若找到最小dist \*/ 15. **return** MinV; /\* 返回对应的顶点下标 \*/ 16. **else return** ERROR;  /\* 若这样的顶点不存在，返回错误标记 \*/ 17. } 18. **bool** Dijkstra( MGraph Graph, **int** dist[], **int** path[], Vertex S ) 19. { 20. **int** collected[MaxVertexNum]; 21. Vertex V, W; 22. /\* 初始化：此处默认邻接矩阵中不存在的边用INFINITY表示 \*/ 23. **for** ( V=0; V<Graph->Nv; V++ ) { 24. dist[V] = Graph->G[S][V]; 25. **if** ( dist[V]<INFINITY ) 26. path[V] = S; 27. **else** 28. path[V] = -1; 29. collected[V] = **false**; 30. } 31. /\* 先将起点收入集合 \*/ 32. dist[S] = 0; 33. collected[S] = **true**; 34. **while** (1) { 35. /\* V = 未被收录顶点中dist最小者 \*/ 36. V = FindMinDist( Graph, dist, collected ); 37. **if** ( V==ERROR ) /\* 若这样的V不存在 \*/ 38. **break**;      /\* 算法结束 \*/ 39. collected[V] = **true**;  /\* 收录V \*/ 40. **for**( W=0; W<Graph->Nv; W++ ) /\* 对图中的每个顶点W \*/ 41. /\* 若W是V的邻接点并且未被收录 \*/ 42. **if** (  collected[W]==**false** && Graph->G[V][W]<INFINITY ) { 43. **if** (  Graph->G[V][W]<0 ) /\* 若有负边 \*/ 44. **return false**; /\* 不能正确解决，返回错误标记 \*/ 45. /\* 若收录V使得dist[W]变小 \*/ 46. **if** (  dist[V]+Graph->G[V][W] < dist[W] ) { 47. dist[W] = dist[V]+Graph->G[V][W]; /\* 更新dist[W] \*/ 48. path[W] = V; /\* 更新S到W的路径 \*/ 49. } 50. } 51. } /\* while结束\*/ 52. **return true**; /\* 算法执行完毕，返回正确标记 \*/ |