

《数据结构》上机报告

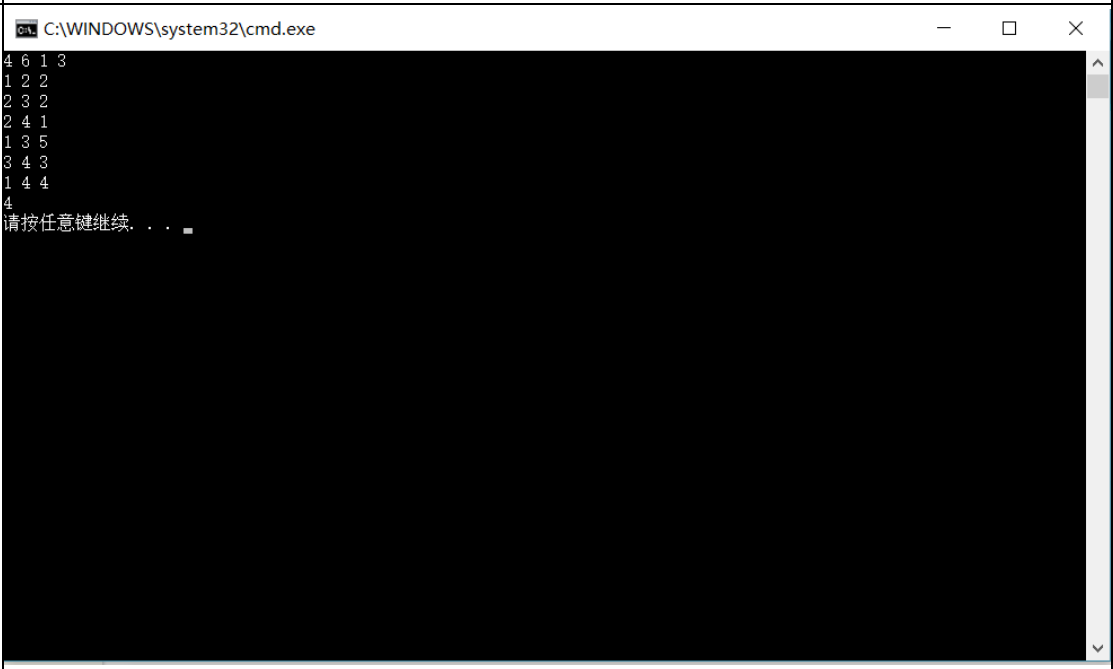
2018 年 12 月 11 日

姓名：刘思源 学号：1651390 班级：电子三班 得分：_____

实验题目	单源最短路径	
问题描述	给定一张 n 个点的无向带权图，节点的编号从 1 至 n，求从 S 到 T 的最短路径长度。	
基本要求	1. (p1) 给定一张 n 个点的无向带权图，节点的编号从 1 至 n，求从 S 到 T 的最短路径长度。	
	已完成基本内容（序号）：	1
选做要求		
	已完成选做内容（序号）	
数据结构设计	<pre>typedef struct MGraph { int vexnum, arcnum; int v1[MAXVEX]; int v2[MAXARC]; int weight[MAXARC]; } MGraph;</pre>	

功能(函数)说明	<p>1、单源最短路径</p> <p>给定的图若存在负权边，这时类似 Dijkstra 等算法便没有了用武之地，而 Bellman-Ford 算法的复杂度又过高，SPFA 算法便派上用场了。我们约定有向加权图 G 不存在负权回路，即最短路径一定存在。当然，我们可以在执行该算法前做一次拓扑排序，以判断是否存在负权回路，但这不是我们讨论的重点。</p> <p>本题不涉及负环，但我们一样可以使用 SPFA，Dijkstra 算法也可。</p> <p>我们用数组 d 记录每个结点的最短路径估计值，用邻接表来存储图 G。我们采取的方法是动态逼近法：设立一个先进先出的队列用来保存待优化的结点，优化时每次取出队首结点 u，并且用 u 点当前的最短路径估计值对离开 u 点所指向的结点 v 进行松弛操作，如果 v 点的最短路径估计值有所调整，且 v 点不在当前的队列中，就将 v 点放入队尾。这样不断从队列中取出结点来进行松弛操作，直至队列空为止</p> <p>期望的时间复杂度 $O(ke)$，其中 k 为所有顶点进队的平均次数，可以证明 k 一般小于等于 2。</p> <p>用数组模拟建立一个队列，初始时队列里只有起始点，再建立一个表格记录起始点到所有点的最短路径（该表格的初始值要赋为极大值，该点到他本身的路径赋为 0）。然后执行松弛操作，用队列里有的点作为起始点去刷新到所有点的最短路，如果刷新成功且被刷新点不在队列中则把该点加入到队列最后。重复执行直到队列为空。</p> <pre> #define MAXVEX 10010 #define MAXARC 500010 #define INF 2147483647 int dep; int v1S[MAXARC], v2S[MAXARC], weightS[MAXARC], nex[MAXARC]; int group[MAXARC * 5], dist[MAXVEX + 1]; void Store(int v1, int v2, int weight) { dep++; v2S[dep] = v2; weightS[dep] = weight; nex[dep] = v1S[v1]; v1S[v1] = dep; } void SPFA(int start, int Vexnum) { bool visited[MAXVEX + 1]; int i, head = 0, tail = 1; for (i = 1; i <= Vexnum; i++) { </pre>
----------	--

	<pre> dist[i] = INF; visited[i] = false; } dist[start] = 0; group[tail] = start; visited[start] = true; while (head <= tail) { head++; int temp = group[head]; int k = v1S[temp]; while (k != 0) { int cur = v2S[k]; if (dist[cur]>dist[temp] + weightS[k]) { dist[cur] = dist[temp] + weightS[k]; if (visited[cur] == false) { visited[cur] = true; tail++; group[tail] = cur; } } k = nex[k]; } visited[temp] = false; } } int main() { int Vexnum, arcnum, start, end; int v1, v2, weight; int i; cin >> Vexnum >> arcnum >> start >> end; dep = 0; for (i = 1; i <= arcnum; i++) { cin >> v1 >> v2 >> weight; Store(v1, v2, weight); Store(v2, v1, weight); } SPFA(start, Vexnum); cout << dist[end] << endl; return 0; } </pre>
开	Visual studio 2017

发 环 境	
调 试 分 析	
心 得 体 会	<p>1、单源最短路径各种方法总结：</p> <p>Floyd算法 求多源、无负权边的最短路。用矩阵记录图。时效性较差，时间复杂度$O(V^3)$。 Floyd算法是解决任意两点间的最短路径的一种算法，可以正确处理有向图或负权的最短路径问题。算法的时间复杂度为$O(N^3)$，空间复杂度为$O(N^2)$。 原理是动态规划： 设$D_{i,j,k}$为从i到j的只以$\{1..k\}$集合中的节点为中间节点的最短路径的长度。 若最短路径经过点k，则$D_{i,j,k} = D_{i,k,k-1} + D_{k,j,k-1}$； 若最短路径不经过点$k$，则$D_{i,j,k} = D_{i,j,k-1}$。 因此，$D_{i,j,k} = \min(D_{i,k,k-1} + D_{k,j,k-1}, D_{i,j,k-1})$。 在实际算法中，为了节约空间，可以直接在原来空间上进行迭代，这样空间可降至二维。描述如下：</p> <pre> for k ← 1 to n do for i ← 1 to n do for j ← 1 to n do if ($D_{i,k} + D_{k,j} < D_{i,j}$) then $D_{i,j} \leftarrow D_{i,k} + D_{k,j}$; </pre> <p>其中$D_{i,j}$表示由点$i$到点$j$的代价，当$D_{i,j}$为 ∞ 表示两点之间没有任何连接。</p> <p>Dijkstra算法 求单源、无负权的最短路。时效性较好，时间复杂度为$O(V*V+E)$。源点可达的话，$O(V*\lg V + E*\lg V) \Rightarrow O(E*\lg V)$。当是稀疏图的情况时，此时$E=V*V/\lg V$，所以算法的时间复杂度可为$O(V^2)$。若是斐波那契堆作优先队列的话，算法时间复杂度，则为$O(V*\lg V + E)$。</p>

Bellman-Ford算法

求单源最短路，可以判断有无负权回路（若有，则不存在最短路），时效性较好，时间复杂度 $O(VE)$ 。此算法日后还会在本BLOG内具体阐述。Bellman-Ford算法是求解单源最短路径问题的一种算法。Dijkstra算法不同的是，在Bellman-Ford算法中，边的权值可以为负数。假如我们可以从图中找到一个环路（即从 v 出发，经过若干个点之后又回到 v ）且这个环路中所有边的权值之和为负。那么通过这个环路，环路中任意两点的最短路径就可以无穷小下去。如果不处理这个负环路，程序就会永远运行下去。而Bellman-Ford算法具有分辨这种负环路的能力。

SPFA算法

是Bellman-Ford的队列优化，时效性相对好，时间复杂度 $O(kE)$ 。（ $k \ll V$ ）。与Bellman-ford算法类似，SPFA算法采用一系列的松弛操作以得到从某一个节点出发到达图中其它所有节点的最短路径。所不同的是，SPFA算法通过维护一个队列，使得一个节点的当前最短路径被更新之后没有必要立刻去更新其他的节点，从而大大减少了重复的操作次数。

SPFA算法可以用于存在负数边权的图，这与dijkstra算法是不同的。与Dijkstra算法与Bellman-ford算法都不同，SPFA的算法时间效率是不稳定的，即它对于不同的图所需要的时间有很大的差别。

在最好情形下，每一个节点都只入队一次，则算法实际上变为广度优先遍历，其时间复杂度仅为 $O(E)$ 。另一方面，存在这样的例子，使得每一个节点都被入队 $(V-1)$ 次，此时算法退化为Bellman-ford算法，其时间复杂度为 $O(VE)$ 。

SPFA算法在负边权图上可以完全取代Bellman-ford算法，另外在稀疏图中也表现良好。但是在非负边权图中，为了避免最坏情况的出现，通常使用效率更加稳定的Dijkstra算法，以及它的使用堆优化的版本。通常的SPFA算法在一类网格图中的表现不尽如人意。