广东金融学院实验报告

课程名称：算法分析与设计

装订线

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验编号  及实验名称 | 算法分析与设计实验9 | | | 系 别 | 计算科学与技术 |
| 姓 名 | 林旋华 | 学 号 | 181543306 | 班 级 | 1815433 |
| 实验地点 | 电教503 | 实验日期 | 2020.11.26 | 实验时数 | 2 |
| 指导教师 | 郭艺辉 | 同组其他成员 | 无 | 成 绩 |  |
| 1. 实验目的及要求   1) 掌握贪心算法的基本思想以及基本原理。  2) 掌握使用贪心算法求解问题的一般特征以及步骤。  3) 掌握贪心算法设计方法以及复杂性分析方法。  4) 掌握贪心法解单源最短路径问题、最小生成树问题算法设计思想、设计过程以及程序实现。 | | | | | |
| 1. 实验环境及相关情况（包含使用软件、实验设备、主要仪器及材料等）   1) 操作系统：Windows操作系统  2) 开发工具：Eclipse、JDK  3) 开发语言：Java | | | | | |
| 1. 实验内容及步骤（包含简要的实验步骤流程）   1、单源最短路径问题的问题提出是，计算带权有向图G =(V, E)中一个点（源点）到其余各顶点的最短路径长度，如下图所示。设源点为顶点1，采用Dijkstra算法求下图中源V0为到其余各顶点的最短路径。     1. 将算法编程实现, 并将程序与运算结果截屏填写入实验结果。 2. 分析Dijkstra算法的时间复杂性。   2、设G=(V,E)是连通带权图，如下图所示：    使用Prim算法建立G的最小生成树：   1. 将算法编程实现, 将程序与运算结果截屏填入实验结果。 2. 分析算法的时间复杂性。 | | | | | |
| 1. 实验结果（包括程序或图表、结论陈述、数据记录及分析等，可附页）  * 代码：   **public** **class** dijkstra {  **public** **static** **final** **float** ***max*** = Float.***MAX\_VALUE***;  **public** **static** **void** main(String[] args) {  **float**[][] a = { { 0, 3, 4, ***max***, ***max***, ***max*** }, { ***max***, 0, 1, 9, 4, ***max*** },  { ***max***, ***max***, 0, 5, 13, ***max*** }, { ***max***, ***max***, ***max***, 0, ***max***, 8 },  { ***max***, ***max***, ***max***, 12, 0, 10 }, { ***max***, ***max***, ***max***, ***max***, ***max***, 0 } };  **float**[] dist = **new** **float**[6];  **int**[] prev = **new** **int**[6];  *dijkstra*(0, a, dist, prev);  **for** (**int** i = 0; i < dist.length; i++) {  System.***out***.println("源点到"+(i+1)+"的最短路径为:"+dist[i] + "，前驱点是" + prev[i]);  }  }  // 单元最短路径问题的Dijkstra算法  **public** **static** **void** dijkstra(**int** v, **float**[][] a, **float**[] dist, **int**[] prev) {  **int** n = dist.length - 1;  **if** (v < 0 || v > n - 1)  **return**;  **boolean**[] s = **new** **boolean**[n + 1];  // 初始化  **for** (**int** i = 1; i <= n; i++) {  dist[i] = a[v][i];  s[i] = **false**;  **if** (dist[i] == Float.***MAX\_VALUE***) {  prev[i] = 0;  } **else** {  prev[i] = v;  }  }  dist[v] = 0;  s[v] = **true**;  **for** (**int** i = 1; i < n; i++) {  **float** temp = Float.***MAX\_VALUE***;  **int** u = v;  **for** (**int** j = 1; j <= n; j++) {  **if** ((!s[j]) && (dist[j] < temp)) {  u = j;  temp = dist[j];  }  }  s[u] = **true**; // 找到了第一个并入S的节点  **for** (**int** j = 1; j <= n; j++) {  **if** ((!s[j]) && (a[u][j] < Float.***MAX\_VALUE***)) {  **float** newdist = dist[u] + a[u][j];  **if** (newdist < dist[j]) {  // dist[j] 减少  dist[j] = newdist;  prev[j] = u;  }  }  }  }  }  }   * 结果截图：      * Dijkstra算法的时间复杂性  1. 初始化dist和 s均为 O(n) 2. 外层循环，每次删除一个顶点，共n个顶点，故为 O(n) 3. 内层为O(n)，遍历邻居为 O(v)(v为每个节点最大的邻居数)   综上，**邻接矩阵的**Dijkstra时间复杂度复杂度为 O(n) + O(n2) + O(vn) ∼ O(n2)   * 代码：   **public** **class** prim {  **public** **static** **void** main(String[] args) {  **float** m = Float.***MAX\_VALUE***;  **float**[][] weight = { { 0, 0, 0, 0, 0, 0, 0 }, { 0, m, 34, m, m, 12, m }, { 0, 34, m, 46, m, m, 19 },  { 0, m, 46, m, 17, m, 25 }, { 0, m, m, 17, m, 38, 25 }, { 0, 12, m, m, 38, m, 26 },  { 0, m, 19, 25, 25, 26, m } };// 上图的矩阵  *prim*(weight.length - 1, weight);  }  **public** **static** **void** prim(**int** n, **float**[][] c) { // n为顶点数，c为权  **float**[] lowcost = **new** **float**[n + 1]; // 到新集合到最小值  **int**[] closest = **new** **int**[n + 1]; // 与s集合相连到最小权边到点  **boolean**[] s = **new** **boolean**[n + 1]; // s[i] == true表示i点在s集合中  s[1] = **true**; // 将第一个点放入s集合  **for** (**int** i = 2; i <= n; i++) { // 初始化  lowcost[i] = c[1][i];  closest[i] = 1;  s[i] = **false**;  }  **for** (**int** i = 1; i < n; i++) {  **float** min = Float.***MAX\_VALUE***;  **int** j = 1;  **for** (**int** k = 2; k <= n; k++) {  **if** ((lowcost[k] < min) && (!s[k])) { // 根据最小权加入新点  min = lowcost[k];  j = k;  }  }  System.***out***.println("加入点" + j + "---" + closest[j]);// 相邻的点  s[j] = **true**; // 加入新点  **for** (**int** k = 2; k <= n; k++) {  **if** ((c[j][k] < lowcost[k]) && (!s[k])) {// 根据新加入点j,求得最小值  lowcost[k] = c[j][k];  closest[k] = j;  }  }  }  }  }   * 结果截图：      * Prim算法的时间复杂性  1. 初始化为 O(n) 2. 重复下列操作，直到Vnew = V：   a.在集合E中选取权值最小的边<u, v>，其中u为集合Vnew中的元素，而v不在Vnew集合当中，并且v∈V（如果存在有多条满足前述条件即具有相同权值的边，则可任意选取其中之一）；  b.将v加入集合Vnew中，将<u, v>边加入集合Enew中；  因为上述步骤内层循环的时间复杂度为O（n），而它的外层循环的时间复杂度也是O（n）,所以，prim算法时间复杂度为n2。 | | | | | |
| 1. 实验总结（包括心得体会、问题回答及实验改进意见，可附页）   1.Dijkstra算法是一种基于贪心策略的算法。每次新扩展一个路程最短的点，更新与其相邻的点的路程。当所有边权都为正时，由于不会存在一个路程更短的没扩展过的点，所以这个点的路程永远不会再被改变，因而保证了算法的正确性。  2.Prim算法是从点的方面考虑构建一颗MST（Minimum Spanning Tree，最小生成树），大致思想是：设图G顶点集合为U，首先任意选择图G中的一点作为起始点a，将该点加入集合V，再从集合U-V中找到另一点b使得点b到V中任意一点的权值最小，此时将b点也加入集合V；以此类推，现在的集合V={a，b}，再从集合U-V中找到另一点c使得点c到V中任意一点的权值最小，此时将c点加入集合V，直至所有顶点全部被加入V，此时就构建出了一颗MST。因为有N个顶点，所以该MST就有N-1条边，每一次向集合V中加入一个点，就意味着找到一条MST的边。  通过本次实验,我学习了掌握贪心算法的基本思想以及基本原理，基本掌握了使用使用贪心算法求解问题的一般特征以及步骤，也掌握了贪心算法设计方法以及复杂性分析方法，对贪心法解单源最短路径问题、最小生成树问题算法设计思想、设计过程以及程序实现有了进一步的认识. | | | | | |
| 六、教师评语  1、完成所有规定的实验内容，实验步骤正确，结果正确；  2、完成绝大部分规定的实验内容，实验步骤正确，结果正确；  3、完成大部分规定的实验内容，实验步骤正确，结果正确；  4、基本完成规定的实验内容，实验步骤基本正确，所完成的结果基本正确；  5、未能很好地完成规定的实验内容或实验步骤不正确或结果不正确。  6、其它：  评定等级：优秀 良好 中等 及格 不及格  教师签名：郭艺辉 | | | | | |