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 | 实验5 | | | 系 别 | 互联网金融与信息工程 |
| 姓 名 | 谢潮锋 | 学 号 | 17B543157 | 班 级 | 1715431 |
| 实验地点 | 实验503 | 实验日期 | 2019.11.21 | 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 2. 掌握贪心算法的基本思想以及基本原理。 3. 掌握使用贪心算法求解问题的一般特征以及步骤。 4. 掌握贪心算法设计方法以及复杂性分析方法。 5. 掌握使用动态规划解0-1背包问题以及贪心算法解单源最短路径设计思想、算法设计过程以及程序编码实现。 | | | | | |
| 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） 2. 0-1背包的问题提出是，有*n*个物品，其中物品*i*的重量是，价值为，有一容量为*C*的背包，要求选择若干物品装入背包，使装入背包的物品总价值达到最大。此问题的形式化描述是：给定，要求找出*n*元0-1向量, , ，使得目标函数达到最大，并且要满足约束条件。设。采用动态规划算法解决该0-1背包问题。 3. 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填写入实验结果。 2. 分析Dijkstra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）   第一题：  【运行截图】    【代码实现】  **package** DyPlaning; **import** java.util.Arrays;  **public class** AllBags {  **int bag**; *//容量* **int**[] **weight**; *//物品重量* **int**[] **value**; *//物品价值* **int point**; *//定位指针* **int**[][] **m**;    *//构造函数* AllBags(**int** bag,**int**[] weight, **int**[] value){  **this**.**bag** = bag;  **this**.**weight** = weight;  **this**.**value** = value;  **this**.**point** = weight.**length** -1; *// 记录表：横坐标从0~C；纵坐标为物品种类* **this**.**m** = **new int**[weight.**length**][bag+1];  }    **void** pull() {  *//当背包容量比物品小时，放不进去* **for**(**int** j=0 ;j < **weight**[**point**]; j++) {  **m**[**point**][j] = 0;  }  *//背包容量可容第一个物品（初始化表格第一行）* **for**(**int** j=**weight**[**point**] ; j <= **bag**; j++) {  **m**[**point**][j] = **value**[**point**];  }  *//从第二行开始* **for**(**int** i=**point** -1 ; i>=0;i--) {  *//当背包小于当前物品重量时，价值为0* **for**(**int** j=0;j<**weight**[i];j++) {  *//装不进去，那就是相当于不装，结果为子问题的解* **m**[i][j] = **m**[i+1][j];  }  *//当恰能容纳当前物品重量时* **for**(**int** j = **weight**[i]; j <= **bag** ;j++) {  **m**[i][j] = Math.*max*(**value**[i] + **m**[i+1][j-(**weight**[i])], **m**[i+1][j]);  }  }  }   **void** printBag(){  **int**[] x = **new int**[**weight**.**length**];  **for** (**int** i = 0; i < **weight**.**length**-1; i++) {  **if** (**m**[i][**bag**] == **m**[i+1][**bag**]){  x[i] = 0;  }**else** {  x[i] = 1;  **bag** -= **weight**[i];  }  } *// 判断容量是否能够放入最小子问题对应的物品* x[**weight**.**length**-1] = (**m**[**weight**.**length**-1][**bag**]>0)?1:0;  System.***out***.println(**"放入背包情况为："**+Arrays.*toString*(x));  }   **public static void** main(String[] args) {  **int**[] w = {5,4,8,6,9};  **int**[] v = {20,6,8,15,18};  AllBags allBags = **new** AllBags(18,w,v);  allBags.pull();  allBags.printBag();  System.***out***.println(**"价值记录如下："**);  **for**(**int**[] value:allBags.**m**) {  System.***out***.println(Arrays.*toString*(value));  }  }     }  第二题：  将图转化为二维表进行初始化：  值表示行走该路径的权，等于-1表示不可行走。    【运行截图】    【代码实现】  //测试程序  **import** greed.MiniRounte; **import** org.junit.Test;  **public class** GreedTest {  @Test  **public void** greedTest(){  MiniRounte miniRounte = **new** MiniRounte(1);  miniRounte.go();  } }  //实现代码  **package** greed;  **import** java.util.ArrayList; **import** java.util.Arrays;  *//贪心算法实现有向图最短路径搜索* **public class** MiniRounte {**int start**,**len**;**int**[] **allNode**;**int**[][] **margin**;**int**[] **pre**;**int**[] **dist** ;ArrayList<Integer> **source**;ArrayList<Integer> **notSource**; **int MAX**;**public** MiniRounte(**int** s){  **this**.**start** = s - 1;**MAX** = Integer.***MAX\_VALUE***;  **int**[][] margin = {  {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**,**MAX**,0}  };  **int**[] allNode = {1,2,3,4,5,6};  **len** = allNode.**length**;  **source** = **new** ArrayList<Integer>();  **notSource** = **new** ArrayList<Integer>();**source**.add(allNode[**start**]);**for** (**int** i = **start**; i < allNode.**length** ; i++) {  **notSource**.add(allNode[i]);  }  **dist** = **new int**[**len**];  **pre** = **new int**[**len**];  **for** (**int** i = 0; i < **len**; i++) {  **dist**[i] = margin[**start**][i];  **if** (margin[**start**][i] != **MAX**) **pre**[i] = allNode[**start**];  }  **this**.**allNode** = allNode;  **this**.**margin** = margin;  }   **public void** go(){  **int** next = -1;   **while** (**true**){  **if** (**source**.size() == **len**) **break**;   next = arrMin(**notSource**); **source**.add(next);  **notSource**.remove((Integer) next);  next --; **for** (**int** i = 0; i < **len**; i++) {**if** (**margin**[next][i] != **MAX**){  **if** (**margin**[next][i] + **dist**[next] < **dist**[i]){  **dist**[i] = **margin**[next][i] + **dist**[next];**pre**[i] = next;  }  }  }  }  System.***out***.println(**"从"**+(**start**+1)+**"出发，到达个点的最小代价依次为："**+Arrays.*toString*(**dist**));  System.***out***.println(**"相对应的经过的前一个结点为:"**+Arrays.*toString*(**pre**));  }  *// 扫描数组中最小的元素（主要用于扫描非源）* **int** arrMin(ArrayList a){  **int** min = **MAX**;  **for** (**int** i = 0; i < a.size() ; i++) {  min = Math.*min*((**int**) a.get(i),min);  }  **return** min;  } } | | | | | |
| 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）   本实验中可以采用邻接矩阵或者邻接表来记录网的情况，采用邻接矩阵实现方便数据访问，但会造成空间上的浪费。本算法时间复杂度为O(n2) | | | | | |
| 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 | | | | | |