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10 | | | 系 别 | 计算机科学与技术 |
| 姓 名 | 林旋华 | 学 号 | 181543306 | 班 级 | 1815433 |
| 实验地点 | 电教503 | 实验日期 | 2020.12.3 | 实验时数 |  |
| 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. 掌握回溯法的基本思想以及基本原理。 3. 掌握使用回溯法求解问题的一般特征以及步骤。 4. 掌握回溯法算法设计方法以及复杂性分析方法。 5. 掌握使用回溯法求解装载问题、批处理作业调度问题算法设计思想、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. 装载问题的问题提出是，有一批共*n*个集装箱要装上2艘载重量分别为和的轮船，其中集装箱*i*的重量为，且。问是否有一个合理的装载方案能将这*n*个集装箱装上这两艘轮船。该问题形式化描述为：     设*n*=4, =10, =12, *w*={5, 2, 1, 3}。  采用回溯法解决该装载问题，并回答以下问题：   1. 如何定义装载问题解空间？如何组织装载问题的解空间结构？ 2. 写出算法实现代码并截屏程序的运行结果。 3. 对算法做复杂性分析。 4. 批处理作业调度问题的问题提出是，设有n个作业需要处理，每一个作业（）都有两项任务组成，第一项任务必须在机器1处理完成之后才能在机器2处理。不同的作业调度方案处理完所有作业所需的时间不同。批处理作业调度问题要求制定最佳作业调度方案，使其完成时间和达到最小。例如，有5个作业需要处理，作业需要机器1和机器2的处理时间如表所示。采用回溯法求该作业最佳调度方案。  |  |  |  | | --- | --- | --- | |  | 机器1 | 机器2 | | 作业1 | 3 | 5 | | 作业2 | 6 | 1 | | 作业3 | 5 | 2 | | 作业4 | 4 | 4 | | 作业5 | 3 | 2 |  1. 写出算法实现代码并截屏程序的运行结果。 2. 对算法做复杂性分析。 | | | | | |
| 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） 2. 装载问题  * **定义**装载问题**解空间** 对于有4个集装箱要装上轮船的装载问题，其解空间由长度为4的0-1向量即{(0,0,0,0),(0,0,0,1),(0,0,1,0),(0,0,1,1),(0,1,0,0),(0,1,0,1),(0,1,1,0),(0,1,1,1)(1,0,0,0),(1,0,0,1),(1,0,1,0),(1,0,1,1),(1,1,0,0),(1,1,0,1),(1,1,1,0),(1,1,1,1)}组成。 * 组织装载问题的解空间结构   装载问题的解空间结构是一棵完全二叉树。解空间树中每个结点都有左右两个分支，左分支用1标识，表示把第 i 个集装箱放上轮船，右分支用0标识，表示不把集装箱 i 放上轮船。解空间树的第i层到第 i+1层边上的标号给出了变量的值，从树根到叶的任意一条路径表示解空间中的一个元素。例如，从根结点1到叶结点 23 的路径对应于解空间中的元素(1,0,0,0)。装载问题的解空间树如图所示。  4822c21652b27afeb37cb41b929649b  **package** backtrack;  **public** **class** Loading1 {  **int** n; //集装箱的数量  **int**[] w; //集装箱重量数组，记录每个集装箱的重量  **int** totalWeight;//集装箱总重量  **int** c1; // 第一艘轮船的载重量  **int** c2; // 第二艘轮船的载重量  **int** cw2;// 第二艘轮船需要装载的重量  **int** cw; //当前的载重量  **int** bestw; // 当前最优载重量  **int** r; //剩余集装箱重量  **int**[] x; //当前解  **int**[] bestx; //当前最优解  **public** **void** maxLoading(**int** num,**int**[] ww,**int** c1,**int** c2) {  //初始化数据成员  w=ww;  **this**.c1=c1;// 第一艘轮船的载重量  **this**.c2=c2;// 第一艘轮船的载重量  cw2=0;  cw=0;//当前的载重量  bestw=0;// 当前最优载重量  n=num;  x=**new** **int**[n+1];  bestx=**new** **int**[n+1];  **for**(**int** i=1;i<=n;i++) {  r+=w[i]; //初始化r  totalWeight+=w[i]; //初始化totalWeight  }  //调用backtrack(i)函数计算最优载重量  backtrack(1);  //计算第二艘轮船需要的装载量  cw2=totalWeight-bestw;  //输出最佳装载方案  **if**(cw2<= c2) { //如果第二艘轮船需要的装载量小于其载重量，则两艘轮船可以装载所有物品  System.***out***.println("可以装载所有货物！");  System.***out***.println("第一艘轮船的最优装载方案为：");  **for**(**int** k=1;k<=n;k++) {  System.***out***.print(bestx[k]+" ");  }  System.***out***.println();  //输出最优装载量  System.***out***.println("第一艘轮船的最优装载量为："+bestw);  System.***out***.println("第二艘轮船需要装载的重量为："+cweight2);  }  **else** {  System.***out***.println("无法装载所有货物！");  }  }  //回溯法实现求解最优装载问题  **public** **void** backtrack(**int** i){  //搜索第i层结点  **if**(i>n)  { //到达叶节点  **for**(**int** j=1;j<=n;j++) {  bestx[j]=x[j];  }  bestw=cw;  **return**;    }  //搜索子树  r-=w[i];  **if**(cw+w[i]<=c1)  { //搜索左子树  x[i]=1;  cw+=w[i];  backtrack(i+1);  cw-=w[i];  }  **if**(cw+r>bestw) {// 搜索右子树  x[i]=0;  backtrack(i+1);  }  r+=w[i];  }  **public** **static** **void** main(String[] args) {  **int** n=4;//集装箱的数量  **int** c1=10;//第一艘轮船的载重量  **int** c2=12;  **int**[] weight= {0,5,2,1,3};//集装箱重量数组  Loading1 l=**new** Loading1();  l.maxLoading(n,weight,c1,c2);  }  }     1. 回溯法解决装载问题算法的时间复杂度为O(n2n). 2. 批处理作业调度问题   1)  **package** backtrack;  **public** **class** FlowShop1 {  **int** n;//作业数  **int** f1;//机器1完成处理时间  **int** f;//完成时间和  **int** bestf;//当前最优值  **int**[][] m;//各作业所需的处理时间  **int** []x;//当前作业调度  **int**[] bestx;//当前最优作业调度  **int**[] f2;//机器2完成处理时间  **public** FlowShop1(**int** n,**int**[][] m){  **this**.n=n;  **this**.m=m;  f1=0;  f=0;  bestf=10000;//给定初始值  bestx=**new** **int**[n+1];  x=**new** **int**[n+1];  //初始化，x[i]为原始排序  **for**(**int** i=1;i<=n;i++){  x[i]=i;  }  f2=**new** **int**[n+1];  }  **public** **void** swap(**int**[] x,**int** i,**int** j){  **int** temp=x[i];  x[i]=x[j];  x[j]=temp;  }  **public** **void** backtrack(**int** i){  **if**(i>n){  **for**(**int** j=1;j<=n;j++)  bestx[j]=x[j];  bestf=f;  }  **else**{  **for**(**int** j=i;j<=n;j++){  f1+=m[x[j]][1];//作业x[j]在第一台机器的时间  f2[i]=((f2[i-1]>f1)?f2[i-1]:f1)+m[x[j]][2];//f2[i]等于f2[i-1]和f1中较大者加上作业x[j]在第2台机器的时间  f+=f2[i];  **if**(f<bestf){  swap(x,i,j);  backtrack(i+1);  swap(x,i,j);  }  f1-=m[x[j]][1];  f-=f2[i];  }  }  }  **public** **static** **void** main(String[] args) {  **int** n=5;  **int**[][] m={{0,0,0},{0,3,5},{0,6,1},{0,5,2},{0,5,4},{0,3,2}};//m的下标从1开始，因此第一行的0和每一行第一列的0无用  FlowShop1 f=**new** FlowShop1(n,m);  f.backtrack(1);  System.***out***.println("最优批处理作业调度顺序为：");  **for**(**int** i=1;i<=n;i++)  System.***out***.print(f.bestx[i]+" ");  System.***out***.println();  System.***out***.println("最优调度所需的最短时间为："+f.bestf);  }  }    2)批处理作业调度问题的回溯算法的总时间耗费为O(nn!). | | | | | |
| 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）   通过本次实验，我了解到了回溯法的基本思想以及基本原理，掌握了使用回溯法求解问题的一般特征以及步骤，也掌握了回溯法算法设计方法以及复杂性分析方法，还掌握使用回溯法求解装载问题、批处理作业调度问题算法设计思想、设计过程以及程序编码实现。  由此，我们可以总结出回溯法的一般步骤:(1)针对所给问题，定义问题的解空间;(2)确定易于搜索的解空间结构;(3)以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。通过DFS思想完成回溯。可以看出，回溯法的优点在于其程序结构明确，可读性强，易于理解，而且通过对问题的分析可以大大提高运行效率。但是，对于可以得出明显的递推公式迭代求解的问题，还是不要用回溯法，因为它花费的时间比较长。 | | | | | |
| 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 | | | | | |