算法设计与分析文档

张梦华 201706401229 计172班

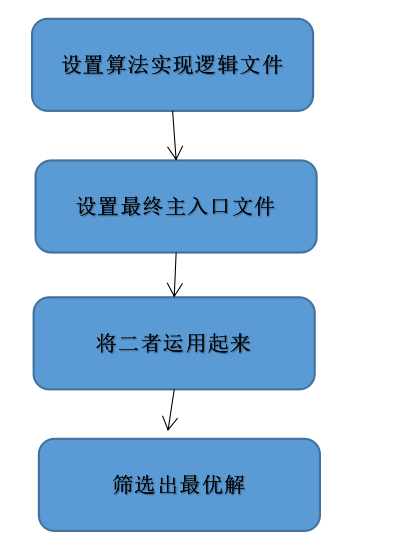
一、问题描述

有n件物品和m个约束的背包。将物品装入背包中，求解以最优价值。

二、基本思路

此算法类似于0-1背包，属于找最优解问题，用回溯法构造解的子集树。只是这里并不是只有一个约束，而是有多个约束，也就是我们需要解决的问题。对于每一个物品i，对于该物品只有选与不选2个决策，总共有n个物品，可以顺序依次考虑每个物品，这样就形成了一棵解空间树： 基本思想就是遍历这棵树，以枚举所有情况，最后进行判断，如果重量不超过背包容量，且价值最大的话，该方案就是最后的答案。在搜索状态空间树时，只要左子节点是可一个可行结点，搜索就进入其左子树。对于右子树时，先计算上界函数，以判断是否将其减去（剪枝）。

1. 流程图



四、优化空间复杂度

因为物品只有选与不选2个决策，而总共有n个物品，所以时间复杂度为IMG_256。因为递归栈最多达到n层，在每一层下面要判断该物品是否满足条件，是否能放进去则又要循环m层，则函数的复杂性为O(n×m)。

五、算法设计

有两个java文件，Runner表示最终主入口，Algorithms是辅助Runner的算法实现逻辑。定义一个Algorithms类，定义相应的成员变量，接着定义构造方法，用回溯法来搜索子集空间数，伪代码如下：

**public void** b(**int** i) { *//i用来指示到达的层数（第几步，从0开始），同时也指示当前选择了几个物品  
 //递归的基准情形，当遍历到叶子节点* **if**( i>=**n**) {  
 **if**(**cv**>=**maxvalue**) {  
 **maxvalue** = **cv**;  
 **bestv** = **cw**;  
 **for** (**int** k = 0; k < **n**; k++) {  
 **bestway** += Integer.*toString*(**path**[k]);  
 }  
 **a**++;  
 **flags**[**flag**][0]=Integer.*toString*(**a**);  
 **flags**[**flag**][**a**]=**bestway**;  
 **flagss**[**flag**][**a**-1]=**maxvalue**;  
 System.***out***.println(**bestway** + **" "** + **maxvalue** + **" "** + **bestv**+**" "**);  
 **bestway** = **""**;  
  
 }  
  
  
 **return**;  
 }  
 **if** (**cv** + **v**[i] <= **mc**&& **cw** + **w**[i] <=**c** ) { *//将物品i放入背包,搜索左子树* **path**[i] = 1;  
 **cw** += **w**[i];*//同步更新当前背包的重量* **cv** += **v**[i];*//同步更新当前背包的总价值* b(i + 1); *//深度搜索进入下一层* **cw** -= **w**[i];*//回溯复原* **cv** -= **v**[i];*//回溯复原* }  
 **if** (*Bound*(i + 1,**c**,**cw**,**cv**,**n**,**w**,**v**) > **bestv**) { *//如若符合条件则搜索右子树* **path**[i] = 0;  
 b(i + 1);  
 }  
 }  
 **private static int** Bound(**int** p,**int** c,**int** cw,**int** cv,**int** n,**int** []w,**int** v[]) {  
 **int** cleft = c - cw;*//剩余背包容量* **int** b = cv;*//记录当前背包的总价值cv,最后求上界  
 //以物品单位重量价值递减次序装入物品* **while** (p <=n && w[p] <= cleft) {  
 cleft -= w[p];  
 b += v[p];  
 p++;  
 }  
 **if** (p <n) { *// 拆分* **int** tmp = v[p] / w[p];  
 b += tmp \* cleft;  
 }  
 **return** b;*//返回计算出的上界* }  
}

在递归函数Backtrack中，当i>n时，算法搜索至叶子结点，得到一个新的物品装包方案。此时算法适时更新当前的最优价值。当i<n时，当前扩展结点位于排列树的第（i-1）层，此时算法选择下一个要安排的物品，以深度优先方式递归的对相应的子树进行搜索，对不满足上界约束的结点，则剪去相应的子树。

定义一个Runner类，定义相应的成员变量，调用方法用于输出n个物品和m个约束及最大价值x，以及相对应的约束值。根据Algorithm的实际算法，来对输入的数据进行选择和优化，也就是计算每个约束的可选算法和价值，从中选择最优的，最终得到最优解。部分代码如下：

**public static int**[][][] initData() {  
 **int** i = 0;  
 **try** {  
 System.***out***.println(**"请请输入商品数n和约束数m及最大价值x:"**);  
 Scanner scanner = **new** Scanner(System.***in***);  
 String fnums = scanner.nextLine();  
 String s = fnums;  
 String[] basicData = s.split(**" "**); *//存放商品数+约束数+最大价值  
 dimensions* = Integer.*parseInt*(basicData [0]); *// 读取出商品个数  
 givevalue* = Integer.*parseInt*(basicData [2]); *// 读取价格  
 splitss* = **new** String[Integer.*parseInt*(basicData [1]) + 2]; *// 计算总行数=商品个数+约束+结果  
 sp* = **new** String[Integer.*parseInt*(basicData [0])]; *// 除去第一行有多少行  
 a* = **new int**[1][Integer.*parseInt*(basicData [1]) + 2][Integer.*parseInt*(basicData [0]) \* Integer.*parseInt*(basicData [0]) \* 2]; *// 转换成二维数组* System.***out***.println(**"请输入"** + basicData [0] + **"个商品价格，用空格隔开："**);  
 String price = scanner.nextLine();*//价格* List<String> list = **new** ArrayList<>();  
 list.add(price);  
 System.***out***.println(**"请输入"** + basicData [0] + **"个商品的"** + basicData [1] + **"个约束"**);  
 **for** (**int** i1 = 0; i1 < Integer.*valueOf*(basicData [1]); i1++) {  
 System.***out***.println(**"请输入第"** + (i1 + 1) + **"个约束："**);  
 String strp = scanner.nextLine();  
 list.add(strp);  
 }  
 System.***out***.println(**"请输入"** + basicData [0] + **"个商品的约束值："**);  
 String constraint = scanner.nextLine();  
 list.add(constraint);

这次的作业通过查阅大量的资料，并找寻相类似的解法，学习有关知识，问过其他同学，也许有许多不够完善的地方，也有一些不是很理解的地方，但还是尽力完成，也不负这么些天来的努力。