**算法分析与设计实验报告**

**第 三 次实验**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 李平凡 | 学号 | 201907040102 | | 班级 | 计科1905 |
| 时间 | 5.20 | 地点 | 软件大楼 | | | |
| 实验名称 | 回溯法求解0-1背包 | | | | | |
| 实验目的 | 通过搜索资料，要求掌握一些基本问题和典型例题的处理方式 | | | | | |
| 实验原理 | 采用回溯法进行有效率的穷举搜索0-1背包问题的解向量，得到一个最有价值可行解。 | | | | | |
| 实验步骤 | ①首先输入背包数，然后输入各个背包对应的重量和价值  ②采用回溯法如果这个物品能够放入背包那么就去判断下一个物品，如果不能放就去右子树，直到走到叶子节点也就是当所有物品都被判断后（已经构造出一个解向量）然后对于每种解找到其最大价值同时更新解向量  ③ 将解向量和最大价值输出 | | | | | |
| 关键代码 | void backtrace(int i,vector<int> w,vector<int> v,int c,vector<int> x,int value)  {  if(i>=n)  {  if(value>bestv)  {  bestv=value;  for(int i=0;i<n;i++)  {  best[i]=x[i];  }  }  return;  }  if(w[i]<=c)  {  c=c-w[i];  x[i]=1;  value=value+v[i];  rv-=v[i];  backtrace(i+1,w,v,c,x,value);  }  rv=rv+v[i];  if(rv-v[i]+value>bestv)  {  c=c+w[i];  x[i]=0;  value=value-v[i];  backtrace(i+1,w,v,c,x,value);  }  }  这里用了两个if，第一个if是判断这个物品能不能放进背包，也就是所谓的约束函数，如果能放，那么就可以去判断下一个物品是否能放，这里都是采用的左子树优先，我们在进入左子树之前，将其价值和可放容量都进行了一次更新，对于右子树的if是一个限界函数，如果剩下所有物品的价值加起来并加上当前价值也比最大价值小，那么就不需要遍历右子树了，这两个函数也就是区别于一般穷举法的重要标志，性能的提升也体现在这两个方面。 | | | | | |
| 测试结果 | 剩下两组测试数据是没有结果的 | | | | | |
| 实验心得 | 因为是回溯法求解0-1背包问题，这是个子集树的问题，所以当它是十个背包的时候，这个问题求解速度还算快也就是1024种情况，但这个回溯法本身就是指数级的问题求解，用的本质是穷举法，深度优先搜索，所以当有100个背包的时候这个问题就已经是2的100次方的情况问题难度，所以之后是求不出来数据，就算能求出也需要大量的时间，因此我们可以看到一个好的算法，比在硬件层面提高一个程序的性能是提高很多的，那么自然对于时间复杂度我认为是O(2^n)这个n是指背包个数。 | | | | | |
| 实验得分 |  | 助教签名 | |  | | |

**附录：完整代码**