**算法分析与设计实验报告**

**第 四 次实验**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 李平凡 | 学号 | 201907040102 | | 班级 | 计科1905 |
| 时间 | 6.2 | 地点 | 软件大楼 | | | |
| 实验名称 | 优先队列式分支限界法求解0-1背包问题 | | | | | |
| 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 | 通过每次依据优先级找到下一个有潜力达到最大价值的节点并对其处理将其后面的有潜力的子节点放入最大值堆中，直到处理到最后一层，这时得到最大价值 | | | | | |
| 实验步骤 | ①首先定义各个类包括了节点类（包括优先级，当前价值，当前重量，是否为左子节点标记，层级），物品类（存储各个物品信息）然后定义一个输入函数将物品信息都全部对应的输入进去  ②通过一个while循环判断处理到了那一层，如果到达叶子节点便返回并输出最终的最优价值和对应的物品编号，同时记录解向量，如果没有到达，便通过约束条件判断是否可以进入左子节点，如果可以便将左子节点加入到子集树中同时加入到最大值堆中并对该节点信息进行初始化，如果不能就通过（限界条件）bound借助于优先级判断是否有拥有最大价值的潜力判断是否可以进入右子节点，如果可以也将其加入到最大值堆中，然后从最大值堆中找到下一个要处理的点  ③ 最终通过判断层级的关系来指导循环的结束，当循环结束就已经达到了最大价值，此时的价值就是最优的。就可以输出结果 | | | | | |
| 关键代码 | int Bound(int i) //计算出优先级，当前价值加上背包剩余容量与剩下最大单位重量价值的乘积  {  int tmp\_cleft = c - cw;  int tmp\_cp = cp;  while(tmp\_cleft >= obj[i].weight && i <= n)  {  tmp\_cleft -= obj[i].weight;  tmp\_cp += obj[i].price;  i++;  }  if(i <= n)  {  tmp\_cp += tmp\_cleft \* obj[i].d;  }  return tmp\_cp;  }  void AddAliveNode(priority\_queue<MaxHeapQNode \*, vector<MaxHeapQNode \*>, cmp> &q, MaxHeapQNode \*E, int up, int wt, int curp, int i, int ch)  {  MaxHeapQNode \*p = new MaxHeapQNode; //将节点加入到最大值堆中更新节点信息  p->parent = E;  p->lchild = ch;  p->weight = wt;  p->upprofit = up;  p->profit = curp;  p->lev = i + 1;  q.push(p);  // cout << "加入点的信息为 " << endl;  // cout << "p->lev = " << p->lev << " p->upprofit = " << p->upprofit << " p->weight = " << p->weight << " p->profit = " << p->profit << endl;  }  void MaxKnapsack()  {  priority\_queue<MaxHeapQNode \*, vector<MaxHeapQNode \*>, cmp > q; // 大顶堆 存储各节点信息包括优先级并依据优先级排列找到下一个处理的点  MaxHeapQNode \*E = NULL; //E存储的是子集树  cw = cp = bestp = 0;  int i = 1;  int up = Bound(1); //Bound(i)函数计算的是i还未处理时候的上限值  while(i != n + 1)  {  int wt = cw + obj[i].weight;  if(wt <= c)  {  if(bestp < cp + obj[i].price)  bestp = cp + obj[i].price;  AddAliveNode(q, E, up, cw + obj[i].weight, cp + obj[i].price, i, 1);  }  up = Bound(i + 1); //注意这里 up != up - obj[i].price而且 up >= up - obj[i].price  if(up >= bestp) //注意这里必须是大于等于  {  AddAliveNode(q, E, up, cw, cp, i, 0);  }  E = q.top();  q.pop(); // 处理下一个要处理的节点  cw = E->weight;  cp = E->profit;  up = E->upprofit;  i = E->lev;  }  for(int j = n; j > 0; --j)  {  bestx[obj[E->lev - 1].id] = E->lchild;  E = E->parent;  }  } | | | | | |
| 测试结果 | 由于最终数组越界，物品类数组存储不了100000个物品对象，所以就没有得到最终结果 | | | | | |
| 实验心得 | 可以发现优先队列分支限界法明显优于简单的队列式分支限界法，但是优先队列分支限界法编码复杂度明显提高，对于队列式分支限界法而言是深度优先搜索同时利用了队列数据结构，而优先队列是利用最大值堆，优先队列数据结构并采用的选择是广度优先搜索，所以优先队列会比队列式快很多，对于队列式而言到达了1000级别的数据规模已经基本上无法计算出结果，但是优先队列可以，因为很大一部分路径优先队列都不需要搜索，那么我们就得注意一个对优先队列而言很致命的问题，和贪心算法类似贪心需要准确的贪心选择策略，而优先队列需要准确的优先级，本例中就是bound函数，它同时也帮助了优先队列式搜索进行限界剪枝操作，从时间复杂度来看O（n\*2^n）因为对于每一个节点有至少O(n)的操作包括bound函数还有最大值堆的处理，2^n代表一共有2^n个节点。 | | | | | |
| 实验得分 |  | 助教签名 | |  | | |

**附录：完整代码**