渤海大学学生实验报告

（信息科学与技术学院）

实验课课程名称： 算法分析与设计

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验室  房间号 | C508 | 日期  时间 | 2022年11月24日 第（1\2）节 | | |
| 年级、班 | 软件工程 | 学号 |  | 姓名 |  |
| 实验项  目名称 | 分析不同算法性能 | | | 指导教师 | 冷泳林 |
| 实验环境 | Dev-c++ | | | 成绩 |  |
| 实验目的 | 比较背包算法 | | | | |
| 【实验内容】（算法、程序、步骤、数据记录与计算、实验结果和讨论等）  动态规划、回溯和分支限界也分别讨论0-1背包算法。请通过实验、学习资料及自己收集资料等方式分析四种算法的性能及适用情况。并写出四种方法的核心算法。  1. 动态规划算法要求所解决的问题存在最有子结构和重叠子问题，动态规划算法一般是n步叠代计算局部最优解，每一步叠代需要计算m个子项，那么时间复杂度就是O(m\*n)。如果只保存一步叠代的结果，空间复杂度就是O(m)；如果需要保存k步叠代结果，空间复杂度就是O(m\*k)。  2. 回溯算法过程是针对所给问题，定义问题的解空间，确定易于搜索的解空间结构（排列树、子集树），从根节点开始，采用深度优先的方式遍历解空间树，在遍历的过程中采用剪枝策略提高搜索速度。  3. 分支限界法（branch and bound method）按广度优先策略搜索问题的解空间树，在搜索过程中，对待处理的节点根据限界函数估算目标函数的可能取值，从中选取使目标函数取得极值（极大或极小）的结点优先进行广度优先搜索，从而不断调整搜索方向，尽快找到问题的解。分支限界法适合求解最优化问题。  4. 解决0-1背包算法可以使用动态规划算法，贪心算法，回溯法和分支界限法等。  5. 核心算法和性能  动态规划：  int KnapSack(int n,struct goods a[],int C,int x[]){  int V[N][N+1];    for(int i = 1; i <= n; i++)  for(int j = 1; j <= C; j++)  if(j < a[i-1].wight)  V[i][j] = V[i-1][j];  else  V[i][j] = MAX(V[i-1][j],V[i-1][j-a[i-1].wight] + a[i-1].value);  for(int i = n,j = C; i > 0; i--)  {  if(V[i][j] > V[i-1][j])  {  x[i-1] = 1;  j = j - a[i-1].wight;  }  else  x[i-1] = 0;  }  return V[n][C];  }  **动态规划法求解 0/1 背包问题的时间复杂度为：O(min{nc,2^n})。**  回溯法:  void Backtrack(int t)  {  if( t > n) Output(x);  else  for(int i =t; i <= n; i++)  {  swap(x[t], x[i]);  if(Constraint(t)&&Bound(t)) Backtrack(t+1);  swap(x[t], x[i]);  }  如果解空间树中从根结点到叶结点的最长路径的长度为 h(n)，则回溯法所需要的计算空间通常为 O(h(n)) 。而显式地存储整个解空间则需要 O(2^h(n)) 或 O(h(n)!) 内存空间。  分支限界法：  int KnapSack(int n,goods a[],int C, int X[])  {  int i, k = 0; //堆中元素个数的计数器初始化为0  int v;  KNAPNODE \*xnode, \*ynode, \*znode;  HEAP x, y, z, \*heap;  heap = new HEAP[n\*n]; //分配堆的存储空间  for(i = 0; i < n; i++)  {  a[i].sign=i; //记录物体的初始编号  }  sort(a,a+n,m); //对物体按照价值重量比排序  xnode = new KNAPNODE; //建立父亲结点  for(i = 0; i < n; i++)  { //初始化结点  xnode->s1[i] = false;  }  xnode->k = xnode->w = xnode->p = 0;  while(xnode->k < n)  {  ynode = new KNAPNODE; //建立结点y  \*ynode = \*xnode; //结点x的数据复制到结点y  ynode->s1[ynode->k] = true; //装入第k个物体  ynode->w += a[ynode->k].w; //背包中物体重量累计  ynode->p += a[ynode->k].p; //背包中物体价值累计  ynode->k ++; //搜索深度++  bound(ynode, C, a, n); //计算结点y的上界  y.b = ynode->b;  y.p = ynode;  insert(heap, y, k); //结点y按上界的值插入堆中  znode = new KNAPNODE; //建立结点z  \*znode = \*xnode; //结点x的数据复制到结点z  znode->k ++; //搜索深度++  bound(znode, C, a, n); //计算节点z的上界  z.b = znode->b;  z.p = znode;  insert(heap, z, k); //结点z按上界的值插入堆中  delete xnode;  x = del\_top(heap, k); //获得堆顶元素作为新的父亲结点  xnode = x.p;  }  v = xnode->p;  for(i = 0; i < n; i++)  { //取装入背包中物体在排序前的序号  if(xnode->s1[i])  {  X[a[i].sign] =1 ;  }  else  {  X[a[i].sign] = 0;  }  }  delete xnode;  delete heap;  return v; //返回背包中物体的价值  }  分支限界法求解 0/1 背包问题的时间复杂度为：O(2^n)。  教师签字：  年 月 日 | | | | | |