**算法分析与设计实验报告**

**第 4 次实验**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 杨杰 | 学号 | 201908010705 | | 班级 | 计科1907 |
| 时间 | 6.2 | 地点 | 软件大楼 | | | |
| 实验名称 | 用优先队列式分支限界法求解0-1背包问题 | | | | | |
| 实验目的 | 通过上机实验，要求掌握用优先队列式分支限界法求解0-1背包问题的问题描述、算法设计思想、程序设计。 | | | | | |
| 实验原理 | 采用优先队列式分支限界法。  优先级定义为：节点的优先级由已装入背包的物品价值加上剩下的最大单位重量价值的物品装满剩余容量的价值和。  如果该左儿子结点是可行结点，则将它加入到子集树和活结点优先队列中；  仅当右儿子结点满足上界约束时才将它加入子集树和活结点优先队列。  一旦当前扩展结点为叶结点就说明找到一个最优解方案。 | | | | | |
| 实验步骤 | 1.对物品按单位重量价值进行排序  2.根据约束函数判断左儿子节点是否是可行节点，若是，加入到子集树和活结点优先队列中  3.根据限界函数判断右儿子节点是否是可行节点，若是，加入到子集树和活结点优先队列中  4.从活结点优先队列中取出优先值最大的节点并扩展  5.重复2-4步直到当前扩展节点为叶节点，构造解向量 | | | | | |
| 关键代码 | float Bound(Pair p[], int n, int level, int cLeft, int value)*//bound背包已有重量+剩下物品将背包装满所得到的重量之和*  {      while (level <= n && p[level].weight <= cLeft)      {          cLeft -= p[level].weight;          value += p[level].value;          level++;      }      if (level <= n)          value += p[level].S \* cLeft;      return value;  }  void AddNode(priority\_queue<Node> &Q, float bound, int cw, int cp, bool select, int level, bbnode \*E)  {      bbnode \*b = new bbnode(E, select);      Q.push(Node(cp, cw, bound, level, select, b));  }  void Knapsack(Pair p[], int n, int c, int bestx[], int &best)*//重量,价值，容量，总数n，最佳选择方式*  {      sort(p + 1, p + n + 1, cmp);*//排序*      priority\_queue<Node> Q;      int i = 1, cw = 0, cp = 0;      bbnode \*E = NULL;      float bound;      while (i <= n)      {          if (cw + p[i].weight <= c)          {*//可以放入背包*              bound = Bound(p, n, i, c - cw, cp);              if (cp + p[i].value > best)                  best = cp + p[i].value;*//及时更新尽管还没到达叶节点*              AddNode(Q, bound, cw + p[i].weight, cp + p[i].value, true, i + 1, E);*//加入堆*          }          bound = Bound(p, n, i + 1, c - cw, cp);*//计算bound值*          if (bound >= best)          {*//处理右子树*              AddNode(Q, bound, cw, cp, false, i + 1, E);          }          Node q = Q.top();*//取出最大值节点*          Q.pop();  */\*取出最大值节点，得到属性值\*/*          i = q.level;          cw = q.weight;          cp = q.value;          E = q.pr;      }  */\*构造解向量\*/*      for (int i = n; i >= 1; i--)      {          int index = p[i].id;          bestx[index] = E->select;          E = E->parent;      }  } | | | | | |
| 算法复杂度分析 | 运用算法分析的知识，分析并阐述求解本问题算法的各个模块的时空复杂度.   1. 时间复杂度：首先需要对数据排序，记为C1nlogn；而在广度搜索的时候，最坏情况下将会搜索所有的2^(n+1)-2个节点，而建立这颗子集树堆需要的频次记为C2\*（2^(n+1)-2log(2^(n+1)-2)）,每个节点还需要bound处理，其时间复杂度为O(n)，综上，时间复杂度为T（n）=O（2^n\*n+2^n\*n）=O(n\*2^n)   ② 空间复杂度：S(n)=O(2^n)，共有2^（n+1）-2个节点，每个节点对应Node结构体，记为C\*2^n，所以空间复杂度为O(2^n). | | | | | |
| 测试结果  （含运行时间） | **小规模数据**      **中规模数据**    **大规模数据**              三种方法得到的答案是一致的，可以验证代码编写正确。 | | | | | |
| 实验心得 | 用优先队列式分支限界法解0-1背包问题的思路如下：  （1）优先级限界函数  对输入数据进行预处理，将各物品依其单位重量价值从大到小进行排列  贪心选择首先选择出能放下的物品，最后判断是否是最后一个物品，如果是则说明已经完全放下，否则等比例加入背包直到背包满。  （2）优先队列分支限界法剪枝  左子节点：  左儿子节点加入背包后重量小于背包总容量wt <= c；  左子节点加入背包后判断当前背包价值是否大于最优价值，如果是那么将会更新bestp，尽管当前还没有找到一个解向量；  右子节点：  当前背包价值+剩下物品装满背包的价值之和大于当前最大价值up = Bound(i+1)，up>=bestp  （3）构造解向量  每次加入一个节点，就将其父指针指向父节点，并且设置LChild为是否为左子节点，最后反向构造出解向量。  回溯法和分支限界法比较：  回溯法和分支限界法都是在问题的解空间树上搜索问题解的算法。回溯法主要利用深度优先搜索策略，通常目标是找到问题的所有可行解；分支限界法主要利用广度优先搜索策略，通常目标是尽快找到一个满足问题约束条件的解。  经过这次实验，我对于用优先队列式分支限界法求解0-1背包问题的相关代码已基本熟悉，算法知识得到了复习与巩固。在写代码与调试的过程中，在解决问题过程中，丰富了个人编程的经历和经验，提高了个人解决问题的能力。 | | | | | |
| 实验得分 |  | 助教签名 | |  | | |

**附录：完整代码**

*//优先队列式分支限界法求解0-1背包问题*

#include <iostream>

#include <algorithm>

#include <windows.h>

#include <queue>

#include <fstream>

#define MAX 2000

using namespace std;

int bestx[MAX];

struct bbnode

{*//构造父节点*

    bbnode(bbnode \*p = NULL, bool s = false) : parent(p), select(s){};

    bbnode \*parent;

    bool select;

};

struct Node

{

    Node(int vv = 0, int ww = 0, float bb = 0, int ll = 0, bool ss = false, bbnode \*p = NULL) : value(vv), weight(ww), bound(bb), level(ll), select(ss), pr(p){};

    friend bool operator<(Node a, Node b) { return a.bound < b.bound; };*//以bound大为优先级*

    int value;*//价值*

    int weight;*//重量*

    float bound;*//背包已有重量+剩下物品将背包装满所得到的重量之和*

    int level;*//层数*

    bool select;*//是否被选择*

    bbnode \*pr;*//扩展节点指针*

};

struct Pair

{*//存放物品*

    Pair(int ww = 0, int vv = 0, int ii = 0) : weight(ww), value(vv), id(ii), S(float(vv) / float(ww)){};

    int weight, value, id;

    float S;

};

Pair p[MAX];

bool cmp(Pair a, Pair b)

{

    return a.S > b.S;

}

float Bound(Pair p[], int n, int level, int cLeft, int value)*//bound背包已有重量+剩下物品将背包装满所得到的重量之和*

{

    while (level <= n && p[level].weight <= cLeft)

    {

        cLeft -= p[level].weight;

        value += p[level].value;

        level++;

    }

    if (level <= n)

        value += p[level].S \* cLeft;

    return value;

}

void AddNode(priority\_queue<Node> &Q, float bound, int cw, int cp, bool select, int level, bbnode \*E)

{

    bbnode \*b = new bbnode(E, select);

    Q.push(Node(cp, cw, bound, level, select, b));

}

void Knapsack(Pair p[], int n, int c, int bestx[], int &best)*//重量,价值，容量，总数n，最佳选择方式*

{

    sort(p + 1, p + n + 1, cmp);*//排序*

    priority\_queue<Node> Q;

    int i = 1, cw = 0, cp = 0;

    bbnode \*E = NULL;

    float bound;

    while (i <= n)

    {

        if (cw + p[i].weight <= c)

        {*//可以放入背包*

            bound = Bound(p, n, i, c - cw, cp);

            if (cp + p[i].value > best)

                best = cp + p[i].value;*//及时更新尽管还没到达叶节点*

            AddNode(Q, bound, cw + p[i].weight, cp + p[i].value, true, i + 1, E);*//加入堆*

        }

        bound = Bound(p, n, i + 1, c - cw, cp);*//计算bound值*

        if (bound >= best)

        {*//处理右子树*

            AddNode(Q, bound, cw, cp, false, i + 1, E);

        }

        Node q = Q.top();*//取出最大值节点*

        Q.pop();

*/\*取出最大值节点，得到属性值\*/*

        i = q.level;

        cw = q.weight;

        cp = q.value;

        E = q.pr;

    }

*/\*构造解向量\*/*

    for (int i = n; i >= 1; i--)

    {

        int index = p[i].id;

        bestx[index] = E->select;

        E = E->parent;

    }

}

int main()

{

    ifstream infile("01beibao\_in.txt", ios::in);

    ofstream outfile("01beibao\_out.txt", ios::out);

    int n;*//物品数量*

    int c = 3000;*//背包容量*

    int best = 0;*//当前最优价值*

    infile >> n;

    int w, v;

    for (int i = 1; i <= n; i++)

    {

        infile >> w >> v;

        p[i] = Pair(w, v, i);*//读入数据*

    }

*/\*测试函数\*/*

    double time = 0;

    LARGE\_INTEGER nFreq, nBeginTime, nEndTime;

    QueryPerformanceFrequency(&nFreq);

    QueryPerformanceCounter(&nBeginTime);*//开始计时*

    Knapsack(p, n, c, bestx, best);

    QueryPerformanceCounter(&nEndTime);*//停止计时*

    time = (double)(nEndTime.QuadPart - nBeginTime.QuadPart) / (double)nFreq.QuadPart;*//计算程序执行时间单位为s*

*/\*输出结果\*/*

    outfile << "分支限界法\n";

    outfile << "数据规模：n=" << n << "  c=" << c << endl;

    outfile << "最优解为：" << best << endl;

    outfile << "解向量为：\n";

    for (int i = 1; i <= n; i++)

    {

        outfile << bestx[i] << ' ';

        if (i % 20 == 0 && i != n)

            outfile << endl;

    }

    outfile << "\n程序耗时：" << time \* 1000 << "ms";

    infile.close();

    outfile.close();

    return 0;

}

*//01背包数据生成器*

#include <iostream>

#include <fstream>

#include <time.h>

#define random(a, b) (rand() % (b - a + 1) + a)

using namespace std;

int main()

{

    int t;

    cin >> t;

    ofstream outfile("01beibao\_in.txt", ios::out);

    outfile << t << endl;

    srand((int)time(NULL));

    for (int i = 1; i <= 2 \* t; i++)

    {

        outfile << random(1, 100) << " ";

        if (i % 20 == 0)

            outfile << endl;

    }

    outfile.close();

    return 0;

}