北京邮电大学软件学院

2017-2018 学年第一学期实验报告

课程名称： 算法分析与设计

项目名称： 实验三：分支限界法

项目完成人：

姓名： 刘禾子 **学号：** 2017526019

指导教师： 李朝晖

日　　期： 2017 年 11 月 20 日

一、 实验目的

1、 深刻理解并掌握分支限界法的设计思想，比较与回溯法的不同之处；
2、 提高应用分支限界法设计算法的技能；
3、 理解这样一个观点：好的限界函数不仅计算简单，还要保证最优解在搜索空间中，更重要的是能在搜索的早期对超出目标函数界的结点进行丢弃，减少搜索空间，从而尽快找到问题的最优解。

二、 实验内容

基本题 1： 0—1 背包问题

给定 n 种物品和一背包。物品 i 的重量是 wi，其价值为 vi，背包的容量为 C。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大?

补充题 1： 任务分配问题的分支限界算法

任务分配问题要求把 n 项任务分配给 n 个人，每个人完成每项任务的成本不同，要求分配总成本最小的最优分配方案。

三、 实验环境

IntelliJ IDEA Community Edition 2017.2.4

四、 实验结果

1. 分支限界法解决 0/1 背包问题

```
"C:\Program Files\Java\jdk1.8.0_112\bin\java" ...
物品0: w:4.0 v:40.0
物品1: w:7.0 v:42.0
物品2: w:5.0 v:25.0
物品3: w:3.0 v:12.0

分支限界求解最大价值为：65.0
取出的物品为：
物品0 物品2
Process finished with exit code 0
```

2. 任务分配问题的分支限界法

```
"C:\Program Files\Java\jdk1.8.0_112\bin\java" ...
任务分配成本矩阵：
9 2 7 8
6 4 3 7
5 8 1 8
7 6 9 4
任务分配情况如下：
1: 任务2  2: 任务1  3: 任务3  4: 任务4
最优分配成本：13
Process finished with exit code 0
```

五、 附录

1. 分支限界法求 0/1 背包问题

（1） 问题分析

给定 n 种物品和一个容量为 C 的背包，物品 i 的重量是 wi,其价值为 vi，对于每种物品只有两种选择：装入背包或者不装入，设计最优方案使得装入背包中的物品的总价值最大。

（2） 设计方案

将 n 种物品按照单位价值由大到小排序，则第 1 个物品给出了单位重量的最大价值，最后一个物品给出了单位重量的最小价值，采用贪心法求得 0/1 背包问题的一个下界 down，然后再求上界，考虑最好的情况，背包中装入的全部是第一个物品且可以将背包装满 up=W*(v1/w1),从而得到目标函数的界[down,up],进而设计评估函数（即限界函数）：

ub=v+(C-w)*(vi+1/wi+1)

（3） 算法分析

设 n 个物品的重量存储在数组 w[n]，价值存储在数组 v[n]，背包容量 C

输入：n 个无哦的重量 w[n]，价值 v[n]，背包容量 C

输出：背包获得的最大价值和装入背包的物品

① 根据限界函数计算目标函数的上界 up;采用贪心法得到下界 down;

② 计算根节点的目标函数值并加入待处理节点表 PT;

③ 循环直到某个叶子节点的目标函数值在表 PT 中取得极大值

Ⅰ i=表 PT 中具有最大值的结点；

Ⅱ 对结点 i 的每个孩子结点 x 执行下列操作：

Ⅱ.1 如果结点 x 不满足约束条件，则丢弃该结点；

Ⅱ.2 否则，估算结点 x 的目标函数值 lb，将结点 x 加入表 PT 中；

④ 将叶子结点对应的最优值输出，回溯法求得最优解的各个分量。

（4） 源代码

```
import java.util.PriorityQueue;

public class Knapsack {
    static class PackNode implements Comparable<PackNode>{//pt 表的节点类型
        double weight,value,upvalue; //upvalue 是评估值
        int left,level;              //left 左节点标志，若左节点可行则为 1，反之则为 0 level 表示当前搜索深度
        PackNode father;             //父节点
        @Override                    //使节点在队列里按 upvalue 值排列
        public int compareTo(PackNode o) {
            return Double.compare(o.upvalue, this.upvalue);
        }
    }
    private static int n=4,c=10;
    private static double up,down;
    private static double[] w={3,5,7,4};
    private static double[] v={12,25,42,40};
```

```
    private static int[] x=new int[n];

    public static void main(String args[]){
        sort();
        for (int i=0;i<n;i++){
            System.out.println("物品"+i+": w:"+w[i]+" v:"+v[i]);
        }
        System.out.println();
        up=calc_up();   //计算上界
        down=calc_down();//用贪心法计算下届
        brand_bound();//分支限界方法
    }

    private static void brand_bound() {
        double maxValue=0;
        PriorityQueue<PackNode> pt= new PriorityQueue<>();
        PackNode root=new PackNode();
        root.level=-1;
        root.upvalue=up;
        pt.add(root);
        while (!pt.isEmpty()){
            PackNode fatherNode=pt.poll();//取出 pt 节点中 ub 值最大的（即在优先队
列中的第一个）
            if (fatherNode.level==n-1){
                if (fatherNode.value>maxValue){
                    maxValue=fatherNode.value;
                    for (int i=n-1;i>=0;i--){
                        x[i]=fatherNode.left;
                        fatherNode=fatherNode.father;
                    }
                }
            }
            else {
                if (w[fatherNode.level+1]+fatherNode.weight<=c){
                    PackNode newNode=new PackNode();
                    newNode.level=fatherNode.level+1;
                    newNode.value=fatherNode.value+v[fatherNode.level+1];
                    newNode.weight=fatherNode.weight+w[fatherNode.level+1];
                    newNode.upvalue=Bound(newNode);
                    newNode.father=fatherNode;

                    newNode.left=1;
                    if (newNode.upvalue>=down)
                        pt.add(newNode);
```

```
                }//向右搜索节点,
                if
(fatherNode.value+(c-fatherNode.weight)*(v[fatherNode.level+1]/w[fatherNode
                        .level+1])>=down){
                    PackNode newNode2=new PackNode();
                    newNode2.level=fatherNode.level+1;
                    newNode2.value=fatherNode.value;
                    newNode2.weight=fatherNode.weight;
                    newNode2.father=fatherNode;

newNode2.upvalue=fatherNode.value+(c-fatherNode.weight)*(v[fatherNode.level+1]/
w[fatherNode
                                .level+1]);
                    newNode2.left=0;
                    pt.add(newNode2);
                }
            }
        }
        System.out.println("分支限界求解最大价值为："+maxValue);
        System.out.println("取出的物品为：");
        for (int i=0;i<n;i++){
            if (x[i]==1)
                System.out.print("物品"+i+" ");
        }
    }
//左子树延伸
    private static double Bound(PackNode node) {
        double maxLeft=node.value;
        double lw=c-node.weight;
        int tempLevel=node.level;
        while (tempLevel<=n-1&&lw>w[tempLevel]){
            lw-=w[tempLevel];
            maxLeft+=v[tempLevel];
            tempLevel++;
        }
        //不能装时 用下一个物品的单位重量折算剩余空间
        if (tempLevel<=n-1){
            maxLeft+=v[tempLevel]/w[tempLevel]*lw;
        }
        return maxLeft;
    }
//按物品单位价值排序
    private static void sort() {
        double tw,tv;
```

```
        for (int i=0;i<n-1;i++){
            for (int j=i+1;j<n;j++){
                if (v[i]/w[i]<v[j]/w[j]){
                    tw=w[i];
                    tv=v[i];
                    w[i]=w[j];
                    v[i]=v[j];
                    w[j]=tw;
                    v[j]=tv;
                }
            }
        }
    }
//计算下界
    private static double calc_down() {
        double maxValue=0;
        double now_w=0;
        for (int i=0;i<n;i++) {
            if (now_w+w[i]<c){
                maxValue += v[i];
                now_w += w[i];
            }
        }
        return maxValue;
    }
//计算上界
    private static double calc_up() {
        return v[0]/w[0]*c;
    }
}
```

2. 任务分配问题的分支限界算法

（1） 问题分析

把 n 项任务分配给 n 个人，每个人完成每项任务的成本不同，任务分配问题要求分配总成本最小的最优分配方案。

（2） 设计方案

应用贪心法求得一个合理的上界 up，为了获得合理的下界，考虑每个人执行所有任务的最小代价之和，这个解可能并不是一个可行解（因为任务分配方案可能存在冲突），但给出了一个参考下界 down，则最优解一定是在[down,up]之间的某个值。一般情况下，假设当前已对人员 1~i 分配了任务，并且花费了成本 v，则限界函数可以定义为:

$$lb = v + \sum_{k=i+1}^{n} \text{第}k\text{行的最小值}$$

（3） 算法分析

① 根据限界函数计算评估函数的下界 down；采用贪心法得到上界 up；

② 将待处理结点表 PT 初始化为空；

③ for (i=1; i<=n; i++)

x[i]=0;

④ k=1; i=0; //为第 k 个人分配任务， i 为第 k-1 个人分配的任务

⑤ while (k>=1)

5.1 x[k]=1;

5.2 while (x[k]<=n)

5.2.1 如果人员 k 分配任务 x[k]不发生冲突， 则

5.2.1.1 根据限界函数计算评估函数值 lb;

5.2.1.2 若 lb<=up， 则将 i,<x[k], k>lb 存储在表 PT 中；

5.2.2 x[k]=x[k]+1;

5.3 如果 k= =n 且叶子结点的 lb 值在表 PT 中最小，

则输出该叶子结点对应的最优解；

5.4 否则， 如果 k= =n 且表 PT 中的叶子结点的 lb 值不是最小， 则

5.4.1 up=表 PT 中的叶子结点最小的 lb 值;

5.4.2 将表 PT 中超出评估函数界的结点删除；

5.5 i=表 PT 中 lb 最小的结点的 x[k]值；

5.6 k=表 PT 中 lb 最小的结点的 k 值； k++;

（4） 源代码

```
import java.util.PriorityQueue;

public class task_problem {
    private static int[][] task={{9,2,7,8},
                        {6,4,3,7},
                        {5,8,1,8},
                        {7,6,9,4}};
    private static int up;
    private static int down;
    private static int[] solu={0,0,0,0};
    private static int[] down_col={0,0,0,0};
    static class PTNode implements Comparable<PTNode>{
        int lb_value;
        int level;
        int job_num;
        PTNode(int lb_value,int level,int job_num,PTNode father){
            this.level=level;
            this.job_num=job_num;
            this.lb_value=lb_value;
            this.father=father;
        }
        PTNode father;
        @Override
```

```
        public int compareTo(PTNode o) {//队列总按 lb 值进行从大到小排列
            return Integer.compare(o.lb_value,this.lb_value);
        }
    }
    public static void main(String args[]){
        System.out.println("任务分配成本矩阵：");
        for (int i=0;i<4;i++){
            for (int j=0;j<4;j++){
                System.out.print(task[i][j]+" ");
            }
            System.out.println();
        }
        down=calc_down();//计算假想下界
        up=calc_up();   //贪心法计算上界
        brand_bound();  //分支限界法
    }

    private static void brand_bound() {
        int max_lb=0;
        PriorityQueue<PTNode> pt=new PriorityQueue<>();//建立优先队列 pt
        PTNode root=new PTNode(down,-1,-1,null);
        pt.add(root);
        int [] col_flag={0,0,0,0};
        while (!pt.isEmpty()){
            PTNode fatherNode=pt.poll();//取出顶端节点
            if (pt.size()>=1){          //由于队列中存在多个 pt 节点，所以标志数
组应当重新赋值
                PTNode                                                  fn1=new
PTNode(fatherNode.lb_value,fatherNode.level,fatherNode.job_num,fatherNode.fathe
r);
                for(int a=0;a<4;a++){
                    col_flag[a]=0;
                }
                while (fn1.level>=0){
                    col_flag[fn1.job_num]=1;
                    fn1=fn1.father;
                }
            }
            if (fatherNode.level==3){   //分配到最后一个物品
                if (fatherNode.lb_value>max_lb){
                    max_lb=fatherNode.lb_value;
                    for (int i=fatherNode.level;i>=0;i--){
                        solu[i]=fatherNode.job_num+1;
                        fatherNode=fatherNode.father;
```

```
                    }
                }
            }
            else {
                int k=fatherNode.level+2;//k 用于定位成本矩阵第 k 行的最小值
                for (int j=0;j<4;j++){
                    if (col_flag[j]==0){ //判断任务是否被分配
                        int t_lb=calc_lb(k,fatherNode.level+1,j,fatherNode);
                        if (t_lb<=up&&t_lb>=down){    //在区间内则生成新节点
                            PTNode                                          newnode=new
PTNode(t_lb,fatherNode.level+1,j,fatherNode);
                            col_flag[j]=1;
                            pt.add(newnode);
                        }
                    }
                }
            }
        }
        System.out.println("任务分配情况如下：");
        for (int i=0;i<4;i++){
            System.out.print((i+1)+": 任务"+solu[i]+"  ");
        }
        System.out.println();
        System.out.print("最优分配成本："+max_lb);
    }
    //计算 lb 值
    private static int calc_lb(int k,int i,int j,PTNode fathernode) {
        int _lb=task[i][j];
        PTNode                                                          _fn=new
PTNode(fathernode.lb_value,fathernode.level,fathernode.job_num,fathernode.fathe
r);
        while (_fn.level>=0){//向上遍历获得父节点的任务成本值
            _lb+=task[_fn.level][_fn.job_num];
            _fn=_fn.father;
        }
        for (;k<4;k++){
            _lb+=down_col[k];//加上第 k 行的最小值
        }
        return _lb;
    }
    //计算上界
    private static int calc_up() {
        int up_v=0,temp;
        int[] flag={0,0,0,0};
```

```
        for (int i=0;i<4;i++){
            temp=task[i][0];
            for (int j=0;j<4;j++){
                if (task[i][j]<temp){
                    if (flag[j]==0){
                        temp=task[i][j];
                        flag[j]=1;
                    }
                }
            }
            up_v+=temp;
        }
        return up_v;
    }
    //计算下届
    private static int calc_down() {
        int temp,down_v=0;
        for (int i=0;i<4;i++){
            temp=task[i][0];
            for (int j=0;j<4;j++){
                if (task[i][j]<temp){
                    temp=task[i][j];
                }
            }
            down_v+=temp;
            down_col[i]=temp;
        }
        return down_v;
    }
}
```

3. 调试心得

经过此次实验加强了我对 java 优先队列的数据结构的掌握，并对分支限界法的大概思路有了一定的认识，分支限界的限界函数剪枝的效果大大优化了算法的复杂度，搜索空间大大降低。调试过程中出现了很多小毛病，数组越界的问题还是存在，搜索树中父节点依次传值时也出现了点问题，细心一点就不会出错，还有对于限界函数值的计算也是需要注意的。