实验 4. 分别基于回溯和分支限界求解 01 背包问题

实验内容

本实验要求基于算法设计与分析的一般过程（即待求解问题的描述、算法设计、算法描述、算法正确性证明、算法分析、算法实现与测试），在针对 0-1 背包问题求解的实践中理解回溯和分支限界方法的思想、求解策略及步骤。

实验目的

- 理解贪心方法的核心思想以及贪心方法的求解过程；
- 从算法分析与设计的角度，对 01 背包问题分别用回溯和分支限界方法求解

环境要求

对于环境没有特别要求。对于算法实现，可以自由选择 C, C++, Java，甚至于其他程序设计语言。

实验步骤

步骤 1：理解问题，给出问题的描述。

1. 问题描述

N 个物品，一个容量 m 的背包，每个物品都有不同的价值与重量，求解怎样放置物品使得背包中所乘物品价值最大。

2. 问题理解

此题是典型的 0-1 背包问题，每个物品都有不被选择与选择两种状态，即 0-1，前面实习二中解决此问题采用动态规划方法求解。通过新一轮的学习，此类问题可以通过回溯法以及分支界限法求解

步骤 2：算法设计，包括策略与数据结构的选择

（一）、回溯法求解 01 背包问题

1. 约束条件

$$\begin{cases} \sum_{i=1}^{n} w_i x_i \le W \\ x_i \in \{0,1\}, (1 \le i \le n) \end{cases}$$

在选择物品过程中，存在边界约束即装入物品背包的重量不超过背包的容量，有限界约束即装入物品背包的价值最大。这两个约束都是隐式约束，本问题没有显示约束，

因此在算法设计的时候只需考虑两个约束条件，针对两个约束条件对解空剪枝即可。

2. 目标函数

$$\max\sum_{i=1}^{n} v_i x_i$$

3. 数据结构

一维数组 ord 存放解向量，元素值为 0 表示此物品没有被选择，元素值为 1 表示此物品被选择。

一维数组 v 存放物品价值

一维数组 w 存放物品重量

W：背包容量

cp:当前背包价值

cw:当前背包已容纳重量

bestp:当前已经计算得最优价值

4. 算法设计

通过上述分析，设计回溯与分支界限法求解本问题。

约束条件为 cw+w[i]<=W,当下一个物品可以放进背包时才进入下一层。

界限条件为 cp+rp>bestp，当回溯到结点 i 时，当且仅当 i 后的物品全部放入背包后所得价值大于 bestp 当前最优价值才会继续进入解空间下一层递归，否则剪枝。

综上，此算法策略是能进则进，不进则换，不换则退。

（二）、分支界限法求解 01 背包问题：

1. 数据结构

```
public static Queue<Qnode> queue = new LinkedList<>();//结点队列
int cw=0;//存储当前背包重量
int cp=0;//存储当前背包价值
int up=0;//存储此节点物品之后背包可以存储的最大价值即价值上界
int lev=1;//广搜树的层数（物品序号）
int bestp=0;//当前搜索到的最大价值
Qnode cnode = new Qnode();//正在扩展的节点
Vector<Boolean> s_path = new Vector<>();//结点路径（解向量）
int[] w;//重量数组
int[] p;//价值数组
```

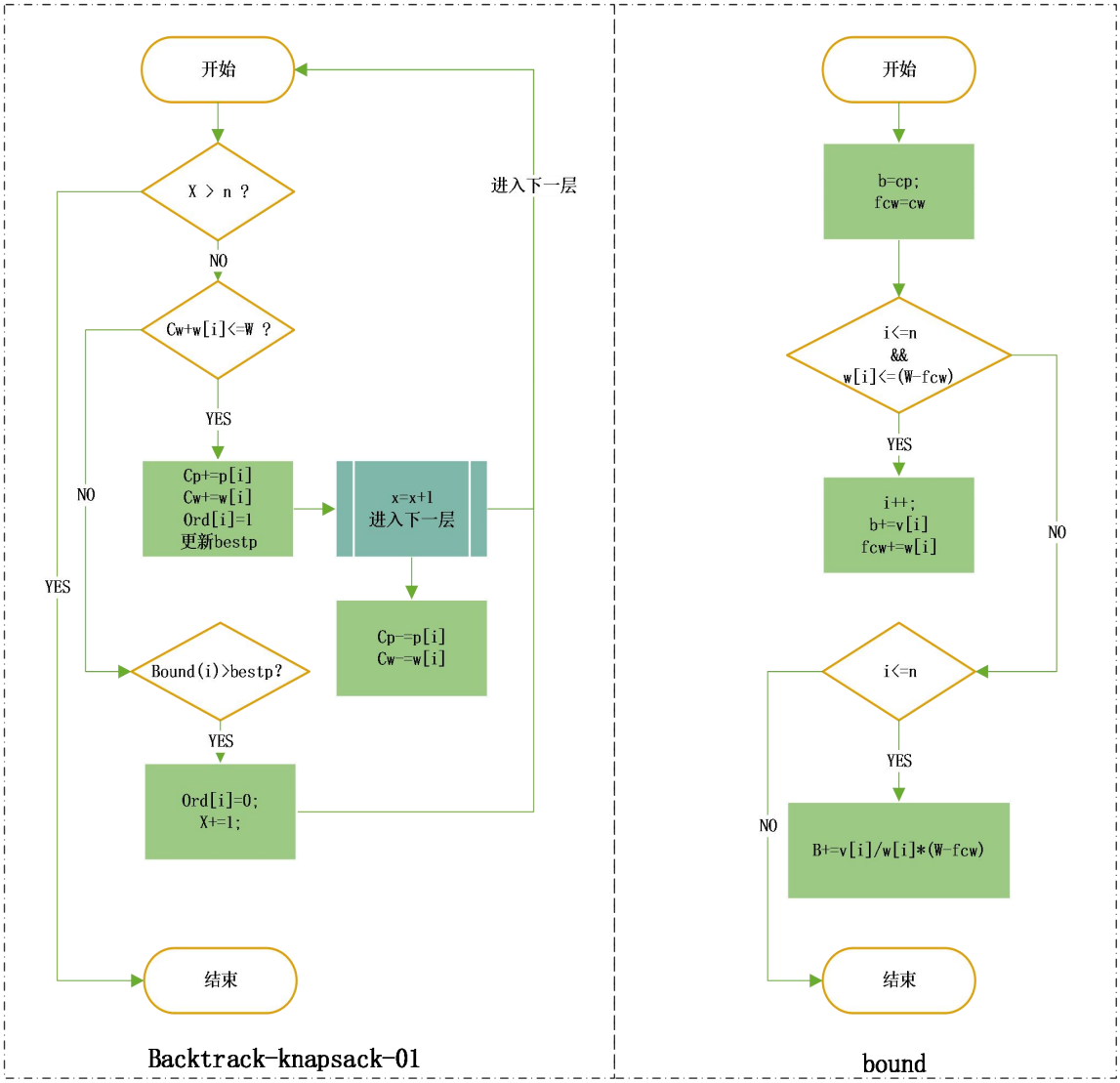
2. 算法设计

以广度优先的方式搜索解空间树，从当前节点开始，一次性扩展所有节点，在 01 背包问题中，对于一个结点只有放和不放两种情况，所以解空间树每个结点最多有两个度。将每个结点扩展之后，将满足约束条件和限界条件（同回溯法）的放入活结点队列 queue 的队尾。

从活结点队列队首中依次取出结点，扩展并更新结点对象的路径属性 s_path，更新最大价值 bestp，直到找到第一个解，算法结束。

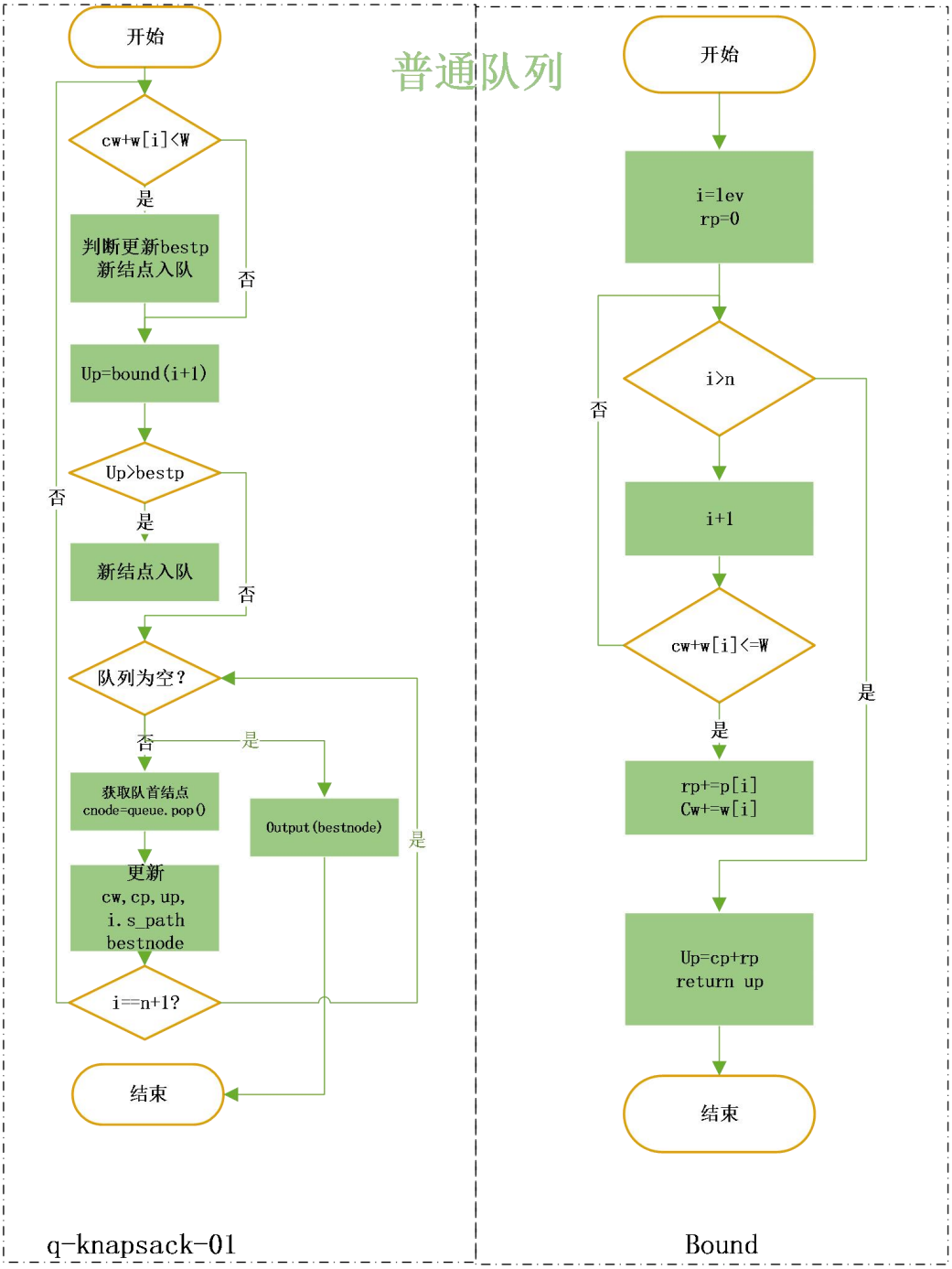
步骤 3：描述算法。希望采用源代码以外的形式，如伪代码或流程图等；

（一）、回溯法算法流程图

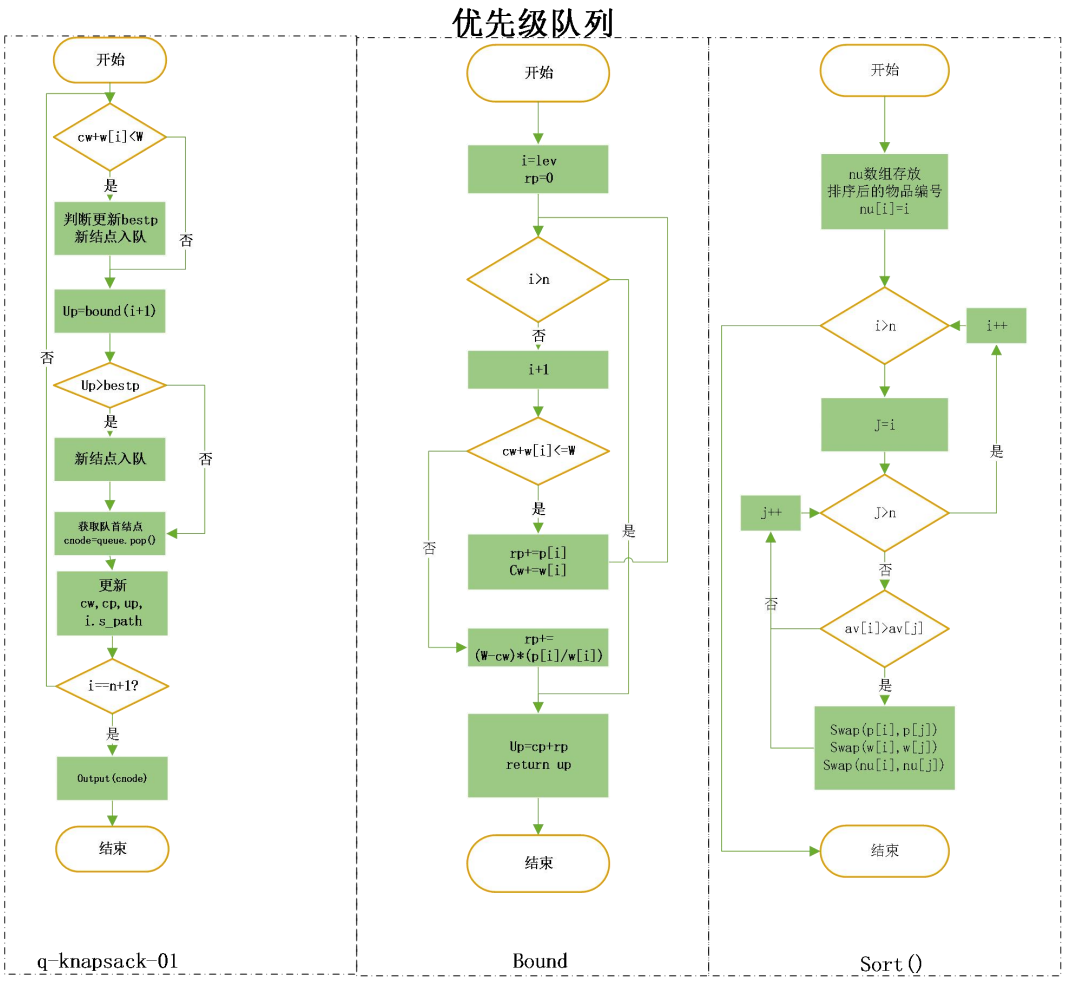

开始
X > n ?
NO
YES
进入下一层
Cw+w[i]<=W ?
NO
YES
Cp+=p[i]
Cw+=w[i]
Ord[i]=1
更新bestp
x=x+1
进入下一层
Cp-=p[i]
Cw-=w[i]
Bound(i)>bestp?
YES
Ord[i]=0;
X+=1;
结束
Backtrack-knapsack-01
开始
b=cp;
fcw=cw
i<=n
&&
w[i]<=(W-fcw)
YES
NO
i++;
b+=v[i]
fcw+=w[i]
i<=n
YES
NO
B+=v[i]/w[i]*(W-fcw)
结束
bound

（二）、分支界限法算法流程图

普通队列：



优先级队列：



步骤 4：算法的正确性证明。需要这个环节，在理解的基础上对算法的正确性给予证明；

（一）回溯法

考虑多米诺性质进行证明：

在 0-1 背包问题中，解空间为

(x1,x2,...,xn)(x1,x2,...,xn)

如果当前结果

P1=(x1,x2,...,xn)

是最优解，那么

P2=(x1,x2,...,xn−1)

也就是减少一个物品但不改变背包容量的时候，可以想到 P2 依然是该问题的最优解。从子集树角度来看，也就是最后一层结点全部去掉后的结果，那么当前结果也是最优的

（二）分支限界法

在算法循环中不断扩展结点，直到子集树的一个叶子结点成为扩展结点时停止。此时队列中的所有活结点的价值上界都不超过该叶子结点的价值。因此，该叶子结点相应的解为问题的最优解。

步骤 5：算法复杂性分析，包括时间复杂性和空间复杂性；

（一）回溯法

时间复杂性：O(n!)

空间复杂性：O(n)

（二）分支限界法

时间复杂性：$O(n*2^n)$

空间复杂性：$O(2^n)$

实验结果

步骤 6：算法实现与测试。附上代码或以附件的形式提交，同时贴上算法运行结果截图；

（一）回溯法

1. cw+rw<bestp 剪枝时

```
package huisu;
import java.util.Scanner;
public class bagof01 {
    static public int W;//背包容量
    static public int n;//物品数目
    static public int[] v;//价值列表
    static public int[] w;//重量列表
    static public int[] ord;//选择列表
    static public int cp;//当前价值
    static public int cw;//当前重量
    static public int bestp;//当前最优价值
    static int bound(int x)
    {
        int b=cp;
```

```
    int cc=cw;
    while(x<=n && w[x]<=(W-cc))
    {
        b+=v[x];
        cc+=w[x];
        x++;
    }
    if(x<=n)
    {
        b+=v[x]/w[x]*(W-cc);
    }
    return b;
}
static void cal(int x)//回溯递归函数
{
    if(x>n) {
        System.out.print("解向量为: ");
        for(int i=1;i<=n;++i)
        {
            System.out.print(ord[i]+" ");
        }
        System.out.println();
        System.out.println("此价值为"+cp);
        return;
    }
    if(cw+w[x]<=W)
    {
        cw+=w[x];
        cp+=v[x];
        ord[x]=1;
        bestp= bestp>cp? bestp:cp;
        cal(x+1);//进入下一层
        cw-=w[x];
        cp-=v[x];

    }
    /*
     * 递归到cal(x+1)回来的时候
     * 要判断是否递归右子树
     * 首先
     * cw-=w[x];
       cp-=v[x];
       清除痕迹
       然后剪枝判断
```

```
            if(bound(x)>bestp)
            有递归的必要
            ord[x]=0;
            则不选 x 个物品
            然后
            cal(x+1);
            进入下一层递归
            如果没有递归的必要
            此函数运行完毕，回退到上一级
         */
        if(bound(x)>bestp)//递归右子树
        {
            ord[x]=0;//不选 x 个物品
            cal(x+1);//进入下一层
        }
    }
    public static void main(String[] args) {

        // TODO Auto-generated method stub
        bestp=0;
        cp=0;
        cw=0;
        Scanner scan = new Scanner(System.in);
        W = scan.nextInt();
        n= scan.nextInt();
        v=new int[n+1];
        w=new int[n+1];
        ord=new int[n+1];
        ord[0]=0;
        for(int i=1;i<=n;++i)
        {
            ord[i]=0;
            v[i]=scan.nextInt();//输入价值
        }
        for(int i=1;i<=n;++i)
        {

            w[i]=scan.nextInt();//输入重量
        }
        scan.close();

        cal(1);
        System.out.println("\n 综上，最大价值为"+bestp);

    }
```

运行结果

```
mize
    }
    public static void main(String[] args) {

        // TODO Auto-generated method stub
        bestp=0;
        cp=0;
        cw=0;
        Scanner scan = new Scanner(System.in);
        W = scan.nextInt();
        n= scan.nextInt();
        v=new int[n+1];
        w=new int[n+1];
        ord=new int[n+1];
        ord[0]=0;
        for(int i=1;i<=n;++i)
        {
            ord[i]=0;
            v[i]=scan.nextInt();//输入价值
        }
        for(int i=1;i<=n;++i)
        {

            w[i]=scan.nextInt();//输入重量
        }
        scan.close();

```

Console ×

<terminated> bagof01 [Java Application] E:\app\bin\javaw.exe (2022年4月17日 下午5:40

```
7 4
 9 10 7 4
3 5 2 1
解向量为: 1 0 1 1
此价值为20

综上，最大价值为20
```

}

2. cw+brw<bestp 剪枝时

```
package huisu;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import java.util.Scanner;
class obj implements Comparable<obj>
{
    int v;
    int w;
    double aver;
    int no;//物品编号
    public obj()
    {

    }
    public obj(int vv,int ww,int n)
    {
        v=vv;
        w=ww;
        aver=v/ww;
        no=n;
    }
    @Override
    public int compareTo(obj o) {
        // TODO Auto-generated method stub
        if(this.aver>o.aver) return -1;
        else return 1;
    }

}
public class youhua {
    static public int n;
    static public int W;
    static public int cp;//当前价值
    static public int cw;//当前重量
    static public int[] ord;//选择列表
    static public int bestp;//当前最优价值
    static public List<obj> la = new LinkedList<>();
```

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    Scanner scan = new Scanner(System.in);
    n = scan.nextInt();
    W= scan.nextInt();
    ord = new int[n];

    for(int i=0;i<n;++i)
    {
        int vv=scan.nextInt();
        int ww=scan.nextInt();
        la.add(new obj(vv,ww,i));
    }
    scan.close();
//输入数据
    Collections.sort(la);
    cal(0);
    System.out.println("\n综上，最大价值为"+bestp);
}
static public double bound(int x)
{
    double b=cp;
    int w = cw;
    for(int i=x+1;i<n;++i)
    {
        if(la.get(i).w+w<W)
        {
            w+=la.get(i).w;
            b+=la.get(i).v;
        }
        else
        {
            b+=la.get(i).v/la.get(i).w*(W-w);
            return b;
        }
    }
    return b;
}

static public void cal(int x)
{

    if(x>n-1)
```

```
        {
            System.out.print("解向量为: ");
            for(int i=0;i<n;++i)
            {
                System.out.print(ord[i]+" ");
            }
            System.out.println("此时背包的价值为: "+cp);
            return;
        }
        if(cw+la.get(x).w<=W)
        {
            cw+=la.get(x).w;
            cp+=la.get(x).v;
            ord[la.get(x).no]=1;

            bestp = bestp>cp? bestp:cp;

            cal(x+1);
            cw-=la.get(x).w;
            cp-=la.get(x).v;
        }
        if(bound(x)>bestp)
        {
            ord[la.get(x).no]=0;
            cal(x+1);

        }
    }
```

运行结果

```
26          // TODO Auto-generated method stub
27          if(this.aver>o.aver) return -1;
28          else return 1;
29      }
30
31  }
32  public class test {
33      public static void main(String[] args) {
34          List<oboj> a = new LinkedList<>();
35          for(int i=0;i<5;++i)
36          {
37              a.add(new oboj(i,1,i));
38          }
39          for(int i=0;i<5;++i)
40          {
41              System.out.println(a.get(i).aver);
42          }
43          Collections.sort(a);
44          for(int i=0;i<5;++i)
45          {
46              System.out.println(a.get(i).aver);
47          }
48      }
49  }
50
```

Console ×

<terminated> bagof01 [Java Application] E:\app\bin\javaw.exe (2022年4月17日 下午5:40:20 – 下午5:40:29)

```
7 4
 9 10 7 4
3 5 2 1
解向量为: 1 0 1 1
此价值为20

综上，最大价值为20
```

（二）分支界限法

1. 普通队列：

```
package ss;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;
import java.util.Vector;

class Qnode{
    public int cw;
```

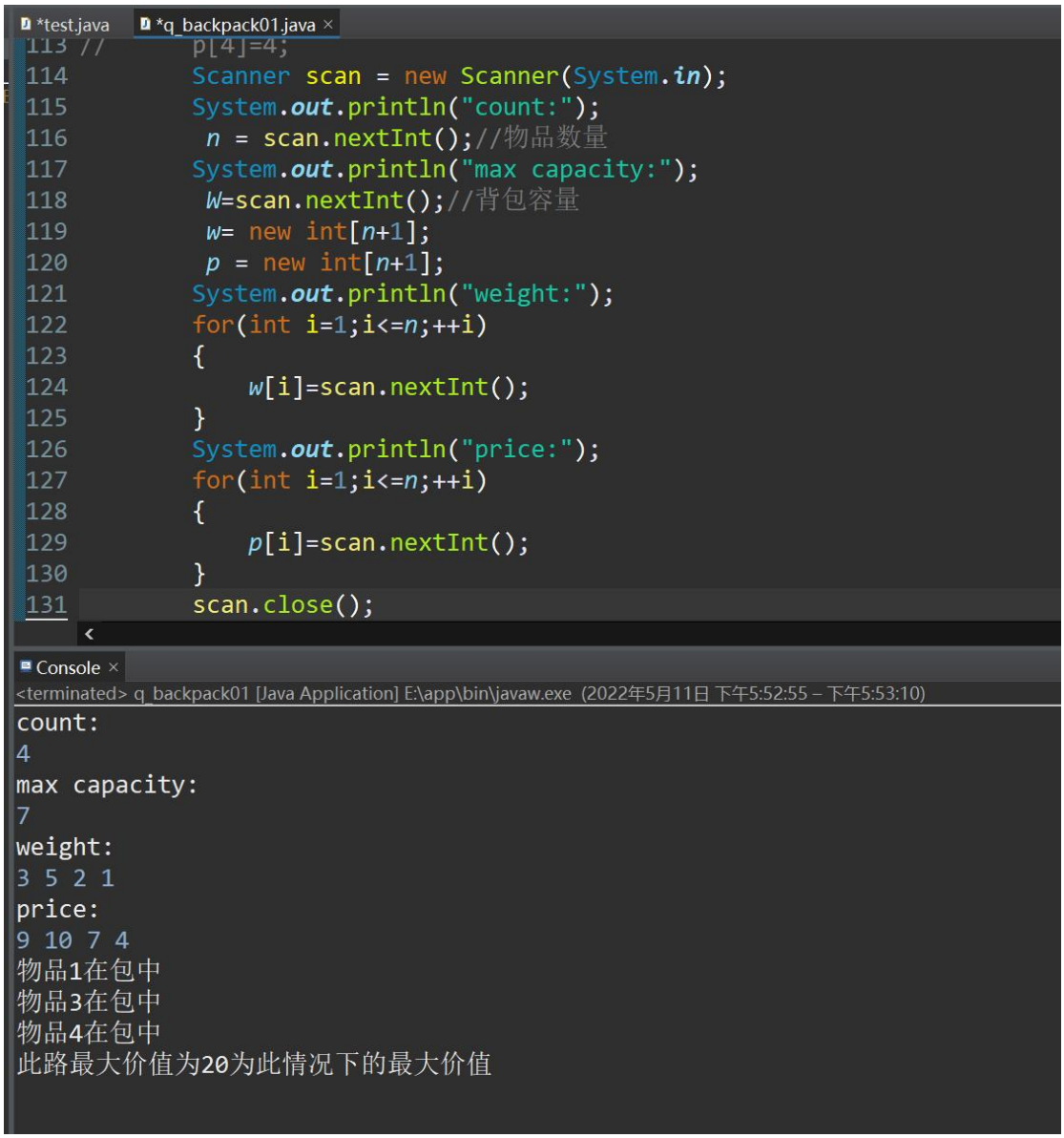
```
        public int cp;
        public int up;
        public int level;
        public Vector<Boolean> path;
        public Qnode() {};
        public Qnode(int w,int p,int u,Vector<Boolean> pv,Boolean
isleft,int le)
        {
            path=new Vector<>();
            for(int i=0;i<pv.size();++i)
            {
                path.add(pv.get(i));
            }
            path.add(isleft);
            cw=w;
            cp =p;
            up = u;
            level=le;
        }
    }
    public class q_backpack01 {
        public static int n;
        public static int W;
        public static int[] w;
        public static int[] p;
        public static Queue<Qnode> queue = new LinkedList<>();

        static int bound(int lev,int cw)//lev 表示当前所在的求解树的
层数，所以是从第 lev 个物品开始
        {
            int rp=0;
            for(int i=lev;i<=n;++i)
            {
                if(w[i]+cw<=W)
                {
                    rp+=p[i];
                    cw+=w[i];
                }
            }
            return rp;
        }
        static void output(Vector<Boolean> path)
        {
            for(int i=1;i<=path.size();++i)
```

```
            {
                if(path.get(i-1)==true)
                System.out.println("物品"+i+"在包中");
            }
        }

        static void cal()
        {
            int cw=0;
            int cp=0;
            int up=0;
            int lev=1;
            int bestp=0;
            Qnode cnode = new Qnode();
            Qnode bnode = new Qnode();//最优结点
            Vector<Boolean> s_path = new Vector<>();

            here:
            while(true)
            {
            //System.out.println("lev"+lev);
            if(w[lev]+cw<=W)
            {
                Qnode nnode = new Qnode(cw+w[lev], cp+p[lev], up,
s_path, true,lev+1);
                queue.add(nnode);
                 if(bestp<cp+p[lev])
                   {
                    bestp = cp+p[lev];
                    bnode = nnode;
                   }
    //          System.out.println("bestp"+bestp);
    //          System.out.println("cp"+cp);
    //          System.out.println("p[lev]"+p[lev]);
            }
            up=cp+bound(lev, cw);
            //System.out.println("up"+up);
            if(up>bestp)
            {
                queue.add(new    Qnode(cw,    cp,    up,    s_path,
false,lev+1));
            }

                while(true)
```

```
            {
                if(queue.isEmpty())
                    break here;//队空才结束
                cnode = queue.remove();
                if(cnode.level==n+1)
                {
                    if(cnode.cp>bestp)
                    {
                        bnode=cnode;
                    }
                    continue;
                }
                else
                {
                 cw=cnode.cw;
                 cp=cnode.cp;
                 up=cnode.up;
                 lev=cnode.level;
                 s_path = cnode.path;
                 break;
                }
            }


        }

         output(bnode.path);
         System.out.println("此路最大价值为"+bestp+"为此情况下
的最大价值");

    }

    public static void main(String[] args) {
        n=4;
        W=7;
        w=new int[5];
        p = new int[5];
        w[1]=3;
        w[2]=5;
        w[3]=2;
        w[4]=1;
        p[1]=9;
        p[2]=10;
        p[3]=7;
```

```
        p[4]=4;
//      Scanner scan = new Scanner(System.in);
//      System.out.println("count:");
//       n = scan.nextInt();//物品数量
//      System.out.println("max capacity:");
//       W=scan.nextInt();//背包容量
//       w= new int[n+1];
//       p = new int[n+1];
//      System.out.println("weight:");
//      for(int i=1;i<=n;++i)
//      {
//         w[i]=scan.nextInt();
//      }
//      System.out.println("price:");
//      for(int i=1;i<=n;++i)
//      {
//         p[i]=scan.nextInt();
//      }
//      scan.close();
        cal();


    }
}
```

运行截图：



2. 优先级队列

```
package ss;
//优先级队列版
import java.util.LinkedList;
import java.util.PriorityQueue;
import java.util.Queue;
import java.util.Scanner;
import java.util.Vector;

class Qnode{
    public int cw;
    public int cp;
    public int up;
    public int level;
```

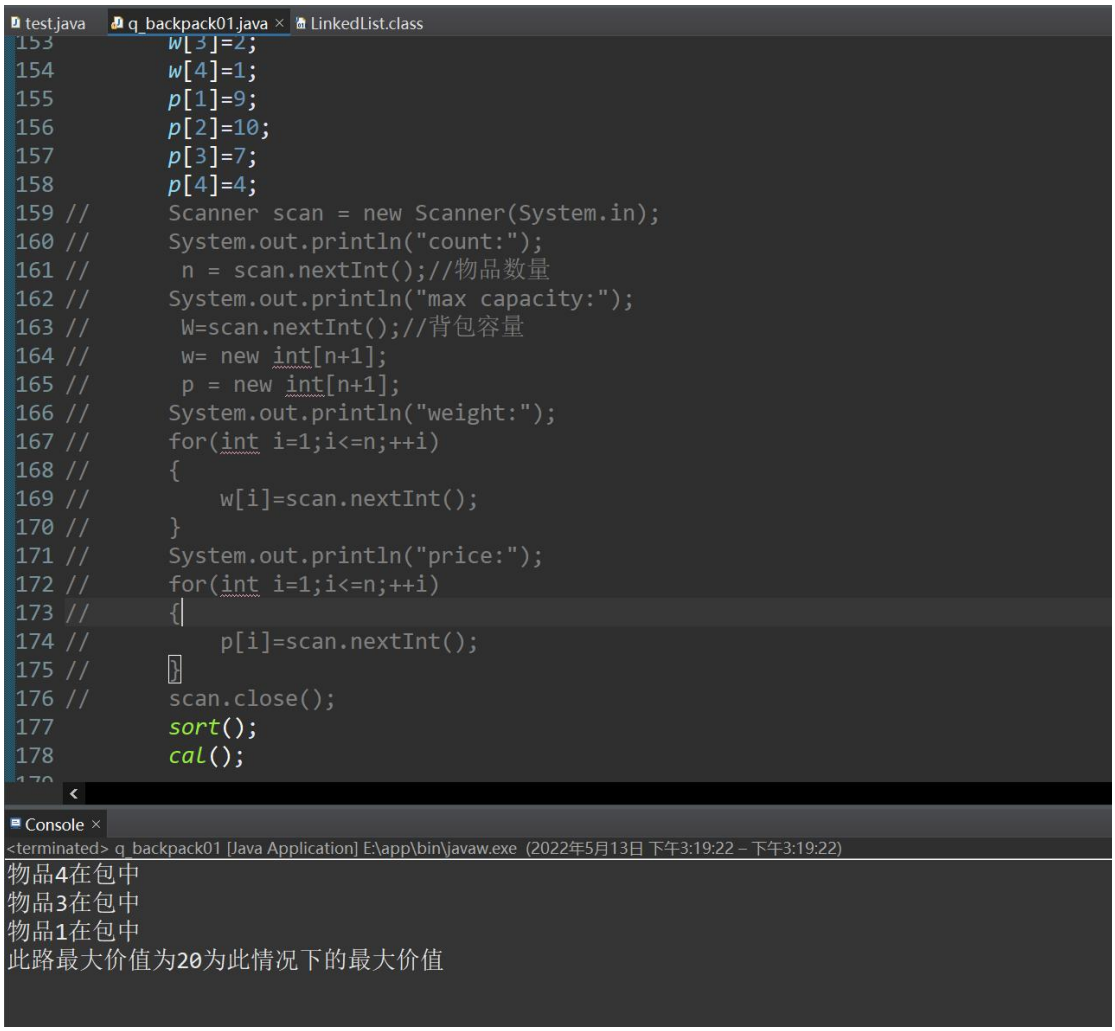
```
    public Vector<Boolean> path;
    public Qnode() {};
    public Qnode(int w,int p,int u,Vector<Boolean> pv,Boolean
isleft,int le)
    {
        path=new Vector<>();
        for(int i=0;i<pv.size();++i)
        {
            path.add(pv.get(i));
        }
        path.add(isleft);
        cw=w;
        cp =p;
        up = u;
        level=le;
    }
}
public class q_backpack01 {
    public static int n;
    public static int W;
    public static int[] w;
    public static int[] p;
    public static int[] nu;//正确顺序
    public static Queue<Qnode> queue = new PriorityQueue<>((a,b)->
    {
        return a.up-b.up;
    });

    static void sort()
    {
        nu = new int[n+1];
        for(int i=1;i<=n;++i)
        {
            nu[i]=i;
        }
        for(int i=1;i<=n;++i)
        {
            double x = (double) w[i]/p[i];
            for(int j=i;j<=n;++j)
            {
                double y = (double) w[j]/p[j];
                if(x>y)
                {
                    int t=nu[i];
```

```
                nu[i]=nu[j];
                nu[j]=t;

                t=p[i];
                p[i]=p[j];
                p[j]=t;

                t=w[i];
                w[i]=w[j];
                w[j]=t;

            }
        }
    }

}
    static int bound int lev,int cw //lev 表示当前所在的求解树的层数，所以是从第 lev 个物品开始
    {
        int rp=0;

        for(int i=lev;i<=n;++i)
        {
            if(w[i]+cw<=W)
            {
                rp+=p[i];
                cw+=w[i];
            }
            else
            {
                rp+=(W-cw)*(p[i]/w[i]);
                break;
            }

        }
        return rp;
    }
    static void output(Vector<Boolean> path)
    {
        for(int i=1;i<=path.size();++i)
        {
            if(path.get(i-1)==true)
            System.out.println("物品"+nu[i]+"在包中");
        }
    }
```

```
    static void cal()
    {
        int cw=0;
        int cp=0;
        int up=0;
        int lev=1;
        int bestp=0;
        Qnode cnode = new Qnode();
        Vector<Boolean> s_path = new Vector<>();
        while true)
        {
        //System.out.println("lev"+lev);
        if(w[lev]+cw<=W)
        {
            queue.add(new Qnode(cw+w[lev], cp+p[lev], up, s_path,
true,lev+1));
            bestp = bestp>cp+p[lev]? bestp:cp+p[lev];
//          System.out.println("bestp"+bestp);
//          System.out.println("cp"+cp);
//          System.out.println("p[lev]"+p[lev]);
        }
        up=cp+bound(lev, cw);
        //System.out.println("up"+up);
        if(up>bestp)
        {
            queue.add(new Qnode(cw, cp, up, s_path, false,lev+1));
        }
        if(queue.isEmpty()) break;
        else
        {
            cnode = queue.remove();
             cw=cnode.cw;
             cp=cnode.cp;
             up=cnode.up;
             lev=cnode.level;
             s_path = cnode.path;
             if(lev==n+1)
             {
                 break;
             }
        }
        }
         output(cnode.path);
         System.out.println("此路最大价值为"+cp+"为此情况下的最大价
```

```
值");

    }

    public static void main(String[] args) {
        n=4;
        W=7;
        w=new int[5];
        p = new int[5];
        w[1]=3;
        w[2]=5;
        w[3]=2;
        w[4]=1;
        p[1]=9;
        p[2]=10;
        p[3]=7;
        p[4]=4;
//      Scanner scan = new Scanner(System.in);
//      System.out.println("count:");
//       n = scan.nextInt();//物品数量
//      System.out.println("max capacity:");
//       W=scan.nextInt();//背包容量
//       w= new int[n+1];
//       p = new int[n+1];
//      System.out.println("weight:");
//      for(int i=1;i<=n;++i)
//      {
//          w[i]=scan.nextInt();
//      }
//      System.out.println("price:");
//      for(int i=1;i<=n;++i)
//      {
//          p[i]=scan.nextInt();
//      }
//      scan.close();
        sort();
        cal();


    }
}
```

运行结果：



}
}

实验总结

通过本次实习，分别用回溯法和分支限界法实现了 01 背包问题，理解了回溯法和分支限界法的核心思想体会了回溯法和分支限界法的求解过程，将老师上课讲的知识付诸实际，感觉不错，受益匪浅。相比前次实习用动态规划求解 01 背包问题，回溯法和分支限界法的基本思想是搜索，回溯法是深度优先搜索，分支限界是广度优先搜索。相比动态规划，搜索类的算法对我来说却是有些难度，因此在实现算法的过程中遇到了很多问题，总结如下：

1. 回溯法求解问题时，在当前扩展结点处，搜索向纵深方向移至一个新结点。这个新结点成为新的活结点，并成为当前扩展结点。如果在当前扩展结点处不能再向纵深方向移动，则当前扩展结点就成为死结点。在用递归实现算法时，进入下一层，转入兄弟节点，返回上一层的顺序一定要正确，不然很容易陷入未知错误，导致算法可以正常运行，但是没有正确结果。

2. 在用分支限界法求解时，在当前节点处，先生成其所有的子节点，然后再从当前的活节点表中选择下一个扩展节点。为了有效地选择下一个扩展节点，加速搜索的进程，在每一个活节点处，计算一个函数值限界，并根据函数值，从当前活节点表中选择一个最有利的节点作为扩展节点，使搜索朝着解空间上有最优解的分支推进，以便尽快地找出一个最优解。在优先队列求解的时候，up=cp+r`p，所以找到的第一个节就是最优解，找到第一个解的时候就该结束循环了。但是普通队列找到的第一个解不一定是最优解，要等到队列为空才能结束循环。

最后感谢老师在实习时的指导与帮助！

软工 2001
2020012249
张宇晨