

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: 实验一: 蛮力法

项目完成人:

姓名: 刘禾子 学号: 2017526019

指导教师: 李朝晖

日 期: 2017 年 10 月 30 日

一、 实验目的

1. 深刻理解并掌握蛮力法的设计思想；
2. 提高应用蛮力法设计算法的技能。

二、 实验内容

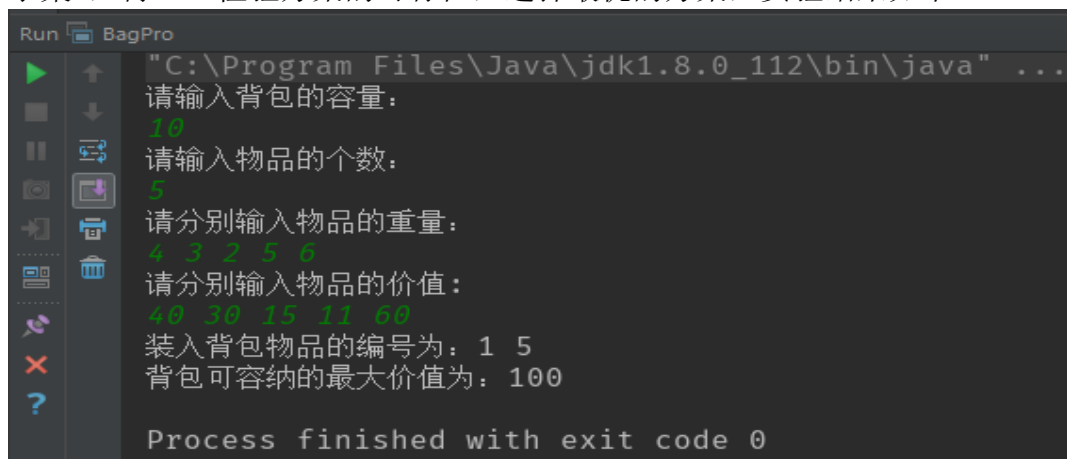
1. 用蛮力法解决 0/1 背包问题对所设计的算法进行时间复杂性分析；
2. 选做二：王伯买鱼。

三、 实验环境

IntelliJ IDEA Community Edition 2017.2.4

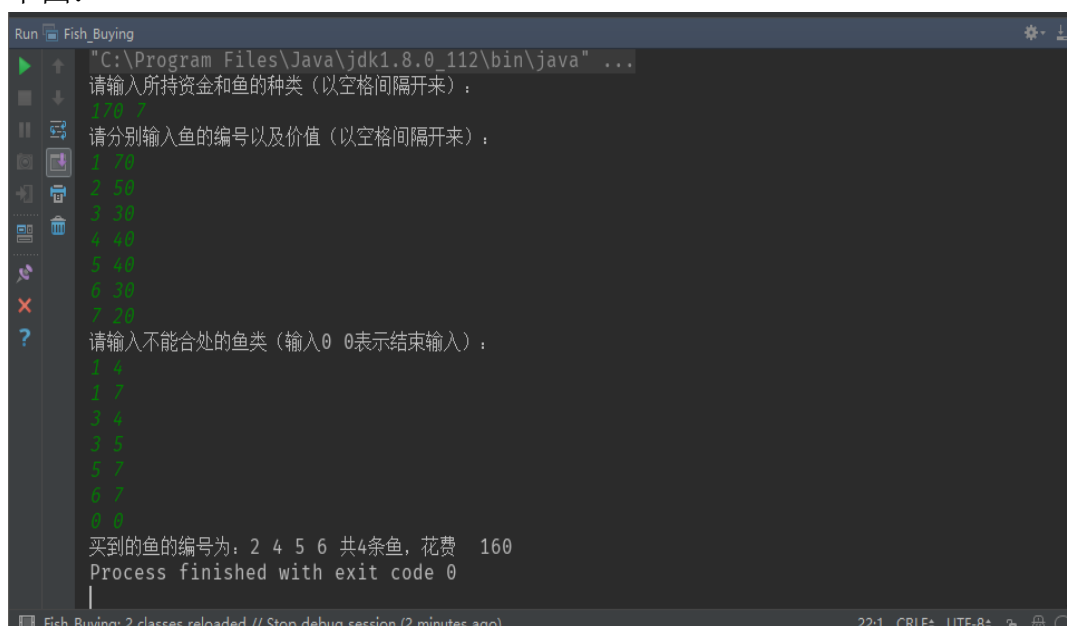
四、 实验结果

1. 蛮力法解决 0/1 背包问题，根据用户输入的参数，求出所有物品组合（即求子集），再一一检验方案的可行性，选择最优的方案，实验结果如下：



```
Run BagPro
"C:\Program Files\Java\jdk1.8.0_112\bin\java" ...
请输入背包的容量:
10
请输入物品的个数:
5
请分别输入物品的重量:
4 3 2 5 6
请分别输入物品的价值:
40 30 15 11 60
装入背包物品的编号为: 1 5
背包可容纳的最大价值为: 100
Process finished with exit code 0
```

2. 王伯买鱼与上题类似，穷尽所有可行方案求最优，不过这里增加了“有些鱼不能共处”的条件，在求子集的时候需要加上这个条件的判断，实验结果如下图：



```
Run Fish_Buying
"C:\Program Files\Java\jdk1.8.0_112\bin\java" ...
请输入所持资金和鱼的种类（以空格间隔开来）:
170 7
请分别输入鱼的编号以及价值（以空格间隔开来）:
1 70
2 50
3 30
4 40
5 40
6 30
7 20
请输入不能合处的鱼类（输入0 0表示结束输入）:
1 4
1 7
3 4
3 5
5 7
6 7
0 0
买到的鱼的编号为: 2 4 5 6 共4条鱼, 花费 160
Process finished with exit code 0
```

五、 附录

1. 蛮力法求 0/1 背包问题

(1) 问题分析

N 个物品，放与不放的组合总共就有 2^N 种，穷尽每种可能，若是超出背包容量的方案则不考虑，判断完毕所有可能后从可行方案中求出价值最高的方案输出。

(2) 设计方案

goods 类:int weight, int value 以及构造方法 goods(int weight,int value);

void subnet(int[][] s,int n):求子集方法，结果存放在 s 二维数组中;

void judge(int[][] s,goods[] goods,int[] mark,int n,int c):判断方案可行性，可行的方案在 mark[] 中标记该方案总价值;

int getmax(int[] mark,int n):求可行方案中最大价值的方案,返回最优价值;

void output(int flag,int[][] s,int n):输出可行方案的物品编号

(3) 算法

采用蛮力法，关键语句是判断所有方案可行性时下图所示的语句:

```
for (int i=0;i<Math.pow(2,n);i++){  
    v=0;w=0;  
    for (int j=0;j<n;j++){
```

时间复杂度 $O(n)=n*2^n$ ，可见其复杂度很大，不适合处理较多物品的 0/1 背包问题，求解过程如下图:

物品编号 方案编号 (w,v)	1 (4,40)	2 (3,30)	3 (2,15)	4 (5,11)	5 (6,60)	weight	value
0	0	0	0	0	0		
1	0	0	0	0	1	6	60
2	0	0	0	1	0	5	11
3	0	0	0	1	1	11	
4	0	0	1	0	0	2	15
5	0	0	1	0	1	8	75
6	0	0	1	1	0	7	26
7	0	0	1	1	1	13	
8	0	1	0	0	0	3	30
9	0	1	0	0	1	9	90
10	0	1	0	1	0	8	41
11	0	1	0	1	1	14	
12	0	1	1	0	0	5	45
13	0	1	1	0	1	11	
14	0	1	1	1	0	10	56
15	0	1	1	1	1	16	
16	1	0	0	0	0	4	40
17	1	0	0	0	1	10	100
18	1	0	0	1	0	9	51
19	1	0	0	1	1	15	
20	1	0	1	0	0	6	55
21	1	0	1	0	1	12	
22	1	0	1	1	0	11	
23	1	0	1	1	1	17	
24	1	1	0	0	0	7	70

(4) 源代码

```
import java.util.Scanner;
public class BagPro {
    static int flag=0;//标记可获得最大价值的可行方案下标
    public static class goods{
        private int weight;
        private int value;
        goods(int weight, int value){//构造方法
            this.weight=weight;
            this.value=value;
        }
    }
    //主函数
    public static void main(String args[]){
        int[][] s=new int[1024][10];//存放子集的二维数组，最大容量为10，
        物品组合有  $2^{10}$  种
        int[] mark=new int[1024];//用来标记可行的方案，可行，数组内容为当前方案价值，不可行则为0
        int n,max_v,c;//n为物品个数，max_v为最大总价值，c为背包容量，flag为可行方案在s数组中的下标
        //初始化对象数组
        goods[] goods= {new goods(0,0),new goods(0,0),new goods(0,0),new goods(0,0),new goods(0,0),
            new goods(0,0),new goods(0,0),new goods(0,0),new goods(0,0),new goods(0,0)};
        System.out.println("请输入背包的容量：");
        Scanner sc=new Scanner(System.in);
        c=sc.nextInt();
        System.out.println("请输入物品的个数：");
        n=sc.nextInt();
        System.out.println("请分别输入物品的重量：");
        for (int i=0;i<n;i++){
            goods[i].weight=sc.nextInt();
        }
        System.out.println("请分别输入物品的价值：");
        for (int i=0;i<n;i++){
            goods[i].value=sc.nextInt();
        }
        sc.close();
        subnet(s,n);
        judge(s, goods, mark, n, c);
        max_v=getmax(mark,n);
        output(flag,s,n);
        System.out.println("背包可容纳的最大价值为："+max_v);
    }
}
```

```

    }
//输出可行方案的装入物品编号
    private static void output(int flag, int[][] s, int n) {
        System.out.print("装入背包物品的编号为: ");
        for (int i=0;i<n;i++){
            if (s[flag][i]==1){
                System.out.print((i+1)+" ");
            }
        }
        System.out.println();
    }
//判断所有方案是否可行
    private static void judge(int[][] s, goods[] goods, int[] mark, int n,
int c) {
        int v,w;
        for (int i=0;i<Math.pow(2,n);i++){
            v=0;w=0;
            for (int j=0;j<n;j++){
                w+=goods[j].weight*s[i][j];
                v+=goods[j].value*s[i][j];
            }
            if (w<c)
                mark[i]=v;
            else
                mark[i]=0;
        }
    }
//求得物品放入的所有组合
    private static void subnet(int[][] s, int n) {
        int k, m;
        for (int i = 0; i < Math.pow(2, n); i++) {
            k = i;
            for (int j = n - 1; j >= 0; j--) {
                //if((s[i][0]&& s[i][3]) || (s[i][2]&& s[i][3]) || (s[i][4]&& s[i][6]) || (s[i][0]
                //&& s[i][6]) || (s[i][2]&& s[i][4]) || (s[i][5]&& s[i][6])) {
                // break;}
                m = k % 2;
                s[i][j] = m;
                k = k / 2;
            }
        }
    }
//求可行方案中的最优价值

```

```

private static int getmax(int[] mark,int n){
    int max=0;
    for (int i=0;i<Math.pow(2,n);i++){
        if (mark[i]>max){
            max=mark[i];
            flag=i;
        }
    }
    return max;
}
}

```

2. 王伯买鱼

(1) 问题分析

与上题类似，根据用户输入的条件，求子集穷尽所有可能，将所有鱼类放入生成的 fish 类对象数组中，再用一个 $c \times 2$ 的数组（c 为常数用户输入 0，0 之后确定下来）not_together[][2] 存储所有不能共处鱼类的组合，相似地，从所有符合要求的组合中求出花费最多的一种方案。

(2) 设计方案

fish 类:int num,int value, 以及构造方法 fish(int num,int value);
void subnet(int[][] s,int n,int[][] not_together,int c):求子集
void judge(int[][] s,int n,int[] mark,fish[] fishs,int money):
盘但方案的可行性，将可行方案所花费的金额存入 mark[] 中;
int getmax(int[] mark,int n):获取最优方案;
void output(int[][] s,int flag,int n,fish[] fishs):输出最优方案
所有所购买的鱼类。

(3) 算法

使用蛮力法，遍历所有可能，关键语句在求子集方法中，在求完子集之后再一一从 not_together[][2] 数组中取出相互冲突的鱼类编号判断当前方案是否存在这两种鱼存在，若存在则把该方案 s[i][j] 全部清零（即该方案不可行）如下图所示：

```

for (int i=0;i<Math.pow(2,n);i++){
    k=i;
    for (int j=n-1;j>=0;j--){
        m=k%2;
        s[i][j]=m;
        k=k/2;
    }
    for (int z=0;z<c;z++){
        if (s[i][not_together[z][0]-1]==1&&s[i][not_together[z][1]-1]==1){
            for (int j=0;j<n;j++){
                s[i][j]=0;
            }
        }
    }
}

```

时间复杂度 $(n+c) \times 2^n$, $O(n) = (n+c) \times 2^n$.

```
import java.util.Scanner;
public class Fish_Buying {
    static int flag=0;
    public static class fish{
        private int num;//鱼的编号
        private int value;//鱼的价值
        fish(int num,int value){
            this.value=value;
            this.num=num;
        }
    }

    public static void main(String args[]){
        int[][] s=new int[1024][10]; //存放子集，最大容量 10 共有 1024 种组合
        int [] mark=new int[1024]; //标记可行的方案
        int [][] not_together=new int[45][2]; //两两不能共存的鱼类的集合
        int n,max_value,money; //n为总共有多少种类的鱼，max_value为最大价值总和，money为所持有资金
        int a=1,b=1,c=0; //临时变量，ab用于判断输入不能相处鱼类的结束状态，c是not_together数组下标
        fish fishes[]={
                new fish(0,0),new fish(0,0),new fish(0,0),new fish(0,0),new fish(0,0),
                new fish(0,0),new fish(0,0),new fish(0,0),new fish(0,0),new fish(0,0),
                new fish(0,0),new fish(0,0)
        };
        System.out.println("请输入所持资金和鱼的种类（以空格间隔开来）：");
        Scanner input=new Scanner(System.in);
        money=input.nextInt();
        n=input.nextInt();
        System.out.println("请分别输入鱼的编号以及价值（以空格间隔开来）：");

        for (int i=0;i<n;i++){
            fishes[i].num=input.nextInt();
            fishes[i].value=input.nextInt();
        }
        System.out.println("请输入不能合处的鱼类（输入 0 0 表示结束输入）：");

        a=input.nextInt(); b=input.nextInt();
        while (a!=0&&b!=0){
            not_together[c][0]=a;
            not_together[c][1]=b;
```

```

        c++;
        a=input.nextInt();b=input.nextInt();
    }
    input.close();
    subnet(s,n,not_together,c);
    judge(s,n,mark,fishs,money);
    max_value=getmax(mark,n);
    output(s,flag,n,fishs);
    System.out.print("花费 "+max_value);
}

private static void output(int[][] s, int flag, int n,fish[] fishs) {
    System.out.print("买到的鱼的编号为: ");
    int count=0;
    for (int i=0;i<n;i++){
        if(s[flag][i]==1){
            count++;
            System.out.print(fishs[i].num+" ");
        }
    }
    System.out.print("共"+count+"条鱼, ");
}

private static int getmax(int[] mark, int n) {
    int max=0;
    for (int i=0;i<Math.pow(2,n);i++){
        if (mark[i]>max){
            max=mark[i];
            flag=i;
        }
    }
    return max;
}

private static void judge(int[][] s, int n, int[] mark, fish[] fishs,
int money) {
    int cost;
    for (int i=0;i<Math.pow(2,n);i++){
        cost=0;
        for (int j=0;j<n;j++){
            cost+=fishs[j].value*s[i][j];
        }
        if (cost<money)
            mark[i]=cost;
        else
            mark[i]=0;
    }
}

```



```

    }
    private static void subnet(int[][] s, int n, int[][] not_together, int
c) {
        int k, m;
        for (int i=0; i<Math.pow(2, n); i++) {
            k=i;
            for (int j=n-1; j>=0; j--) {
                m=k%2;
                s[i][j]=m;
                k=k/2;
            }
            for (int z=0; z<c; z++) {
                if
(s[i][not_together[z][0]-1]==1&&s[i][not_together[z][1]-1]==1) {
                    for (int j=0; j<n; j++) {
                        s[i][j]=0;
                    }
                }
            }
        }
    }
}

```

3. 调试心得

了解了大概的算法思路后，对蛮力法有了深刻的认识，加上编程的练习更是加强了自己的调试能力，调试过程中出现了很多次数组下标越界的问题，都是粗心导致，修改下标后能够正确执行，还有循环次数的条件也有几次输错>与>=这两种情况没有考虑清楚，鱼的编号从 1 开始的所以注意从 not_together 数组中取出的数要-1 才能和 s 数组的下标一一对应。