华东师范大学数据科学与工程学院上机实践报告

课程名称：算法设计与分析　　**年级：**21 级　　**上机实践成绩：**

指导教师：金澈清　　**姓名：**杨茜雅

上机实践名称：动态规划　　**学号：**10215501435　　**上机实践日期：**

上机实践编号：No.8

一、目的

1．熟悉动态规划的基本思想和实现

二、内容与设计思想

1. 完成 OJ 上的两道题目；
2. 分析动态规划的过程及答案的正确性；

三、使用环境

推荐使用 C/C++集成编译环境。

四、实验过程

1. 编写相关实现代码；
2. 解释题目中动态规划算法的思路，并论证其正确性；
3. 比较两道题目的相同点和不同点；

相同点：都牵涉到在给定余额的情况下求满意度的最大值，因为情况有很多种，需要一个最优解，所以都用到动态规划的思想。

不同点：疫情下的大学生问题是一个 0-1 背包问题，一种零食最多只能购买一次，而解封后的大学生可以不限量。

1 疫情下的大学生

```cpp
#include<iostream>
using namespace std;
int price[10000];
int BOTTOM_UP_CUT_ROD(int price[],int saty[],int snacktype,int bala)
{
    int q;
   int r[10000];
    r[0]=0;
    for(int j=1;j<=bala;j++)
    {
        q=0;
        for(int i=1;i<=snacktype;i++)
        {
            if(j>=price[i])
            q=max(q,saty[i]+r[j-price[i]]);//选择第i中商品，剩下的余额按最优分配，循环计算q
        }
        r[j]=q;//余额为J时的最佳收益
    }
    return r[bala];
}
int main()
{
    int snacksaty[10000];
    int snack_type;
    cin>>snack_type;
    int balance;
```

```
        r[j]=q;//余额为j时的最佳收益
    }
    return r[bala];
}
int main()
{
    int snacksaty[10000];
    int snack_type;
    cin>>snack_type;
    int balance;
    cin>>balance;
    for(int i=1;i<=snack_type;i++)
    {
        cin>>price[i];
        cin>>snacksaty[i];

    }
    cout<<BOTTOM_UP_CUT_ROD(price,snacksaty,snack_type,balance);
    return 0;
}

```

过程：

用数组输入第 i 种零食的满意度和价格，要求在规定余额内获取最大的满意度。

r[i]为余额为 i 时的最优解，从小到大求 r[]最后得出 r[bala]，在求每个 r[]时设置一个内循环，求 q 和“选中第 i 种商品，剩下的余额按最优分配”中的较大者，并遍历所有零食的种类即可求出最优的 r[]，在求 saty[i]+r[j-price]时会调用之前求的 r[]数组。

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    int snack_type, balance;
    cin >>snack_type>>balance;      //snack_type是总共有多少件snack， balance是指总共的钱
    int *price, *value;
    price = new int[snack_type+10];
    value = new int[snack_type+10];
    vector<vector<int>> array(snack_type+10,vector<int>(balance+10));
    for(int i= 1; i<= snack_type; i++)    //输入第i个零食的价钱和满意度
    {
        cin >> price[i];
        cin >> value[i];
    }
   for(int i= 1; i<=snack_type; i++)     //全部为零
    {
        for(int j = 1; j <=balance; j++)
        {
           array[i][j]=0;//在编号小于等于i的i种零食中拿j钱去买，得到的最大满意度
        }
    }
    for(int i= 1; i<= snack_type; i++)
    {
        for(int j = 1; j <=balance; j++)
        {
            if(j >= price[i])      //当余额足够买第i个零食时 ，对于第i种零食只有买和不买两种选择，没有买几次的选择
            {
                array[i][j] = max (array[i - 1][j], array[i- 1][j - price[i]] + value[i]);
            }
```

```
        cin >> value[i];
    }
    for(int i= 1; i<=snack_type; i++)     //全部为零
    {
        for(int j = 1; j <=balance; j++)
        {
           array[i][j]=0;//在编号小于等于i的i种零食中拿j钱去买，得到的最大满意度
        }
    }
    for(int i= 1; i<= snack_type; i++)
    {
        for(int j = 1; j <=balance; j++)
        {
            if(j >= price[i])      //当余额足够买第i个零食时 ，对于第i种零食只有买和不买两种选择，没有买几次的选择
            {
                array[i][j] = max (array[i - 1][j], array[i- 1][j - price[i]] + value[i]);
            }
            else//否则就不买
            {
               array[i][j] =array[i-1][j];
            }
        }
    }
    cout << array[snack_type][balance] << endl;
    return 0;
}
```

过程：这里也有双循环，但是外循环是零食中类，内循环是余额，一套套循环走完以后就可以得到 array[i][j],即在 i 种零食中每种都有买或者不买的结果，余额为 J,最后得到的最佳满意度。这里内外循环刚好跟解封后的大学生问题反过来，因为如果这里也把 j 作为内循环的话无法控制每个零食最多买一次。

五、总结

在都要求运用贪心算法的条件下，为了匹配不同的需求，我们需要在代码中根据情况对函数的调用加上不同的循环。