## **引言：**

一个人的精力是有限的，我们需要合理分配自己时间，合理分配这些所谓固定的精力于无穷的事情当中，求得一个最优解，借此最简单的01背包问题应运而生，背包容量有穷，需要拿物品且拿完的物品价值需要最大，在针对0-1背包问题求解的实践中理解动态规划 (Dynamic Programming, DP) 方法的思想、求解策略及步骤！

## 算法复杂性分析：

容量记Bag\_Capa 物品记n

### 求解背包问题部分

时间复杂度：

T(n) =O(n \* Bag\_Capa)

空间复杂度：

T(n)=O(n \* Bag\_Capa) / T(n)=O(Bag\_Capa)

## 常见的算法分析设计策略:

设计：

设计用二维数组对价值信息保存，DP[i][j] 代表 含有I + 1个物品 在j 容量下 可以取的最大价值。

很容易发现：当未曾放入东西时，价值为0即：

DP[i][j] = 0 (0<=i<n,0<=j<=Bag\_Capa)

对于一般情况

dp[i][j] =max( dp[i-1][j],value[i] + dp[i-1][j-weight[i]](max(A,B)意为A,B中取最大值)

可以这样理解：比较 不拿该物品时候的最大价值 与 拿了物品后 得到价值 + 剩余容量 i-1个物品下 最大价值。保证从前到后都是最大价值！

## 0-1背包问题在动态规划（贪心算法或回溯法）中的实现

#include <iostream>

using namespace std;

// class OT\_Bag

// {

//     void OT\_Bag\_Easy()

//     {

//     }

// };

int max(int x,int y)

{

    return x>y?x:y;

}

int main()

{

    // dp  物品数量 物品质量 物品价值

    int dp[101][101] = {0};

    int n, bag\_Capa,answer;

    scanf("%d", &n); //物品数量

    int value[n] = {0};

    int weight[n] = {0};

    for (int i = 0; i < n; i++)

    {

        scanf("%d", &weight[i]); //物品质量

        scanf("%d", &value[i]);  //物品价值

    }

    scanf("%d", &bag\_Capa);     //容量

    //初始化

    //dp数组中每个元素都代表 i个物品  j容量下 装取最大价值为多少。

    //通过前面情况最大价值 推 后续情况 最大价值：

    //dp[i][j] = max(dp[i-1][j],value[i]+dp[i-1][j-weight[i]]);

    //恰好配合 该物品要不要拿

    //最后输出  n个物品 bag 容量下 能取的最大价值

    for (int j = weight[0]; j <= bag\_Capa; j++) {

         dp[0][j] = value[0];

    }

    for (int i = 1; i < n; i++) //物品优先 初始化 一行

    {

        for (int j = 0; j <= bag\_Capa; j++) //变化容量

        {

            //初始化最上边 最右边

            // if (i == 0 || j == 0)

            // {

            //     if (j < weight[i]) //放不进去

            //     {

            //         dp[i][j] = 0;

            //     }

            //     else

            //     {

            //         dp[i][j] = value[i];

            //     }

            // }

            //初始化开始写在里面会影响后续判断

            //dp[i][j] 看第i个值不值得放进去

            //将不放进去最大价值 与放进去 价值 + 剩余容量且在0-i-1任取 情况下的最大价值比较

            //根本放不进去

            if(j<weight[i]){

                dp[i][j] = dp[i-1][j];

            }

            else{

                dp[i][j] = max(dp[i-1][j],value[i]+dp[i-1][j-weight[i]]);

            }

            //放进去了然后比较

        }

    }

    cout << dp[n-1][bag\_Capa]<<endl;//n-1 代表第n个物体  bag下标正好对应bag容量

    return 0;

}

## 应用场景：

辰辰是个天资聪颖的孩子，他的梦想是成为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到一个到处都是草药的山洞里对他说：“孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。”

给定 N个正整数 A1,A2,…,AN，从中选出若干个数，使它们的和为 M，求有多少种选择方案。