**背包问题程序报告**

姓名：吴静 学号：2113285 班级：信息安全二班

**一、源码部分**

dp算法：

#include<iostream>

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

using namespace std;

int n;

int object[100][2] = { 0 };

int dp(int n,int V) {

if (n == -1)

return 0;

if (V - object[n][0] >= 0)

return max(object[n][1] + dp(n - 1, V - object[n][0]), dp(n - 1, V));

else

return dp(n - 1, V);

}

int main() {

clock\_t start1, finish1, start2, finish2;

int V;

cout << "请输入背包容量V：" << endl;

cin >> V;

cout << "请输入物品个数n：" << endl;

cin >> n;

cout << "请依次输入物品所占背包容量和物品价值：" << endl;

for (int i = 0; i < n; i++) {

cin >> object[i][0] >> object[i][1];

}

//cout << "由随机数产生的物品所占背包容量和物品价值如下所示：" << endl;

//for (int i = 0; i < n; i++) {

// object[i][0] = rand() % (2 \* n);

// object[i][1] = rand() % (V / n) + 2 \* n;

// cout << object[i][0] << ' ' << object [i][1] << endl;

//}

cout << "由dp算法求出该背包所能容纳物品的最大价值为：" << endl;

start1 = clock();

cout << dp(n, V) << endl;

finish1 = clock();

cout << "dp算法程序运行时长为： " << (double)(finish1 - start1) / CLOCKS\_PER\_SEC << endl;

}

枚举法：

void badd(int used[]) {

for (int i = 0; i < n; i++) {

if (used[i] != 1) {

used[i] = 1;

return;

}

else

used[i] = 0;

}

return;

}

int sum = 0;

start2 = clock();

for (int i = 0; i < pow(2, n); i++) {

badd(used);

int tempw = 0;

int tempV = 0;

for (int j = 0; j < n; j++) {

tempw += used[j] \* object[j][1];

tempV += used[j] \* object[j][0];

}

if (tempV <= V && tempw > sum)

sum = tempw;

}

cout << "由枚举法求出该背包所能容纳物品的最大价值为：" << endl;

cout << sum << endl;

finish2 = clock();

cout << "枚举法程序运行时长为： " << (double)(finish2 - start2) / CLOCKS\_PER\_SEC << endl;

**二、说明部分**

本题主要考察对dp算法（动态规划）和01背包问题的理解运用。

动态规划算法，是将复杂问题拆分成子问题，并在子问题的基础上，求解复杂问题，子问题之间不是独立的，而是相互依存的，其过程大致可以分为以下四步：

1.划分状态，即划分子问题, 按照问题的时间或空间特征，把问题分为若干个阶段。在划分阶段时，注意划分后的阶段一定要是有序的或者是可排序的.

2.状态表示

3.状态转移

4.确定边界，找出递推的终止条件或边界条件

总结：在使用动态规划时，最重要的是确定动态规划三要素——问题的阶段，每个阶段的状态，从每一个阶段转化到后一个阶段之间的递推关系。

他有两种实现方式：递归和非递归，以递归为例，最终是要确定表达式**f(n,m)=max{f(n-1,m), f(n-1,m-w[n])+P(n,m)}**：

约定object[i][0]装载物品的重量，object[i][1]装载物品的价值，而装与不装，是受背包容量V和价格最大化限制。

我们假设dp(n,V)代表对第n个物品操作且容量为V时的最优解（价值最大）。那么对于第n个物品而言，受背包容量限制：

1.当容量够的时候，我们有两种对n物品的操作，选n，不选n:

* 选n时其对应最优解为dp[n][1]+dp(n-1,V-object[n][0])
* 不选n时其对应最优解为dp(n-1,V)

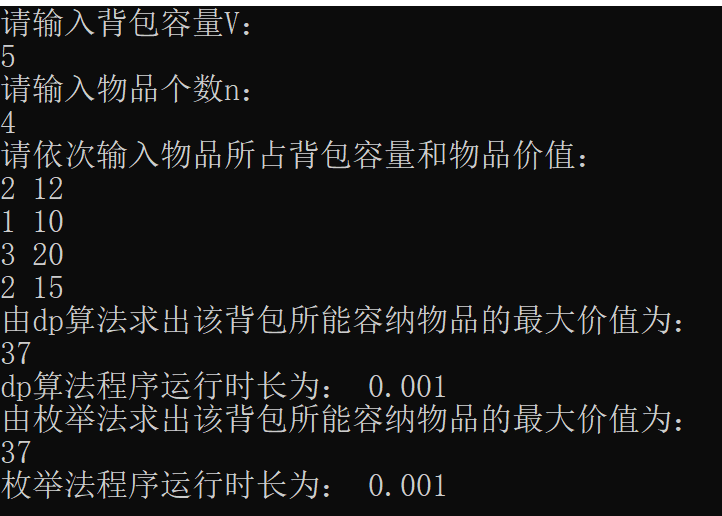
故此时dp(n,V)=MAX{ dp[n][1]+dp(n-1,V-object[n][0]),dp(n-1,V)}

2.当容量不够时，我们只有一种对n物品的操作，即不选n

此时dp(n,V)=dp(n-1,V)

由于以上的运算流程是基于递归完成的，所以要设置递归结束条件：当n<0时，说明没有物品了，return 0;

**三、运行示例**

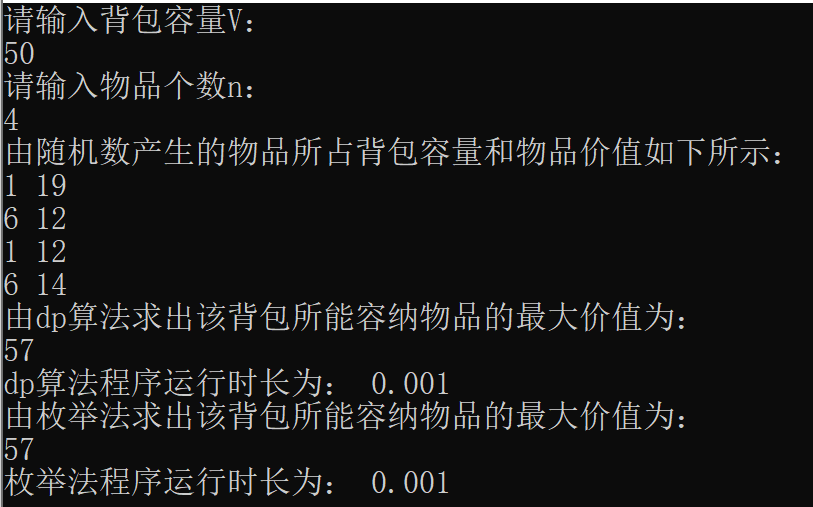


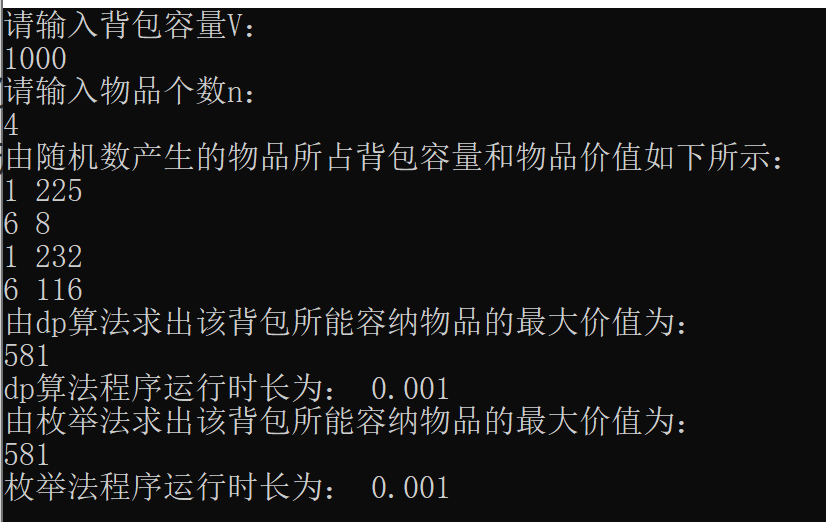
物品个数为4时，dp算法和枚举法运行时长均为0.01s。

例题案例通过。

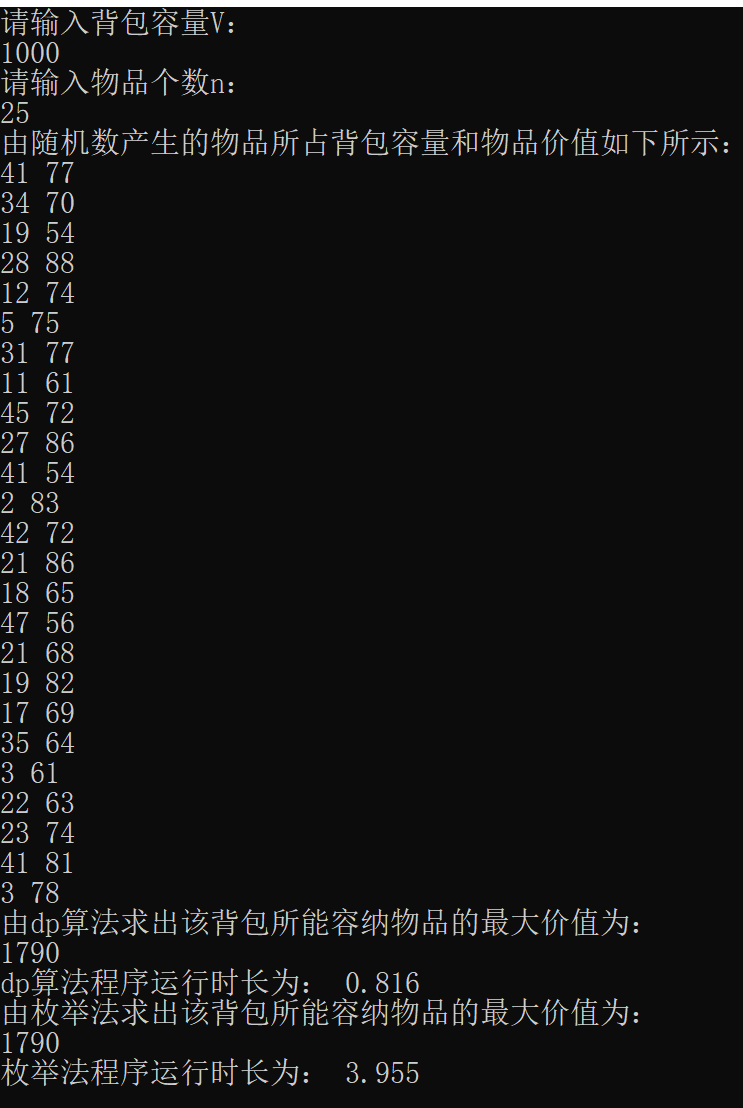
另外，我们再利用随机数给出几个测试用例：

1.物品个数为4时：

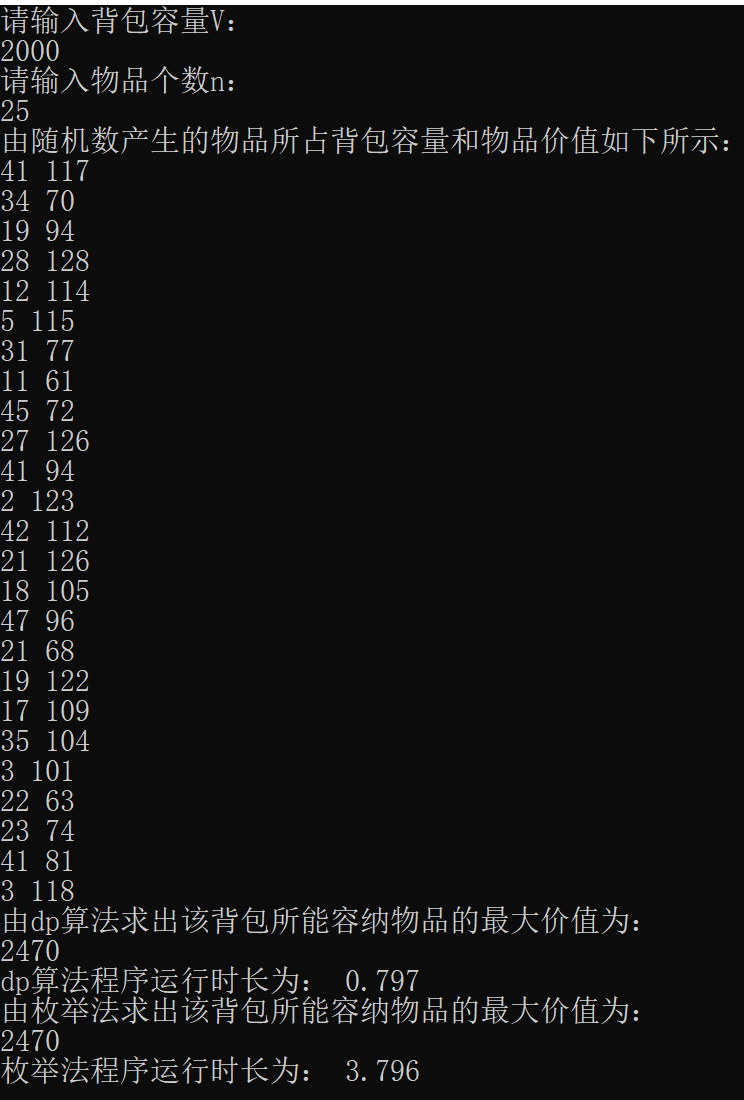




2.物品个数为25时：

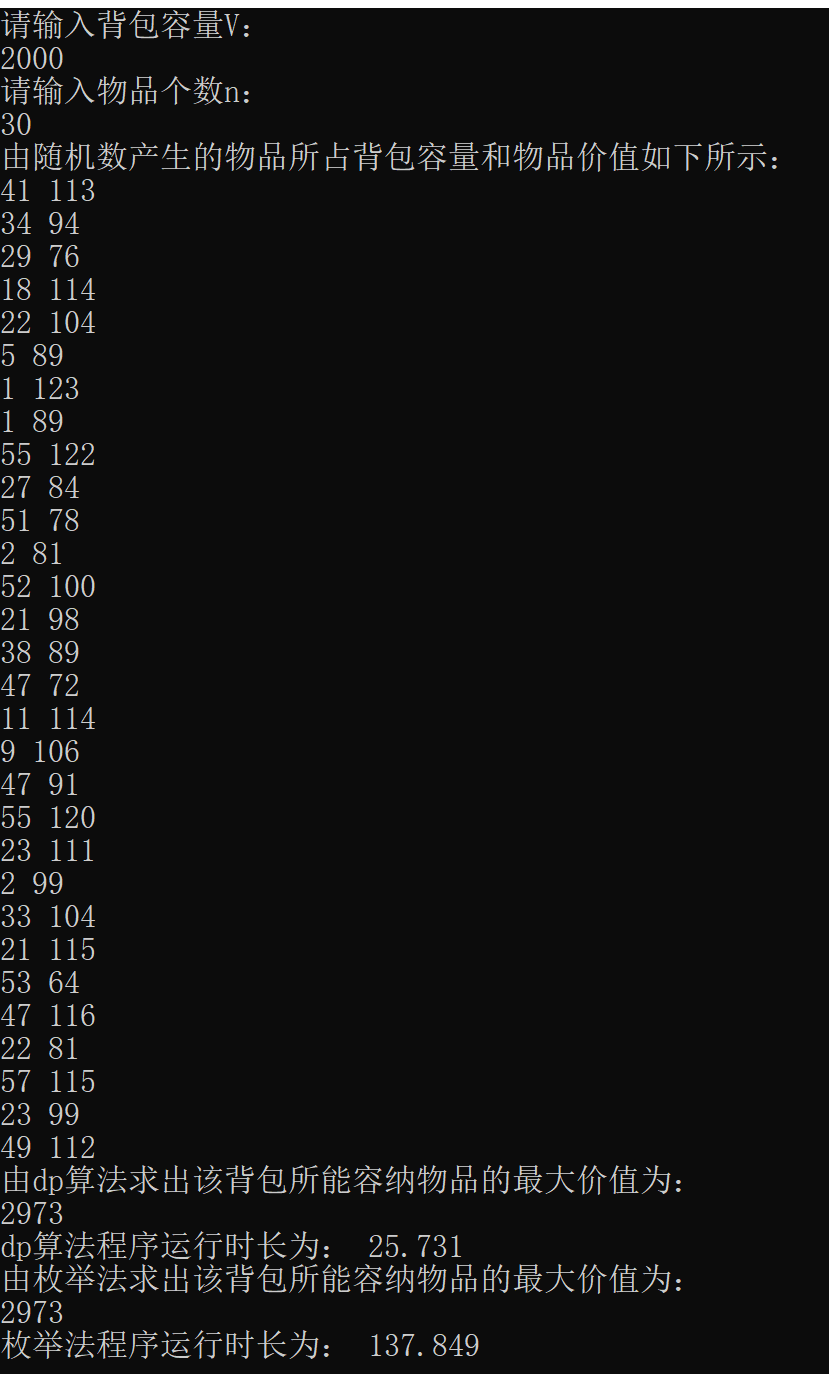


可以看到，此时dp算法的运行时长为0.816s，大大小于枚举法的运行时长3.955s。



此时dp算法的运行时长为0.797s，大大小于枚举法的运行时长3.796s。

3.物品个数为30时



此时dp算法和枚举法运行时间都大大增加，同时，dp算法运行时间25.731s，远远小于枚举法运行时间137.849s