**★★★☆☆**

**題組：Problem Set Archive** **with Online Judge**

**題號：10898:Combo Deal**

**解題者：楊捷評**

**解題日期：2022年5月19日**

**題意：**

輸入菜單中各餐點的單賣價格，以及套餐中包含的餐點數量及總價格，最後輸入欲購買的各餐點數量，輸出最低的應付價格。

**題意範例：**

4 349 99 109 219 => 4個餐點及價格

2 => 2種套餐

1 1 1 0 479 => 包含的餐點數量及價格

2 2 2 1 999

2 => 2個餐點訂單數量

9 6 8 0

9 6 8 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 　 | 　 | 　 | 數量 | 　 | 價格 |
| 餐點1 | 1 | 0 | 0 | 0 | 349 |
| 餐點2 | 0 | 1 | 0 | 0 | 99 |
| 餐點3 | 0 | 0 | 1 | 0 | 109 |
| 餐點4 | 0 | 0 | 0 | 1 | 219 |
| 套餐1 | 1 | 1 | 1 | 0 | 479 |
| 套餐2 | 2 | 2 | 2 | 1 | 999 |

**解法：**

用vector來儲存Menu的內容(各餐點的數量及價格，其中數量是一串digit)，單點也視為一個只有單一份餐點的套餐。接著利用動態規劃的top-down approach，將各種可能的組合都跑過一次，但不會重複計算。

演算法:

int recur(n, vector<menu>&v)

 If (n == 0) return 0;

 If (dp有找過了) return dp[n]

 Otherwise,

 Min = 100000000

 For (跑過各個menu組合)

 If (餐點數沒有超過訂購數)

 Min = min(Min, recur(剩餘訂購數, v)+目前的套餐價格)

 dp[n] = Min // memorization

 return dp[n]

**解法範例：**

2 15 10 => 2個餐點及價格

1 => 2種套餐

1 1 20 => 包含的餐點數量及價格

2 => 2個餐點訂單數量

1 2

3 5



 以上是計算有兩個餐點種類的訂單，各有效節點的情形。其中每個節點的最低價(Min)都會存到dp陣列中，在找之後的訂單35的時候就不用一直recursive下去。

在看完所有可能情形之後(在此指套餐的種類)，找出應付的最低金額，這個值即是答案，以上面的圖(12)為例，答案為30。

**討論：**

(1) 如果是使用數字串來儲存餐點個數，要注意每個餐點不可以被訂購超過9次。(例如輸入10，結果變成餐點2被訂購10次。)

**程式：**

#include<iostream>

#include<vector>

#include<algorithm>

#include<cstring>

#include<cmath>

const int SIZE = 1000000;

using namespace std;

int dp[SIZE];

int num;

struct DATA

{

 int v = 0; // kinds

 int p = 0; // price

 DATA(int a, int b) :v(a), p(b)

 {

 }

};

int recur(int n, const vector<DATA>& v);

int main()

{

 while (cin >> num)

 {

 int item;

 int s;

 vector<DATA> v;

 int t = pow(10, num -1);//t是把單賣也算成只有一份餐點的套餐(combo)

 for (int i = 0; i < num; i++)

 {

 cin >> item;

 v.push\_back(DATA(t, item));

 t /= 10;

 }

 int combo;

 cin >> combo;//套餐數量

 for (int i = 0; i < combo; i++)

 {

 s = 0;//kinds of a single combo

 for (int j = 0; j < num; j++)

 {

 cin >> t;

 s = s \* 10 + t; //套餐內的個別餐點個數可以超過一個

 }

 cin >> t;//套餐價格

 v.push\_back(DATA(s, t));

 }

 // initialize the dp array

 memset(dp, 0, sizeof(dp));

 int q;

 cin >> q;//requested orders quantity

 for (int i = 0; i < q; i++)

 {

 s = 0;

 for (int j = 0; j < num; j++)

 {

 cin >> t;

 s = s \* 10 + t;

 }

 cout << recur(s, v) << endl;

 }

 }

 return 0;

}

int recur(int n, const vector<DATA>& v)

{

 int mi, t;

 if (dp[n] || n == 0) //有找過了或剛好完成order(dp[0] == 0)

 {

 return dp[n];

 }

 else

 {

 mi = (int)1e9;

 for (int i = 0; i < v.size(); i++)

 {

 //每次都從單點先處理，所以dp[n]永遠不會return 1e9

 t = n - v[i].v;

 bool b = true;

 for (int j = 1 ; j <= num ; ++j)

 {

 //避免order:10有10(>9)個單點二的問題

 if (n % (int)(pow(10, j)) < (t % (int)(pow(10, j))))

 {

 b = false;

 }

 }

 if (t >= 0 && b)

 {

 //如果order需求數<0，則mi會是上一次的min

 mi = min(mi, recur(t, v) + v[i].p);

 }

 }

 }

 dp[n] = mi;//存到dp，下次就可以直接拿出來用

 return dp[n];

}