# NCKU Programming Contest Training Course
# Course 9
# 2013/03/13

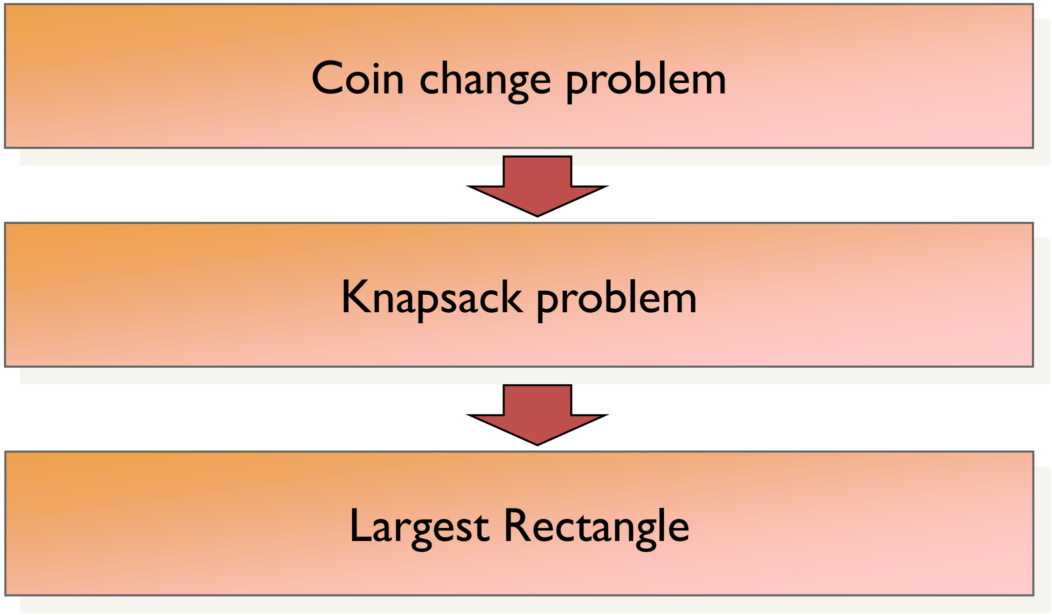
**Sheng-Chi You(rabbit125)**

*jay_s6215@hotmail.com*

***http://myweb.ncku.edu.tw/~f74986133/Course_9.rar***

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

# Outline



Coin change problem

Knapsack problem

Largest Rectangle

# Coin change problem

- Coin change problem
  - 0/1 Coin change problem
  - Unbounded Coin change problem
  - Limited Coin change problem
  - …

# Coin Change

## 類型

- 硬幣限制各一個(0/1背包變型)
  - 是否湊得某個價位 /湊得某價位的方法數
- 硬幣無限
  - 是否湊得某個價位 / 湊得某價位的方法數 / 湊得某個價位的最少硬幣用量
  - 湊得某個價位的硬幣用量 (錢用量不多時)
- 硬幣有限
  - 是否湊得某個價位 / 湊得某個價位的最少硬幣用量

## 相同觀念

- 按照題意設計
- 注意設定紀錄維度, 意義, 以及初始化數值

# Coin Change

- 硬幣**限制各一個**，是否湊得某價位
- *Dp[0]=1;*
- *Money value v[i]= 2,5*
- *if ( dp[ j-v[i] ] == true ) dp[j]=dp[ j-v[i] ];*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Coin Change

- 硬幣限制各一個，是否湊得某價位
- *Dp[0]=1;*
- *Money value v[i]= 2,5*
- *if ( dp[ j-v[i] ] == true ) dp[j]=dp[ j-v[i] ];*

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Coin Change

- 硬幣限制各一個，是否湊得某價位
- *Dp[0]=1;*
- *Money value v[i]= 2,5*
- *if ( dp[ j-v[i] ] == true ) dp[j]=dp[ j-v[i] ];*

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Coin Change

- 硬幣限制各一個，是否湊得某價位
- *Dp[0]=1;*
- *Money value v[i]= 2,5*
- *if ( dp[ j-v[i] ] == true ) dp[j]=dp[ j-v[i] ];*

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

# Coin Change

- 硬幣限制各一個，是否湊得某價位
- *Dp[0]=1;*
- *Money value v[i]= 2,5*
- *if ( dp[ j-v[i] ] == true ) dp[j]=dp[ j-v[i] ];*

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

# Coin Change

- 硬幣限制各一個，是否湊得某價位
- *Dp[0]=1;*
- *Money value v[i]= 2,5*
- *if ( dp[ j-v[i] ] == true ) dp[j]=dp[ j-v[i] ];*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |

# Coin Change

- 硬幣限制各一個，是否湊得某價位
- *Dp[0]=1;*
- *Money value v[i]= 2,5*
- *if ( dp[ j-v[i] ] == true ) dp[j]=dp[ j-v[i] ];*

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

# Coin Change

- 硬幣無限，是否湊得某個價位
- *Dp[0]=1;*
- *Money value v[i]= 2,5*
- *if ( dp[ j-v[i] ] == true ) dp[j]=dp[ j-v[i] ];*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Coin Change

- 硬幣無限，是否湊得某個價位
- *Dp[0]=1;*
- *Money value v[i]= 2,5*
- *if ( dp[ j-v[i] ] == true ) dp[j]=dp[ j-v[i] ];*

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

***made by electron, free999, rabbit125***

# Coin Change

- 硬幣無限，是否湊得某個價位
- *Dp[0]=1;*
- *Money value v[i]= 2,5*
- *if ( dp[ j-v[i] ] == true ) dp[j]=dp[ j-v[i] ];*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

# Coin Change

- 硬幣無限，是否湊得某個價位
- *Dp[0]=1;*
- *Money value v[i]= 2,5*
- *if ( dp[ j-v[i] ] == true ) dp[j]=dp[ j-v[i] ];*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

# Coin Change

- 硬幣無限，是否湊得某個價位
- *Dp[0]=1;*
- *Money value v[i]= 2,5*
- *if ( dp[ j-v[i] ] == true ) dp[j]=dp[ j-v[i] ];*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

# Coin Change

- 硬幣無限，是否湊得某個價位
- *Dp[0]=1;*
- *Money value v[i]= 2,5*
- *if ( dp[ j-v[i] ] == true ) dp[j]=dp[ j-v[i] ];*

|    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|----|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |

# Coin Change

- 硬幣無限，是否湊得某個價位
- *Dp[0]=1;*
- *Money value v[i]= 2,5*
- *if ( dp[ j-v[i] ] == true ) dp[j]=dp[ j-v[i] ];*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |

# Coin Change

- 硬幣無限，湊得某個價位有幾種？
- *Dp[0]=1;*
- *Money value v[i]= 2,3*
- *if ( dp[ j-v[i] ] == true ) dp[j]+=dp[ j-v[i] ];*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Coin Change

- 硬幣無限，湊得某個價位有幾種？
- *Dp[0]=1;*
- *Money value v[i]= 2,3*
- *if ( dp[ j-v[i] ] == true ) dp[j]+=dp[ j-v[i] ];*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

# Coin Change

- 硬幣無限，湊得某個價位有幾種？
- *Dp[0]=1;*
- *Money value v[i]= 2,3*
- *if ( dp[ j-v[i] ] == true ) dp[j]+=dp[ j-v[i] ];*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

# Coin Change

- 硬幣無限，湊得某個價位有幾種？
- *Dp[0]=1;*
- *Money value v[i]= 2,3*
- *if ( dp[ j-v[i] ] == true ) dp[j]+=dp[ j-v[i] ];*

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

# Coin Change

- 硬幣無限，湊得某個價位有幾種？
- *Dp[0]=1;*
- *Money value v[i]= 2,3*
- *if ( dp[ j-v[i] ] == true ) dp[j]+=dp[ j-v[i] ];*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |

# Coin Change

- 硬幣無限，湊得某個價位有幾種？
- *Dp[0]=1;*
- *Money value v[i]= 2,3*
- *if ( dp[ j-v[i] ] == true ) dp[j]+=dp[ j-v[i] ];*

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

# Coin Change

- 硬幣無限，湊得某個價位有幾種？
- *Dp[0]=1;*
- *Money value v[i]= 2,3*
- *if ( dp[ j-v[i] ] == true ) dp[j]+=dp[ j-v[i] ];*

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 1 | 1 | 1 | 2 | 0 | 1 |

1+1

# Coin Change

- 硬幣無限，湊得某個價位有幾種？
- *Dp[0]=1;*
- *Money value v[i]= 2,3*
- *if ( dp[ j-v[i] ] == true ) dp[j]+=dp[ j-v[i] ];*

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 1 | 1 | 1 | 2 | 1 | 1 |

# Coin Change

- 硬幣無限，湊得某個價位有幾種？
- *Dp[0]=1;*
- *Money value v[i]= 2,3*
- *if ( dp[ j-v[i] ] == true ) dp[j]+=dp[ j-v[i] ];*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 1 | 0 | 1 | 1 | 1 | 1 | 2 | 1 | 2 |

1+1

# Coin Change

- 硬幣無限，湊得某個價位的最少硬幣用量
- *Dp[0]=0;*
- *Money value v[i]= 2,3*
- *dp[j]=min ( dp[j], dp[ j-v[i] ]+1 );*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 0 | INF | INF | INF | INF | INF | INF | INF | INF |

# Coin Change

- 硬幣無限，湊得某個價位的最少硬幣用量
- *Dp[0]=0;*
- *Money value v[i]= 2,3*
- *dp[j]=min ( dp[j], dp[ j-v[i] ]+1 );*
- *Greedy ： Money value 大的優先( 找錢問題 )*

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| dp | 0 | INF | INF | INF | INF | INF | INF | INF | INF |

# Practice

- 基礎：UVA 674
- 進階：UVA 10306

# Coin Change

- 硬幣有限，是否湊得某個價位的最少硬幣用量
- *Money value v[i]= {2,4}*
- *Number of v[i] → m[] = {2,1}*

- *硬幣無限的方法跑 m 次??*

|    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|----|---|---|---|---|---|---|---|---|---|
| dp | 0 | INF | INF | INF | INF | INF | INF | INF | INF |

# Coin Change

- 硬幣有限，是否湊得某個價位的最少硬幣用量
- *Money value v[i]= {2,4}*
- *Number of v[i] → m[] = {2,1}*

- *硬幣無限的方法跑* m *次??*

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 0 | INF | 1 | INF | 2 | INF | INF | INF | INF |

# Coin Change

- 硬幣有限，是否湊得某個價位的最少硬幣用量
- *Money value v[i]= {2,4}*
- *Number of v[i]* → *m[]* = {2,*1*}

- *硬幣無限的方法跑* m *次??*

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| dp | 0 | INF | 1 | INF | 1 | INF | INF | INF | INF |

# Coin Change

- 硬幣有限，是否湊得某個價位的最少硬幣用量
- *Money value v[i]= {2,4}*
- *Number of v[i] → m[]* = {2,*1*}
- *硬幣無限的方法跑* m *次??*

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| dp | 0 | INF | 1 | INF | 1 | INF | 1 | INF | INF |

# Coin Change

- 硬幣有限，是否湊得某個價位的最少硬幣用量
- *Money value v[i]= {2,4}*
- *Number of v[i] → m[] = {2,1}*

- *硬幣無限的方法跑 m 次??*

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 0 | INF | 1 | INF | 1 | INF | 1 | INF | 2 |

# Coin Change

- 硬幣有限，是否湊得某個價位的最少硬幣用量
- *Money value v[i]= {2,4}*
- *Number of v[i] → m[] = {2,1}*

- *硬幣無限的方法跑 m 次??* → *It is wrong !*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 0 | INF | 1 | INF | 1 | INF | 1 | INF | 2 |

impossible

# Coin Change

- 硬幣有限，是否湊得某個價位的最少硬幣用量
- *Money value v[i]= {2,4}*
- *Number of v[i] → m[]   = {2,1}*
- *for j = max to v[i]*

  *run m[i]*次

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| dp | 0 | INF | INF | INF | INF | INF | INF | INF | INF |

# Coin Change

- 硬幣有限，是否湊得某個價位的最少硬幣用量
- *Money value v[i]= {2,4}*
- *Number of v[i]* → *m[] = {2, 1}*
- *for j = max to v[i]*

  *run m[i]次*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 0 | INF | 1 | INF | 2 | INF | INF | INF | INF |

# Coin Change

- 硬幣有限，是否湊得某個價位的最少硬幣用量
- *Money value v[i]= {2,4}*
- *Number of v[i]* → *m[]* = {2, 1}
- *for j = max to v[i]*

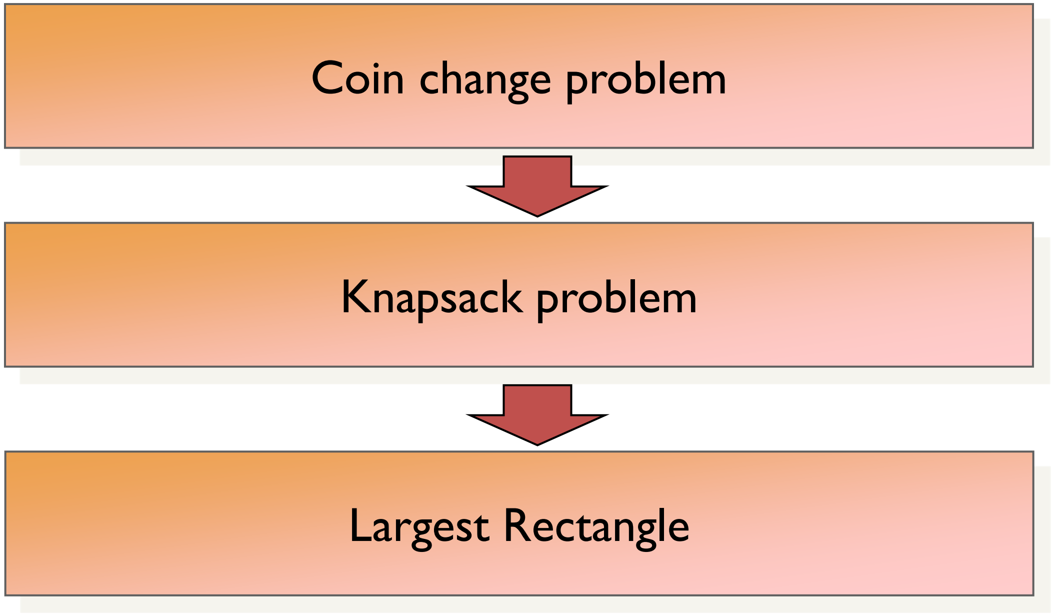
  *run m[i]*次

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| dp | 0 | INF | 1 | INF | 2 | INF | 2 | INF | 3 |

# Practice

- UVA 166

# Outline



Coin change problem

Knapsack problem

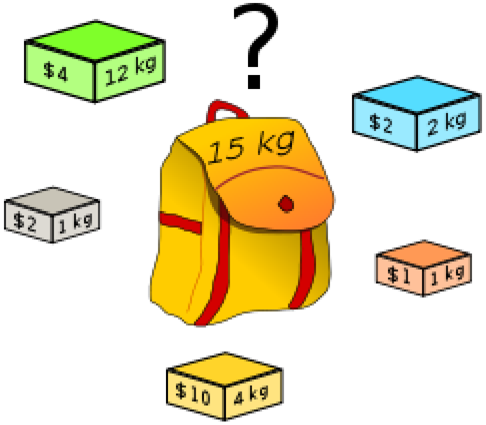
Largest Rectangle

# Knapsack problem

- Knapsack problem
  - 0/1 Knapsack problem
  - Unbounded Knapsack problem
  - …

# 0/1 Knapsack problem

- Problem Description:
  - Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than a given limit and the total value is as large as possible.



- Brute force
  - Each item has 2 status: put in the bag or not
  - If there are N items, it will cost O(2^N) to check all the possibilities.

# 0/1 Knapsack problem

- Status representation and transfer function:
  - dp[n][m] store the maximum value that we put some of first n items in the bag and weight m
  - $dp[n][m] = \max(\underbrace{dp[n-1][m]}_{\text{不取第n個物品}}, \underbrace{dp[n-1][m-w[n]] + v[n]}_{\text{取第n個物品放入背包}})$
- Top-down DP can be written as follow:

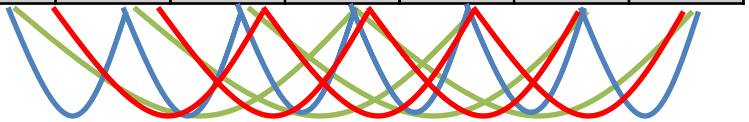
```
int dp[N+1][W+1], v[N], w[N];                    // top-down, N items with maximum total weight W
bool isfind[N][W];
int knapsack(int n, int m){
        if (m < 0) return -INF;                            // basic constrain
        if (n == 0 ) return 0;
        if (isfind[n][m]) return dp[n][m] ;                // isfind before
        dp[n][m] = max(knapsack(n-1, m),
                                knapsack(n-1, m-w[n]) + v[n]);   // recursively call
        isfind[n][m ] = true;                              // record isfind
        return  dp[n][m];
}
```

# 0/1 Knapsack problem

- Bottom-up DP can be written as follow:

```
int c[W+1], v[N], w[N];                           // bottom-up
int knapsack( int n, int w){
        memset(c, 0, sizeof(c));                  // Initialize basic constrain
        for (int i = 0; i < n; i++)
                for (int j = W; j - w[i] >= 0; j--)   // Back to the front
                        c[j] = max( c[j], c[j - w[i]] + v[i] );   // Update lookup table
        return c[w];
}
```

| c[0] | c[1] | c[2] | c[3] | c[4] | c[5] | c[6] |
|---|---|---|---|---|---|---|
| 0 | 20 | 30 | 50 | 50 | 60 | 60 |



| item | w | v |
|---|---|---|
| 0 | 3 | 10 |
| 1 | 1 | 20 |
| 2 | 2 | 30 |

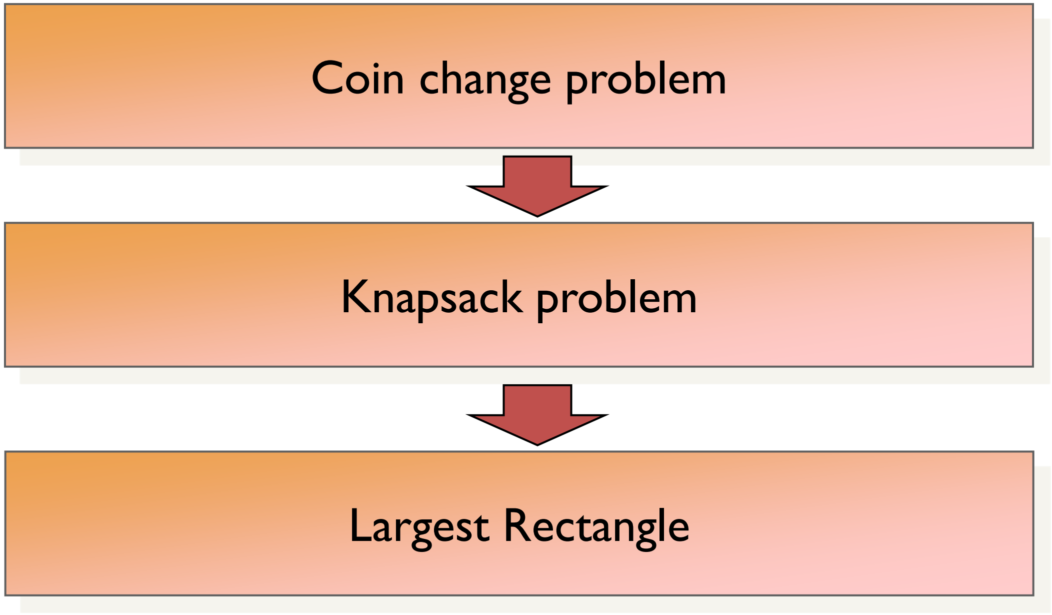
| W |
|---|
| 6 |

Initialize basic constrain
Update lookup table
Get the answer

# Practice

- ZeroJudge2 d155

# Outline



Coin change problem

Knapsack problem

Largest Rectangle

# Largest Rectangle

- Largest Rectangle
  - Maximum Sub-array Sum problem 1D (Array )
  - Maximum Sub-array Sum problem 2D (Rectangle)
  - Max size of Rectangle expansion
  - …

# MSS

- Maximum Sub-Array Sum Problem
  - Maximum subarray summation problem is to find a subarray which contains a set of continuous elements in which the summation is maximum.
  - The elements in the subarray must be continuous.
  - This problem can be extended to multiple dimension.
  - Two general method to solve this problem
    - Brute force method
    - DP based method

# MSS

- Maximum Sub-Array Sum Problem
  - Example for one dimension
    - Given an array a[10] = {1, 2, -6, 3, -2, 4, -1, 3, 2, -4}
    - Subarray can be {1, 2, -6} with summation = 1+2-6 = -3, can be also be {-1, 3, 2, -4} with summation = -0, and so on.
  - Example for two dimension
    - Given an array:

|   |   |   |   |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 2 | 6 | -4 | 3 |
| 4 | 3 | -3 | 3 |
| 5 | 2 | -1 | -1 |

Subarray with summation value = 14

# MSS (1D)

- Maximum Sub-Array Sum Problem (1D)

- Rule 1
  - Order?

- Rule 2
  - Category

# MSS (1D)

- Rule 3
  - Define a *max_sum* that represents the optimal value and define a variable *sum* that represents a temporary summation.
  - Given an array *a[1…N]* .

```
int maximum_subarray()
{
        int max_sum = 0, sum = 0;
        for (int i=0; i<N; ++i)
        {
                sum += a[i];                                   // 隨時計算總和
                if (sum < 0) sum = 0;                          // 零總比負數好
                if (sum > max_sum) max_sum = sum;              // 隨時紀錄最大值
        }
        return max_sum;
}
```

# MSS (1D)

- Rule 4
  - Program

```
int maximum_subarray()
{
         int max_sum = 0, sum = 0;
        for (int i=0; i<N; ++i)
        {
                sum += a[i];                                  // 隨時計算總和
                if (sum < 0) sum = 0;                         // 零總比負數好
                if (sum > max_sum) max_sum = sum;             // 隨時紀錄最大值
        }
         return max_sum;
}
```

- Rule 5
  - Trace

# MSS (2D)

- Maximum Sub-Array Sum Problem (2D)
  - UVA 108

$$\begin{array}{rrrr}
0 & -2 & -7 & 0 \\
9 & 2 & -6 & 2 \\
-4 & 1 & -4 & 1 \\
-1 & 8 & 0 & -2
\end{array}$$

# DP method

- Exercise
  - UVA 10684 (1D MSS problem)
  - UVA 108 (2D MSS problem)

- Review
  - Time complexity O( ? )
  - Space complexity O( ? )
  - Compare with the brute force method.

# References

- 演算法筆記
  - http://www.csie.ntnu.edu.tw/~u91029/KnapsackProblem.html#a4
  - http://acm.nudt.edu.cn/~twcourse/MoneyChangingProblem.html
  - http://www.csie.ntnu.edu.tw/~u91029/LargestEmptyRectangle.html

# Homework 9

- ZeroJudge2
  - d207, d208, d197, d167, ,d155, d156, d146, d267, d171
- PKU
  - 1050
- Uva
  - 10285
- NCKUOJ
  - 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 132

# Notice

- 例行月賽! 記得先報名!!
- 3/26(二) CPE 18:30~9:30
- 3/27(三) ITSA 18:00~21:00
- 3/28(四) PTC 19:00~22:00
- 3/30~4/7 Practice!!!! Practice!!!! Practice!!!!

# Homework 9+

- **PKU (total 124 problems)**

1837, 1836, 1260. 2533, 3176, 3034, 1925, 2948, 3280, 1054, 1191, 2250, 1159, 1018, 1050, 1083, 1088, 1125, 1143, 1157, 1163, 1178, 1179, 1185, 1208, 1276, 1322, 1414, 1456, 1458, 1609, 1644, 1664, 1690, 1699, 1740, 1742, 1887, 1926, 1936, 1952, 1953, 1958, 1959, 1962, 1975, 1989, 2018, 2029, 2033, 2063, 2081, 2082, 2181, 2184, 2192, 2231, 2279, 2329, 2336, 2346, 2353, 2355, 2356, 2385, 2392, 2424, 1019, 1037, 1080, 1112, 1141, 1170, 1192, 1239, 1655, 1695, 1707, 1733, 1737, 1837, 1850, 1920, 1934, 1937, 1964, 2039, 2138, 2151, 2161, 2178, 1015, 1635, 1636, 1671, 1682, 1692, 1704, 1717, 1722, 1726, 1732, 1770, 1821, 1853, 1949, 2019, 2127, 2176, 2228, 2287, 2342, 2374, 2378, 2384, 2411, 1579, 1080, 3356, 2533, 1631, 1157, 1014, 1160

# Homework 9+

- **UVA (total ? problems)**
  103, 108, 111, 116, 147, 164, 166, 231, 348, 357, 437, 473, 481, 497, 507, 531, 562, 590, 607, 620, 624, 674, 709, 711, 714, 787, 825, 836, 882, 907, 909, 910, 926, 944, 986, 988, 990, 991, 10003, 10029, 10032, 10036, 10037, 10051, 10066, 10069, 10074, 10081, 10100, 10111, 10118, 10128, 10130, 10131, 10149, 10151, 10154, 10157, 10159, 10163, 10166, 10169, 10185, 10192, 10201, 10207, 10247, 10259, 10261, 10271, 10280, 10285, 10296, 10304, 10306, 10313, 10340, 10358, 10400, 10401, 10404, 10405, 10453, 10482, 10496, 10534, 10549, 10558, 10559, 10564, 10593, 10599, 10604, 10605, 10616, 10617, 10618, 10625, 10626, 10635, 10643, 10645, 10648, 10650, 10651, 10654, 10663, 10664, 10665, 10667, 10681, 10684, 10688, 10690, 10700, 10702, 10712, 10721, 10722, 10723, 10739, 10755, 10759, 10817, 10827, 10891, 10910, 10911, 10912, 10913, 10917, 10918, 10943, 10953, 10970, 11002, 11003, 11008, 11022, 11026, 11052, 11081, 11087, 11125, 11126, 11133, 11137, 11149, 11151, 11153, 11158, 11162, 11171, 11176, 11238, 11258, 11259, 11284, 11307, 11311, 11312, 11331, 11341, 11370, 11372, 11391, 11394, 11400, 11404, 11420, 11421, 11427, 11432, 11438, 11441, 11450, 11471, 11472, 11485, 11500

# Homework 9+

- **ZJ2 (total 9 problems)**

d013, d018, d023, d025, d034, d039,  d061,  d078, d079, d083

*made by electron, free999, rabbit125*

# Thank You For Attention!