NCKU Programming Contest Training Course
Course 8
2015/03/18

Mong-Ting Wu(aj211y)
aj211y@gmail.com
http://myweb.ncku.edu.tw/~f74006323/course_8_dp2.pptx

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

made by aj211y

Outline

- Coin change problem
- Knapsack problem

Coin change

- 類型：
  - 硬幣限制各 1 個
  - 硬幣無限多個
  - 硬幣有限

  求
  - 是否湊得某個價位
  - 湊得某價位的方法數
  - 湊得某價位的最少硬幣用量

made by aj211y

Coin change

- 硬幣限制各一個，是否湊得某價位
- dp[i]：能否湊得價位 i (0 - 不可 / 1 - 可以)
  (dp[0]=1 ➔ 0元必可湊得)
- v[k]：第k種幣值
- If (dp[i – v[k]] == true) dp[i] = 1;

如果價位 i 可以由v[k]湊得，
表示價位 i – v [k]可以被湊
得(dp [i - v[k])=1)

Coin change

- v[k] = 2, 5 (有2跟5這兩種幣值)
- If (dp[i – v[k]] == true) dp[i] = 1;

注意：每個硬幣只有一個，所以可以湊得**最大的價位**就是每個硬幣都用過一次 Σv[k]

初始化：零元必可湊得

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

v[k] = 2

i = 2

dp[2] 更新值為1
因為價位 2可以被湊出來

Coin change

- v[k] = 2, 5 (有2跟5這兩種幣值)
- If (dp[i – v[k]] == true) dp[i] = 1;

注意：每個硬幣只有一個，所以可以湊得**最大的價位**就是每個硬幣都用過一次 Σv[k]

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

v[k] = 5

i = 5

dp[7] 更新值為1
因為價位 7可以被湊出來

- dp[i] = 1 ➔ 可以湊出價位 i

Coin change

- 硬幣無限，是否湊得某價位
- dp[0]=1
- v[k] = 2, 5
- If (dp[i - v[k]] == true) dp[i] = 1; (i 從v[k]開始跑)

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

v[k] = 2

i = 8

dp[2] 更新值為1
因為價位 2可以被湊出來

要注意 i 的邊界值，只需要從 v[k] 跑到 Max

Coin change

- 硬幣無限，是否湊得某價位
- dp[0]=1
- v[k] = 2, 5
- If (dp[i - v[k]] == true) dp[i] = 1; (i 從v[k]開始跑)

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

v[k] = 5

i = 7

dp[5] 更新值為1
因為價位 5可以被湊出來

Coin change

- 硬幣無限，湊得某價位有幾種可能
- dp[0]=1
- v[k] = 2, 3
- If (dp[i - v[k]] == true) dp[i] += dp[i - v[k]];

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

v[k]=2, 作法同p.7

	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，湊得某價位有幾種可能
- dp[0]=1
- v[k] = 2, 3
- If (dp[i - v[k]] == true) dp[i] += dp[i - v[k]];

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

v[k] = 3

i = 6

價位 3 的可能組合有 dp[3] 種
每一種加上硬幣 3 之後皆可組成價位 6
所以組成價位 6 的可能組合
多了dp [3] 種

dp[6] = dp [6] + dp [3] ;

made by aj211y

Practice

- 基礎：Uva 674
- 進階：Uva 10898

0/1 Knapsack Problem

- Knapsack Problem：背包問題
  將一堆物品塞進背包，要使背包裡的**物品總價值最高**，但背包有**耐重限制**，所以塞的太重的話，背包就會撐破。
- 0/1：
  物品只會放進背包**0個或1個**，物品不可切割，所以只有不放或者全放兩種可能。

0/1 Knapsack Problem

- 窮舉法
  每個物品有放或不放2種選擇，有N個物品，所有的情況列舉需O(2^N)
- DP
  對某個物品可以選擇放或不放，然後移去這個物品來縮小問題範圍，時間複雜度 O(NW)。(N種物品，W是限重)

0/1 Knapsack Problem

- dp[n][m]：
  前n項物品，在背包載重m的情況下，所能擁有的**最大價值**
- v[n]： 物品n的價值
- w[n]：物品n的重量
- dp[n][m] =
  max (dp[n-1][m], dp[n-1][m-w[n]]+v[n])

以dp[n-1][m’]來模擬dp[n][m]的情況

不放物品 n 的情況下
背包載重m’=m 不變

放了物品 n 的情況下
背包載重m’ = m – w[n]
最大價值要加上物品n 的value

0/1 Knapsack Problem

- dp[n][m] =
  max (dp[n-1][m], dp[n-1][m-w[n]]+v[n])
- 邊界限制
  dp[n][m]=
  { 0 , n==0} 沒有物品
  {dp[n-1][m] , n > 0 && m-w[n] < 0} 有物品但背包放不下
  {max (dp[n-1][m], dp[n-1][m-w[n]]+v[n])
  ,n>0 && m-w[n]>=0}}
  有物品且背包還有空間可以放

0/1 Knapsack Problem

- Top-down DP

```
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
- c[m]：背包載重m的情況下，所能擁有最大的價值

```
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 m = W; m - w[i] >= 0; m--)        // Back to the front
            c[m] = max( c[m], c[m - w[i]] + v[i] );// Update lookup
                                                   // table
    return c[w];
}
```

0/1 Knapsack Problem

- c[m] = max(c[m], c[m - w[i]] + v[i]);

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

W
6

max(c[5] , c[5 - 3] + v[0]) = 10

初始化為零
c[0]=0

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]
0	0	0	10	10	10	10

因為m=0表示
背包能載重0
故背包最大價值0

0/1 Knapsack Problem

- c[m] = max(c[m], c[m - w[i]] + v[i]);

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

W
6

max(c[5] , c[5 - 1] + v[1]) = 30

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]
0	20	20	20	30	30	30

0/1 Knapsack Problem

- c[m] = max(c[m], c[m - w[i]] + v[i]);

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

W
6

max(c[5] , c[5 - 2] + v[2]) = 50

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

ans

Practice

- POJ-3624

Thank for Your Attention

made by aj211y