Computer Programming 2 Lab

2023-05-03

Mozix Chien

Outline

- Dynamic Programming
  - 非波那契數列
  - Coin Change
  - 背包問題
  - Minimum Path Sum
- Homework & Exercise
  - HW
  - EX

Dynamic Programming

Dynamic Programming

非波那契數列

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

```
fib(n) = fib(n-1) + fib(n-2)
```

Dynamic Programming

非波那契數列

遞迴求解

```
int fib(n) {
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;

    return fib(n-1) + fib(n-2);
}
```

- 效能如何？

Dynamic Programming

非波那契數列

遞迴求解 + 陣列紀錄

```
int fibo[N+1] = {0, 1, -1, -1, -1 ...};
int fib(n) {
    if (n == 0)
        return fibo[0];
    if (n == 1)
        return fibo[1];

    if (fibo[n] != -1)
        return fibo[n]
    else
        return fibo[n] = fib(n-1) + fib(n-2);
}
```

- 效能如何？

Dynamic Programming

非波那契數列

陣列迭代

```
int fib(n) {
    int fibo[n+1] = {0, 1};
    for(int i=2;i<=n;i++)
        fibo[i] = fibo[i-1] + fibo[i-2];
    return fibo[n];
}
```

- 效能如何？

Dynamic Programming

- 透過將問題轉化為數個子問題，遞推來求解
- 可以先試著統整歸納「遞推式」，在將其轉為程式碼

Dynamic Programming

Coin Change

給定數個金幣價值 `coins` ，以及目標價值 `amount` ，問 `amount` 可由多少最少的金幣組成，如果無法組成輸出 -1

輸入

```
coins = [1, 3, 4]
amount = 6
```

輸出

```
2
```

Dynamic Programming

Coin Change

給定數個金幣價值 `coins`，以及目標價值 `amount`，問 `amount` 可由多少最少的金幣組成，如果無法組成輸出 -1

考慮今天沒有任何金幣

```
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
-----------------------------
| 0 | N | N | N | N | N | N |
```

- N 表示無限大、不可能達成

Dynamic Programming

Coin Change

給定數個金幣價值 `coins`，以及目標價值 `amount`，問 `amount` 可由多少最少的金幣組成，如果無法組成輸出 -1

考慮今天有金幣 1

```
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
-----------------------------
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
```

- N 表示無限大、不可能達成

Dynamic Programming

Coin Change

給定數個金幣價值 `coins`，以及目標價值 `amount`，問 `amount` 可由多少最少的金幣組成，如果無法組成輸出 -1

考慮今天有金幣 1、3

```
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
-----------------------------
| 0 | 1 | 2 | 1 | 2 | 3 | 2 |
```

- 其中 V[3] 本來為 3，但其更好的解法應該是從 V[0] 加上 1 個價值為 3 的金幣
- 即 V[3] = V[0] + 1

Dynamic Programming

Coin Change

給定數個金幣價值 `coins`，以及目標價值 `amount`，問 `amount` 可由多少最少的金幣組成，如果無法組成輸出 -1

考慮今天有金幣 1、3、4

```
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
-----------------------------
| 0 | 1 | 2 | 1 | 1 | 2 | 2 |
```

- 其中 V[5] 本來為 3，但其更好的解法應該是從 V[1] 加上 1 個價值為 4 的金幣
- 即 V[5] = V[1] + 1

Dynamic Programming

Coin Change

給定數個金幣價值 `coins`，以及目標價值 `amount`，問 `amount` 可由多少最少的金幣組成，如果無法組成輸出 -1

考慮今天有金幣 1、3、4

```
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
-----------------------------
| 0 | 1 | 2 | 1 | 1 | 2 | 2 |
```

- 其中 V[6] 本來為 2，如果我們考慮 V[2] 加上 1 個價值為 4 的金幣
- 即 V[6] = V[2] + 1，效果並沒有比原本的 2 枚硬幣來的好

Dynamic Programming

Coin Change

給定數個金幣價值 `coins`，以及目標價值 `amount`，問 `amount` 可由多少最少的金幣組成，如果無法組成輸出 -1

遞推式

```
V[i] = min(V[i-C] + 1, V[i])
```

- V 為目標價值，C 為硬幣價值
- 亦即 **考慮目標價值為 V 時，從價值 V - C 的數量再加 1 會不會比原本來的還要好**

Dynamic Programming

0 1 背包問題

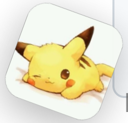
有 n 個重量與價值分別是 w_i 和 v_i 的物品。想裝入一個最大限重為 W 的背包，想求背包可裝入的最大價值。

輸入

```
n = 4
物品 [(w, v)] = [(2, 3), (1, 2), (3, 4), (2, 2)]
W = 5
```

輸出

```
7 # 裝入 0, 1, 3 號物品
```



Dynamic Programming

0 1 背包問題

有 n 個重量與價值分別是 w_i 和 v_i 的物品。想裝入一個最大限重為 W 的背包，想求背包可裝入的最大價值。

直接爆搜

- 枚舉所有物品放入背包的狀況，找其最佳解
- 時間複雜度 $O(2^n)$

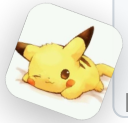
Dynamic Programming

0 1 背包問題

直接爆搜

```
int w[n] = {2, 1, 3, 2};
int v[n] = {3, 2, 4, 2};
int sol(int k, int cur_w){
    if(k == n) // 走到底了
        return 0;
    if(cur_w + v[k] > W) // 背包裝不下這個物品了
        return sol(k+1, cur_w);

    return max(
        sol(k+1, cur_w+w[k]) + v[k],    // 選擇這個物品
        sol(k+1, cur_w)                 // 捨棄這個物品
    );
}
```



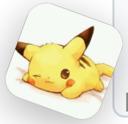
01背包問題

直接爆搜 + 陣列紀錄

```
int w[n] = {2, 1, 3, 2};
int v[n] = {3, 2, 4, 2};
int dp[n][W];
int sol(int k, int cur_w){
    if(dp[k][cur_w] >= 0)
        return dp[k][cur_w];

    if(k == n) // 走到底了
        return 0;
    if(cur_w + v[k] > W) // 背包裝不下這個物品了
        return dp[k][cur_w] = sol(k+1, cur_w);

    return dp[k][cur_w] = max(
        sol(k+1, cur_w+w[k]) + v[k],    // 選擇這個物品
        sol(k+1, cur_w)                 // 捨棄這個物品
    );
}
```



Dynamic Programming

01背包問題

動態規劃

```
dp[i][j] = dp[i-1][j] // 背包重量裝不下這個物品
         = max(
             dp[i-1][j-w[i]] + v[i], // 裝看看這個物品
             dp[i-1][j]              // 不裝這個物品
         )
```

- 時間複雜度 $O(n * W)$

0 1 背包問題

動態規劃

考慮物品 0 (2, 3)

i\j	0	1	2	3	4	5
-1	0	0	0	0	0	0
0	0	0	3	3	3	3
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0

0 1 背包問題

動態規劃

考慮物品 1 (1, 2)

i\j	0	1	2	3	4	5
0	0	0	3	3	3	3
1	0	2	2	5	5	5
2	0	0	0	0	0	0
3	0	0	0	0	0	0

01背包問題

動態規劃

考慮物品 2 (3, 4)

i\j	0	1	2	3	4	5
0	0	0	3	3	3	3
1	0	2	3	5	5	5
2	0	2	3	5	6	7
3	0	0	0	0	0	0

0 1 背包問題

動態規劃

考慮物品 3 (2, 2)

i\j	0	1	2	3	4	5
0	0	0	3	3	3	3
1	0	2	3	5	5	5
2	0	2	3	4	6	7
3	0	2	3	4	6	7

01背包問題

動態規劃

```
int w[n] = {2, 1, 3, 2};
int v[n] = {3, 2, 4, 2};
int dp[n][W+1];
for(int i=0;i<n;i++){
    for(int j=W;j>=0;j--){
        if(j-w[i] < 0){
            dp[i][j] = dp[i-1][j]
        }else{
            dp[i][j] = max(
                dp[i-1][j-w[i]] + v[i],
                dp[i-1][j]
            )
        }
    }
}
return dp[n-1][W];
```

Dynamic Programming

多重背包問題

有 n 個重量與價值分別是 w_i 和 v_i 的物品，**物品可重複拾取**。想裝入一個最大限重為 W 的背包，想求背包可裝入的最大價值。

輸入

```
n = 4
物品 [(w, v)] = [(2, 3), (1, 2), (3, 4), (2, 2)]
W = 5
```

輸出

```
10 # 裝入 5 個 1 號物品
```

Dynamic Programming

可重複拾取背包問題

1. 爆搜 ... 如果物品很多（n很大）搜得完嗎？
2. Greedy ... 這樣還對嗎？
    - n = 2, [(5, 5), (4, 3)], W = 8
3. 那就是動態規劃

Dynamic Programming

Minimum Path Sum

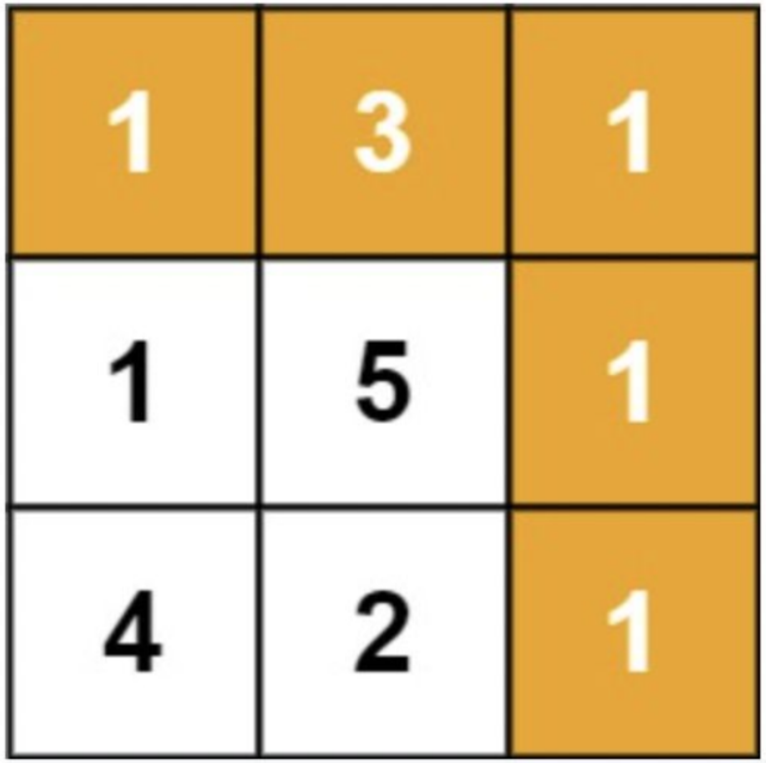
給定一個 `m x n` 的 `grid` ，你只能從左上角開始，每次往右或往下移動，問當你走到右下角時，數字的最小總和為多少？

輸入

```
grid = [[1,3,1],[1,5,1],[4,2,1]]
```

輸出

```
7
```



Dynamic Programming

Minimum Path Sum

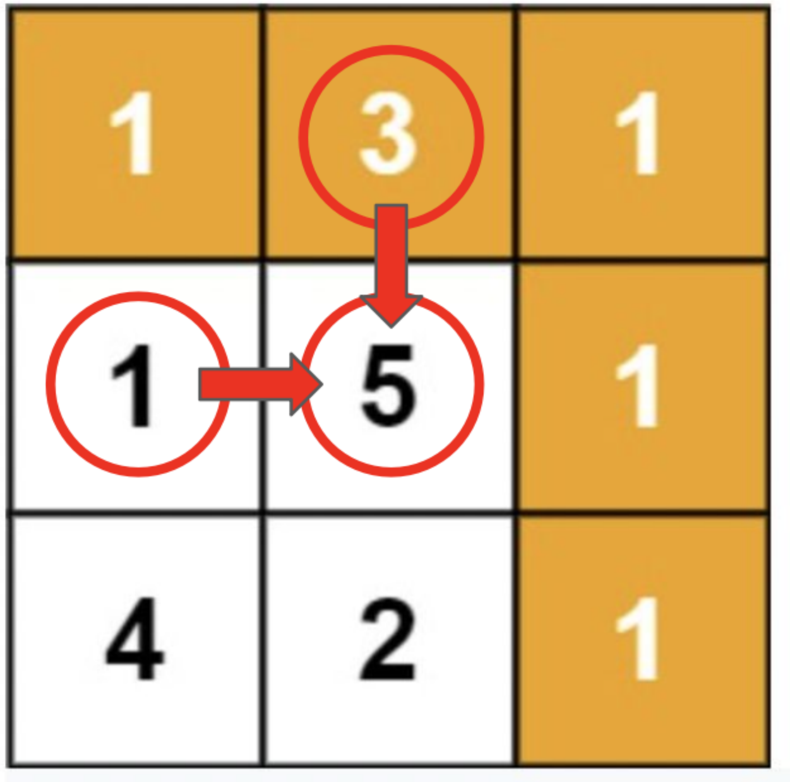
1. 爆搜 ... $O(\frac{(m+n)!}{m! \times n!})$
2. 那就是動態規劃力

1	3	1
1	5	1
4	2	1

Dynamic Programming

Minimum Path Sum

- 只能往右或往下，反過來說，對於當前的格子 `grid[i][j]` ，他的來源只可能是 `grid[i-1][j]` 或 `grid[i][j-1]` 。
- 遞推式： `grid[i][j] = grid[i][j] + min(grid[i-1][j], grid[i][j-1])`
- $O(m \times n)$



Homework & Exercise

Arbitrage

Arbitrage

Target

- Determine whether a sequence of exchanges exists that results in a profit of more than 1 percent (0.01)
- If so, print out the sequence of exchanges that results in this profit (the shortest one)
- If not, print out "***no arbitrage sequence exists***"

Arbitrage

Example

Given the exchange rates' table below:

```
1.0   1.2   0.89
0.88  1.0   5.1
1.1   0.15  1.0
```

Output:

```
1 2 1
```

The sequence of exchanges that results in a profit of more than 1 percent is:

```
1 -> 2 -> 1
1.0 * 1.2 * 0.88 = 1.056
1 -> 3 -> 1
1.0 * 0.89 * 1.1 = 0.979

2 -> 1 -> 2
1.0 * 0.88 * 1.2 = 1.056
2 -> 3 -> 2
1.0 * 5.1 * 0.15 = 0.765

3 -> 1 -> 3
1.0 * 1.1 * 0.89 = 0.979
3 -> 2 -> 3
1.0 * 0.15 * 5.1 = 0.765
```

Arbitrage

Another Example

Given the exchange rates' table below:

```
1.0  2.0
0.45 1.0
```

The sequence of exchanges that results in a profit of ther is no more than 1 percent:

```
1 -> 2 -> 1
1.0 * 2.0 * 0.45 = 0.9

2 -> 1 -> 2
1.0 * 0.45 * 2.0 = 0.9
```

Output:

```
no arbitrage sequence exists
```

Perfect Squares

Any Questions?