

# Informatics II

## Exercise 11

March 16, 2022

### Dynamic Programming

#### Task 1

1. Coin change problem can be solved by
  - A. Greedy algorithm, can obtain the global optimal solution
  - B. Divide and conquer recursion, can obtain the global optimal solution
  - C. Dynamic programming, can obtain the global optimal solution
  - D. Greedy algorithm, can not obtain the global optimal solution
2. Consider for coin change of amount 14 with coins 10, 7, 2, 1. Give a solution with least coins by dynamic programming.

#### Task 2

Consider a 2-D matrix  $M$ . Value of each cell of  $M$  is a wall 'W' or a bug 'B' or empty '0'. You can place a bomb at any empty cell with value '0'. The bomb will explode in the following four directions: up, down, left and right. The explosions will be stopped by the walls or the border of the matrix. Consequently, bugs that are covered by the explosions will be eliminated.

We use the following:

- $V_l[i][j]$  the total number of bugs when walking in the *left* direction of grid( $i, j$ ) (included) before hitting the wall.
- $V_r[i][j]$  the total number of bugs when walking in the *right* direction of grid( $i, j$ ) (included) before hitting the wall.
- $V_u[i][j]$  the total number of bugs when walking in the *up* direction of grid( $i, j$ ) (included) before hitting the wall.
- $V_d[i][j]$  the total number of bugs when walking in the *down* direction of grid( $i, j$ ) (included) before hitting the wall.

When placing the bomb at the position  $(i, j)$  in  $M$ , we calculate the number of eliminated bugs in sum of four corresponding values, *i.e.*,  $V_l[i][j] + V_r[i][j] + V_u[i][j] + V_d[i][j]$ . For example, when we place bomb at "Empty"  $M[1][1]$ , the number of eliminated bugs would be  $V_l[1][1] + V_r[1][1] + V_u[1][1] + V_d[1][1] = 1 + 0 + 1 + 1 = 3$ . The four matrices  $V_l$ ,  $V_r$ ,  $V_u$  and  $V_d$  have the same dimensions as  $M$ .

	0	1	2	3
0	0	B	0	0
1	B	0	W	B
2	0	B	0	0

Figure 1: Illustration of  $M_1$  with "Bug", "Wall" and "Empty"

	0	1	2	3
0	0	1	1	1
1	1	1	0	1
2	0	1	1	1

$V_l$  for left direction

	0	1	2	3
0	1	1	0	0
1	1	0	0	1
2	1	1	0	0

$V_r$  for right direction

	0	1	2	3
0	0	1	0	0
1	1	1	0	1
2	1	2	0	1

$V_u$  for up direction

	0	1	2	3
0	1	2	0	1
1	1	1	0	1
2	0	1	0	0

$V_d$  for down direction

Figure 2: Illustration of solution matrices

**Task 2.1** Fill in four direction matrices below.

	0	1	2	3
0	0	0	0	B
1	0	W	0	0
2	0	0	W	0
3	B	B	0	0

Figure 3: Illustration of  $M_2$  with "Bug", "Wall" and "Empty"

	0	1	2	3
0				
1				
2				
3				

$V_l$  for left direction

	0	1	2	3
0				
1				
2				
3				

$V_r$  for right direction

	0	1	2	3
0				
1				
2				
3				

$V_u$  for up direction

	0	1	2	3
0				
1				
2				
3				

$V_d$  for down direction

Figure 4: Illustration of solution matrices

**Task 2.2** Write down Recursive problem formulation for  $V_l[i][j]$ ,  $V_r[i][j]$ ,  $V_u[i][j]$  and  $V_d[i][j]$  for grid  $M[i][j]$ .

**Task 2.3** Implement the C function `calculate(M, x, y)` that returns the maximum number of eliminated bugs when a bomb is placed in  $M$ . The matrix  $M$  has the dimension  $x \times y$ .

### Task 3

A parentheses string contains only left and right parentheses. A parentheses string is balanced if and only if left and right parentheses are correctly matched. For example, the strings `"()"` and `"()()"` are balanced while `"())()"` is not. Given a parentheses string  $S[0 \dots n-1]$  with  $n$  parentheses, determine the length of **longest** balanced subsequence of string  $S$ . The **subsequence** is derived by selecting/not selecting elements, without changing the order of the selected elements. For example, for the parentheses string `"()()()"`, the length of longest balanced subsequence is 4. We choose the 1st, 2nd, 5th, and 6th characters to form an balanced subsequence `"()()"`.

Let  $dp$  be a 2-D matrix with the dimension  $(n+1) \times (n+1)$  for the parentheses string  $S[0 \dots n-1]$ , and  $dp[i,j]$  be the length of longest balanced subsequence of subarray  $S[i \dots j]$ .

	0	1	2	3
0	0	0	2	4
1	0	0	2	2
2	0	0	0	0
3	0	0	0	0

Matrix  $dp'$  for string `"()()"`

	0	1	2	3
0	0	2	2	4
1	0	0	0	2
2	0	0	0	4
3	0	0	0	0

Matrix  $dp''$  for sting `"()()()"`

Figure 5: Illustration of solution matrix  $D$

**Task 3.1** Fill in  $dp'''$  for string `"((()))"`.

	0	1	2	3	4	5
0	0					
1	0	0				
2	0	0	0			
3	0	0	0	0		
4	0	0	0	0	0	
5	0	0	0	0	0	0

Figure 6: Matrix  $dp'''$  for sting `"((()))"`

**Task 3.2** Write down the recursive problem formulation for  $D[i, j]$ .

**Task 3.3** Write C function `calculate(S, n)` that return the length of the longest balanced subsequence of  $S$ .