

# COMP3027 ASM 2

490399992

April 2021

## Task 1

### a) Define the subproblems needed by your algorithm

Consider the array  $R$  of locations. Define  $OPT(j)$  as the value of the optimal solution to the subproblem consisting of locations  $R[0, 1, 2, \dots, j]$  with index  $j$  representing the potential bus location. When  $j = 0$ , this corresponds to an empty set of locations.

### b) Define the recurrence

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ R[j] & \text{if } j < d \\ \max\{OPT(j-d) + R[j], OPT(j-1)\} & \text{otherwise} \end{cases}$$

### c) Define the base cases

There are two base cases. The first case is  $OPT(0) = 0$ , representing an empty street. The second case is  $OPT(j) = R[j]$  when  $j < d$ . This is when there is no eligible location to the left of  $j$  that satisfies the social distancing requirement.

### d) Describe how the maximum revenue is obtained from $OPT(..)$

The maximum revenue is obtained with the subproblem  $OPT(n-1)$ ,  $n$  being the size of  $R$ .

### e) Prove correctness

The base cases are sufficient as  $OPT(j)$  depends on some  $OPT(j-d)$ . Thus, the recurrence will always end up at either  $OPT(0)$  or  $OPT(j)$  where  $j < d$ . The first case represents the start of the street, and the second case represents the case where no further eligible locations exist to the left of  $j$ . We have  $OPT(0) = 0$ , as there are no eligible locations so no revenue can be earned, and  $OPT(j) = R[j]$  where  $j < d$  as  $R[j]$  is the last eligible location.

We now prove the correctness of the recurrence. Let  $S(j)$  be the optimal solution for the first  $j$  locations. The recurrence for  $OPT(j)$  is correct because:

- In any situation, either location  $j$  is part of the solution or it is not
  - If location  $j$  is in  $S(j)$ , then the remaining locations of  $S(j)$  is a subset of the first  $j-d$  locations, so  $S(j) = S(j-d) + R[j]$  and so  $OPT(j) = OPT(j-d) + R[j]$ .
  - If location  $j$  is not in  $S(j)$ , then  $S(j)$  must be a subset of the first  $j-1$  items, so  $S(j) = S(j-1)$  and  $OPT(j) = OPT(j-1)$ .

Finally, the optimal value for the original problem is  $OPT(n-1)$  as by the definition of  $OPT$ , this is the most value that can be achieved by a subset of all  $n$  items in  $R$ .

### f) Prove time complexity

- Space complexity: The space required by our algorithm is dominated by the number of subproblems, which is  $O(n)$  as we have subproblem  $OPT(j)$  for each  $0 \leq j < n$ .
- Time complexity: The base cases take time  $O(n)$  as each base case takes constant time, however there are up to  $n$  base cases (in the case where  $d > n$ ). For the recursive cases, there are  $O(n)$  iterations and each iteration takes  $O(1)$  time since it only requires a constant number of comparisons, arithmetic operations and array lookups. Thus, the recursive cases takes time  $O(n)$ . Returning the optimal value takes  $O(1)$  time for an array lookup. Thus, the total time is  $O(n) + O(n) + O(1) = O(n)$ .

## Task 2

### a) Define the subproblems needed by your algorithm

Consider the grid  $S$ . Define  $OPT(i, j)$  to be the value of the optimal solution to the subproblem consisting of the grid position  $(i, j)$  with  $i$  representing the column of the grid and  $j$  representing the row, indexed from 0. When  $(i, j) = (0, 0)$ , this represents the bottom left corner of the grid.

### b) Define the recurrence

$$OPT(i, j) = \begin{cases} S[0,0] & \text{if } i = 0 \text{ and } j = 0 \\ -1 & \text{if } ic_r + jc_u > B \\ \max\{OPT(i-1, j) + S[i, j], OPT(i, j-1) + S[i, j]\} & \text{otherwise} \end{cases}$$

### c) Define the base cases

There are two base cases. The first base case is the bottom left position in the grid, where  $OPT(0, 0) = S[0, 0]$ . The second case is  $OPT(i, j) = -1$  where  $ic_r + jc_u > B$ . This is where the position cannot be reached with the given energy budget  $B$ .

### d) Describe how the maximum total score is obtained from $OPT(..)$

The maximum score is obtained by the maximum  $OPT$  of all positions in the grid from the bottom left position,  $OPT(0, 0)$ , to the top right position,  $OPT(n-1, m-1)$  where  $n$  is the grid's width and  $m$  is the grid's height.

### e) Prove correctness

The base cases are sufficient as  $OPT(i, j)$  always depends on either the position directly left,  $OPT(i-1, j)$ , or below,  $OPT(i, j-1)$ . The first base case to consider is where the position cannot be reached with the given energy budget. As position  $(i, j)$  can only be reached with  $i$  "right" moves and  $j$  "up" moves, a position will be unable to be reached if  $ic_r + jc_u > B$ . In this case,  $S(i, j) = -1$  as there is no valid maximum score, and so  $OPT(i, j) = -1$ .

The second base case is for the bottom left position. As each stage of the recurrence includes either the position to the left or below as a subproblem, if the position can be reached within the budget, the recurrence will always reach  $OPT(0, 0)$ . This is the only position where there does not exist a position to the left or below. In this case, there is only one position to be considered, which has a cost of 0 and so will always be selected, so  $OPT(0, 0)$  is the score  $S[0, 0]$ . As each position can either be reached or not reached within the budget, all positions will eventually reach a base case.

We now prove the correctness of the recurrence. Let  $S(i, j)$  be the optimal solution for the position  $(i, j)$  (that can be reached within the energy budget). The recurrence for  $OPT(i, j)$  is correct because in any situation, the only two options to reach  $(i, j)$  is either a right move from the position to the left, or an up move from the position below. As the cost to reach  $(i, j)$  is constant, whichever selection we make will not affect the cost of other positions. Thus, the best option will be whichever path will yield the highest score:

- if the left item has a larger maximum score and is selected by  $S(i, j)$ , then  $S(i, j) = S(i-1, j) + S[i, j]$  and so  $OPT(i, j) = OPT(i-1, j) + S[i, j]$
- if the item below has a larger maximum score and is selected by  $S(i, j)$ , then  $S(i, j) = S(i, j-1) + S[i, j]$  and so  $OPT(i, j) = OPT(i, j-1) + S[i, j]$

Finally, the optimal value for the original problem is the maximum  $OPT$  in of all positions in the grid, as by the definition of  $OPT$ , this represents the position that yields the maximum score that can be traversed to within the energy budget.

### f) Prove time complexity

- Space complexity: The space required by our algorithm is dominated by the number of subproblems, which is  $O(nm)$  as we have subproblem  $OPT(i, j)$  for each  $0 \leq i < n$  and  $0 \leq j < m$ .
- Time complexity: The base case for  $OPT(0, 0)$  takes time  $O(1)$  as the base case takes constant time, and only needs to be computed once. For the recursive cases (including the other base case), there are  $O(nm)$  iterations and each iteration takes  $O(1)$  time since it only requires a constant number of comparisons, arithmetic operations and array lookups. Thus, the recursive cases takes time  $O(nm)$ . Returning the optimal value takes  $O(nm)$  time for an array traversal to find the maximum  $OPT$ . Thus, the total time is  $O(1) + O(nm) + O(nm) = O(nm)$ .