CS 374 HW 5 Problem 3

Hieu Huynh, Aldo Sanjoto, quddus2

TOTAL POINTS

98 / 100

QUESTION 1

1 3A **50 / 50**

✓ - **0 pts** **Correct**

- **20 pts** algorithm implementation incorrect
- **20 pts** recursive idea incorrect
- **10 pts** time analysis incorrect
- **37.5 pts** IDK
- **50 pts** Illegible/Unreadable
- **10 pts** slight error in recursive idea
- **10 pts** slight error in algorithm
- **5 pts** base case of recursion not specified/incorrect
- **12.5 pts** correctness of idea not argued/wrong argument
- **40 pts** Illegible/Unreadable

QUESTION 2

2 3B **8 / 10**

- **0 pts** Correct

✓ - **2 pts** **Minor error in algorithm**

- **7.5 pts** IDK
- **0 pts** Click here to replace this description.
- **5 pts** recursive step modification not specified/wrong
- **5 pts** base case modification not specified/wrong
- **10 pts** unreadable/illegible

What if i is n or j is m

QUESTION 3

3 3C **40 / 40**

✓ - **0 pts** **Correct**

- **10 pts** Small errors in algorithm/idea
- **10 pts** Time analysis wrong/not specified
- **30 pts** IDK
- **20 pts** recursive idea wrong
- **20 pts** Algorithm wrong

(Q3)

a) Let isPossible(i,j) denote whether it is possible to move from (p_i, q_j) to (p_n, q_m) or not. This function obeys the following recurrence

$$\text{isPossible}(i,j) = \begin{cases} \text{True}, & \text{if } i=n \text{ and } j=m \\ \text{False}, & \text{if } i>n \text{ or } j>m \\ \text{isPossible}(i,j+1) \wedge (d(i,j+1) \le \ell), & \text{if } i=n \\ \text{isPossible}(i+1,j) \wedge (d(i+1,j) \le \ell), & \text{if } j=m \\ (\text{isPossible}(i,j+1) \wedge d(i,j+1) \le \ell) \vee (\text{isPossible}(i+1,j) \wedge d(i+1,j) \le \ell), & \text{otherwise.} \end{cases}$$

- We need to compute isPossible$(1,1)$.
- We can memoize the function isPossible into a 2-Darray $A[1..n, 1..m]$
Each entry $A[i,j]$ depends on entry in next column $A[i,j+1]$, and entry in next row $A[i+1,j]$ (if they exist).
So, we can fill the array from right to left in a row, and move to the row above when finish.

⟹ code is on next page

```
isPossible(d[1...n, 1...m], l, n, m){
    A[1...n, 1...m]

    For j ← m to 1 {
        For i ← n to 1 {
            if (i==n && j==m) { //Base case
                A[i,j] = True;
            }
            else{
                temp = False
                if (i+1 ≤ n) {
                    if (d[i+1,j] ≤ l){
                        temp = temp ∨ A[i+1,j];
                    }
                }
                If (j+1 ≤ m) {
                    if (d[i,j+1] ≤ l){
                        temp = temp ∨ A[i,j+1];
                    }
                }
                A[i,j] = temp;
            }
        }
    }
    return A[1,1];
}
```

This block takes $O(1)$

Running Time Analysis: To fill 1 entry in the array, it takes $O(1)$. And we have $m \cdot n$ entries

$\Rightarrow$ The total running time is $O(m \cdot n)$

Running Time Analysis:

To fill 1 entry in the table, it takes $O(1)$.

And we have $m \cdot n$ entries

$\Rightarrow$ Total running time is $O(m \cdot n)$.

1 3A **50 / 50**

✓ **- 0 pts** **Correct**

- 20 pts algorithm implementation incorrect

- 20 pts recursive idea incorrect

- 10 pts time analysis incorrect

- 37.5 pts IDK

- 50 pts Illegible/Unreadable

- 10 pts slight error in recursive idea

- 10 pts slight error in algorithm

- 5 pts base case of recursion not specified/incorrect

- 12.5 pts correctness of idea not argued/wrong argument

b) After filling the array A[1..n, 1..m], we check if A[1,1] is True or False. If it's False, it's impossible to schedule. If it's true, we follow the following rules: when we are at entry A[i,j], if A[i+1,j] == True, output (i+1, j), and move to entry A[i+1,j]. Else if A[i,j+1] == True output (i, j+1) and move to entry A[i, j+1]. Stop when we reach (n, m). We start at A[1,1]. Note: A[i,j+1] doesn't exist, we go to A[i+1,j]

```
Find_Schedule (d[1...n, 1...m], l, n, m) {
    // Repeat code in part (A) to fill table A[1..n, 1..m].
    if (A[1,1] == False) {
        return: impossible to schedule
    }
    else {
        result [1, ... m+n-1].        // m+n-1 pair of (i, j)
        result.add (1, 1);
        i ← 1 ; j ← 1
        while (i ≠ n && j ≠ m) {
            if (i < n && A[i+1, j] == True) {
                result.add (i+1, j);
                i++;
            }
            else {
                result. add (i, j+1);
                j++;
            }
        }
        return result;
    }
}
```

Running time Analysis:

This algorithm takes $O(n \cdot m)$ time. Because to fill the table, we need $O(n \cdot m)$ time. To traverse through the table to get the schedule take less than $O(nm)$

2 3B 8 / 10

- 0 pts Correct

✓ - 2 pts **Minor error in algorithm**

- 7.5 pts IDK
- 0 pts Click here to replace this description.
- 5 pts recursive step modification not specified/wrong
- 5 pts base case modification not specified/wrong
- 10 pts unreadable/illegible

What if i is n or j is m

Q3 C)

• We modify the table $A[1...n, 1...n]$ in part (A), such that the entry $A[i][j]$ stores the minimum threshold such that it is possible to move from (p_i, q_j) to (p_n, q_m). We have the following recurrence:

$$MinL(i,j) = \begin{cases} 0 & \text{, if } i=n \text{ \&\& } j=m \\ \infty & \text{, if } i>n \text{ or } j>m \\ \max(d[i,j+1], MinL[i,j+1]) & \text{if } i=n \\ \max(d[i+1,j], MinL[i+1,j]) & \text{if } j=m \\ \min\begin{cases} \max(d[i,j+1], MinL[i,j+1]) \\ \max(d[i+1,j], MinL[i+1,j]) \end{cases} & \text{, otherwise.} \end{cases}$$

• We need to comput $MinL(1,1)$.
• We can fill the table in the same way we did in part (A)

```
MinL(d[1...n, 1...m], m, n){
    A[1...n, 1...m];
    For j ← m to 1{
        For i ← n to 1{
            if(j == m && i == n)   // base case
                A[i,j] = 0
            else{
                l1 ← ∞ ; l2 ← ∞ ;
                if(i+1 ≤ n)
                    l1 = max(d[i+1,j], A[i+1,j]);
                if(j+1 ≤ m)
                    l2 = max(d[i,j+1], A[i,j+1]);
                A[i,j] = min(l1, l2);
            }
        }
    }
    return A[1][1];
}
```

This block takes O(1) time

Running Time Analysis:

To fill 1 entry in the table, it takes $O(1)$.

And we have $m \cdot n$ entries

$\Rightarrow$ Total running time is $O(m \cdot n)$.

3 3C **40 / 40**

✓ **- 0 pts** Correct

- 10 pts Small errors in algorithm/idea

- 10 pts Time analysis wrong/not specified

- 30 pts IDK

- 20 pts recursive idea wrong

- 20 pts Algorithm wrong

- 40 pts Illegible/Unreadable