CS 3510 – Assignment 3

Due `Friday, June 17, 2022` at `11:59pm` on Canvas

Instructor: Shahrokh Shahi

- Please type your answers (LaTeX is highly recommended) and upload a single PDF file named `<Your-GT-Account>.pdf`, e.g., GBurdell3.pdf, including all your answers. You can submit multiple times. Canvas keeps track of the submissions and append a version number when you re-submit. We always grade your most recent submissions.
- Please read the policies, and do not forget to acknowledge your collaborators and cite your references.
- If you do not understand a question, please ask on Piazza or come to office hours well ahead of the due date.
- It is recommended to start reviewing the course material by reading the lecture slides and reviewing the demo codes. Then, the suggested readings from textbooks and solving the practice problems can provide the additional preparation for solving the homework problems. Please note, for the textbook readings, you do not need to cover the topics which have not been covered in the lectures.

Suggested Reading

	CLRS	KT
Section(s)	Chapter 15	Chapter 6

Suggested Practice Problems

	CLRS	KT
Practice problems	Exercise: 15.1-3, 15.1-4 Problems: 15-2, 15-3	Solved Exercise: 1 Exercise: 1-10, 19-21

Additional reading and problems:
– DPV (Chapter 6)

1 Dynamic Programming: Binary Board [25 pts]

Given a binary matrix B of size $n \times m$, where the entries are either 0 or 1, design a dynamic programming algorithm to find the maximum width w of a square of ones in B, as well as the coordinates (x, y) of the top left corner of such a square. Therefore, for all i and j such that $x \leq i < x + w$ and $y \leq j < y + w$, we have $B[i, j] = 1$.

a) (10 pts) Design a dynamic programming algorithm to find the maximum width w and the corresponding top left coordinate (x, y). (Provide the recurrence relation including the base case(s)).

b) (10 pts) Write a pseudocode presenting your algorithm (bottom-up or top-down).

c) (5 pts) Analyze the time and space complexity of your algorithm.

(Hint: For solving this problem, you may consider $OPT[i, j]$ as the width of the largest square of ones whose top left corner is $B[i, j]$.)

Solution

a) Let's define $S_{i,j}$ as the largest square of ones in the given binary board B whose top left corner is $B[i, j]$, and define $OPT[i, j]$ as the width of $S_{i,j}$. If $B[i, j] = 0$, then the width of the square $S_{i,j}$ is zero, thus $OPT[i, j] = 0$. Otherwise, for all the elements of $B[i', j']$ within the range of $i \leq i' < i + OPT[i, j]$ and $j \leq j' < j + OPT[i, j]$ should be equal to 1. Therefore, for $S_{i,j}$ to have the width of $OPT[i, j]$, then

- $S_{i,j+1}$ must have a size at least $OPT[i, j] - 1$.
- $S_{i+1,j}$ must have a size at least $OPT[i, j] - 1$.
- $S_{i+1,j+1}$ must have a size at least $OPT[i, j] - 1$.

Therefore, for all $1 \leq i \leq n$ and $1 \leq j \leq m$, the recurrence relation will be as follows:

$$OPT[i, j] = \begin{cases} 0 & \text{if } B[i, j] = 0 \\ 1 + \min \begin{cases} OPT[i, j + 1], \\ OPT[i + 1, j], \\ OPT[i + 1, j + 1] \end{cases} & \text{otherwise} \end{cases}$$

b) Pseudocode:

Algorithm 1: `largest_square`

Input: B
Result: w, x, y (the width and the top left position of the largest square)

```
for (i = 1 : n) do
    OPT[i, m] ← B[i, m]
end
for (j = 1 : m − 1) do
    OPT[n, j] ← B[n, j]
end
for (i = n − 1 : 1) do
    for (j = m − 1 : 1) do
        if B[i, j] = 0 then
            OPT[i, j] ← 0
        else
            OPT[i, j] ←
              1 + min OPT[i, j + 1], OPT[i + 1, j], OPT[i + 1, j + 1]
        end
    end
end

w ← 0
for (i = 1 : n) do
    for (j = 1 : m) do
        if OPT[i, j] > w then
            w ← OPT[i, j]
            x ← i
            y ← j
        end
    end
end

return w, (x, y)
```

c) Time: $O(n \times m)$
Space: $O(n \times m)$

2 Dynamic Programming: Atlanta MARTA [25 pts]

The Metropolitan Atlanta Rapid Transit Authority (MARTA) is the principal public transport operator in the Atlanta metropolitan area. It was Formed in 1971 as strictly a bus system, and today, it is transporting almost 450,000 passengers a day (bus and train). Currently, MARTA Passes are the cheapest option for those who regularly use MARTA for transportation. Assume the MARTA Passes are sold in three following forms:

- Daily: A 1-day pass sold for $tickets[0]$ dollars;
- Weekly: A 7-day pass sold for $tickets[1]$ dollars;
- Monthly: A 30-day pass sold for $tickets[2]$ dollars.

For instance, $tickets = 2, 7, 20$ means we need to pay \$2, \$7, and \$20 for each daily, weekly,and monthly pass, respectively. The passes allow consecutive days of travel. For example, if we get a weekly pass on day 5, then we can travel for 7 consecutive days which are: day 5, 6, 7, 8, 9,10, and 11. George P. Burdell is a student at Georgia Tech and he has already organized his commuting plan for the upcoming year. In his plan, each day of year is specified by an integer identification number from 1 to 365. Therefore, he can represent his commuting plan as an array of integers. For instance, days = 8, 9, 10, 11, 14, 17, 18, ... means George needs to commute to the school on the 8th, 9th, 10th, 11th, ... days of year. He asked you to help him find the minimum amount of money that he should spend to purchase MARTA Passes for commuting to school in the next year.

a) (10 pts) Explain the optimal substructure of this problem.

b) (10 pts) Write a recursive expression for calculating min Cost including the base case.

c) (5 pts) Give the pseudocode of a linear Dynamic Programming algorithm to return the minimum cost of commuting every day in the array "days", if the cost of MARTA passes is given in a three-element array "tickets". Analyze the space and time complexity of your algorithm.

Solution

Optimal Substructure

We are given an array of integer numbers representing the days that George wants to commute to school. $days = \{d_1, d_2, \ldots, d_n\}$, and a three-element array $tickets = \{t_1, t_2, t_3\}$. we want to calculate the minimum possible value that George has to pay for buying MARTA passes. Let $OPT[i]$ denotes the minimum amount of money George needs to pay to fulfill the plan from day i to the end of the plan. Therefore, the minimum cost of commuting for the entire plan is $OPT[1]$. There can be two approaches to solve this problem:

- *Approach 1: Iterate over all days*
  Starting from day one, if George doesn't want to travel today, there is no need to buy a MARTA pass today. Therefore, it is strictly better to wait until the day that he needs to travel, say day i. Then, he will have three options: buying a 1-day, 7-day, or 30-day MARTA pass:
  - If he buys a 1-day pass: he needs to pay $tickets[0]$ dollars and the next subproblem is $OPT[i+1]$ because the 1-day pass is only valid for one day and for the next day he has to pay $OPT[i+1]$ dollars for commuting from day $i+1$ to the end of the plan.
  - If he buys a 7-day pass: he needs to pay $tickets[1]$ dollars and the next subproblem is $OPT[i+7]$ since the 7-day pass is valid for 7 days (from the day of purchase), and thus after that 7 days, he has to pay $OPT[i+7]$ dollars for commuting from day $i+7$ to the end of the plan.

– Similarly, if he buys a 30-day pass: he needs to pay $tickets[2]$ dollars and the next subproblem is $OPT[i+30]$

Therefore, if George needs to travel in day i, the optimum amount of money can be obtained by taking the minimum values of these three:

$$OPT[i] = \min\{tickets[0] + OPT[i+1], tickets[1] + OPT[i+7], tickets[2] + OPT[i+30]\}$$

The solution of these three subproblems, $OPT[i+1]$, $OPT[i+7]$, and $OPT[i+30]$ are minimum. (Note: We can show that using contradiction. For the sake of contradiction, assume these three values are not the minimum solutions. Therefor, there must exist other optimum solutions with the less ticket cost. Then, we can substitute the solution(s) of subproblem(s) with these minimum values. That implies we obtain less value for $OPT[i]$, the cost of tickets for the commuting plan from day i which is a contradiction because we started with this assumption that $OPT[i]$ is the minimum ticket cost. Thus, this problem has optimal substructures.)

- *Approach 2: Iterate over the days of the commuting plan*
In the first approach, we iterate over all days of the year (starting from day 1 to the last possible day in the plan e.g. 365 , regardless of their presence in the commuting plan. However, instead of all days, we can only check the days that their identification numbers are in the commuting plan. This approach will be slightly faster than the first one. The second approach will particularly be more preferable when the number of days in the plan is much less than the maximum day index, i.e. $|days| << \max\{days\}$. In the worst case, number of days of the commuting plan will be equal to the maximum day index which happens when George wants to commute everyday. In this case, the subproblem $OPT[i]$ denotes the minimum amount of money George needs to pay to fulfill the plan from day $days[i]$ to the end of the plan. (In approach 1, i denotes identification number of each day. In approach 2, i denotes the index of each day in $days$ array). Now, to pass the unnecessary checks, we need to define three additional indices. Let $j1$ be the largest index such that $days[j1] < days[i]+1$, $j7$ be the largest index such that $days[j7] < days[i]+7$, $j30$ be the largest index such that $days[j30] < days[i]+30$. In this way, if George needs to travel in day i, the optimum amount of money can be obtained by taking the minimum values of these three:

$$OPT[i] = \min\{tickets[0] + OPT[j1], tickets[1] + OPT[j7], tickets[2] + OPT[j30]\}$$

The discussion of optimal substructure is similar to the first approach.

Recursive Expression

- *Approach 1*

$$OPT[i] = \begin{cases} 0 & i > \max\{days\} \\ \min \begin{cases} tickets[0] + OPT[i+1], \\ tickets[1] + OPT[i+7], \\ tickets[2] + OPT[i+30] \end{cases} & i \leq \max\{days\} \end{cases}$$

- *Approach 2*

$$OPT[i] = \begin{cases} 0 & i > |days| \\ \min \begin{cases} tickets[0] + OPT[j1], \\ tickets[1] + OPT[j7], \\ tickets[2] + OPT[j30] \end{cases} & i \leq |days| \end{cases}$$

where $j1$, $j7$, and $j30$ are the largest indices such that $days[j1] < days[i]+1$, $days[j7] < days[i]+7$, and $days[j30] < days[i]+30$, respectively.

Pseudocode

- *The driver code for Top-Down algorithm*

Algorithm 2: `MinMARTACost`

Input: $days = \{d_1, d_2, \ldots, d_n\}$, $tickets = \{t_1, t_2, t_3\}$
Result: $minCost$ (the minimum amount to pay for MARTA passes)

```
1 memo ← {}
2 minCost ← minCostRecur(days, tickets, memo, 1)

3 return minCost
```

- *Approach 1*

Algorithm 3: `minCostRecur`

Input: $days = \{d_1, d_2, \ldots, d_n\}$, $tickets = \{t_1, t_2, t_3\}, memo, i$
Result: $minCost$ (the minimum amount to pay for MARTA passes starting from day i)

```
1  if (i > max{days}) then // base case
2  |  return 0
3  end

4  if (memo[i] ≠ ϕ) then
5  |  return memo[i]
6  end

7  if (i ∈ days) then // i is in the commuting plan
8  |
   |   memo[i] = min{tickets[0] + minCostRecur(days, tickets, memo, i + 1),
   |                 tickets[1] + minCostRecur(days, tickets, memo, i + 7),
9  |                 tickets[2] + minCostRecur(days, tickets, memo, i + 30)}
10 else
11 |  memo[i] = minCostRecur(days, tickets, memo, i + 1)
12 end

13 return memo[i]
```

- *Approach 2*

Algorithm 4: `minCostRecur`

Input: $days = \{d_1, d_2, \ldots, d_n\}$, $tickets = \{t_1, t_2, t_3\}, memo, i$
Result: $minCost$ (the minimum amount to pay for MARTA passes starting from day i)

```
1  if (i > length({days})) then // base case
2  |  return 0
3  end

4  if (memo[i] ≠ φ) then
5  |  return memo[i]
6  end

   // Finding the largest indices j1, j7, and j30
7  j1 ← i
8  while (j1 < |days| && days[j1] < days[i] + 1) do j1++;
9  j7 ← j1
10 while (j7 < |days| && days[j7] < days[i] + 7) do j7++;
11 j30 ← j7
12 while (j30 < |days| && days[j30] < days[i] + 30) do j30++;
13
```

$$memo[i] = \min\{tickets[0] + \texttt{minCostRecur}(days, tickets, memo, j1),\\ tickets[1] + \texttt{minCostRecur}(days, tickets, memo, j7),\\ tickets[2] + \texttt{minCostRecur}(days, tickets, memo, j30)\}$$

```
14 return memo[i]
```

Time and Space Complexity

In approach 1, the recursive calls is executed for the maximum number of days identifiers. For instance, if the commuting plan is written for one year, the algorithm will be executed 365 times. Therefore, the time complexity is $O(\max\{days\})$. The space complexity is the same as the time complexity due to the memory required for the memoization.
In approach 2, as explained earlier, the algorithm is executed for the number of days of the commuting plan. Therefore, both time and space complexity are $O(|days|)$.

Note

In the presented solution, we solved the subproblems from the end of the commuting plan (the last day of the plan) to obtain the minimum cost estimation in the first day of the plan. Therefore, $OPT[i]$ is defined as the minimum amount of money George needs to pay for commuting from day i to the end of the plan, and the minimum value for the entire plan is $OPT[1]$. It is also correct and accepted to start from the beginning of the plan and solve the subplroblems as the minimum amount of money spent on MARTA passes from the first day of the plan until day i. In this case, the recursive expression can be written as,

$$OPT[i] = \begin{cases} 0 & i \leq 0 \\ \min \begin{cases} tickets[0] + OPT[i-1], \\ tickets[1] + OPT[i-7], \\ tickets[2] + OPT[i-30] \end{cases} & 0 < i \leq \max\{days\} \end{cases}$$

Thus, $OPT[\max\{days\}]$ gives the minimum amount of money spent on MARTA passes.