Data Structure & Algorithms

Assignment 2. Innopolis University, Spring 2019

Name: Hany Hamed
Group Number: BS18-06

Section 2: Spell autocorrection

(a) **Submission Number**: 50868134
Submission Link: https://codeforces.com/group/lk8Ud0ZeBu/contest/239900/submission/50868134.

(b) **Submission Number**: 50868163
Submission Link: https://codeforces.com/group/lk8Ud0ZeBu/contest/239900/submission/50868163.

(c) **Submission Number**: 50869068
Submission Link: https://codeforces.com/group/lk8Ud0ZeBu/contest/239900/submission/50869068.

Section 3: Theoretical part

1. *(a).*

$$a = 16, b = 4, f(n) = n$$
$$n^{\log_b a} = n^{\log_4 16} = n^2$$
$$\textit{That's imply that } f(n) < n^2\textit{, then it is the } 1^{st} \textbf{\textit{ case of Master Theorem}}$$
$$\textit{Then, } T(n) = \Theta\ (n^2)$$

(b).

$$a = 1, b = 2, f(n) = 2^n$$
$$n^{\log_b a} = n^{\log_2 1} = 1$$
$$\textit{That's imply that } f(n) > 1\textit{, then we need to check first the regularity condition.}$$

$$\textit{if } af(n/2) \leq cf(n)\ \textit{, and } \exists\ c < 1\textit{, then it will be the } 3^{rd} \textit{case.}$$

$$\textbf{\textit{Checking the Regularity condition:}}$$
$$2^{n/2} \leq c2^n$$
$$\textit{as } 2^n \textit{ always positive}$$

DSA, Innopolis University, Spring 2019
Author: Hany Hamed

$$Then,\ 2^{\frac{n}{2}-n} \le c$$
$$Then,\ 2^{\frac{-n}{2}} \le c$$
$$Now\ find\ out\ when\ \exists\ c < 1,$$
$$2^{\frac{-n}{2}} < 1\ (By\ taking \log_2 for\ both\ sides$$
$$\frac{-n}{2} < \log_2 1$$
$$\frac{-n}{2} < 0$$
$$Then,\ n > 0\ \exists\ c < 1$$
That's imply that it is the 3^{rd} ***case of Master Theorem***
$$Then,\ T(n) = \Theta\ (2^n)$$

(c).

$$a = 2, b = 2, f(n) = \frac{n}{\log n}$$
$$n^{\log_b a} = n^{\log 2} = n$$
$$\frac{n}{\log n}\ ?\ 1\ As,\ n > \log n$$
$$Then,\ \frac{n}{\log n} > 1$$

That's imply that $f(n) > 1$*, then we need to check first the regularity condition.*
if $af(n/2) \le cf(n)$ *, and* $\exists\ c < 1$*, then it will be the* 3^{rd}*case.*

Checking the Regularity condition:

$$2\frac{n/2}{\log n/2} \le c\frac{n}{\log n}$$
$$Then,\ \frac{1}{\log n - \log 2} \le \frac{c}{\log n}$$
$$Then,\ \frac{\log n}{\log n - 1} \le c$$
$$As,\ \log n > \log(n-1)$$
$$Then,\ \frac{\log n}{\log(n-1)} > 1$$
$$Then,\ 1 \le c,\ \exists!\ c < 1$$

Then, Regularity condition is not holding.
Then, we cannot use Master Theorem.

(d).

$$a = 4, b = 2, f(n) = n^2$$
$$n^{\log_b a} = n^{\log_2 4} = n^2$$
$$f(n) = n^2 = n^{\log_b a}$$
That's imply that it is the 2^{nd} ***case of Master Theorem***
$$Then,\ T(n) = \Theta\ (n^2 * \log n)$$

2.

- ***Step 1:***
Sub-problem is a part of the problem but it can be solved easily in order to get the overall problem.
The sub-problem here is the best cost from a city (s) to another city (the end city: n).
Using the DP memomization technique, we are going to use an 1D array $m[i]$ which the index is the start city (s) and the value is the best cost (the minimum) to go to n, and using Bottom-Up approach we will build this array.

- ***Step 2:***
The recurrence definition for the problem:

$$m[i] = \begin{cases} 0, & \textit{if } i == n. \\ \min\limits_{i \le k < n} f_{i,k} + m[k], & \textit{otherwise: } (i < n). \end{cases} \tag{1}$$

Where k is an arbitrary element in the middle between [i,n)

- ***Step 3:***
The recurrence will be correct as we are going to use Bottom-Up approach which will start from m[n-1] and the cost of it will be the direct direction from n-1 to n, then we will calculate for n-2 which has 2 options: (1) go to n-1 then n. (2) go to n directly and here we take the minimum between the two options, So, it will be local optimal solution and so on for the rest of the array until we get a global optimal solution which consists of the local optimal solutions of the sub problems.

- ***Step 4:***
The 1^{st} base case is when $i = n$ in this case the best cost = 0.Time complexity of this step: O(1).
The 2^{nd} base case which is implicitly declared in the recurrence relation is when $i = n - 1$ and in this case the best cost = $f_{i,n}$. Time complexity of this step: O(1).

- ***Step 5:***
- Take the input.
- Store it in 2D Matrix (Lower Matrix). ($f_{i,n}$)
- Create 1D array with size n (zero indexed array). (m[n])
- Initialize the array with $m[i] = f_{i,n}$.
- Iterate over the array from the end (from n-1).
- If the iterator (i) = n-1, store 0
otherwise, Iterate over the array (k) from i to n.

m[i] = min(m[i], $f_{i,k} + m[k]$)

- return m[1]

***P**sudocode:*

```
input n
for i = to n:
    f_{i,j} = 0
  for j = i+1 to n:
    f_{i,j} = input

Create m[n]
for i = 0 to n:
  m[i] = f_{i,n}
for i = n-1 to 0:
  if i == n-1: m[i] = 0
    else: for k = i to n:
      m[i] = min(m[i], f_{i,k} + m[k])
```

- ***Step 6:***
  According to the provided algorithm, the Time complexity can be observed as $T(n) = \theta(n^2)$ *as in initialize the values of the array m we do 1+2+3+...+n operations which is Arithmetic series where the value =* $\frac{n*(n+1)}{2} = \theta(n^2)$. *So,* $T(n) = \theta(n^2)$