Esra
Eryılmaz
171044046

1

-CSE 321-

Homework 4

Deadline: 19.01.2021

1) Consider a text with n zeros. How many character comparisons (in terms of n) will the brute-force string matching algorithm make in searching the pattern 0010? What is the worst case input pattern of length 3 (3 bits) for the brute-force algorithm?

Total number of comparisons (in terms of n) for the search pattern $\underbrace{0010}_{m}$ is;

$m(n-m+1) = m(n-4+1) = m(n-3) = \underline{m(n-3)}$

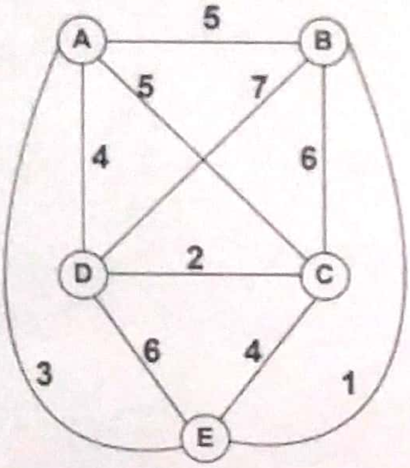
For the worst case input pattern of length $\underbrace{3}_{m}$ is;

$m(n-m+1) = m(n-3+1) = \underline{m(n-2)}$ comparisons

$O(m \cdot n) = O(3n) = O(n)$

Scanned with CamSc

2) **Apply brute-force algorithm for the travelling salesman problem.**



$$\frac{(n-1)!}{2} = \frac{(5-1)!}{2} = 12$$ routes

start with → A vertex

1) ABCDEA → 5 + 6 + 2 + 6 + 3 = 22
2) ABCEDA → 5 + 6 + 4 + 6 + 4 = 25
3) ABDCEA → 5 + 7 + 2 + 4 + 3 = 21
4) ABDECA → 5 + 7 + 6 + 4 + 5 = 27
5) ABECDA → 5 + 1 + 4 + 2 + 4 = 16
6) ABEDCA → 5 + 1 + 6 + 2 + 5 = 19
7) ACBDEA → 5 + 6 + 7 + 6 + 3 = 27
8) ACBEDA → 5 + 6 + 1 + 6 + 4 = 22
9) ACDBEA → 5 + 2 + 7 + 1 + 3 = 18
10) ACEBDA → 5 + 4 + 1 + 7 + 4 = 21
11) ADBCEA → 4 + 7 + 6 + 4 + 3 = 24
12) ADCBEA → 4 + 2 + 6 + 1 + 3 = 16

optimal routes

Time complexity → $O(n!)$

3) Design a decrease-by-half algorithm for computing log n (base 2). Calculate its time efficiency.

Algorithm FloorOfLog (n)

// Input : A positive integer n

// Output : Returns $\log_2 n$

```
if n == 1                  // Base case
        return 0
else
        return FloorOfLog (n/2) + 1
```

The recurrence relation for this algorithm is ;

$\boxed{T(n) = T\left(\frac{n}{2}\right) + 1}$ for $n > 1$, $T(1) = 0$

By using Master Theorem the time efficiency is;

$a = 1$
$b = 2$
$d = 0$

$\boxed{a = b^d}$

$\Theta(n^d \log n) = \Theta(n^0 \cdot \log n)$

$= \Theta(\log n)$

4) **A bottle factory produces bottles of equal mass. During a production, the weight of one of the bottles is set incorrectly. The factory scale will be used to find this bottle. Design a decrease-andconquer algorithm which finds the that bottle. Analyze the worst-case, best-case and average-case complexities of your algorithm. Explain your algorithm in the report file.**

- I think, I can solve this problem like fake coin problem. They are similar.
- with using decrease-by-a-constant-factor;

Algorithm decrease-by-factor-2

if n = 1

bottle found

else

divide the bottles into two piles of $\lfloor n/2 \rfloor$ bottle each, leaving one extra bottle if n is odd weigh the two bottle.

if their weigh the same

return the one extra bottle as incorrect weight

else

continue with the lighter of the two bottles

- worst case:

$W(n) = W(\lfloor \frac{n}{2} \rfloor) + 1$ for $n > 1$, $W(1) = 0$

$\rightarrow W(n) = \lfloor \log_2 n \rfloor \longrightarrow \Theta(\log_2 n) = \Theta(\log n)$ //

- Best case:

if $n=1 \rightarrow \Theta(1)$ //

- Average case:

Same as worst case //

Scanned with CamSc

5) **Assume you have 2 arrays which are both unsorted. Provide a divide and conquer algorithm which finds the x th element of the merged array of these two arrays. Write the pseudocode of your algorithm and calculate its worst-case running time. Tabu: Merging these arrays at first and then finding the x th element is forbidden.**

First, assume I apply merge sort for both unsorted arrays. Then, I have two different sorted arrays.
So now, I write divide and conquer algorithm;
→ I compare the middle elements of array A[] and B[]
Let assume A[midA] x, then clearly the elements after midB cannot be the required element. We then set the last element of B[] to be B[midB]
In this way, we define a new subproblem with half the size of one of the arrays.

```
Algorithm FindX(A, B, endA, endB, x)
    if A = endA
        return B[x]
    if B = endB
        return A[x]

    midA = (endA - A)/2
    midB = (endB - B)/2

    if midA + midB < x
        if A[midA] > B[midB]
            return FindX(A, B + midB + 1, endA, endB, x - midB - 1)
        else
            return FindX(A + midA + 1, B, endA, endB, x - midA - 1)
    else
        if A[midA] > B[midB]
            return FindX(A, B, A + midA, endB, x)
        else
            return FindX(A, B, endA, B + midB, x)
```

Worst-Case Time Complexity :

- I first assume that I applied Merge Sort for both arrays. Time complexity of Merge sort is $O(n \log n)$ in all the 3 cases (worst, best, average)

  For two arrays $\longrightarrow$ $O(n\log n) + O(m\log m)$

- FindX( ) time complexity $\rightarrow$ $O(\log n + \log m)$

$$+$$

$$O(n\log n) + O(m\log m) + O(\log n + \log m)$$

have control on others

$$= \underline{O(n\log n)}$$