CSE - 331

Homework - 4

Türker Tercan

171044032

1) Consider a text with n zeros. How many 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(bits) for the brute-force algorithm?

pseudocode BFSringMatch (T[0,.. n-1], P[0,.. m-1])

```
for i ← 0 to n-m do
    j ← 0
    while j < m and P[j] = T[j+i] do
        j ← j+1
    end while
    if j=m
        return i
    end if
end for
end
```

Text : 0000_____0 (n)

Pattern : 0010

The algorithm does it follows

1- Iterate every element of text 0000...0 (n)

2- If current text element and pattern's first element are the same

000...00

0010

a. Iterate text and pattern both, and check every elements are matching.

0000...0 → 0000...0

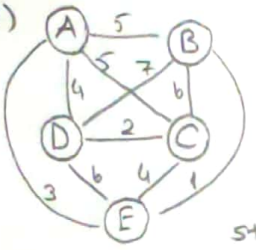
0010 → 0011

b. Dismatch occurred return

Analysis: Algorithm needs to make m comparisons before shifting the pattern. So in this case, there will be there comparisons before shifting because first two elements of the pattern will always be matched.

Total Comparisons : $3(n-4+1) = 3n-3$ comparisons

If we have a pattern that is like "00X" and X can be any digit, it will be the worst case input and total comparisons will be same as "0010".

2) 

Apply brute-force algorithm for travelling salesman problem.

- All following algorithms makes brute-force way to solve travelling salesman problem
- It calculates all permutation which salesman can travel from a starting vertex and traverses every edges and selects minimum cost of them for every starting vertex.

Pseudocode Travelling Salesman (Graph, size)

```
    min_cost ← ∞
    for i ← 0 to size do
        vertex = []
        for j ← 0 to size do
            if j != i
                vertex.append(j)
            end if
        end for
        min_path ← ∞
        all_permutations = permutations (vertex)
        for permutation in all_permutations
            current_weight = 0
            k = i
            for index in permutation
                current_weight += Graph[k][i]
                k = index
            end for
            current_weight += Graph[k][i]
            min_path = min (min_path, current_weight)
        end for
        min_cost = min (min_path, min_cost)
    end for
end
procedure permutations(vertex)
    if len(vertex) == 0
        return []
    end if
    else if len(vertex) == 1
        return [vertex]
    end if
    newVertex = []
    for i ← 0 to len(vertex) do
        m = vertex[i]
        remVertex = []
        for j ← 0 to i do
            remVertex.add(vertex[j])
        end for
        for j ← i+1 to len(vertex) do
            remVertex.add(vertex[j])
        end for
        for p in permutations(remVertex)
            newVertex.append([m] + p)
        end for
    end for
    return newVertex
end
```

3) Design decrease by half algorithm for computing logn (base 2). Calculate its time efficiency.

```
pseudocode logarithmBase2(int n)
    if n==1
      return 0
    end if
    else
      1 + logarithmBase2(n/2)
    end else
end
```

Time Efficiency:

$T(n) = T(n/2) + 1$

We can use Master's Theorem to solve this

$n^{\log_b a} = n^{\log_2 1} = n^0 = 1$

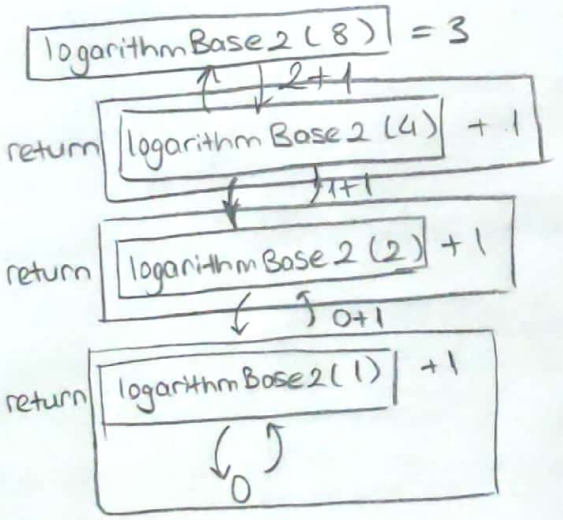
$T(n) = \Theta(n^{\log_b a})$

Case-2 Master's Theorem is applicable

$T(n) = \Theta(\log_2 n) = \Theta(\log n)$

For example:

Take 8 for int 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-and-conquer algorithm which finds the that bottle. Analyze the worst-case, best-case, average-case complexities of your algorithm. Explain your algorithm in the report file.

```
procedure incorrect_bottle(bottle[0:n], mass, firstIndex, lastIndex)
    if firstIndex == lastIndex
       if bottle[firstIndex] != mass
          return firstIndex
       end if
    end if
    middle = (firstIndex + lastIndex + 1) / 2
    first_part = 0
    first_count = 0
    for i = 0 to middle do
       first_part += bottle[i]
       first_count += 1
    end for
    if first_part / first_count != mass
       return incorrect_bottle(bottle, mass, firstIndex, middle - 1)
    end if
    else
       return incorrect_bottle(bottle, mass, middle, lastIndex)
    end if
end
```

For example: Let the bottle as this, bottle = [1, 1, 1, 1, 2, 1], mass = 1

1- [1, 1, 1, 1, 2, 1]
middle = 3
[1, 1, 1, 1, 2, 1]
first_part = 3
first_part / first_count = 1
Average of the first part of the array is equal to mass then, select last_part

2- [1, 2, 1]
middle = 2
[1, 2, 1]
first_part = 3
first_part / first_count != mass
It is not equal to mass
select first

3- [1, 2]
middle = 1
[1, 2]
first_part = 1
first_part / first_count == mass
is equal to mass
select second

4- [2]
return

Average Time Complexity:

$T(n) = T(n/2) + n/2, \quad T(1) = 0$

$T(n) = T(n/4) + n/4 + n/2$

$T(n) = T(n/8) + n/8 + n/4 + n/2$

$T(n) = T(n/2^k) + n\underbrace{\left[\frac{1}{2^k} + \frac{1}{2^{k-1}} + \cdots \frac{1}{2}\right]}_{1}$

$= T(n/2^k) + n$

Lets assume $n/2^k = 1 \quad k = \log_2 n$

$T(n) = T(1) + n \quad T(n) \in \vartheta(n)$
$= n$

Best and Worst complexities are the same because we need traverse the element and its depends on the input we are working.

$B(n) \in \vartheta(n) \quad W(n) \in \vartheta(n)$
$A(n) \in \vartheta(n)$