|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Kingdom of Saudi Arabia**  **Ministry of Education**  **University of Jeddah**  **College of Science and Computer Engineering** |  | **المملكة العربية السعودية**  **وزارة التعليم**  **جامعة جدّة**  **كلية علوم و هندسة الحاسب** | |  |  |

**CCCS 314 – Design and Analysis of Algorithms**

**LAB 3**

**Topics:**

1. **Brute Force: String Matching**
2. **Brute Force: Closest-Pair Problem**

**Total Marks: 2**

**Student Name: Bassam AL-Ghamdi**

**Student ID: 2141362**

**Marks:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Exercises | 1 | 2 | 3 | Total |
| Allocated | 0.5 | 1 | 0.5 | 2 |
| Obtained |  |  |  |  |
| **CLO, PLO** | 1.1, K1 | 2.1, S1 | 2.1, S1 |  |

**CLO** **Marks:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | CLO1.1, K1 | CLO2.1, S1 | Total |
| Allocated | 0.5 | 1.5 | 2 |
| Obtained |  |  |  |

**Exercise 1:** **(Brute Force: String Matching)**

How many comparisons (both successful and unsuccessful) are made by the brute-force string-matching algorithm in searching for each of the following patterns in the binary text of 1000 zeros?

1. 00001 b. 10000 c. 01010

**Answer**:

**a. For pattern "00001":**

The length of the pattern is m=5. The algorithm would compare the first character of the pattern with each of the 1000 characters of the text, then it would move one character to the right and compare the pattern again with the next m=5 characters of the text, and so on until the end of the text is reached. Therefore, the total number of comparisons would be:

1000 - 5 + 1 = 996 (both successful and unsuccessful).

**b. For pattern "10000":**

Following the same logic, we have:

1000 - 5 + 1 = 996 comparisons.

**c. For pattern "01010":**

Following the same logic, we have:

1000 - 5 + 1 = 996 comparisons.

Therefore, for each of the given patterns, the brute-force string-matching algorithm would make 996 comparisons in searching for it in the binary text of 1000 zeros.

**Exercise 2:** **(Brute Force: String Matching)**

Consider the problem of counting, in a given text, the number of substrings that start with an A and end with B. (For example, there are four such substrings in CABAAXBYA.)

**Answer**:

To solve this problem using brute-force string-matching algorithm, we can consider all possible substrings of the given text and count the number of substrings that start with an A and end with a B. The algorithm would iterate over the text and for each character that is equal to A, it would iterate over the remaining characters and count the number of Bs that appear after it. The total number of substrings that start with A and end with B would be the sum of these counts.

Here's a Python code to implement this algorithm:

```

def count\_AB\_substrings(text):

count = 0

n = len(text)

for i in range(n):

if text[i] == 'A':

for j in range (i+1, n):

if text[j] == 'B':

count += 1

return count

```

Using the example given in the question (text = "CABAAXBYA"), this algorithm would count the following four substrings that start with A and end with B:

AB, AXB, AB, A.

Therefore, the output of the algorithm for this example would be 4.

Note that this algorithm has a time complexity of O(n^2), where n is the length of the text. Therefore, for large texts, it may be necessary to use more efficient algorithms to solve this problem.

**Exercise 3:** **(Closest-Pair Problem)**

Can you design a more efficient algorithm than the one based on brute-force strategy in the lecture slides to solve the closest-pair problem for *n* inputs , ,..., on the real line if we can use a sorting algorithm of efficiency class ? If you can, explain it and analyze why it is more efficient.

**Answer**:

Yes, we can design a more efficient algorithm than the brute-force strategy to solve the closest-pair problem for n inputs x\_1, x\_2, ..., x\_n on the real line, by utilizing the sorting algorithm of efficiency class Θ (n log n).

Here's an outline of the algorithm:

1. Sort the given n inputs x\_1, x\_2, ..., x\_n in non-decreasing order using the sorting algorithm of efficiency class Θ (n log n). This step takes Θ (n log n) time.

2. Initialize a variable min\_distance to infinity.

3. For each i from 1 to n-1, compute the distance between x\_i and x\_i+1, and update min\_distance if the computed distance is smaller than min\_distance.

4. Return min\_distance as the closest pair distance.

The idea behind this algorithm is to reduce the problem of finding the closest pair to computing the distances between adjacent pairs of points. Since the points are sorted, the closest pair must be adjacent in the sorted order, so we only need to compute the distances between adjacent pairs and keep track of the minimum distance found so far.

The time complexity of this algorithm is dominated by the sorting step, which takes Θ (n log n) time. The loop in step 3 takes O(n) time. Therefore, the overall time complexity of the algorithm is Θ (n log n).

Compared to the brute-force strategy, which has a time complexity of O(n^2), this algorithm is much more efficient, especially for large values of n.