**Vipul Krishna**

**Assignment 8**

**INFO 6205 – Program Structures and Algorithms**

Question 1:

Java Code Attached

Question 2:

Text = ABCABCDABABCDABCDABDE

Pattern = BCD

Explanation:

In the brute force method, we check all the possible substring of length of the pattern and return the index of the starting substring wherever a match occurs. We start the match from the first index of the main string and check if it matches the given pattern. Then we shift the position one-by-one in the main string and match its substrings with the pattern. Programmatically, in each iteration, we match the starting character of the string with the pattern. If it matches, we move to the next character till the time match occurs. If not, then we break from that iteration and move to the next.

Step 1 trace:

ABCABCDABABCDABCDABDE

BCD

Result – No match

Step 2 trace:

ABCABCDABABCDABCDABDE

BCD

Result – No match

Step 3 trace:

ABCABCDABABCDABCDABDE

BCD

Result – No match

Step 4 trace:

ABCABCDABABCDABCDABDE

BCD

Result – No match

Step 5 trace:

ABCABCDABABCDABCDABDE

BCD

Result – Match Found. Index of the substring returned which is 4

……………….. and so on.

Step 11 trace:

ABCABCDABABCDABCDABDE

BCD

Result – Match Found. Index of the substring returned which is 10

…….

Step 15 trace:

ABCABCDABABCDABCDABDE

BCD

Result – Match Found. Index of the substring returned which is 14

……..

Step 15 trace:

ABCABCDABABCDABCDABDE

BCD

Result – No Match

Question 3:

a)

In Rabin-Karp algorithm, we do two checks. First, we calculate the hash value of the pattern. Then we calculate the hash value of each possible substring of length equal to the pattern length. Then match the hash values of each substring with the pattern. If there is a match in the hash values, only then we match the characters of the substring and the pattern.

Hash Value is found using the following formula:

Hash Value = Hash(previous Substring) – Hash(Excluded Character) + Hash(Included Character)

Example: If Hash(ABC) = X

Then, Hash(BCD) = X – Hash(A) + Hash(D)

Step 1 trace:

ABCABCDABABCDABCDABDE

BCD

Result – The hash values do not match. We move to the next iteration

Step 2 trace:

ABCABCDABABCDABCDABDE

BCD

Result – The hash values do not match. We move to the next iteration

Step 3 trace:

ABCABCDABABCDABCDABDE

BCD

Result – The hash values do not match. We move to the next iteration

Step 4 trace:

ABCABCDABABCDABCDABDE

BCD

Result – The hash values do not match. We move to the next iteration

Step 5 trace:

ABCABCDABABCDABCDABDE

BCD

Result – The hash value of the pattern and the substring matches. Now, we match the characters of substring and the pattern. The result is true. Index of the substring returned which is 4

……………….. and so on.

Step 11 trace:

ABCABCDABABCDABCDABDE

BCD

Result – The hash value of the pattern and the substring matches. Now, we match the characters of substring and the pattern. The result is true. Index of the substring returned which is 4. Index of the substring returned which is 10

…….

Step 15 trace:

ABCABCDABABCDABCDABDE

BCD

Result – The hash value of the pattern and the substring matches. Now, we match the characters of substring and the pattern. The result is true. Index of the substring returned which is 4. Index of the substring returned which is 14

……..

Step 15 trace:

ABCABCDABABCDABCDABDE

BCD

Result – The hash values do not match. We stop the iteration here.

b)

Yes, there is a difference in time complexity between Brute Force and Rabin Karp algorithm.

Consider the length of pattern as m and length of string as n.

So, the time complexity of brute force algorithm = O(mn)

The time complexity of Rabin Karp algorithm = O(mn), but here, the size of n is smaller when compared to brute force.

In Rabin Karp, we match the substring with the pattern only when there is a match in their hash values.

Finding hash values takes constant time. Thus, the substring of the main string we match will have a length equal to the pattern already. Thus, O(mn) in case of Rabin Karp algorithm is more efficient than brute force algorithm

Question 4:

1. Step-by-Step demonstration of Dijkstra shortest path algorithm:
2. Consider 0 as the starting vertex. Mark its distance from 0 as ‘0’ and all other vertex as ∞.
3. Find the distance of all adjacent unvisited vertices from 0. We have 1 and 7. Since their distance from 0 is less than ∞, we replace the distance values.
4. Next, visit the adjacent unvisited vertex which is at the smallest distance from the start vertex(greedy). We have 1 whose distance is 4.
5. From the current vertex, calculate the minimum distance of each unvisited neighbor from the start vertex via the visited vertex. We have 2 and 7. Distance is 12 and 15 respectively. Replace 2’s distance but skip 7’s (since 8<15).
6. Visit 7. Replace vertices 6 and 8 distances as 9 and 15 respectively.
7. Visit 6. Replace vertices 5 distance as 11.
8. Visit 5. Replace 3’s distance as 25. Replace 4’s distance as 21. Skip 2 because 12<15.
9. Visit 2. Replace 3’s distance as 19. Replace 8’s distance as 14.
10. Visit 8. No unvisited neighbor. Skip.
11. Visit 3. Skip 4’s as 21<28.
12. Visit 4. No unvisited neighbor. Skip. No unvisited vertex. Process ends.

|  |  |  |
| --- | --- | --- |
| Vertex | Minimum distance from 0 | Previous Vertex |
| 0 | ∞ 🡪 0 |  |
| 1 | ∞ 🡪 4 | 0 |
| 2 | ∞ 🡪 12 | 1 |
| 3 | ∞ 🡪 25 🡪 19 | 5 🡪 2 |
| 4 | ∞ 🡪 21 | 5 |
| 5 | ∞ 🡪 11 | 6 |
| 6 | ∞ 🡪 9 | 7 |
| 7 | ∞ 🡪 8 | 0 |
| 8 | ∞ 🡪 15 🡪 14 | 7 🡪 2 |

1. Space Complexity = O(V^2).

Time Complexity

1. Time Complexity = O(V^2) – for matrix representation and priority queue implementation
2. Time Complexity = O (Vlog V + E) – with min-priority queue using Fibonacci heap implementation and adjacency list representation.
3. Time Complexity = O (ELog V) – with min-priority queue using binary heap implementation
4. Java Code attached

Question 5

Greedy Algorithm – This is a technique which strives to choose a locally optimal choice assuming that it will lead to a globally optimal solution

Give time array for each work, A = {7,6,5,4,3,2,1}

Total time available, T = 10

Steps:

Arrange A in ascending order, A = {1,2,3,4,5,6,7}

1. First Iteration

Time elapsed = 1;

Number of work = 1;

1. Second Iteration

Time elapsed = 1 + 2 = 3;

Number of work = 2;

1. Third Iteration

Time elapsed = 3 + 3 = 6;

Number of work = 3;

1. Fourth Iteration

Time elapsed = 6 + 4 = 10;

Number of work = 4;

After the fifth iteration, Time elapsed = 10 + 5 = 15 which is greater than T = 10.

Thus, the number of work done = 4

Question 6:

1. Goal of Genetic Algorithm - The goal of genetic algorithm is to optimize a problem by finding out such values of the input which will give the best output values as the result. This algorithm is based on Charles Darwin theory of natural evolution.

Its main elements consist of:

1. finding the population on which the test should be conducted.
2. After the population is established, we figure out a method to generate input values for the output function.
3. Then we select the best input values from the generated set.
4. Once we have the best input set, we get the outputs using these inputs.
5. The steps involved in Genetic Algorithm are as follows:
6. Initialization – This is the first step in the algorithm. Here, we define the problem and the expected result. We also set the population on which the test should be conducted.
7. Fitness Function – This function generates the set of input values that will give us the output. This function gives a fitness score to each input value.
8. Selection – Once we have the input set ready, we select the best input values which would give us the best possible output values. The inputs are selected based on the fitness score given by the fitness function.
9. Crossover – This is the step where is process the selected input and get the output values. These output values would be the best of all the values that could have possibly been generated.

1. Mutation – Even though we select the best input set to get the best possible output values, there are certain changes in the actual outputs. This phenomenon is known as mutation. These outputs will again be sent to the fitness function to test if they are acceptable or not.

Flow Diagram:

Validated Output