Bahria University, Islamabad

Department of Software Engineering

Computer Programming Lab

(Fall-2023)

Teacher: Dr. Raja Suleiman

Group: Sher Hassan and Aryan

Enrollment: 01-131232-084, 01-131232-014

Date: 29/10/23

Task No:	Task Wise Marks		Documentation Marks		Total Marks (20)
	Assigned	Obtained	Assigned	Obtained	
1	3		5		
2	3				
3	3				
4	3				
5	3				

Comments:

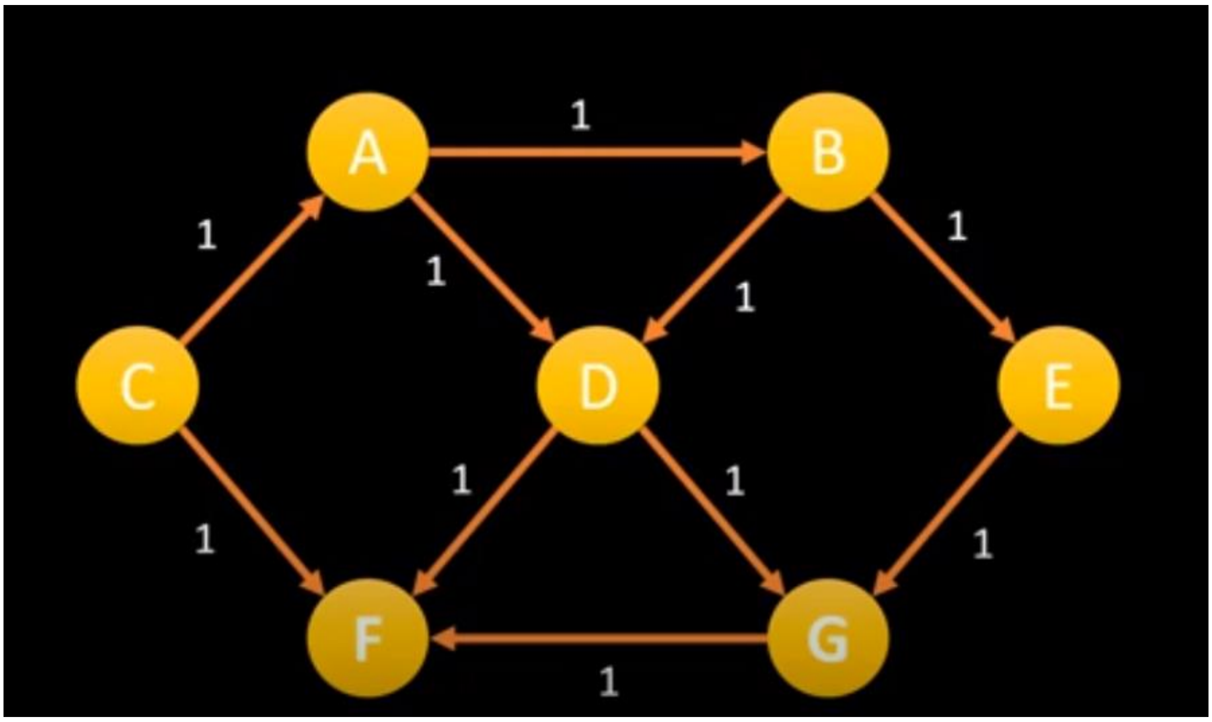
Signature

LAB

Problem Solving

Problem 1: Finding the Shortest Path.

Imagine you are developing a GPS navigation system. You are given a map with various locations and the roads connecting them. Your task is to write an algorithm to find the shortest path from one location to another. You can assume that you have a list of locations and the distance between each pair of locations. Your algorithm should output the shortest path and the total distance.



Algorithms:

1. The following are the locations of which we are going to find the shortest path.
2. As there is no weight given so we will take a default value 1 as weight.
3. We will take the first given location as our initial location and its value will be 0.
4. Now, we must find the shortest path to the other location as quickly as possible through the shortest distance.
5. So, for this purpose we will make a distance table for this scenario.

This is our initial Table.

Vertex	**Distance[V]**	**Previous vertex which gave distance**
A	-1	
B	-1	
C	0	
D	-1	
E	-1	
F	-1	
G	-1	

Initially, the values for all the vertices will be -1 except the starting because we have not visited them.

6. By using this table which will update we can find the shortest path.

Vertex	**Distance[V]**	**Previous vertex which gave distance**
A	1	C
B	2	A
C	0	-
D	2	A
E	3	B
F	1	C
G	3	D

7. Now, we can visit point A by going from point C so the distance will be 1. Now to go to point B first we will go to point C then go to point B that's why its distance is 2. By this way we update the table. And we will not revisit any location in this program because we want to find the shortest path.
8. We must create a distance array which will hold the distance from the starting vertex.
9. Then we must have created a path array which held the vertex through which we got the shortest distance.
10. Lastly, we need a queue which will have the path and the path with the smallest number will be given as output.

Problem 2: Sorting a List of Numbers.

You are working on a project where you need to sort a list of numbers in ascending order. Design an algorithm to efficiently sort a list of integers. You should consider various sorting algorithms, evaluate their time complexity, and choose the most suitable one for the task.

Algorithms:

Bubble Sort:

1. We take an array of list of numbers, and you compare each pair of neighboring numbers.

2.Then we use the If statement to compare the values with each other.

3. If they are in the wrong order (the larger one comes before the smaller one), we swap them.
4. We keep doing this until you go through the entire list without making any swaps.
5. It's like the larger numbers come to their correct positions.

Complexities:

It takes quadratic time, meaning it can be slow for large lists.

Selection Sort:

1. In Selection Sort, we repeatedly find the smallest (or largest) number from the unsorted part of the list.
2. Move it to the beginning (or end) of the sorted part.
3. We keep doing this until the entire list is sorted.
4. It's like selecting the right number for each position one at a time.
5. Selection Sort is also straightforward but not very efficient for big lists.

Complexities:

Also quadratic time, so it's not very efficient for large datasets.

Insertion Sort:

1. Think of Insertion Sort as if we're sorting random numbers.
2. The program will randomly pick a number.
3. The program inserts each card into its correct position among the numbers, making sure they are in order.
4. The program keeps doing this until all the numbers are sorted.
5. It's efficient for small lists or almost sorted data.

Complexities:

Quadratic time, but it's more efficient for nearly sorted or small lists

Merge Sort:

1. We divide our list into smaller halves,
2. We sort those smaller halves,
3. We merge them back together to create a fully sorted list.
4. It's like sorting smaller pieces and then combining them.
5. Merge Sort works well for larger lists and is always reasonably efficient.

Complexities:

It takes n log n time and works well for large lists.

Quick Sort:

1. In Quick Sort, we pick a "pivot" element from the list.
2. We rearrange the other elements so that all elements smaller than the pivot come before it, and all elements larger come after it.
3. We then repeat this process for the smaller and larger parts.
4. Quick Sort is usually fast, but it can be slower in certain situations.

Complexities:

It's typically n log n but can be quadratic in the worst case.

Heap Sort:

1. We repeatedly remove the largest element and place it in the sorted part of the list.
2. We then fix the heap.
3. This process continues until all elements are sorted.
4. Heap Sort is efficient and works well for large lists.

Complexities:

N log n time, making it efficient for large lists.

Most suitable sorting algorithm for the task:

Using Quick Sort will be the best because we need an algorithm that is usually faster and don't mind the worst-case time complexity of O(n^2) (which is rare and can be mitigated with optimizations). Quick Sort is efficient and commonly used in practice so, we will use Quick start algorithm to sort our list in ascending order.

Problem 3: Calculating Fibonacci Numbers.

The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding. Ones (e.g., 0, 1, 1, 2, 3, 5, 8, 13, ...). Write an algorithm to calculate the nth Fibonacci number. Your algorithm should be efficient and capable of handling large values of n.

Algorithms:

1. Set up a 2x2 matrix, named 'F', with initial values [[1, 1], [1, 0]].

2. If n is 0, return 0 as the result, since the 0th Fibonacci number is 0.

3. To calculate 'F' raised to the power of n - 1, use a function called 'power'.

4. In the 'power' function, we create another 2x2 matrix, named 'M', with values [[1, 1], [1, 0]].

5. Loop from 2 to n.

6. Within each loop, multiply 'F' by 'M' and update 'F' with the result. This step is where there is matrix multiplication.

7. The top-left element of 'F' now contains the nth Fibonacci number.

8. Return this top-left element as the result. By this way we can find the nth Fibonacci number.

Problem 4: Inventory Management.

You are tasked with creating an algorithm for a store's inventory management system. Your algorithm should be able to add and remove items from the inventory, update the quantity of existing items, and generate reports of the items and their quantities. Design an algorithm that efficiently manages the store's inventory based on these requirements.

Algorithms:

Initialize Inventory:

First, we start with an empty inventory to store the items and their quantities. We can use a data structure like a list to do this.

Add Items:

Then we add items to the inventory, we create a function that takes the item name and quantity as input. This function will add the item to the inventory or update the quantity if the item already exists.

Remove Items:

Then we create another function to remove items from the inventory. This function will take the item name as input and remove it from the inventory if it exists.

Update Quantities:

Then we can update the quantity of existing items, we can create a function that takes the item name and the new quantity as input. This function will update the quantity for that item in the inventory.

Generate Reports:

We design a function to generate reports. This function will provide options to generate different types of reports, such as a list of all items and their quantities, items with low stock, or any other relevant reports.

User Interface:

Depending on the complexity of your project, we design a user interface (UI) for store employees to interact with the inventory system. This UI calls the functions we've created and provide an easy way to manage the inventory.

Github Repository:

https://github.com/Sher-Hassan/Assignment11.git