Algorithms: Theory, Design and Implementation

5SENG003C.2

Report

Name	- Amantha Sandupa Mihiranga
IIT ID	- 20201262
UOW ID	- w1839499

Array

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. Arrays are also very easy to use for sorting. Such as Bubble sort, Insertion sort. Also we can search in binary.

Advantages

- An array can store multiple values in a single variable.
- Access to an element is very easy by using the index number.
- Arrays are fast as compared to primitive data types.
- 2D Array is used to represent matrices.
- We can use multi dimensional arrays
- Arrays can be used for CPU scheduling.

Disadvantages

- The array has a fixed size. Once declared the size of the array cannot be modified.
- We cannot increase or decrease the size of the array at runtime.
- Taking a lot of memory
- Can store only similar data types

Linked List

A linked list is a sequence of data structures, which are connected together via links. They can be used to implement stacks, queues, and other abstract data types. That's why i use linked list for the pathfinding implementation.

Advantages

- The size of the linked list can increase or decrease at run time so there is no memory wastage.
- A linked list is a dynamic data structure.

Disadvantages

- reverse traversing is not possible in the linked list.
- Random access is not possible in a linked list due to its dynamic memory allocation.

Depth First Search

- Solving a maze puzzle by using DFS is very easy.
- For directed graphs - O(V) + O(E) = O(V + E). For undirected graphs - O(V) + O (2E) ~ O(V + E).
- There are 3 main steps in DFS
    1. Visit a node
    2. Mark node as visited.
    3. Recursively visit every unvisited node
- DFS can be implement using stack data structure

Benchmark Test

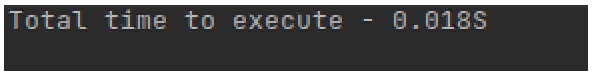
This is a very simple benchmark test. I used currentTimeMills() method to calculate the execution starting time and ending time

```
//calculating the execution time for a benchmark
long start = System.currentTimeMillis();

/*
the program
 */

//calculate the execution time
double end = System.currentTimeMillis();
double totalTime = (end - start)/1000;
System.out.println("Total time to execute - "+totalTime +"S");
```

To check this benchmark, I used 3 different text files to see if it works properly. These are some of the output screenshots ,

Test - 1

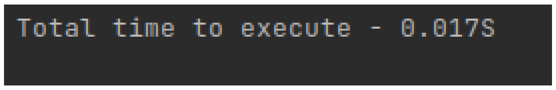
Input file	- maze30_2.txt
Rows	- 30
Colloms	- 30



```
Total time to execute - 0.018S
```

Test - 2

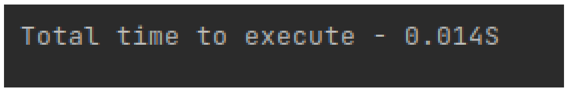
Input file	- maze20_2.txt
Rows	- 20
Colloms	- 20



```
Total time to execute - 0.017S
```

Test - 3

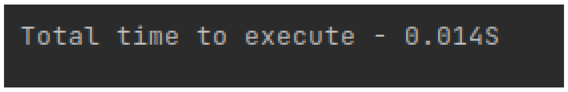
Input file	- maze25_2.txt
Rows	- 20
Colloms	- 20



```
Total time to execute - 0.014S
```

Test - 4

Input file	- maze10_2.txt
Rows	- 10
Colloms	- 10



```
Total time to execute - 0.014S
```