****

**CZ2001 Algorithms Example Class 4**

**Report**

**SS3 Team 2**

**Joshua Martin Tjahjadi**

**Kwek Ai Hui**

**Jason Lee**

**Shantanu Kamath**

**Jordan Kwong**

**Suyash Lakhotia**

SCHOOL OF COMPUTER ENGINEERING

NANYANG TECHNOLOGICAL UNIVERSITY

Contents

[1. INTRODUCTION 3](#_Toc434177655)

[1.1 Description of Problem 3](#_Toc434177656)

[2. IMPLEMENTATION 4](#_Toc434177657)

[2.1 Breadth First Search 4](#_Toc434177658)

[2.2 Preprocessing to compute all-pairs shortest path 4](#_Toc434177659)

[2.3 Time Complexity 5](#_Toc434177660)

[3. STATISTICS 6](#_Toc434177661)

[3.1 5000 Vertices 6](#_Toc434177662)

[3.2 10000 Vertices 7](#_Toc434177663)

[4. CONCLUSION 8](#_Toc434177664)

[5. REFERENCE 9](#_Toc434177665)

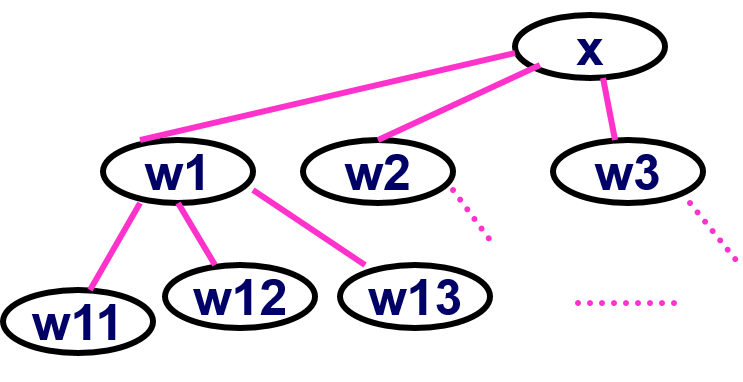
# 1. INTRODUCTION

## Description of Problem

In this example class, we are expected to find the shortest-path in an undirected unweighted graph (i.e. given two vertices u and v, return the shortest path and the length measured by the number of edges between u and v in the graph). The program should return a special value (e.g. -1) if there is no path between the two vertices.

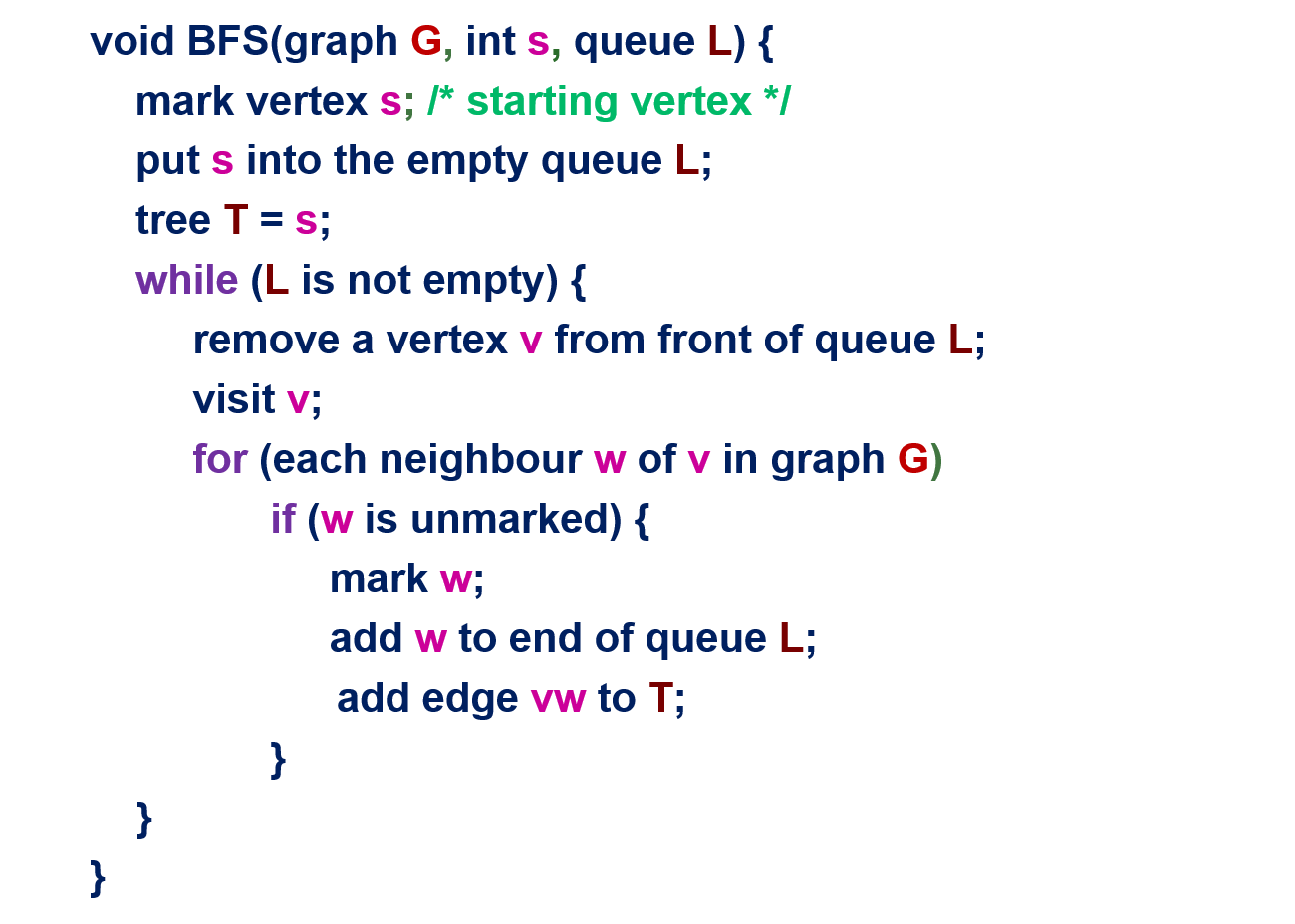
# 2. IMPLEMENTATION

## 2.1 Breadth First Search

****

**Figure 1: Graph (Taken from the lecture notes)**

Given a graph G = (V, E) and a source vertex x, the BFS algorithm will search the edges directly connected to x before visiting the further vertices. To monitor which vertices will be visited next, we used a queue.



**Figure 2: BFS Algorithm**

## 2.2 Preprocessing to compute all-pairs shortest path

In this example class, preprocessing can be used to run BFS on every vertex as the starting vertex. The result of the preprocessing is saved in a linked list so that it is possible to answer a query by looking up the records of shortest paths.

## **2.3 Time Complexity**

Using the breath first search algorithm, we know that:

* Every edge will be processed once in the while loop for a total cost of Θ (|E|).
* Every vertex is queued and dequeued once, for a total cost of Θ (|V|).
* Theoretical worst-case time complexity for breath first search is Θ (|V| + |E|) if graph is represented by an array of adjacent lists, or Θ (|V|2 + |E|) if graph is represented by an adjacency matrix as each vertex may take Θ (|V|) to scan for its neighbours.
* For this experiment, the worst case time complexity would be Θ (|V|2 + |E|\*|V|) since we are checking the code for each vertex.

# 3. STATISTICS

## 3.1 5000 Vertices

|  |  |
| --- | --- |
| Number of Edges | CPU Time (Nanoseconds) |
| 1000 | 149113224 |
| 5000 | 992811512 |
| 10000 | 1578954037 |
| 50000 | 3399709203 |
| 100000 | 8097287082 |

## 3.2 10000 Vertices

|  |  |
| --- | --- |
| Number of Edges | CPU Time (Nanoseconds) |
| 1000 | 144516804 |
| 5000 | 220176445 |
| 10000 | 3684219384 |
| 50000 | 11461491827 |
| 100000 | 23898366503 |

# 4. CONCLUSION

The theoretical time complexity for BFS matches with our results showing that the time complexity of a BFS is indeed directly related to number of edges and number of vertices.

Hence, the time complexity is θ (|V| + |E|) for adjacency lists.

The results prove the correctness of the implementation of our algorithm.

# 5. REFERENCE

1. Lecture materials on NTULearn.