**Algorithms and Data Structures CMPU2001: Assignment 1**

**Vilim Mikic C21737525**

# Introduction

In this assignment, I will try to explain and present different types of diagrams for minimum spanning trees, algorithms they entice as well as graph traversal algorithms.

I will focus on depth-first traversal algorithm which was implemented with recursion (Cormen’s version) and breadth-first graph traversal algorithm which was implemented by using a queue (Cormen’s version).

Regarding Prim’s Minimum Spanning Tree algorithm, Kruskal’s Minimum spanning tree algorithm and Dijkstra’s Shortest path tree algorithm, there will be graphs with an explanation.

This is the graph that above mentioned algorithms will be tested on.

Diagram

Description automatically generated

# Explanation

## Cormen’s Depth-first algorithm

* Traverses a network in a recursive manner
* The colour, parent and ending time arrays are initialized by the DF method using the starting vertex “s” as an input. It keeps track of the time using a global variable.
* It calls the DFS\_Visit method for the starting vertex. The method takes a vertex “u” as an input and recursively explores all its neighbours.
* It keeps track of the colours of the vertices (white, grey, black), discovery and finishing time of each vertex.
* Algorithm then backtracks and sets the colour and finishing time for the current vertex. When all adjacent nodes have been visited, the node is blackened. This step is important, so we don’t end up in an infinite loop.
* The complexity of this algorithm: O (V + E) where V is number of vertices, and E is number of edges.
* It is a greedy algorithm and it goes deep into child node until the end and then changes to a different child node.

## Cormen’s Breadth-first algorithm

* Traverses a network using an iterative approach by visiting all its neighbours before moving to the next level.
* BFS takes a starting vertex “s” as an input and initialises the colour and parent arrays. It also creates a queue and adds the starting vertex to it.
* It keeps the track of the colours of the vertices (white, grey, black).
* It will dequeue the first vertex and process it (print message to display which vertex is currently being visited along which edge. After that, it will enqueue all of the neighbours of the dequeued vertex that haven’t been visited as of yet to update their colours and parents.
* This process will be repeated until the queue is empty.
* The complexity of BFS is O(V+E) where v is number of vertices e is number of edges, same as DFS.
* The main difference is that BFS goes broad, and then deep. It is also a greedy algorithm.

## Prim’s Minimum Spanning Tree algorithm

* Basic idea is to grow the MST one vertex at a time, starting with a vertex “s” and always adding the vertex that has the smallest weight edge, connecting it to the current minimum spanning tree. It will keep doing so until all of the vertices are used up, and we have a full tree.
* Algorithm initialises three arrays – dist (distance of each vertex from the starting vertex), hPos (the position of the vertices in the heap) and parent (parent of each vertex in the MST). Distance is initially zero and is inserted into the heap as such.
* While loop continues until the heap is empty. In each iteration, vertex with the smallest weight is removed from the heap and it’s weight is added to the weight sum. Following that, the weight is then made negative to mark it as visited.
* For each neighbour of the vertex “v”, if the weight of the neighbour is less than its current distance in the “dist” array, the distance of the neighbour is updated to the weight, and parent is updated to “v”.
* If the neighbour is not already in the heap, it’s inserted, and if it already is there, its position in the heap is updated (siftUp).
* In this case, the implementation was with heap, but it can also be implemented with priority queues.
* Time complexity of Prim’s MST algorithm is O (E log V) where E is number of edges and V is the number of vertices, meaning that the algorithm execution time will increase proportionally with the number of edges and logarithmically with the number of vertices in the graph. It is also considered a greedy algorithm.

## Dijkstra’s Shortest Path algorithm

* Implementation is quite similar to Prim’s MST in java. Initialisation part is very same, just taking away wgt\_sum.
* Finds a shortest path from a source vertex “s” to all vertices in a weighted graph.
* It works by maintaining a set of visited vertices and a set of unvisited vertices. Initially, distance from source to all other vertices is set to infinity, apart from the distance to itself, which is set to 0.
* The algorithm runs a loop until the heap is empty.
* In each iteration of the loop, the function removes the vertex with the smallest weight from the heap and updates the distances of its neighbours if a shorter path is found.
* If neighbour vertex “u” is not in the heap, it gets inserted, otherwise the function sifts up the neighbour vertex to maintain the heap.
* After the algorithm is done, the shortest path tree gets reconstructed using the “parent” array and prints the SPT from the source vertex to each vertex in the graph.
* Time complexity of Dijkstra’s SPT algorithm is O ( E log V), same as Prim’s MST. It is important to note that the cost of inserting and removing a vertex from a binary heap is O(logV) which counts toward the overall cost.

## Kruskal’s Minimum Spanning Tree algorithm