**3.5 Homework Task Submission**

|  |  |
| --- | --- |
|  |  |
| **Answer all parts of this Homework task and submit your work into VSV Online as PDF and SnapApps/Edgy .xml or Python3/Trinket code files. This homework is required to demonstrate learning outcomes to a satisfactory standard.**  **For this Homework Submission 2 Files are expected:**  **• a PDF file with text responses to Problem, parts a, b, c, e**  **• an exported coding file in .xml from SnapApps/Edgy or using Python3/Trinket addressing Problem, part d** |
|  |

**Spreading rumours**

|  |  |
| --- | --- |
| Image result for friendship network | **Consider a friendship network modelled by a graph G={V,E}, where V={people} and E={friendship}.** |

1. In a friendship network, describe how can you find out how long it will take for a particular rumour to spread from a single source, if everyone is a snitch and passes the rumour onto their friends? Describe how this information can be modelled by the graph ADT (Abstract Data Type).

In the friendship network, you can model who is friends with who and see how long it would take for one person to tell another friend. As such, you can use a graph ADT to create relationships between nodes (which represent people/friends) and edges that represent friendships, each of which have a label denoting how long it would take for one person to tell the other. The answer to how long it would take would then simply be a sum of the traversal of the rumour from one node to the other.

1. When would you stop measuring the spread of the rumour in a friendship network? How is this action modelled by the graph ADT? Justify your response.

You would stop measuring the spread of the rumour when all friends have heard the rumour. This could be modelled in the ADT by checking when all nodes have been visited, i.e. when all people have received the rumour.

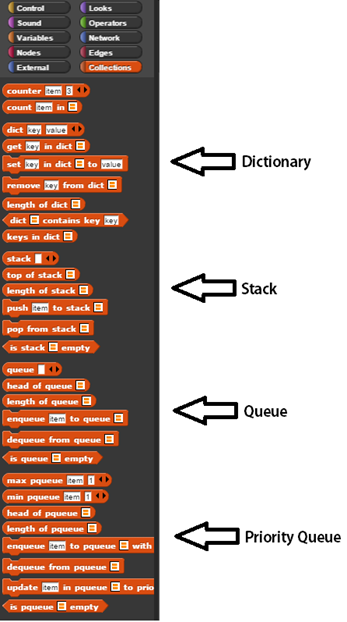
1. Which of the graph traversal algorithm(s) studied and ADT(s) could be implemented to find the solution to the rate of rumour spread? Justify your choice.

A BFS algorithm could be implemented to find the solution to the rate of rumour spread, since the behaviour of this traversal matches how a rumour would spread as well, where each node spreads to each of its adjacent nodes. Since the behaviour matches, the spread could be modelled by this algorithm

1. Code your selected algorithm in SnapApps/Edgy or Python3/Trinket using one of the graph traversal algorithms studied in our course {Prims MST, Depth First Search, Breadth First Search, Best First Search}. As a guide, you can use the pseudocode provided on the last page of this task or other **correct** **versions** of these algorithms.
2. In another friendship network, the same rumour is started by 3 different people, again everyone is a snitch and passes this rumour onto their friends. What kind of ADTs and algorithm could be used to measure how long it will it take for this particular rumour to spread to friends of friends of friends? Describe if, and what modifications would be needed to your selected ADTs and algorithm from **part d)** in order to solve this problem.

In order to solve this problem, you could populate the initial queue with all the initial nodes, so the queue could start with the 3 different people instead of just the one initial node. As such, the BFS would spread from the 3 nodes and you could use a similar technique of measuring the depth to find how long it would take on an unweighted graph.

ADT functionality available in SnapApps/Edgy for stacks and queues can be found under “Collections”, these model the Data Signatures of ADTs as outlined in the VSV Online course.



If you are coding using Python3/Trinket then you will have to implement your own stack and queue operations using lists. Refer to examples from the textbook “An Introduction to Algorithmic Thinking – Algorithmics (HESS) Student Guide” on page …….

|  |  |
| --- | --- |
| **Algorithm DFS (G, Vnode, Target)**  *// Input G is a graph of nodes and edges, connected and undirected*  *// Input Vnode is a starting node within the graph G*  *// Input Target is the target node within the graph G*  Create stack S  Create list NodesVisited  Add Vnode to NodesVisited  **While** (Vnode doesn’t equal Target) **do**  *// Mark Vnode attribute as seen*  Vnode.seen:=True  **For each** neighhouring node X of Vnode **do**  **If** (X.seen is False) **then**  Push X onto S  **End if**  **End do**  Vnode:=Pop item from S  Add Vnode to NodesVisited  **End do**  Print NodesVisited  **End Algorithm** | **Algorithm BFS (G, Vnode, Target)**  *// Input G is a graph of nodes and edges, connected and undirected*  *// Input Vnode is a starting node within the graph G*  *// Input Target is the target node within the graph G*  Create queue Q  Create list NodesVisited  Add Vnode to NodesVisited  **While** (Vnode doesn’t equal Target) **do**  *// Mark Vnode attribute as seen*  Vnode.seen:=True  **For each** neighbouring node X of Vnode **do**  **If** (X.seen is False) **then**  Enqueue X into Q  **End if**  **End do**  Vnode:=Dequeue item from Q  Add Vnode to NodesVisited  **End do**  Print NodesVisited  **End Algorithm** |
| **Algorithm PrimsMST(G, v0)**  *// Input: G={V,E} a weighted, undirected, connected graph*  *// Input: v0 is a starting node in the graph G, this can be any node chosen at random*  *// Output: T={V,E} a minimum spanning tree*  *// Output: MSTCost, the sum of weights of the MST*  *// Initialise the MST T={v0}*  Add node v0 to graph T  Set MSTCost to zero  **While** (T does not include all the nodes of G) **do**  Choose minimal weighted edge {U-V} where node U is in tree T and node V is not  Add node V to T  Add edge {U-V} to T  Add weight of edge {U-V} to MSTCost  **End do**  **End Algorithm** | **Algorithm BestFS (G, Vnode, Target)**  *// Input G is a graph of nodes and edges, connected and undirected*  *// Input Vnode is a starting node within the graph G*  *// Input Target is the target node within the graph G*  Create priority queue Q  Create list NodesVisited  Add Vnode to NodesVisited  **While** (Vnode doesn’t equal Target) **do**  *// Mark Vnode attribute as seen*  Vnode.seen:=True  **For each** neighbouring node X of Vnode **do**  **If** (X.seen is False) **then**  Enqueue X with ranking into priority Q  **End if**  **End do**  Vnode:=Dequeue item from Q  Add Vnode to NodesVisited  **End do**  Print NodesVisited  **End Algorithm** |