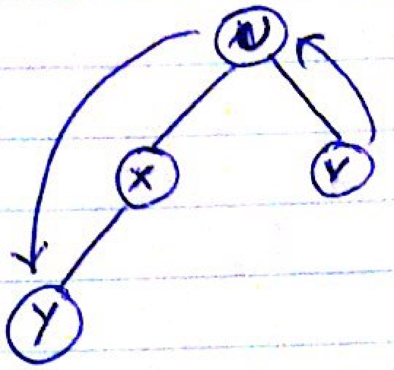
CS430 Assignment - 8

1) Breadth First Search:

a) Undirected Graph:

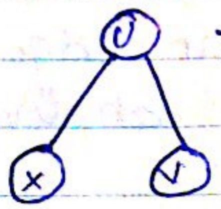
i) 

In this graph VU → Back edge
UY → Forward edge.

In Breadth first search, all the edges of parent node (U) is explored before exploring the edges of child node (V). Since it is an undirected graph, we would have explored the edge U — Y while finding the edges of U. So backedge V—U is not possible, similarly there can't be any back edge in an BFS of an undirected graph.

An edge between U at level 0 and Y at level 2 is known as a forward edge. Let's assume that edge U — Y exist. Conceptually in BFS, all the edges of node "U" would have been explored in level 1 and there is no possibility for an edge to occur between a node at level 0 and a node at level 1. This proves that our assumption is wrong and so there are no forward edges in a breadth first search of an undirected graph.

ii)



→ height = U.d

→ height = V.d

To prove : $V.d = U.d + 1$

By BFS algorithm,

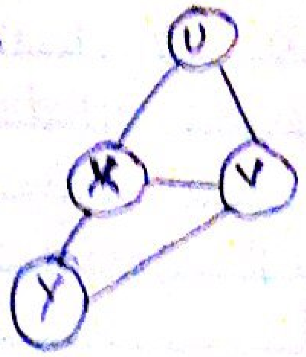
1. if a child of a node is in WHITE colour, we would change it's colour to GREY. (V.colour = GREY)
2. Calculate the distance of the node (V) by the formula ($V.d = U.d + 1$)
3. Enqueue the vertex.

By step we know that height of the node v is always height of the parent node + 1.

iii) 0

1

2



V-X and V-Y are cross edges.

Let's assume that we have an edge from node (V) at level 0 to node (Z) at level 3 such that ($Z.d = V.d + 2$)
By BFS algorithm, we would have explored all the edges of node V at level 2. The node V can have cross edges to level 1 and level 2. i.e. to node X and to node Y and can never have an edge to nodes at levels greater than 2.
This proves that our assumption is wrong.
(VZ)→ ∴ $Z.d = V.d$ (or) $Z.d = V.d + 1$ (~~cr~~)
for cross edge (UV) $V.d = U.d$ (or) $V.d = U.d + 1$

b) Directed Graph:

i) Let's assume we have an edge $(\overrightarrow{UV})$. Forward edge is an edge from node (U) to the child of node (V). If there exists an ~~node~~ edge from ~~node~~ (U) to child of node (V), we would have visited ~~explored~~ it while exploring the edges of node (u). so there are no forward edge.

ii) For tree edge $(\vec{uv})$, to occur, node v must be a child of node u. By BFS algorithm the distance of child node is calculated by the formula

$$\text{distance of child node} = \text{distance of parent node} + 1$$

In case eg. (uv) → $v.d = u.d + 1$

iii) While using BFS of an undirected graph, a cross edge $(\vec{uv})$ can occur only if the edge v is at the same level of node u (or) one level greater than node u. Beyond that it is not possible for a node to occur sharing edge with node u. By this definition, always, $\boxed{v.d \le u.d + 1}$

iv) For Back edge $(\vec{uv})$ to occur, the edge u should be at same level/one level greater than node v

By BFS height of node u is always greater than or equal to the height of node v

∴ $\boxed{0 \le v.d \le u.d}$

② The inputs → Graph
L-values (for each vertex)

$R(u) = \{$ set of vertices reachable from u $\}$.

A vertex v is reachable from u if and only if there is a path $u \rightarrow v$ in G.

Algorithm:

for v in G:
    mark v undiscovered.
for every undiscovered vertex v in G, with the vertices sort based on L-value:
    $R(v) = L(v)$
    Mark v discovered
    perform reverse-DFS from v
        for every undiscovered vertex u:
            $R(u) = R(v)$
            Mark u discovered.

By reverse DFS, we explore every edge and vertex exactly once giving $O(V+E)$ time complexity.

③ a) Euler Theorem:

A connected Eulerian graph has even degrees for all vertices.

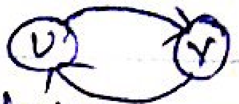
Proof:

Let P be the Euler circuit. It is listed as $v_1, e_1, v_2, e_2 \dots v$. Every time a vertex v_k is listed, two edges e_{k-1} and e_k are listed before and after the vertex. Since the circuit can only visit each edge once, edges are not repeated. Thus each vertex should have even degree.

A Connected graph with even degree vertices has Euler tour.

Proof by Induction:

Base case: Consider a graph G with 2 vertices and 2 edges between them



Graph G is eulerian.

Assumption: Assume that all connected graphs with m edges, where each vertex has even degree has an Eulerian tour.

Proof: Consider a connected Graph G with $k > m$ and each vertex has even degree. We shall start at vertex v and keep following edges arbitrarily selecting one after another until we return to v. Consider the trial as W and E be edges of W

Graph G-F has components C_1, C_2, C_3 C_k where each component will clearly have less than m edges and every degree is even because when we removed F, we removed even edges from each vertex

By induction each component has an eulerian circuit $F_1, F_2 \cdot F_k$. since G is connected there is a vertex a_i in each component C_i on both W and E_i

An Euler circuit can be defined in G by starting at v in W until reaching a_i, following the edges in C_p to come back to a_i, and then adding edges in W to reach a vertex a_j in another component and so on until we reach v.

b) Algorithm:

Let G be a graph with V vertices and E edges.

1. If graph has any odd vertex return that Euler tour does not exist.
2. Choose a vertex V at random
3. Follow edge one at a time. If there is a choice between a bridge and non-bridge, choose the non-bridge.
4. Stop when we run out of edge.

④ Running time for prim's algorithm (with fibonacci heap implementation) consist of

Extract min for each vertex $\rightarrow$ $V \times O(\log V)$
Decrease key for each edge $\rightarrow$ $E \times O(1)$

$$O(V \log V + E)$$

Let's implement the queue as an array of linked lists, with one element per weight value. This changes the running time of Extract-min(). consisting of scanning the array for first non-empty list element.

① If weight ranges from $1 \rightarrow |V|$ then it takes $O(V)$ per scan for Extract-min().

Total time $\rightarrow$ $O(V(V)+E) = O(V^2)$, which is worse than implementing queue with heap

② If weight ranges from $1 \rightarrow W$ ($W \rightarrow$ constant), then it takes $O(1)$ for Extract-min()

Total time $\rightarrow$ $O(V(1)+E) = O(V+E)$
$= \underline{O(E)}$

This is the best possible way to increase run time of prim's algorithm.

⑤ To prove: For A Graph G with distinct edge weights, there exists an unique MST.

Proof by contradiction:

Let's assume that there exists 2 MSTs. MST1 and MST2.

$E \rightarrow$ set of edges in MST2 but not in MST1.

Adding an edge to MST1 should create a cycle since it is a minimum spanning tree. Create a cycle by adding an edge $\boxed{e \in E}$ to MST1.

To create MST out of MST1, we have to remove one expensive edge from it. Since Graph G has distinct edge weights, we will have only one edge with most expensive weight.

2 condition:

1. e is most expensive edge:
   You will not get multiple MSTs.
2. e is not most expensive edge:
   MST1 was not a minimum spanning tree.