Australian National University
Research School of Computer Science

COMP3600/COMP6466 in 2016 – Quiz four

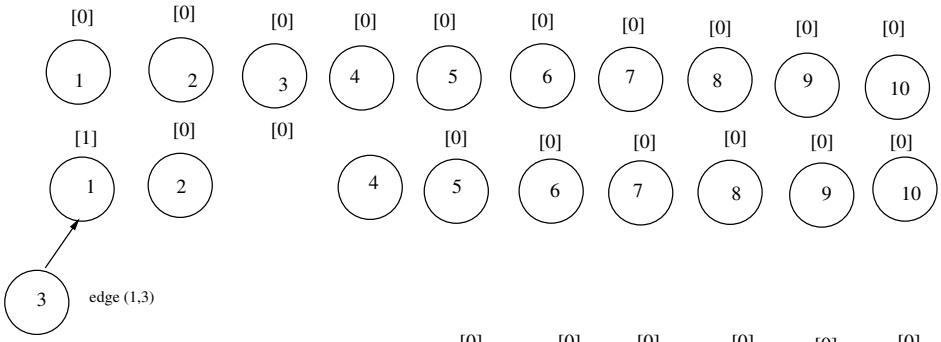
Due: 23:55pm Friday, October 14

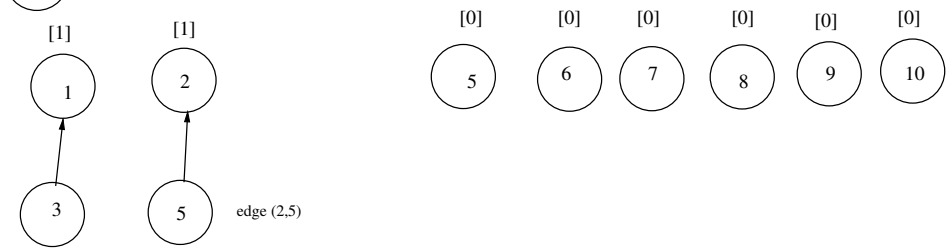
Submit your work electronically through Wattle. The total mark of this quiz worths 20 points which is worth 4 of the final mark.

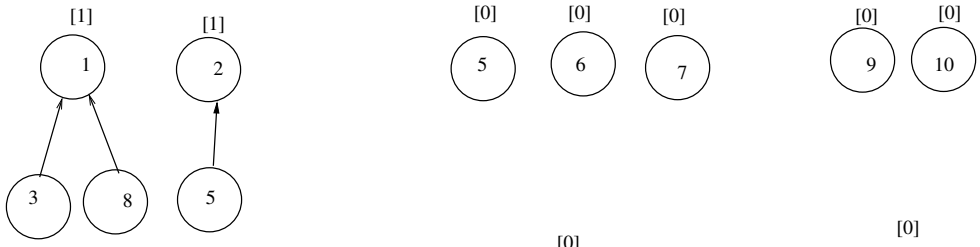
Question 1 (5 points).

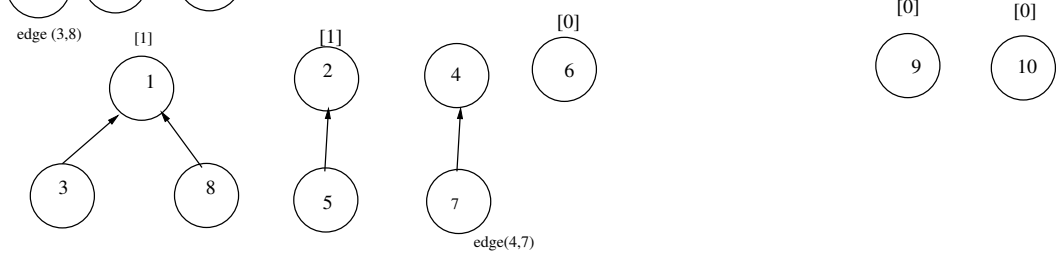
(a) Apply the forest consisting of (inverted) directed tree implementation of disjoint-set data structure, using both heuristics (path compression and union by rank), to find the connected components in a graph with 10 vertices and edges provided in this order: 1-3, 2-5, 3-8, 4-7, 6-7, 8-9, 1-10. (3 points)

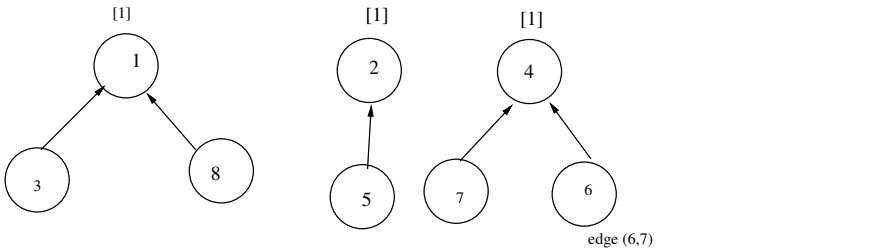
Answer:

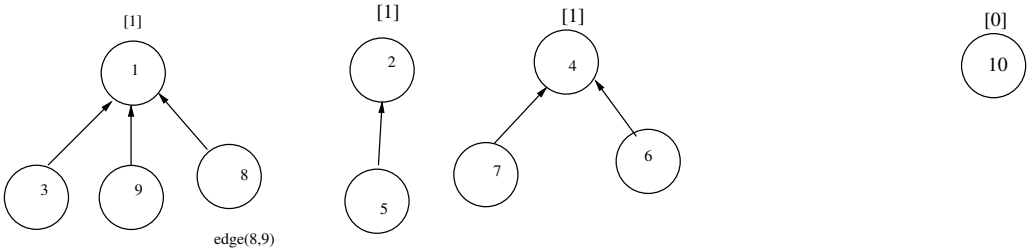

[0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
1 2 3 4 5 6 7 8 9 10
[1] [0] [0] [0] [0] [0] [0] [0] [0]
1 2 4 5 6 7 8 9 10
3
edge (1,3)

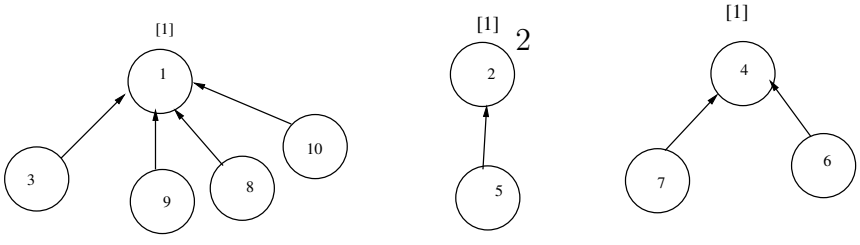

[1] [1] [0] [0] [0] [0] [0] [0]
1 2 5 6 7 8 9 10
3 5
edge (2,5)


[1] [1] [0] [0] [0] [0] [0]
1 2 5 6 7 9 10
3 8 5
edge (3,8)


[1] [1] [0] [0] [0] [0]
1 2 4 6 9 10
3 8 5 7
edge(4,7)


[1] [1] [1] [0]
1 2 4 9
3 8 5 7 6
edge (6,7)


[1] [1] [1] [0]
1 2 4 10
3 9 8 5 7 6
edge(8,9)


[1] [1] [1]
1 2 4
3 9 8 10 5 7 6
edge(1,10)

Notice that the numbers in square brackets are ranks. There is a choice when merging two trees of the same rank, so other solutions are possible.

(b) Consider a red-black tree formed by inserting n nodes with RB-INSERT. Argue if $n > 1$, the tree has at least one red node. (2 points)

Answer: Initially inserting the 1st node, its color will be black as it serves as the tree root. If the 2nd node is inserted, it will be colored with red, and there are no other red nodes, done.

Now consider inserting the ith node, it is colored with red using the algorithm of insertion for binary search trees. Let new inserted node be z (with red color).

Case 1: z has a red uncle and parent. For this case, change its parent and uncle as black, and push the red color (originated at its parent) to its grandparent, the grandparent becomes a new z, the original z is red.

Cases 2 and 3: z has a red parent and black uncle, z is either the left or right child of its parent. Thus, a left-right rotation can transform one case to another case. To Case 3, a rotation is applied, and z's color will not be changed.

In summary, it can be seen that at least one node in the tree is red. The claim is true.

(c) Given an algorithm that determines whether or not a given undirected graph $G = (V, E)$ contains a cycle. Your algorithm should be run in $O(|V|)$ time, independent of $|E|$. (2 points)

Answer: We perform DFS search by starting from a node in $s \in V$, when there is a **back** edge, then, there is a cycle in G. As there are only $|V|$ nodes in the graph, if there is such an edge, it can be detected by searching no more than $|V|$ edges. If G is disconnected, a node from each connected component is chosen as the starting node to perform DFS search. So, checking whether there is a cycle in each connected component takes $O(|V|)$ time. Thus, the time for checking a cycle in G is $O(|V|)$, not $O(|E| + |V|)$.

(d) Given a connected undirected graph $G = (V, E)$ where V is the set of nodes and E is the set of edges, devise an $O(|V| + |E|)$ algorithm to verify whether G contains any odd cycles. An odd cycle in a graph is a simple cycle that has odd numbers of edges. (3 points).

Answer: As G is connected, we perform a BFS search in G starting from any node s, let h be the number of layers in the BFS tree, then, the nodes in V is partitioned into h disjoint subsets $V_1 = \{s\}$, V_2 is the set of nodes whose distance to s is 1, and V_i is the set of nodes whose distance is $i - 1$ from s with $1 \leq i \leq h$.

One important property of BFS search is that there is not any edge in G crossing

different layers except neighboring layers.

We say that G does not contain odd cycles if there are no edges between nodes in in V_i for all $2 \leq i \leq h$. Otherwise, let (u, v) be an edge in V_i, and w be the lowest common ancestor in the BFS tree between u and v (i.e., w is the first common ancestor of both u and v in the tree), it forms an odd cycle by the path from w to u, the path from w to v, and edge (u, v), as the cycle contains $2l + 1$ edges assuming that the number of edges for each path is l.

Thus, the construction of BFS tree takes $O(|E| + |V|)$ time, while it takes $O(|E|)$ time by examine each edge in E to see whether its two endpoints are in the same layer. Thus, the odd cycle checking takes $O(|V| + |E|)$ time.

Question 2 (10 points).

(a) In which case should the Bellman-Ford algorithm instead of Dijkstra's algorithm be applied to solve the single-source shortest paths problem? (1 point)

Answer: In the case where the weights associated with the edges in the graph can be negative, only the Bellman-Ford algorithm can be applied to solve the single-source shortest paths problem.

(b) Given a connected undirected graph $G = (V, E)$, assume that each edge $e \in E$ has a non-negative weight, let e_{min} be an edge with the minimum weight in a cycle of G, prove or disprove that e_{min} will be contained by any minimum spanning tree in G. **Hints:** you may just give a counter-example if you disprove the claim. (3 points)

Answer: The claim is false. Consider a graph with a central node v_0, and other n nodes $v_1, v_2, \ldots, v_n$. Assume that there is an edge between v_0 and v_i with $1 \leq i \leq n$ with weight 1. There is an edge between v_i and v_{i+1} for all i with $1 \leq i \leq n - 1$ and an edge between v_n and v_1, the weights associated the edges in the cycle $C = (v_1, v_2, \ldots, v_n, v_1)$ are in the set $\{3, 4, 5\}$. Clearly, the minimum weight of the edges in C is 3, none of the edges in the MST of G which is a star central node v_0.

(c) You are given a set of cities, along with the pattern of highways between them, in the form of an undirected graph $G = (V, E)$. Each stretch of highway $e \in E$ connects two of the cities, and you know its length in kilometers $l(e)$. You want to get from city $s \in V$ to city $t \in V$. There is one problem: your car can only hold enough petrol to cover L kilometers. There are petrol stations in each city, but not between cities. Therefore, you can only take a route if every one of its edges has length no greater than L.

- [i] Given the limitation on your car's fuel tank capacity, show how to determine in linear time whether there is a feasible route from s to t. (3 points)

Answer: We assume that G is represented by adjacency lists, a subgraph $G' = (V, E')$ can be derived by removing all edges with length greater than L, which takes $O(|V|+|E|)$ time by scanning all edges in G. Notice that G' may or may not be connected. We then check whether s and t in the same connected component of G' which can be done within $O(|V|+E|)$ time using a DFS/BFS-based algorithm for finding connected components in a graph. If both s and t in the same connected component, then there is a feasible route from s to t; otherwise, there is no such a route between the two cities. Clearly, this takes linear time to find such a route, using DFS/BFS traversal on the connected component in which both s and t are contained.

- [ii] You are now planning to buy a new car, and you want to know the minimum fuel tank capacity that is needed to travel from s to t. Give an $O((|V|+|E|)\log|V|)$ algorithm to determine this. (3 points)

  Answer: Sort the edges in G in increasing order of the weight, let $e_1, e_2, \ldots, e_m$ be the sorted edge by their weights in increasing order. W then choose an edge (e.g. edge e_i) from the sorted sequence such that both s and t are in the same connected component of an induced subgraph $G'_i = (V', \{e_1, e_2, \ldots, e_i\})$ of G by edges with weight no greater than $l(e_i)$ and $l(e_i)$ is as small as possible, where $l(e_i)$ is the minimum fuel capacity of the new car. Finding such an edge with the minimum length takes $O(\log|E|) = O(\log|V|^2) = O(\log|V|)$ time, and detect whether s and t in the same connected component of G_i takes $O(|V|+|E|)$ time. In total, it takes $O((|V|+|E|)\log|V|)$ time.