Australian National University
Research School of Computer Science

COMP3600/COMP6466 in 2015 – Quiz four

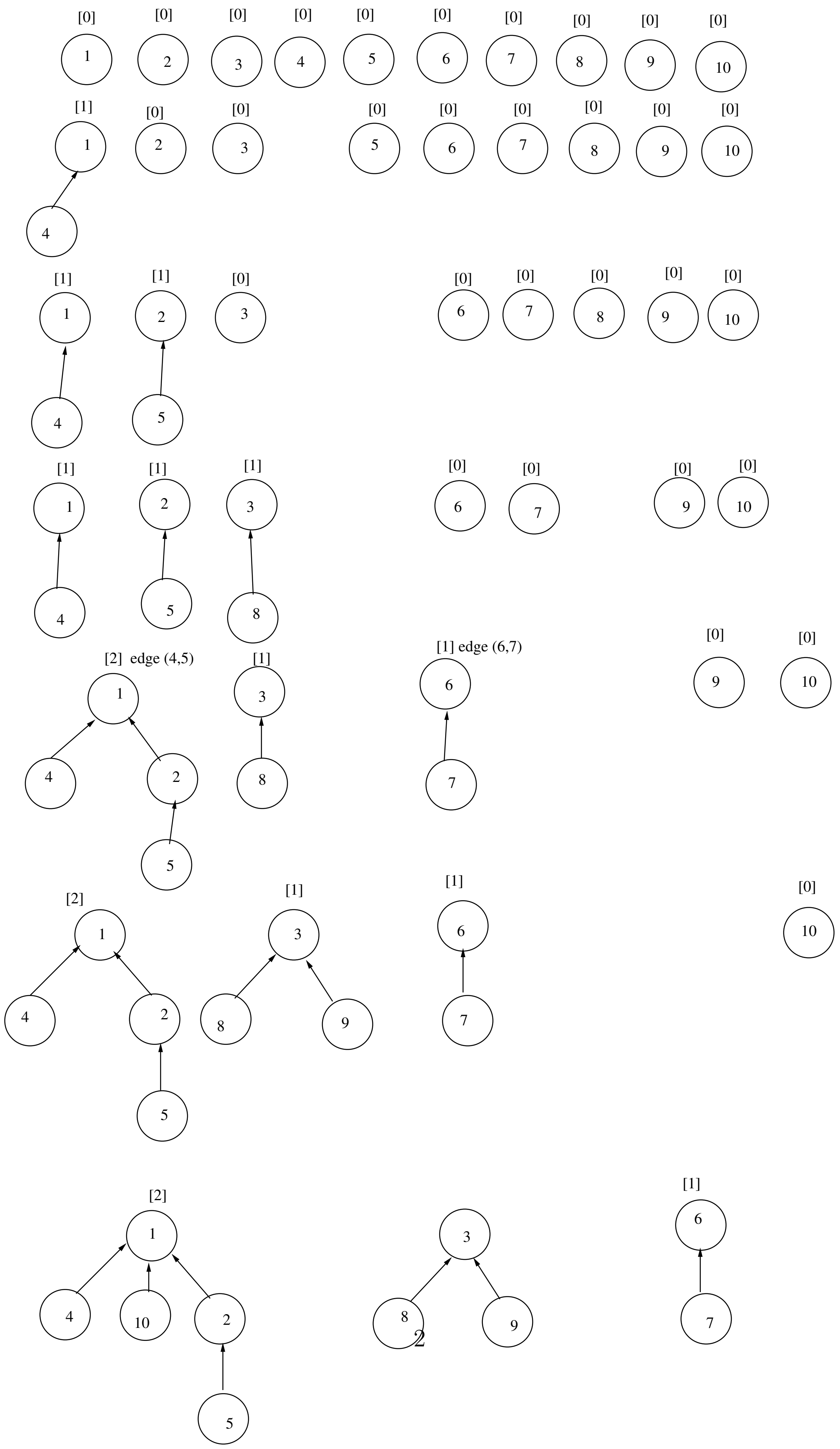
Due: 23:55pm Friday, October 16

Submit your work electronically through Wattle. The total mark of this quiz worths 15 points.

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: 1-4, 2-5, 3-8, 4-5, 6-7, 8-9, 1-10. (2 points)

Answer:Answer:



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

(b) Given an undirected connected graph $G = (V, E)$, apply the Breadth-First-Search to G, then the edges in E are partitioned into two disjoint sets E_1 and E_2, where E_1 is the set of tree edges in the BFS tree and E_2 is the set of non-tree edges, i.e., $E = E_1 \cup E_2$ and $E_1 \cap E_2 = \emptyset$. Show that there is no such non-tree edge $e = (u, v) \in E_2$ with $|d(u) - d(v)| \geq 2$, where $d(u)$ and $d(v)$ are the depths of nodes u and v in the BFS tree, the depth of the tree root is one. (2 points)

Answer: As G is undirected, connected graph, we perform a BFS search in G starting from any node s, then all nodes will be contained by the BFS tree rooted at s.

Assume that there exists a non-tree edge (u, v) with $|d(u) - d(v)| \geq 2$. Assume that $d(v) < d(u)$ (for the case $d(v) > d(u)$ the discussion is similar, omitted), then following the BFS traversal, when visiting node v, (1) node u is either visited and colored with 'black' or 'grey', which means that u is visited prior to v, i.e., $d(u) \leq d(v)$, this contradicts the assumption that $d(v) \leq d(u)$, thus, this is impossible. (2) Node u has not been visited when visiting node v, then u will become a child of v and v becomes the parent of u in the BFS tree, as there is an edge (u, v) incident to node v, which means that $|d(u) - d(v)| \leq 1$, this contradicts the assumption that $|d(u) - d(v)| \geq 2$. The claim thus holds.

Question 2 (5 points).

(a) 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: We first choose a node r as the starting node and perform BFS traversal in G. We obtain a BFS tree rooted at r. Let $d(v)$ be the depth of node v in the tree from the root r and $d(r) = 1$. We then check each non-tree edge $(u, v) \in E$, we distinguish into two cases:

- if $d(u) = d(v)$, G contains an odd cycle.
- if $d(u) \neq d(v)$ but both of them are either even or odd, then G contains an odd cycle

Otherwise, G does not contain any odd cycles.

The analysis of time complexity of the algorithm is as follows. The construction of the BFS tree takes $O(|V| + |E|)$ time; while checking there is a non-tree edge $e \in E$

meeting either of the two conditions takes $O(1)$ time for each non-tree edge, there are no more than $O(|E|)$ non-tree edges. Thus, the algorithm takes $O(|V| + |E|)$ time.

(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).

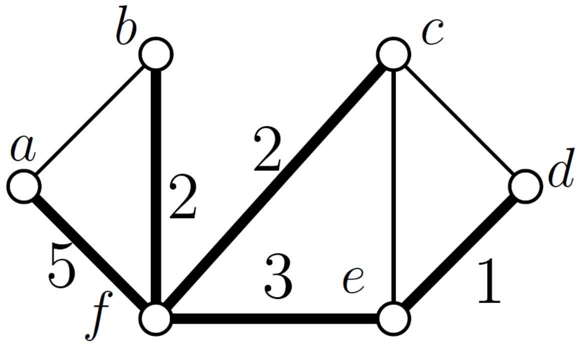
Case 1: z has a red uncle and parent. For this case, change its parent and uncle as black, and push the red color (originated at its parent) to its grandparent, the grandparent becomes a new z, the original z is red.

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.

Question 3 (5 points).

(a) The solid edges of the graph G shown below form a spanning tree T of G. The edge-weights of T are $w(a, f) = 5$, $w(b, f) = 2$, $w(c, f) = 2$, $w(d, e) = 1$, and $w(e, f) = 3$. Determine the minimum possible weights of the edges (a, b), (c, d), and (c, e) if T is a minimum spanning tree of G. (2 points)



Answer: If T is an MST, then, the minimum possible weights of the edges (a, b),

(c, d), and (c, e) are $w(a, b) = 5$, $w(c, d) = 3$ and $w(c, e) = 3$. If any of these weights was smaller, we would get a spanning tree with smaller cost by applying Kruskal's or Prim's algorithm.

(b) Given a connected undirected graph $G = (V, E)$, assume that each edge $e \in E$ has a non-negative weight, let e_{max} be an edge with the maximum weight in a cycle of G, show that e_{max} will not be contained by any minimum spanning tree in G. (1.5 points)

Answer: We show this by contradiction. Assume that T is an MST that contains the edge e_{max}, as e_{max} is an edge in a cycle C, then, at least one edge e in C is not contained by T, and the weight of e, $w(e)$ is strictly less than the weight $w(e_{max})$ by the assumption, i.e., $w(e) < w(e_{max})$. We now remove edge e_{max} from T, the vertex set of T now is partitioned into two disjoint subset V_1 and V_2, the nodes in each V_i a subtree induced by the removal of e_{max} from the T, clearly, $e \in (V_1 \times V_2) \cap E$, the addition of edge e results in another tree $T' = T \setminus \{e_{max}\} \cup \{e\}$, the cost of T' thus is $W(T') = W(T) - w(e_{max}) + w(e) < W(T)$, where $W(T)$ is the weight of the MST T, this contradicts that T is an MST of G. Thus, e_{max} will not be contained by any MST in G.

(c) Suppose a CS curriculum consists of n courses, all of them mandatory. The prerequisite graph G has a node for each course, and an edge edge from course v to course u if and only if u is a prerequisite for u. Find an algorithm that works directly with this graph representation, and computes the minimum number of semesters necessary to complete the curriculum, assuming that a student can take any number of courses in each semester. The running time of your algorithm should be linear. (1.5 points)

Answer: It can be seen that the prerequisite graph is a DAG. The algorithm proceeds as follows. Let S_0 be the set of nodes whose in-degrees are zeros (i.e., no prerequisite), remove all nodes in S_0 from $G_0 = G$ and reduce the in-degree of other nodes if there are incoming edges from the nodes in S_1, let G_1 be the resulting graph. Let S_i be the nodes in G_i with in-degree zero. Thus, a sequence of sets $S_0, S_1, \ldots, S_k$ is derived, assuming that $G_{k+1} = \emptyset$. Then, there are $k + 1$ semesters to finish all courses.

The implementation of the proposed algorithm is as follows. We construct an auxiliary graph $G' = (V \cup \{v_0\}, E \cup \{\langle v_0, u\rangle$ if the in-degree of u is zero $\})$. Clearly G' is a DAG and its construction takes $O(|V| + |E|)$ time. Given the DAG G' and S_i, we construct S_{i+1} from S_i. The algorithm proceeds iteratively until $S_{i+1} = \emptyset$. Specifically, initially $S_0 = \{v_0\}$, $i = 0$. Let i be the current iteration, the set S_{i+1} is constructed as follows.

$S_{i+1} = \emptyset$ initially. For each node $u \in S_i$, the algorithm traverses along the linked list of node u, i.e., for $v \in Adj(u)$, the in-degree of node v is reduced by 1, if its in-degree now becomes zero, v is added to S_{i+1}. When the neighbors of each node in S_i have been examined. $i \leftarrow i + 1$, if $S_{i+1} \neq \emptyset$, the algorithm starts its next iteration. Otherwise, it terminates. Assume that there are k disjoint subsets $S_1, S_2, \dots, S_k$, then $\cup_{i=1}^{k} S_i = V$. As each node $u \in V$ belongs to one of these k subsets, the time of constructing these subsets is $\sum_{u \in \cup_{i=0}^{k} S_i} out\_degree_{G'}(u) = O(|V| + |E|)$, where $out\_degree_{G'}(u)$ is the outgoing degree of node u in G'. Thus, the algorithm takes $O(|V|+|E|)$ time, as the constructions of both G' and S_i for all i with $0 \leq i \leq k$ take $O(|V| + |E|)$ time.