Computer Science Tripos Part IA and IB

2019-2020
Exam Question
Cover sheet

Student BGN2023A**…..……**

Paper1**……**

Question number9**……**

How did you answer this question?

Timed	☐	Open Book	☑
Untimed	☑	Closed Book	☐

Questions

List all the questions you have answered for this paper here. 1,4,6,8,9

Computer Science Tripos Honour Code

1. We take it as a principle that maintaining the integrity and fairness of examinations should be regarded as a collaboration between students and the Department.
2. The students undertake that they will not help others in examinations and will not receive any help from others (students or non-students).
3. Students will actively contribute to ensuring that all students adhere to the code.
4. Students will keep to the conditions of the assessment and will accurately report those conditions when asked.
5. The Department will not make any attempt at remote invigilation of online examinations.

I **undertake to respect the Computer Science Tripos honour code**

Tick the box to confirm ☑

9a) Support that `dfs_recurse()` terminates, and has visited a vertex u, but there is a vertex v which is reachable from u that has not been visited.

Since v is reachable from u, this means that there exists a path $u \to v_1 \to \ldots \to v$.

In the `dfs_recurse` algorithm, when a vertex is visited, all of its unvisited neighbours are then visited before the algorithm terminates. This means that upon termination, all neighbours of a visited vertex have also been visited.

Since u has been visited by the algorithm, this means that v_1 has been visited, and so too v_2, and so on, until v is reached. So this is a contradiction, which means that all vertices reachable from a visited vertex have also been visited.

b)

```
def findOrigin(g):
    for v in g.vertices:
        v.visited == False
    toVisit = Stack()
    nUnvisited = len(g.vertices)

    for s in g.vertices:
        if s.visited = False:
            toVisit.push(s)
            while not toVisit.isEmpty():
                v = toVisit.pop()
                v.visited = True
                nUnvisited -= 1
                for w in v.neighbours:
                    if not w.visited:
                        toVisit.push(w)
            if nUnvisited == 0:
                return s
```

c) The algorithm performs depth-first searches from unvisited vertices, and returns only once every vertex has been visited.

Suppose the algorithm returned a vertex which was not an origin. This means that every node has been visited by some DFS at some point, due to the return criteria. But since we know there is an origin in the graph, this means that an origin was visited at some point during DFS on the returned node (if it was reached in an earlier search, then the function would have returned earlier).

If an origin was reachable by the returned node, then by extension all nodes are reachable from the returned node via that origin, which means that the returned node must have been an origin as well. This forms a contradiction, so the node that was returned must have been an origin.

d)

```
def allOrigins(g):

```