Week 11 Studio Sheet

(To be completed during the Week 11 studio class)

Objectives: The tutorials, in general, give practice in problem solving, in analysis of algorithms and data structures, and in mathematics and logic useful in the above.

Instructions to the class: Aim to attempt these questions before the tutorial! It will probably not be possible to cover all questions unless the class has prepared them in advance. There are marks allocated towards active participation during the class. You **must** attempt the problems under **Assessed Preparation** section **before** your tutorial class and give your worked out solutions to your tutor at the start of the class – this is a hurdle and failing to attempt these problems before your tutorial will result in 0 mark for that class even if you actively participate in the class.

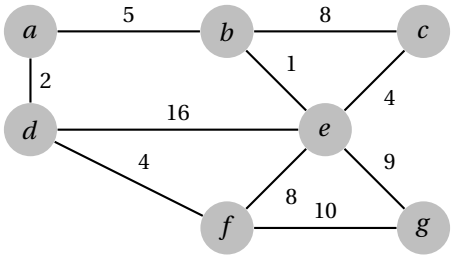
Instructions to Tutors:

1. The purpose of the tutorials is not to solve the practical exercises!
2. The purpose is to check answers, and to discuss particular sticking points, not to simply make answers available.

Supplementary problems: The supplementary problems provide additional practice for you to complete after your tutorial class, or as pre-exam revision. Problems that are marked as **(Advanced)** difficulty are beyond the difficulty that you would be expected to complete in the exam, but are nonetheless useful practice problems as they will teach you skills and concepts that you can apply to other problems.

Assessed Preparation

Problem 1. Show the steps taken by Prim's and Kruskal's algorithms for computing a minimum spanning tree of the following graph. Use vertex a as the root vertex for Prim's algorithm. Make sure that you indicate the order in which edges are selected, not just the final answer.



Problem 2. Consider the following state of a union-find data structure, using the union by size heuristic, as described in lectures.

ID	0	1	2	3	4	5	6	7	8
Parent	-1	2	-3	4	7	-1	7	-4	2

Determine the state of the array after each the following operations are executed (in order). If two trees are the same size, assume the first arguement to `union()` will be chosen as the root:

1. `union(3, 6)`
2. `union(0, 5)`
3. `union(5, 3)`

Studio Problems

Problem 3. In the union-find data structure, when performing a union operation, we always append the set with fewer elements to the set with more elements. This heuristic is called union by size heuristic. Prove that in the worst case we can perform all union operations in $O(V \log(V))$ time when using union by size.

Problem 4. Consider a variant of the shortest path problem where instead of finding paths that minimise the total weight of all edges, we instead wish to minimise the weight of the largest edge appearing on the path. Let's refer to such a path as a *bottleneck path*.

(a) Give an example of a graph in which a shortest path between some pair of vertices is not a bottleneck path and vice versa

(b) Prove that all of the paths in a minimum spanning tree M of a graph G are bottleneck paths in G

(c) Prove that not all bottleneck paths of G are paths in a minimum spanning tree M of G

Problem 5. Consider the following supposed invariant for Kruskal's algorithm: At each iteration, each connected component in the current spanning forest is a minimum spanning tree of the vertices in that component.

(a) Prove or disprove whether Kruskal's algorithm maintains this invariant.

(b) Can this invariant be used to prove that Kruskal's algorithm is correct?

Problem 6. Consider the following algorithm quite similar to Kruskal's algorithm for producing a minimum spanning tree

```
1:  function MINIMUM_SPANNING_TREE(G = (V, E))
2:      sort(E, descending, key((u,v)) = w(u, v))    // Sort edges in order from heaviest to lightest
3:      T = E
4:      for each edge e in nonincreasing order do
5:          if T - {e} is connected then
6:              T = T - {e}
7:          end if
8:      end for
9:      return T
10: end function
```

Instead of adding edges in order of weight (lightest first) and keeping the graph acyclic, we remove edges in order of weight (heaviest first) while keeping the graph from becoming disconnected.

(a) Identify a useful invariant maintained by this algorithm

(b) Prove that this invariant is maintained by the algorithm.

(c) Deduce that this algorithm correctly produces a minimum spanning tree

Problem 7. Can Prim's and Kruskal's algorithm be applied on a graph with negative weights to compute a minimum spanning tree? Why or why not?

Supplementary Problems

Problem 8. Implement Kruskal's algorithm.