# Example sheet week 5

Graphs. Path finding.
Algorithms—DJW—2016/2017

Questions labelled FS are from Dr Stajano's list of exercises. Questions labelled CLRS are from *Introduction to Algorithms, 3rd ed.* by Cormen, Leiserson, Rivest and Stein. Questions labelled $*$ involve more coding or more thinking and are best tackled after the other questions, if you have time.

**Question 1 (FS52).** First, some definitions.

- A *DAG* is a directed graph without cycles (a Directed Acyclic Graph)
- A graph is *reflexive* if the edge $(v,v)$ is present for every vertex $v$, and *anti-reflexive* if $(v,v)$ is absent for every vertex $v$.

Draw an example of each of the following, or explain why no example exists:

(i) An anti-reflexive directed graph with 5 vertices and 7 edges
(ii) A reflexive directed graph with 5 vertices and 12 edges
(iii) A DAG with 8 vertices and 10 edges
(iv) An undirected tree with 8 vertices and 10 edges
(v) A graph without cycles that is *not* a tree

**Question 2*.** You've learned what $O(n)$ means when there's a single variable $n$ that increases. How would you define $O(E+V\log V)$?

**Question 3.** Do the algorithms `dfs_stack` and `dfs_recurse` (as given in the handout) always visit vertices in the same order? If not, modify `dfs_stack` so that they do. Give pseudocode.

**Question 4.** Give pseudocode for a modified version of `dfs_recurse_path`, which does not visit any further vertices once it has reached the destination vertex $t$.

**Question 5 (CLRS-22-4)*.** Consider a directed graph in which every vertex $v$ is labelled with a real number $x_v$. For each vertex $v$, define $m_v$ to be the minimum value of $x_u$ among all vertices $u$ such that either $u = v$ or $u$ is reachable via some path from $v$. Give an $O(E+V\log V)$-time algorithm that computes $m_v$ for all vertices $v$. *[Correction: the printed handout erroneously asked for $O(V+E)$.]*

**Question 6*.** Here is a modified version of `dfs_stack` that keeps track of which vertices are currently being investigated. Prove that the assertion on line 6 always succeeds when the graph is undirected. Give an example of a directed graph where the assertion fails.

```
1  def visit(v):
2    v.visited = True
3    v.active = True
4    for w in v.neighbours:
5      if w.visited:
6        assert w.active == True
7      else:
8        visit(w)
9    v.active = False
```

*[Correction: the assertion should read* `assert (w.active == True) or (w is not v's first neighbour)`*.]*

**Question 7 (FS53).** Give an example DAG with 9 vertices and 9 edges. Pick some vertex that has one or more edges coming in, and run through `dfs_recurse_all` starting from this vertex. Mark

each vertex with two numbers as you proceed: the discovery time (the time we printed "visiting") and the exit time (the time that `visit()` returns). Then draw a linearized DAG by arranging the vertices on a line in reverse order of their finishing time, and reproducing the appropriate arrows between them. Do all the arrows go forwards?

**Question 8*.** Sometimes we want to impose a total order on a collection of objects, given a set of pairwise comparisons that can be thought of as a "DAG with noise". For example, let vertices represent movies, and write $v_1 \to v_2$ to mean "The user has said she prefers $v_1$ to $v_2$." A user is likely to give answers that are by and large consistent, but with some exceptions. Discuss what properties you would like in an "approximate total order", and how you might go about finding it.

**Question 9*.** Give pseudocode for an algorithm that takes as input an arbitrary directed graph $g$, and returns a boolean indicating whether or not $g$ is a DAG.

**Question 10 (FS54)*.** Write out a formal proof of the correctness of the `toposort` algorithm, filling out all the details that are skipped over in the handout. Pay particular attention to step 3, where it is claimed "the recursion stack gives us a path from $v_2$ to $v_1$".

**Question 11.** The `bfs` algorithm given in the handout will visit all vertices and print out a log message at each. Modify the code so that, given a destination vertex $t$, it produces a shortest path from $s$ to $t$. *(Hint. See* `dfs_recurse_path`*.)* Modify it further to produce *all* shortest paths from $s$ to $t$.

**Question 12*.** Consider a graph without edge weights, and write $d(u,v)$ for the length of the shortest path from $u$ to $v$. The *diameter* of the graph is defined to be

$$\max_{u,v \in V} d(u,v).$$

Give an efficient algorithm to compute the diameter of an undirected tree, and analyse its running time.

**Question 13 (FS57).** In a directed graph with edge weights, give a formal proof of the triangle inequality

$$d(u,v) \leq d(u,w) + c(w \to v) \quad \text{for all vertices } u, v, w \text{ with } w \to v$$

where $d(u,v)$ is the minimum weight of all paths from $u$ to $v$ (or $\infty$ if there are no such paths) and $c(w \to v)$ is the weight of edge $w \to v$. Make sure your proof covers the cases where no path exists.

**Question 14*.** Consider a directed graph with edge weights that are $\geq 0$, and suppose we want to find a shortest path from some start vertex $s$ to a destination $t$. Let $h(v)$ be the distance from $s$ to $v$ for any vertex $v$. Write down a recursion from $h(v)$, and give pseudocode for a dynamic programming algorithm that uses recursion and memoization to compute $h(s)$. Comment on the relationship between this and Dijkstra's algorithm.

**Question 15*.** Describe a class hierarchy that you might use for a library of graph algorithms, including depth-first search, breadth-first search, topological sort, and shortest-path computation. Bear in mind

- Each algorithm has its own variables it wants to store with a vertex.
- A programmer using your library may want to construct a graph, then apply to it a variety of different algorithms.
- Sometimes an algorithm only needs to inspect a few nodes to return its answer.