The “Classroom Problems” will be discussed and solved ex tempore at the tutorials in the same week as this problem set is handed out. These are intended to help you in getting started with the material, and will not yield any course credit.

The “Homework Problems” you should solve at home, and attend one of the tutorial sessions in the following week, prepared to present your solutions at class if requested. You earn one tutorial credit point (= 0.33 course points, for a maximum total of 6) for each of these problems that you have solved, and marked as solved at the tutorial session.

The “Demonstration Problems” illustrate further interesting aspects of the material. These will be discussed at the tutorials in the same week as the Homework Problems (time permitting) and printed solutions will be provided. You are encouraged to work on these, but no course credit is given for them.

The “Challenge Problems” are extra problems related to the material, which may be exceptionally difficult and/or require tools beyond the course curriculum.

**Note:** There are no lectures or tutorials on the course in Week 41, i.e. 9–13 October. Thus, the Classroom Problems from this problem set are discussed at the tutorials in Week 40 and the Homework Problems in Week 42.

**Classroom problems:**

1. [Dasgupta et al., Ex. 3.2] Perform a depth-first search on the following graph; whenever there’s a choice of vertices, pick the one that is alphabetically first. Classify each edge as a tree edge, forward edge, back edge, or cross edge, and give the pre and post number of each vertex.

   ![Graph Diagram](attachment:image.png)

2. Prove the following Claim from Lecture 5: In the DFS search of a directed graph, for any vertices \( u \) and \( v \), the intervals \( (\text{pre}[u], \text{post}[u]) \) and \( (\text{pre}[v], \text{post}[v]) \) are either disjoint or one is contained within the other.

3. [Dasgupta et al., Ex. 3.8] Pouring water. We have three containers whose sizes are 10 pints, 7 pints, and 4 pints, respectively. The 7-pint and 4-pint containers start out full of water, but the 10-pint container is initially empty. We are allowed one type of operation: pouring the contents of one container into another, stopping only when the source container is empty or the destination container is full. We want to know if there is a sequence of pourings that leaves exactly 2 pints in the 7- or 4-pint container.

   (a) Model this as a graph problem: give a precise definition of the graph involved and state the specific question about this graph that needs to be answered.

   (b) What algorithm should be applied to solve the problem?
Homework problems:

4. [Dasgupta et al., Ex. 4.1] Suppose Dijkstra’s algorithm is run on the following graph, starting at vertex A.

(a) Draw a table showing the intermediate distance values for all the vertices at each iteration of the algorithm.
(b) Show the final shortest-path tree.

Whenever there is a choice of vertices to explore, always pick the one that is alphabetically first.

Hint: ..... 

5. Vertex r is a root of digraph (directed graph) \( G = (V, E) \), if there exists a path from \( r \) to \( v \) for each \( v \in V \). (Note that a digraph may have several roots, or none at all.)

(a) Design an algorithm that finds a root in a digraph \( G \), represented as an adjacency list, in time \( O(|V| + |E|) \), or determines that none exist. (Hint: A constant number of DFS passes over \( G \) suffices.)
(b) Design an efficient algorithm for determining all roots of a given digraph \( G \), and analyse its worst-case running time.

Hint: Pick an arbitrary vertex \( v_1 \). Let tree \( T_1 = \text{DFS}(v_1) \). If every vertex is visited, what does it mean? If not, remove all visited vertices, pick an arbitrary unvisited vertex \( v_2 \) and run \( \text{DFS}(v_2) \) to get \( T_2 \). Can any vertex in \( T_1 \) be a root of \( G \)? Why? If every vertex is visited now, is it necessary that \( v_2 \) is a root? If not, what should we do next? Assume that we know \( r \) is a root, then every vertex \( u \) that has path to \( r \) is also a root. How can we find these vertices?

6. [Dasgupta et al., Ex. 4.5] Often there are multiple shortest paths between two vertices of a graph. Give a linear-time algorithm for the following task.

Input: Undirected graph \( G = (V, E) \) with unit edge lengths; vertices \( u, v \in V \).
Output: The number of different shortest paths from \( u \) to \( v \).

Hint: BFS gives us layers of vertices, where the first layer is only a vertex \( u \) and the last layer is only a vertex \( v \). There is no forward edge that connects non-adjacent layer (why?). We now keep track the number of shortest paths from the first layer to the last layer. The number of shortest paths from \( u \) to layer \( i \) can be calculated from the number of shortest paths from layer \( i - 1 \) (how?).

Demonstration problems:

7. [Dasgupta et al., Ex. 3.7] A bipartite graph is a graph \( G = (V, E) \) whose vertices can be partitioned into two sets \( (V = V_1 \cup V_2 \text{ and } V_1 \cap V_2 = \emptyset) \) such that there are no edges between vertices in the same set (for instance, if \( u, v \in V_1 \), then there is no edge between \( u \) and \( v \)).
(a) Give a linear-time algorithm to determine whether an undirected graph is bipartite.

(b) There are many other ways to formulate this property. For instance, an undirected graph is
bipartite if and only if it can be coloured with just two colours.\(^1\)

Prove the following formulation: an undirected graph is bipartite if and only if it contains
no cycles of odd length.

(c) At most how many colours are needed to colour an undirected graph with exactly one
odd-length cycle?

Solution:

(a) Pick arbitrary vertex \( u \). For each vertex, color 1 if the distance from \( u \) is odd, otherwise
color 2. This can be easily done by BFS. After finishing the coloring phase, spend linear
time to check if each edge is valid, i.e., two endpoints have different colors.

(b) (If an undirected graph is bipartite, then it contains no cycles of odd length) Prove by
contrapositive. It is easy to see that any odd cycles need at least 3 colors. (If an undirected
graph contains no cycles of odd length, then it is bipartite) Pick an arbitrary vertex \( u \) and
consider a tree formed by \( \text{BFS}(u) \): odd layer with red and even layer with blue. Notice
that there cannot be cross edge that connects the same color because it induces odd cycle.
Hence, this is a valid 2-coloring.

(c) If we remove one edge in this cycle, this is 2-colorable graph. We need the third color once
we add this edge back. So, we need at most 3 colors.

\(^1\) A graph can be *coloured with \( k \) colours*, if it is possible to assign one of a given set of \( k \) colours to each of the vertices,
so that no two neighbouring vertices get the same colour.
Challenge problem:

8. A directed acyclic graph $G$ is **confluent**, if any two vertices $v_1, v_2$ in $G$ which have a common ancestor $u$ also have a common successor $w$. (I.e. if there are paths from some $u$ to both $v_1$ and $v_2$, then there are also paths from $v_1$ and $v_2$ to some $w$. Vertices $u, v_1, v_2$ and $w$ do not need to be distinct, thus e.g. a single “line” of vertices is trivially confluent.) Design a linear-time algorithm that determines whether a given DAG $G$ is confluent.