Tutorial 1 - Graph Representation, Traversal and MST

COMP9418 – Advanced Topics in Statistical Machine Learning

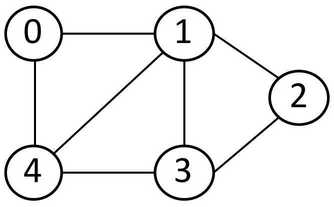
Lecturer: Gustavo Batista

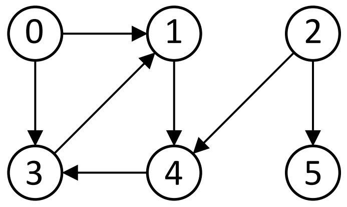
Last revision: Sunday 13th September, 2020 at 20:14

In this tutorial, we will review some basic concepts of graph representation, traversal and minimum spanning trees (MST). In particular, we will look at some relevant problems we can solve with graph traversal and efficient implementations of MST algorithms. This material will not be covered in the course, but it is essential to understand and implement several algorithms of Probabilistic Graphical Models.

Question 1 - Graph representation

There are two frequently used representations for graphs: adjacency matrix and list. Both representations can be used for directed and undirected graphs, like the ones in the next figure.





Represent both graphs using the matrix and list representations. You do not need to write any code here, use a graphical representation such as boxes liked by arrows. Which of these two representations is typically prefered to implement operations over graphs? Why?

Question 2 - Graph traversal

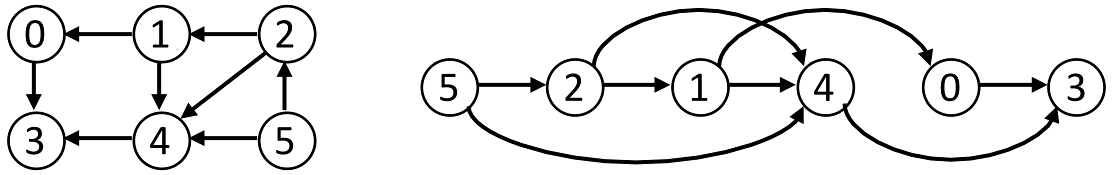
Graph traversal is the process of visiting each vertex in a graph. There are two main variations, depth-first (DFS) and breadth-first search (BFS). Write the algorithms for each of these operations. Use a simplifying notation that allows you to write a single algorithm for both graph representations. For problems that can be solved by both algorithms, is there one search strategy (depth or breadth) that is preferred? Why?

Question 3 - Cycle detection in directed graphs

In the first part of this course, we will use direct acyclic graphs (DAGs) to represent statistical independencies between random variables. Therefore, it may be useful to develop an algorithm to detect cycles to verify if a directed graph is a DAG. The DFS algorithm can be adapted to detect the existence of cycles in a directed graph. A cycle is a non-empty path that links a vertex to itself. A well-known trick is to colour the vertices as we visit them. Vertices are all initially white and become grey as we first visit them. A grey vertex becomes black when we have visited all descendent nodes of this grey vertex, and we are ready to backtrack the recursion. Adapt your DFS algorithm to detect cycles in directed graphs using this colouring scheme. Is a cycle detected when we reach a white, grey or black node?

Question 4 - Topological sorting on DAGs

Many algorithms in this course require iterating through nodes, where all parents need to be processed before their children. For these algorithms, we use an ordering known as *topological ordering*. Topological sort or topological ordering of a DAG is a linear ordering of the vertices. For every directed edge from vertex u to vertex v, u comes before v in the order. The next figure shows a topological ordering for a DAG.



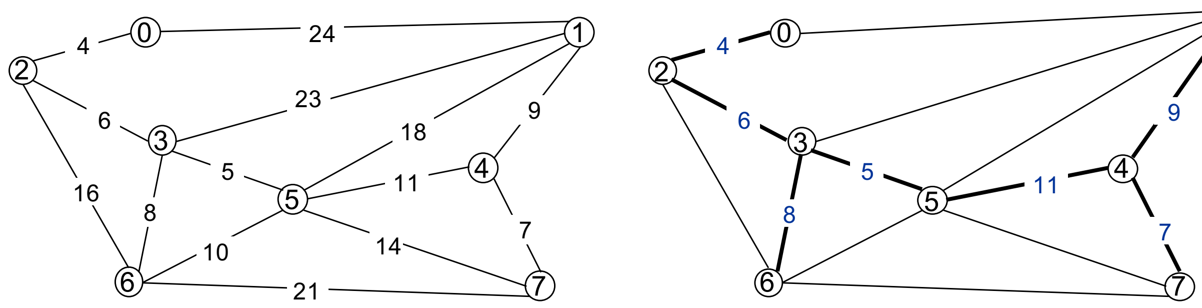
Adapt the DFS algorithm to produce a topological order of a DAG. Do you need any additional data structure to report the results in proper order?

Question 5 - Strongly connected directed graphs

A directed graph is called strongly connected if there is a path in each direction between each pair of vertices. We can easily (but inefficiently) test if a directed graph is strongly connected by running n DFS searches from each vertex and confirming that we can reach every node in each search. However, according to Kosaraju's algorithm, two searches suffice, the first one in the graph G and the second one in its transpose G^T. Explain the intuition behind this algorithm.

Question 6 - Minimal spanning trees

A common operation over undirected graphs is known as a minimum spanning tree (MST). Given an undirected graph G, MST consists of finding a subset of the edges that forms a tree and includes every vertex of G. The sum of the edges' weights in the tree should be minimal. The next figure shows an example of a graph (left) and its minimum spanning tree (right).



Notice that for a graph with n vertices, the spanning tree must have exactly $n-1$ edges. Adding one more edge would create a cycle, and removing one edge would disconnect the tree creating a forest.

There are several algorithms to compute an MST of a graph, including the well-known Prim and Kruskal algorithms. These algorithms can be trivially adapted to calculate the *maximum* spanning tree by selecting the edges with the highest cost instead of the smallest cost.

Write the pseudo-code of an MST algorithm such as PRIM. What is the time complexity of your algorithm? Which data structures are necessary to achieve such complexity?