Section 08: Graph traversal, shortest path algorithms

1. Limitations and properties of shortest path algorithms

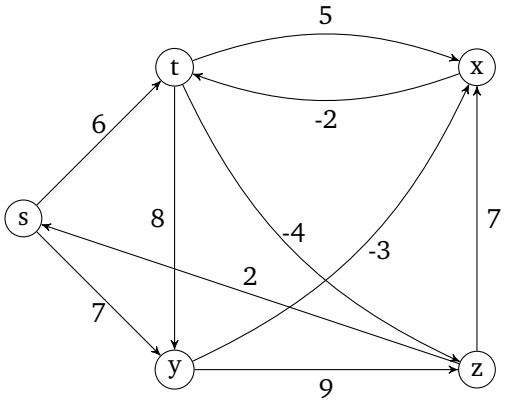
(a) Draw an example of a directed graph where (a) there exists a path between two vertices s and t but (b) there *is no shortest path* between s and t.

(b) Draw an example of a directed graph where there does exist a *shortest path* between two vertices s and t but if we try running Dijkstra's algorithm on s, it fails to find that shortest path and returns an incorrect result.

(c) Given some arbitrary graph, how would you determine if it contained a cycle?

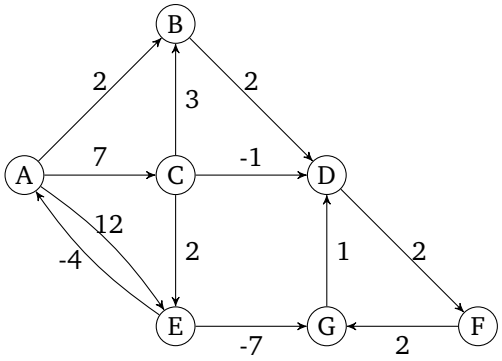
2. Simulating Dijkstra's

(a) Consider the following graph:



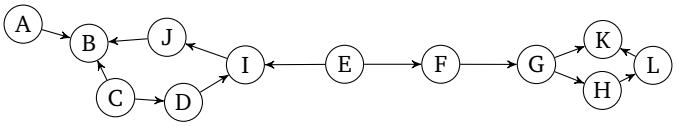
Suppose we run Dijkstra's algorithm on this graph starting with vertex s. (Note: we will very likely get meaningless results due to the negative edges, but this is still a good way to practice running the algorithm.) What are the final costs of each vertex and the shortest paths from s to each vertex?

(b) Here is another graph. What are the final costs and shortest paths if we run Dijkstra's starting on node A?



3. Practicing topological sort

Find a topological sort of the following graph:



4. Graph representations

(a) Suppose we have a directed graph represented in *adjacency list* form. Design an algorithm that produces a new graph (also in adjacency list form) where each edge is reversed.

(b) Implement an algorithm that does the same thing, except that we now accept and return a graph in *adjacency matrix* form.

5. Designing algorithms: Applying algorithms

(a) Explain how you would implement DFS recursively, rather then iteratively. How much memory does the recursive algorithm use vs the iterative one?

(b) Suppose you want to model how a tweet gets re-tweeted by followers on Twitter. You have data on who the users of Twitter are and all the followers of each user. Given a source, a tweet can be re-tweeted by any follower of that source.

Suppose a particular user makes a tweet. Design an algorithm that designs which users could have seen this tweet, assuming the tweet was re-tweeted at most k times.

(c) Suppose we have graph where the edges are unweighted, but the vertices have numerical weights. Explain how you would modify one of the algorithms we've studied in class to find the shortest path between two vertices.

(d) **Challenge:** Suppose you are trying to find the most strenuous hiking trail between two points: the path with the most elevation gain or loss per mile. We model this problem as a connected undirected graph $G = (V, E)$. Each vertex is associated with some height $h(v)$ and each edge is associated with some positive distance $d(e)$.

Let a and b be our starting and ending points. We assume $h(a) = h(b) = 0$.

Now, we define an "up-down path" as a path where there exists some vertex t such that all vertices we visit from a to t have increasing heights, and all vertices we visit from t to b have decreasing heights. Metaphorically, an "up-down path" represents a hill, where t is the very top of the hill.

Next, we define the "difficulty" of the up-down path has:

$$\text{difficulty} = \frac{\text{h(t)}}{\text{sum of the lengths of the edges in path}}$$

That is, the difficulty is the height of the hill divided by the total horizontal length we end up walking.

Design an algorithm that finds an up-down walk with the maximum possible difficulty between two nodes a and b. If there does not exist an up-down path, throw an exception.

6. Designing algorithms: Pathfinding in mazes

(a) Suppose we are trying to design a maze within a 2d top-down video-game. The world is represented as a grid, where each tile is either an impassable wall, an open space a player can pass through, or a *wormhole*. On each turn, the player may move one space on the grid to any adjacent open tile. If the player is standing on a wormhole, they can instead use their turn to teleport themselves to the other end of the wormhole, which is located somewhere else on the map.

Now, suppose the there are several coins scattered throughout the map. Your goal is to design an algorithm that finds a path between the player and some coin in the fewest number of turns possible.

Describe how you would represent this scenario as a graph (what are the vertices and edges? Is this a weighted or unwighted graph? Directed or undirected?). Then, describe how you would implement an algorithm to complete this task.

(b) **Challenge:** Suppose the map now stuffed with a huge number of players. We now want an algorithm that finds the shortest path between every single player and the closest coin.

A naive way of doing this would be to run the same algorithm from the previous part on every single player. However, this would be prohibitively expensive. Design a more efficient algorithm that does the same thing.

When designing your algorithm, you may assume that the map is relatively small/that there are only a few coins, relative to the number of players.

(c) **Challenge:** Now, suppose our map contains just a single player again. The player now wants to collect *all* the coins in the fewest number of turns possible. Design an algorithm to do this efficiently.

7. Designing algorithms: Planning course prerequisites

(a) Suppose you just got into the department of your choice and are trying to plan out the courses you need to take in order to graduate. You have a list of all available courses, their prerequisites, and whether or not that course is optional.

Now suppose you want to all of the available classes. Design an algorithm that prints out one possible order in which you can take these classes. What is the worst-case runtime of your algorithm? Note: if a class has a prerequisite, make sure you take the prerequisites first.

For the sake of simplicity, you may assume you will take only one class per quarter.

(b) What if you only want to take some of the classes? Modify your algorithm so that it accepts a list of all classes you want to take and prints out a schedule where you take only those courses (or prerequisites to those courses). What is the worst-case runtime of your algorithm?

(c) **Challenge:** What if you can take up to k classes per quarter? Adjust your algorithm so it prints out a schedule that lets you graduate as soon as possible. What is the runtime of your algorithm?