Definitions

Definition 1 (Greedy algorithm). A greedy algorithm is one which always makes the choice that looks best at the moment—the *locally optimal* choice—in order to find the best *globally optimal* solution. Greedy algorithms do not always yield optimal solutions, but for many problems they do.

Definition 2 (Shortest path). A shortest path from vertex \( s \) to vertex \( t \) is a directed path from \( s \) to \( t \) with the property that no other such path has a lower total edge weight.

Dijkstra’s Algorithm

Dijkstra’s algorithm finds the shortest path between two given vertices in a weighted graph, assuming that the graph’s edge weights are non-negative. The running time of the algorithm is \( O(E \log V + V \log V) \) when the graph is implemented using adjacency lists. With a special transformation (use of Fibonacci heaps) this can be reduced to \( O(E + V \log V) \), which is the fastest version of this algorithm. The pseudo-code for the algorithm is given below.

Pseudocode

\[ \text{Dijkstra}(G, s) \]

1. for each vertex \( v \in V_G \)
2. \( \text{dist}[v] = \infty \)
3. \( \text{parent}[v] = \text{NIL} \)
4. \( \text{dist}[s] = 0 \)
5. \( Q = V_G \)
6. while \( Q \neq \emptyset \)
7. \( u = \text{Extract-Min}(Q) \)
8. for each vertex \( v \in G.\text{Adj}[u] \)
9. if \( \text{dist}[v] > \text{dist}[u] + w(u, v) \)
10. \( \text{dist}[v] = \text{dist}[u] + w(u, v) \)
11. \( \text{parent}[v] = u \)

Edge-Weighted DAGs (Directed Acyclic Graphs)

The algorithm for shortest path on edge weighted DAGs is simpler and faster than Dijkstra’s algorithm. However, instead of considering vertices by priority of their distance estimates, we consider the vertices of the DAG in a topological order. (Why must a DAG always have a topological order?) Then we just relax each vertex in the topological ordering. Running time: \( O(|V| + |E|) \).
Problem 1. Find the shortest path between vertices E and G in the graph provided

Problem 2. Explain why Dijkstra’s algorithm is a greedy algorithm.

Problem 3. Does Dijkstra’s Algorithm work with negative weights? Why or why not?

Problem 4. True or false: Dijkstra’s algorithm will not terminate if run on a graph with negative edge weights.

Problem 5. True or false: The shortest path algorithm in an edge weighted DAG works even with negative edge weights.

Problem 6. How could you modify Dijkstra’s algorithm to find all shortest paths?

Problem 7. How could you modify Dijkstra’s algorithm to stop once it’s found the shortest path to a particular node?

Problem 8. Explain the running time of Dijkstra’s algorithm.

Problem 9. True or false: If we double the weights of all the edges in a graph, then Dijkstra’s algorithm will produce the same shortest path.

Problem 10. Say we are given a graph G where all edges are positively weighted. Construct graph G’ where for all edges e with weight w(e) and endpoints u and v, we replace e with w(e) edges of weight 1 in G’, such
that the path from $u$ to $v$ in $G'$ consists of $w(e) - 1$ middle nodes.

How could you use this method to find the shortest path between two vertices in $G'$? What problem do you see with this approach?

**Topological Sort**

A few weeks ago we covered an algorithm called topological sort. This is motivated by many problems encountered in the real world. For example, you are running an assembly line where there are a number of tasks required to create a product. Some of the tasks must come before others. You want to maximize the amount of parallel tasks you can complete at once. How can you obtain an ordering of these tasks to make sure the product is assembled properly? The answer is a topological sort!

**Definition 3** (Topological ordering). A topological ordering of a directed acyclic graph $G = (V, E)$ is a linear ordering of $V$ such that whenever $G$ contains a directed edge $(u, v)$, then $u$ appears before $v$ in the ordering.

There are two canonical algorithms for this. It is good for you to understand both of them.

**Using depth-first search (Tarjan’s algorithm)**

- Call DFS and compute finish times for each vertex $v$.
- As each vertex finishes, push each onto a stack.
- Return the stack.

From most recently pushed to the eldest element, the stack contains the nodes in order of decreasing finishing times.

This is equivalent to a reverse postorder traversal.

You should think carefully about the correctness of this algorithm!

**Kahn’s algorithm**

- Maintain a set $S$ of nodes with in-degree 0.
- While $S$ is not empty, remove a node from $S$ and add to the end of ordering.
- Remove all edges going out of that node and update $S$ accordingly.

This is perhaps the more intuitive algorithm based on your understanding of topo sort.

**Problems**

**Problem 1**

Conceptual questions:

1. (True/False) Every DAG has exactly one topological ordering.
2. (True/False) A preorder traversal always produces a topological ordering on a tree.
3. If a graph has a topological ordering, then a depth-first traversal of the same graph will not see any back edges.

**Problem 2**

**Problem** (CLRS 22.4-2). Give a linear-time algorithm that takes as input a directed acyclic graph $G = (V, E)$ and two vertices $s$ and $t$, and returns the number of simple paths from $s$ to $t$ in $G$. You only need count the simple paths, not list them. (An example can be found in the textbook.)