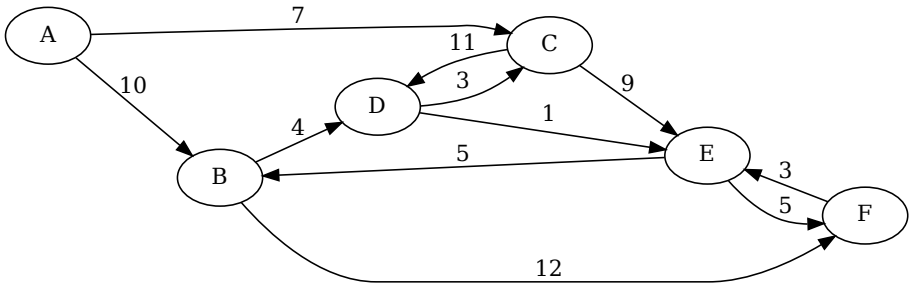
TD 5: SSSP

1 Execute Dijkstra's algorithm

Execute Dijkstra's algorithm on the graph below, starting from vertex A. Show the distances calculated and the shortest-path tree from A.



2 Make Dijkstra Work

Recall that Dijkstra's algorithm may fail if the input graph contains edges with negative weights. We consider several special cases of graphs with negative weights and try to decide if Dijkstra's algorithm may still be correct in those cases. For the questions below, assume that the graph in question does **not** have negative cycles. We are given a digraph G and want to execute Dijkstra's algorithm starting from s.

1. Is Dijkstra's algorithm correct if we are promised that G is a DAG (with possibly negative weights)?
2. Is Dijkstra's algorithm correct if G contains only one arc of negative weight?
3. Is Dijkstra's algorithm correct if all the negative arcs of G are incident on s?

3 Make Bellman-Ford work (less)

As we discussed in class, one possible way to try to speed up the Bellman-Ford algorithm is the following: in each iteration of the main loop, check if any of the distances are modified. If nothing changes, then there is no point in repeating the outer loop, so we can immediately stop. For this exercise we consider this version of the BF algorithm.

1. Give an example where this version of the BF algorithm computes the correct answer in time $O(m)$.
2. Nevertheless, show that this improvement does not affect the worst-case performance of BF by giving an example where even this version takes time $\Omega(mn)$.

Observe that the same graph can be used as an answer to both questions by using a different ordering of the vertices/arcs and you don't even need negative weights. Is the following statement true?

- True or False: For every edge-weighted digraph G there exists an ordering of its **arcs** such that the BF algorithm executes in time $O(m)$.

4 Shortest Simple Paths and Negative Cycles

One annoying aspect of our discussions of graphs with negative weights has been that negative cycles make it complicated to define what exactly is a shortest path (a path may repeat a negative cycle many times, arbitrarily reducing its cost). One way to work around this problem could be to simply forbid this. For this exercise we define the shortest path distance as the weight of the shortest **simple** path, that is, the shortest path that does not repeat any vertex. This is always well-defined, even if the input graph has negative cycles.

Unfortunately, defining the problem in this way makes it impossible to solve! Prove the following:

- If there is a polynomial-time algorithm that takes as input an edge-weighted digraph G and two vertices s, t and computes the simple path from s to t that has minimum weight, then there exists a polynomial time algorithm for the HAMILTONIAN PATH problem.

The HAMILTONIAN PATH problem is the following: given an undirected unweighted graph G, decide if there is a path that visits each vertex of G exactly once. HAMILTONIAN PATH is one of the most famous NP-complete problem and you can take it as a given for this exercise that it admits no polynomial-time algorithm.

5 Constraint Systems

We are given a set of n integer variables $x_1, \ldots, x_n$ and a system of m inequalities, all of which have the form $x_i - x_j \leq b_k$, for some integer b. Our goal is to determine a feasible solution of the system of inequalities, or correctly decide that none exists.

As an example, consider the following system:

$$\begin{aligned} x_1 - x_2 &\leq 5 \\ x_1 - x_3 &\leq 4 \\ x_2 - x_3 &\leq -3 \\ x_3 - x_1 &\leq 3 \end{aligned}$$

1. Show that the system above has an integer solution.
2. Show that the system no longer has a solution if we replace the last inequality by $x_3 - x_1 \leq -3$.
3. Show that if such a system has a feasible solution $(v_1, v_2, \ldots, v_n)$, then for all offset values d, the solution $(v_1 + d, v_2 + d, \ldots, v_n + d)$ is also valid.
4. Give an efficient algorithm that produces a solution of a given system or decides that none exists.
5. Give a modification of your algorithm which handles the more general problem where some constraints are allowed to involve only one variable, that is, have the form $x_i \leq b_k$ or $-x_i \leq b_k$.