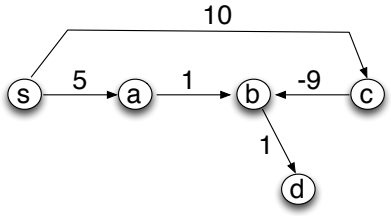
Algorithms Lab 10

In lab

1. Consider the directed graph below and assume you want to compute SSSP(s).



   (a) Run Bellman-Ford algorithm.

   (b) Run Dijkstra's algorithm and reflect on why it does not work.

2. Suppose you had a bunch of undirected edges (given as adjacency lists) and you want to figure out if they form a tree. How would you do it, and how fast?

3. Prove that the following claim is false by showing a counterexample:

   Claim: Let $G = (V, E)$ be a directed graph with negative-weight edges, but no negative-weight cycles. Let $w, w < 0$, be the smallest weight in G. Then one can compute SSSP in the following way: transform G into a graph with all positive weights by adding $-w$ to all edges, run Dijkstra, and subtract from each shortest path the corresponding number of edges times w. Thus, SSSP can be solved by Dijkstra's algorithm even on graph with negative weights.

4. Suppose the degree requirements for a computer science major are organized as a *DAG* (directed acyclic graph), where vertices are required courses and an edge (x, y) means course x must be completed prior to beginning course y. We make the following assumptions:

   - All classes are required to major.
   - All prerequisites must be obeyed.
   - There is a course, CPS1, that must be taken before any other course.
   - Every course is offered every semester, and there are enough slots for everyone who wants to register (unlike our department).
   - These classes involve no work, so you can take as many as you want in any one semester (again, unlike our department).

Describe an efficient algorithm to compute the minimum number of semesters required to complete the major. Analyze its running time.

5. Consider a directed graph G, and assume that instead of shortest paths we want to compute *longest paths*. Longest paths are defined in the natural way, ie the longest path from u to v is the path of largest overall weight among all possible paths from u to v. Note that if the graph contains a positive cycle, then longest paths are not well defined (for the same reason that shortest paths are not well defined when the graph has a negative cycle). So what we want is the *longest simple path*, (a path is called *simple* if it contains no vertex more than once).

   Show that the the longest simple path problem does not have optimal substructure by coming up with a small graph that provides a counterexample.

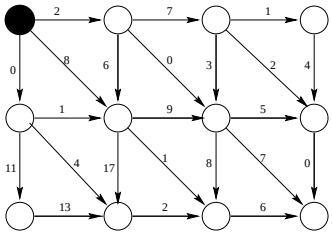
   Note: Finding longest (simple) paths is a classical *hard* problem, and it is known to be NPC (NP-complete).

Homework

1. (CLRS 24.3-6) We are given a directed graph $G = (V, E)$ on which each edge (u, v) has an associated value $r(u, v)$, which is a real number in the range $[0, 1]$ that represents the reliability of a communication channel from vertex u to vertex v. We interpret $r(u, v)$ as the probability tht the channel from u to v will not fail, and we assume that these probabilities are independent. Give an efficient algorithm to find the most reliable path between two given vertices.

2. (GT C-7.7) Suppose you are given a diagram of a telephone network, which is a graph G whose vertices represent switching centers, and whose edges represent communication links between the two centers. The edges are marked by their bandwidth. The bandwidth of a path is the *minimum* bandwidth along the path. Give an algorithm that, given two switching centers a and b, will output a maximum bandwidth path between a and b.

3. Consider a directed weighted graph with non-negative weights and V vertices arranged on a rectangular grid. Each vertex has an edge to its southern, eastern and southeastern neighbours (if existing). The northwest-most vertex is called the root. The figure below shows an example graph with $V = 12$ vertices and the root drawn in black:



Assume that the graph is represented such that each vertex can access **all** its neighbours in constant time.

(a) How long would it take Dijkstra's algorithm to find the length of the shortest path from the root to all other vertices? (Your bound should be function of V).

(b) Describe an algorithm that finds the length of the shortest paths from the root to all other vertices in $O(V)$ time.

(c) Describe an efficient algorithm for solving the all-pair-shortest-paths problem on the graph (it is enough to find the length of each shortest path).