

Practice Problems 3

Problem 1. Dijkstra’s algorithm requires nonnegative edge weights. Throughout this problem, use the version of the algorithm given in class.

- (a) The digraph in Fig. 1 has a single negative-weight edge and no negative-cost cycle. Simulate Dijkstra’s algorithm from s : show the order in which vertices are extracted from the priority queue, and give the d -values of all four vertices after each extraction and the relaxation of its outgoing edges. Which vertices end up with an incorrect distance?

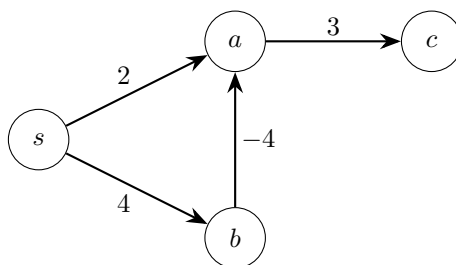


Figure 1: A digraph with one negative-weight edge and no negative-cost cycle.

- (b) Suppose a digraph has exactly one negative-weight edge (u, v) and no negative-cost cycle. Part (a) shows that a single run of Dijkstra’s algorithm may fail. Can the shortest-path distances from s still be computed using Dijkstra’s algorithm as a *subroutine*? You may delete edges, run Dijkstra’s algorithm several times, and combine the results, but you may not modify the algorithm itself. Either give such an algorithm and justify its correctness, or argue that none exists.

Problem 2. Let $G = (V, E)$ be a digraph with n vertices, m edges, and possibly negative edge weights, and let $s \in V$. Recall that Bellman-Ford sets $d[s] = 0$ and $d[v] = \infty$ for $v \neq s$, then performs $n - 1$ rounds, each relaxing every edge once; relaxing (u, v) sets $d[v] \leftarrow \min(d[v], d[u] + w_{uv})$. In class we assumed that G has no negative-cost cycle. Consider the following test for detecting one.

DETECT(G, s): Run Bellman-Ford from s for $n - 1$ rounds. Then make one more pass over all edges, and report “negative-cost cycle” if and only if some edge (u, v) can still be relaxed, i.e., $d[u] + w_{uv} < d[v]$.

- (a) Simulate DETECT on the digraph of Fig. 2 with source s , relaxing the edges within each round in a fixed order that you state. Give the d -values of all five vertices after each round, then state what DETECT reports. Is the report correct?
- (b) Now assume every vertex is reachable from s . Prove the following; together they show that DETECT is correct under this assumption.
- (i) After $n - 1$ rounds, every d -value is finite.

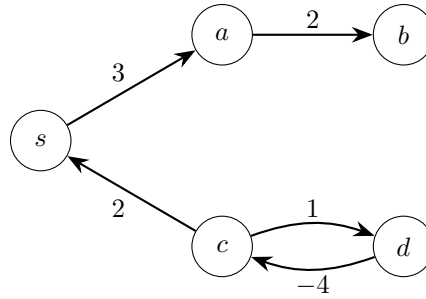


Figure 2: A digraph containing a negative-cost cycle.

- (ii) If G has no negative-cost cycle, then after $n - 1$ rounds no edge can be relaxed.
- (iii) If all d -values are finite and no edge can be relaxed, then G has no negative-cost cycle. (Hint: sum the inequality $d[v] \leq d[u] + w_{uv}$ over the edges of a cycle.)
- (c) Modify the test so that a *single* run of Bellman-Ford detects a negative-cost cycle anywhere in G , reachable from s or not. (Hint: modify the digraph before running Bellman-Ford.) Briefly justify correctness.

Problem 3. Let $G = (V, E)$ be a digraph with n vertices, m edges, and nonnegative edge weights w_{uv} , given as an adjacency list, and let $s \in V$ be a source from which every vertex is reachable. Let $\delta(s, v)$ denote the shortest-path distance from s to v .

You are given a spanning tree $T \subseteq E$ rooted at s with all edges directed away from s , and it is claimed that T is a shortest-path tree. For each v , let $d[v]$ be the weight of the unique path from s to v in T ; the claim is that $d[v] = \delta(s, v)$ for all v . You want to check the claim faster than rerunning Dijkstra's algorithm, which takes $O((n + m) \log n)$ time.

- (a) Give an $O(n + m)$ -time algorithm that accepts if T is a shortest-path tree and rejects otherwise. (Hint: the d -values can be computed in $O(n)$ time. If the claim holds, what inequality relates $d[u]$, $d[v]$, and w_{uv} for *every* edge $(u, v) \in E$?)
- (b) Prove that your algorithm is correct:
 - (i) If it rejects, then T is not a shortest-path tree.
 - (ii) If it accepts, then T is a shortest-path tree. (Hint: sum the inequality from part (a) along an arbitrary path from s to v to show $d[v] \leq \delta(s, v)$. Why is $d[v] \geq \delta(s, v)$?)

Problem 4. There are n lectures, where lecture i occupies the half-open interval $[s_i, f_i)$ with $s_i < f_i$. Lectures whose intervals overlap *conflict* and cannot share a classroom. The times are fixed; the only decision is which classroom each lecture is assigned to. The goal is to use as few classrooms as possible.

The **depth** at time t is the number of lectures whose interval contains t , and the depth d of the instance is the maximum depth over all t .

- (a) Prove that every valid schedule uses at least d classrooms.
- (b) Give an $O(n \log n)$ -time algorithm that uses exactly d classrooms. (Hint: process the lectures in a suitable order, and maintain enough information about the open classrooms to decide in $O(\log n)$ time whether one can be reused.)

- (c) Prove that your algorithm never opens more than d classrooms, and derive its running time.
- (d) Consider instead sorting the lectures by *finish* time and assigning each, in that order, to the lowest-numbered classroom containing no conflicting lecture, opening a new classroom only when necessary. Give a small instance on which this uses more than d classrooms, or argue that it never does.

Problem 5. An electric car travels from College Park to New York City. It starts fully charged, can travel at most R miles on a full charge, and can be recharged to full only at charging stations. The stations are at positions $0 = x_0 < x_1 < \dots < x_n$, in miles from College Park, where x_n is the destination; you may assume $x_i - x_{i-1} \leq R$ for all i . Give an algorithm that, given R and $\langle x_0, \dots, x_n \rangle$, computes the minimum number of recharging stops needed to complete the trip (the start and destination do not count). Briefly justify its correctness and derive its running time.

Problem 6. In compilers for neutral-atom quantum computers, maximal independent sets are used to schedule atom movements. This problem relates them to interval scheduling.

Let $G = (V, E)$ be an undirected graph with $n = |V|$ and $m = |E|$. A set $S \subseteq V$ is **independent** if no edge joins two vertices of S ; it is **maximal** if no vertex can be added while keeping it independent, and **maximum** if no independent set of G is larger. Every maximum independent set is maximal, but not conversely (Fig. 3).



Figure 3: Two maximal independent sets (shaded) in the same graph; only the one on the right is maximum.

Consider the following algorithm.

MAXIMALIS(G): Set $S \leftarrow \emptyset$. While the current graph has a vertex, pick any vertex v , add v to S , and delete v and its neighbors (with their incident edges). Return S .

- (a) Prove that MAXIMALIS always returns a maximal independent set. Then show that for every $n \geq 2$ there is a graph on n vertices for which it may return a set of size 1 while the maximum independent set has size $n - 1$.
- (b) Show that MAXIMALIS can be implemented in $O(n + m)$ time on an adjacency list. (Hint: each vertex is deleted exactly once; keep an array of flags marking which vertices remain.)
- (c) Show that a maximum independent set of an arbitrary graph can be found by examining every subset of V , in $O(2^n n^2)$ time. (No polynomial-time algorithm is known; the problem is NP-hard, as we will see later in the semester. You need not prove this.)

- (d) Now let G be an **interval graph**: the vertices are n half-open intervals $[s_1, f_1), \dots, [s_n, f_n)$, and $\{i, j\} \in E$ when intervals i and j overlap. Prove that $S \subseteq V$ is independent if and only if its intervals are pairwise disjoint. Conclude that the earliest-finish-time greedy algorithm from class finds a *maximum* independent set of G in $O(n \log n)$ time, without constructing G .
- (e) Compare the running times of parts (b), (c), and (d), and what each algorithm guarantees about the size of its output. What does this suggest about exploiting the structure of a problem?