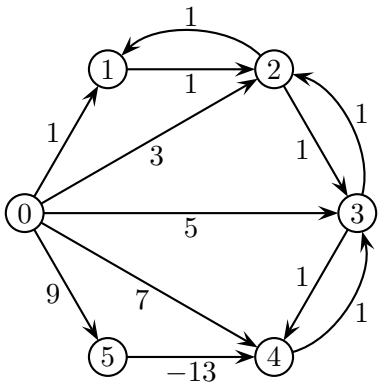
CSCI 0500: Data Structures, Algorithms, and Intractability (Fall 2025)

Assignment 6

Due at 11:59pm ET, Monday, Dec 8

1. (1 point) In this question, we study single-source shortest paths with arbitrary edge weights.

   Consider the following directed graph H. The edge weights are labeled next to each edge. The source node is $s = 0$.



   (a) Consider running Dijkstra's algorithm on H. Write down the order in which nodes are finalized. Write down the distances returned by Dijkstra's algorithm.

   To correctly compute shortest-path distances when edge weights may be negative, we can use the Bellman-Ford algorithm. Let $G = (V, E, w)$ be a weighted directed graph with source node s. Let $n = |V|$ and $m = |E|$. For simplicity, we assume that G has no negative cycles. [1]

   Define $\mathrm{OPT}(i, v)$ as the length of the shortest path from s to v that uses at most i edges, and $\mathrm{OPT}(i, v) = +\infty$ if no such path exists. The Bellman-Ford algorithm computes $\mathrm{OPT}(i, v)$ for all $0 \leq i < n$ and $v \in V$ as follows:

$$\mathrm{OPT}(0, s) = 0$$
$$\mathrm{OPT}(0, v) = +\infty, \ \forall v \neq s \ .$$
$$\mathrm{OPT}(i, v) = \min\Big(\mathrm{OPT}(i-1, v), \ \min_{(u,v)\in E}\big(\mathrm{OPT}(i-1, u) + w(u, v)\big)\Big), \ \forall i \geq 1, \forall v \in V \ .$$

   The algorithm returns $\mathrm{OPT}(n-1, v)$ as the shortest-path distance from s to every $v \in V$.

[1] A negative cycle is a cycle whose total edge weight is negative.

```
def bellman_ford(n, E_w, s):
  d = [[float('inf')] * n for _ in range(n)]
  d[0][s] = 0
  for i in range(1, n):
    for v in range(n):
      d[i][v] = d[i - 1][v]
    for (u, v, w) in E_w:
      d[i][v] = min(d[i][v], d[i - 1][u] + w)
  return d[n-1]
```

Lemma 1. *If G has no negative cycles, the Bellman-Ford algorithm correctly computes the shortest-path distances from s to every $v \in V$.*

You are encouraged to try to prove this lemma, but it is not required for this assignment.

(b) Assume $m = \Omega(n)$. Prove that the Bellman-Ford algorithm runs in time $O(mn)$.

(c) Consider running the Bellman-Ford algorithm on the graph H with source node $s = 0$. Fill in the table below with the values of $\text{OPT}(i, v)$, where rows correspond to i (the maximum number of edges allowed) and columns correspond to nodes $v \in V$.

OPT =

i \ v	$s = 0$	1	2	3	4	5
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1						
2						
3						
4						
5						

2. (1 point) In this question, we introduce Kruskal's algorithm for minimum spanning tree.

Let $G = (V, E, w)$ be a weighted undirected graph with $n = |V|$ nodes and $m = |E|$ edges. For simplicity, we assume that G is connected and that all edge weights are distinct.

Algorithm 1: Kruskal's Algorithm

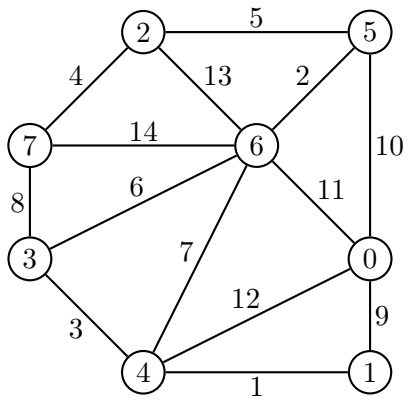
Input : A weighted undirected graph $G = (V, E, w)$
Output: A minimum spanning tree T of G
$T \leftarrow \varnothing$
for *each edge $e \in E$ in increasing order of edge weight* **do**
  if *adding e to T does not create a cycle* **then**
    $T \leftarrow T \cup \{e\}$
return T

Lemma 2. *Kruskal's algorithm correctly computes a minimum spanning tree.*

Lemma 3. *Kruskal's algorithm can be implemented to run in $O(m \log n)$ time.*

You are encouraged to try to prove these lemmas, but it is not required for this assignment.

Consider the following graph H. The edge weights are labeled next to each edge.



(a) Consider running Prim's algorithm on H starting from node $s = 0$. List the minimum-spanning-tree edges in the order they are added by Prim's algorithm.

(b) Consider running Kruskal's algorithm on H. List the minimum-spanning-tree edges in the order they are added by Kruskal's algorithm.

3. (1 point) In this question, we study the longest increasing subsequence problem. The input is a 1-indexed array X of $n > 0$ distinct integers. The goal is to output the length of the longest increasing subsequence of X.

   A subsequence of X is a sequence obtained by deleting (zero or more) elements from X while preserving the order of the remaining elements. A sequence is increasing if each element is greater than the previous one.

   Suppose $X = (2, 7, 5, 3, 8, 4, 6, 1)$. A longest increasing subsequence of X is $(2, 3, 4, 6)$, so the output should be 4.

   Solve one of Parts (a1) or (a2), then build on it to solve Part (b). You can use any algorithms and data structures covered in class or previous assignments without implementing them. You do not need to prove the correctness or analyze the runtime of your algorithms.

   (a1) Define $\mathrm{OPT}(i)$ as the length of the longest increasing subsequence ending at $X[i]$.
   Design an $O(n^2)$ time algorithm for longest increasing subsequence based on this idea.
   (Hint: The final answer is $\max_{1 \le i \le n} \mathrm{OPT}(i)$. For $\mathrm{OPT}(i)$, by definition, the last element of the corresponding subsequence is $X[i]$. Consider all possibilities for the second-to-last element.)

   (a2) Consider all increasing subsequences of $(X[1], \ldots, X[i])$ of length ℓ. Note that it suffices to remember only one of these subsequences: the one with the smallest ending element, because this subsequence is easiest to extend later.
   For all $1 \le i \le n$ and $1 \le \ell \le n$, define $\mathrm{OPT}(i, \ell)$ as the smallest possible value z such that there is an increasing subsequence of $(X[1], \ldots, X[i])$ of length ℓ ending with z. Let $\mathrm{OPT}(i, \ell) = +\infty$ if no such subsequence exists.

Design an $O(n^2)$ time algorithm for longest increasing subsequence based on this idea.

(Hint: The final answer is the largest ℓ such that $\mathrm{OPT}(n, \ell) < +\infty$. For $\mathrm{OPT}(i, \ell)$, consider whether $X[i]$ can extend an earlier subsequence of length $\ell - 1$, and whether doing so gives a better subsequence of length ℓ. More specifically, compare $X[i]$ with $\mathrm{OPT}(i-1, \ell-1)$ and $\mathrm{OPT}(i-1, \ell)$.)

(b) Design an $O(n \log n)$ time algorithm for longest increasing subsequence.

(Hint: If you did Part (a1), consider using a data structure that supports range maximum queries, in particular, for querying the maximum value among items whose keys are smaller than a given key. If you did Part (a2), observe that for a fixed i, the values of $\mathrm{OPT}(i, \ell)$ are monotonically increasing as ℓ increases.)