ENS DE LYON

Fundamentals in Computer Science
Homework (due to 25/10/2024)

Fall 2024
Prof. Thomassé
M. Marin

The homework is due on October 25th, and must be submitted via email to both malory.marin@ens-lyon.fr and stephan.thomasse@ens-lyon.fr with the subject line [DM:FoCS]. You have two options for submission: either a single Jupyter notebook (with code in Python cells and written answers in markdown) or two separate files—a PDF written in LaTeX and a .py file for the code. I encourage you to choose the first option.

You must carefully test every function you write, and the tests should be included in the code. Failure to do so will result in penalties.

You are encouraged to collaborate with one another, but each of you must write your own solutions.

Exercise 1. *Shortest Path Using Dijkstra's Algorithm*

You are given a weighted graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Each edge (u, v) has a non-negative weight $w(u, v)$, representing the cost of traveling from vertex u to vertex v. Your task is to find the shortest path from a given starting vertex s to every other vertex in the graph.

Dijkstra's algorithm is a greedy algorithm used to find the shortest path from a source vertex to all other vertices in a weighted graph with non-negative edge weights. The algorithm maintains a set of vertices for which the shortest path has been found and iteratively updates the shortest paths for the remaining vertices.

The algorithm works as follows:

- Initialize the distance to the source vertex as 0 and all other distances as infinity.
- Use a priority queue to always select the vertex with the smallest tentative distance. A priority queue allows for the storage of pairs (element, value), where each element is associated with a priority value. It enables efficient retrieval of the element with the smallest value. This data structure can be implemented using various methods, such as a list or a heap, with heaps generally providing better performance for insertion and extraction operations.
- For the current vertex, update the distances to its neighboring vertices.
- Continue this process until all vertices have been visited.

Pseudocode:

```
function Dijkstra(Graph, source):
    dist[source] ← 0                      // Distance to source is 0
    for each vertex v in Graph:
        if v != source:
            dist[v] ← infinity            // Initialize other distances to infinity
        previous[v] ← UNDEFINED           // Previous node in the optimal path
    Q ← all vertices in Graph             // Priority queue of all vertices in the graph

    while Q is not empty:
        u ← vertex in Q with min dist[u] // Select the vertex with the smallest distance
        remove u from Q                   // Remove vertex from queue

        for each neighbor v of u:
            alt ← dist[u] + length(u, v) // Tentative distance through u
            if alt < dist[v]:             // If a shorter path is found
                dist[v] ← alt             // Update distance
                previous[v] ← u           // Update previous vertex
```

```
return dist, previous          // Return the shortest distances and the path tree
```

(a) Write a Python function `dijkstra(graph, start) -> Dict[int, float]` that takes a graph G (represented as an adjacency list or matrix) and a source vertex s, and returns the shortest path distances from s to all other vertices.

(b) Analyze the time complexity of Dijkstra's algorithm. Consider the case when a binary heap is used to implement the priority queue, and compare it to the case when a simple list is used.

(c) (*Bonus*) Prove that Dijkstra's algorithm always finds the shortest path for every vertex when all edge weights are non-negative. Discuss why the algorithm fails when negative weights are present.

Exercise 2. ***Dynamic Programming for Longest Common Subsequence (LCS)***

The Longest Common Subsequence (LCS) problem is a fundamental problem in computer science, particularly in areas like text comparison and bioinformatics. Given two sequences, the goal is to find the longest subsequence that appears in both sequences in the same order, but not necessarily consecutively.

For example, for the sequences X="ABCBDAB" and Y="BDCAB", a LCS is "BCAB" with a length of 4.

(a) Write a function that returns the size of a longest common subsequence between two strings. You can use dynamic programming to build a table to store the lengths of common subsequences.

- **Hint**: Consider using a 2D table where the entry at (i, j) represents the length of the LCS of the first i characters of the first string and the first j characters of the second string.

(b) Analyze the time and space complexity of your LCS function.

(c) Write a function that receives an integer (length of the sequence) and an alphabet size, and returns a random sequence of that length using characters from the specified alphabet.

(d) Write a function that draws the average longest common subsequence length for random pairs of strings, using an alphabet of size 2. Observe that the relationship between the length of the strings and the average LCS length is linear.

(e) Let $l(n, s)$ be the average size of the LCS between two random strings of length n in an alphabet of size s. From the previous question, we observe that $l(n, s)$ is linear in n. Therefore, we can express it as $l(n, s) \sim \alpha_s \cdot n$. Write a function that plots α_s versus s, for $s \in \{0, ..., 10\}$.