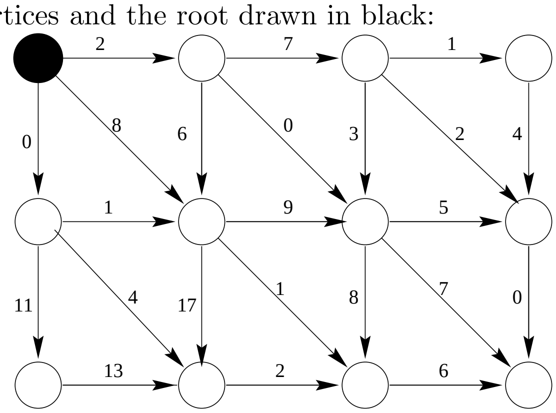
CSci 231 Homework 11 *

Shortest Paths, CLRS Chapter 24.0, 24.3

Dijkstra'a SSSP algorithm works on general graphs with non-negative weights. The running time of Dijkstra's algorithm is $O(|E| + |V| \times$ INSERT $+ |V| \times$ DELETE-MIN $+ |E| \times$ CHANGE-KEY). Assuming the graph is connected and the priority queue is implemented as a heap the running time is $O(|E| \log |V|)$. The running time can be improved to $O(|E| + |V| \log |V|)$ using improved versions of priority queue (for instance the Fibonacci heap, which supports INSERT and CHANGE-KEY in $O(1)$ time amortized, and DELETE-MIN in $O(\lg n)$ amortized). While Dijkstra's algorithms gives the best known upper bounds for general SSSP with general non-negative weights and linear space, improved algorithms are known for special classes of graphs. In this homework you will investigate several examples and derive improved bounds for computing SSSP.

1. **Shortest path for Directed Acyclic Graphs (DAGs):** Let $G = (V, E)$ be a DAG and let s be a vertex in G. Find a linear time $O(|V| + |E|)$ algorithm for computing SSSP(s). What vertices are reachable from s? Sketch a proof that your algorithm is correct. Does your algorithm need the constraint that the edge weights are non-negative?

2. 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.

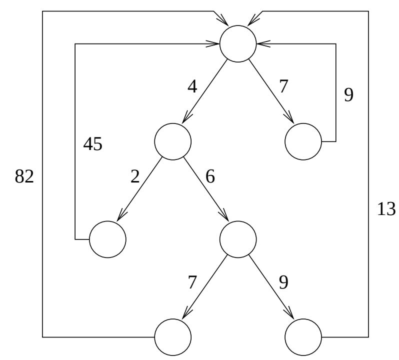
(a) How long would it take Dijkstra's algorithm to find the length of the shortest path from the root to all other vertices?

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

*This is the last homework. Collabortaion is encouraged as usually.

(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).

3. Consider a directed weighted graph with non-negative weights which is formed by adding an edge from every leaf in a binary tree to the root of the tree. Let the graph/tree have n vertices. An example of such a graph with $n = 7$ could be the following:



We want to design an algorithm for finding the shortest path between two vertices in such a graph.

(a) How long time would it take Dijkstra's algorithm to solve the problem?

(b) Describe and analyze a more efficient algorithm for the problem.

4. **All-Pair-Shortest-Paths with dynamic programming:** In the APSP problem, we want to compute the shortest path between any two vertices $u, v \in V$. Note that the output is of size $O(|V|^2)$ so we cannot hope to design a better than $O(|V|^2)$ time algorithm.

(a) We can solve the problem simply by running Dijkstra's algorithm $|V|$ times. What is the running time of this approach? What does the running time become for sparse graphs ($E = \theta(V)$) and for dense graphs ($E = \theta(V^2)$)?

We can obtain another algorithm by working on adjacency matrix A. For weighted graphs, a_{ij} is equal to the weight w_{ij} of the edge (v_i, v_j); w_{ij} is assumed to be ∞ is the edge does not exist.

Let A, B be two matrices, and let $C = A \cdot B$. Remember that

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

We redefine the $\sum$ and $\cdot$ operators in matrix multiplication to mean *minimum* and $+$ respectively. That is,

$$c_{ij} = min_{k=1..n}\{a_{ik} + b_{kj}\}$$

(b) What does $A \cdot A$ represent in terms of paths in graph G? What about $\min\{A, A \cdot A\}$?

(c) Sketch an algorithm for computing APSP using this approach and estimate its running time.