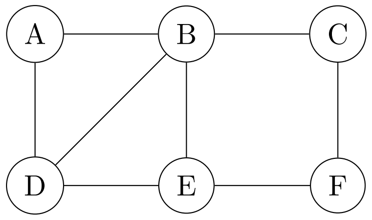
CS 170 DIS 03

Released on 2018-09-17

1 Breadth-First Search



Run breadth-first search on the above graph, breaking ties in alphabetical order (so the search starts from node A). At each step, state which node is processed and the resulting state of the queue. Draw the resulting BFS tree.

2 Dijkstra's Algorithm Fails on Negative Edges

Draw a graph with five vertices or fewer, and indicate the source where Dijkstra's algorithm will be started from.

1. Draw a graph with no negative cycles for which Dijkstra's algorithm produces the wrong answer.
2. Draw a graph with at least two negative weight edge for which Dijkstra's algorithm produces the correct answer.

3 Fixing Dijsktra's Algorithm with Negative Weights

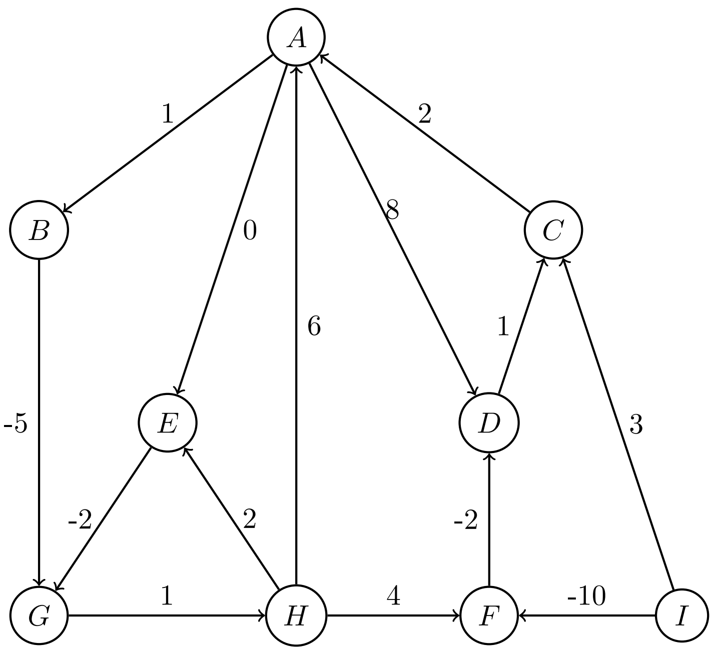
Dijkstra's algorithm doesn't work on graphs with negative edge weights. Here is one attempt to fix it:

1. Add a large number M to every edge so that there are no negative weights left.
2. Run Dijkstra to find the shortest path in the new graph.
3. Return the path Dijkstra found, but with the old edge weights (i.e. subtract M from the weight of each edge).

Show that this algorithm doesn't work by finding a graph for which it must give the wrong answer.

4 Bellman-Ford Practice

(a) Run the Bellman-Ford algorithm on the following graph, from source A. Relax edges (u, v) in lexicographic order, sorting first by u then by v.



(b) What problem occurs when we change the weight of edge (H, A) to 1? How can we detect this problem when running Bellman-Ford? Why does this work?

(c) Let $G = (V, E)$ be a directed graph. Under what condition does the Bellman-Ford algorithm returns the same shortest path tree (from source $s \in V$) regardless of the ordering on edges?

5 Midterm Prep: Graph Short Answer

Answer each question below *concisely* (one short sentence or a number should suffice). Do not justify your answer. Do not show your work.

(a) Suppose we are given a directed graph $G = (V, E)$ represented in adjacency list format, and we want to test whether G is a dag or not, using a method that is as asymptotically efficient as possible. In a sentence, what approach would you use?

(b) What's the running time of your solution in (a), using $O(\cdot)$ notation?

(c) Let $G = (V, E)$ be a directed graph with $|V| = 1000$ vertices, $|E| = 5000$ edges, and 700 strongly connected components. How many vertices does the metagraph have?

(d) Let $G = (V, E)$ be a dag. Let s be a source vertex in G. Suppose we set the weight of each edge to 1 and run Dijkstra's algorithm to compute the distance from s to each vertex $v \in V$, and then order the vertices in increasing order of their distance from s. Are we guaranteed that this is a valid topological sort of G?

Circle Yes or No.

(e) Justify your answer to part (d) as follows: If you circled Yes, then give one sentence that explains the main idea in a proof of this fact. If you circled No, then give a small counterexample (a graph with at most 4 vertices) that disproves it.

(f) Suppose we run Dijkstra's algorithm on a graph with n vertices and $O(n \lg n)$ edges. Assume the graph is represented in adjacency list representation. What's the asymptotic running time of Dijkstra's algorithm, in this case, if we use a binary heap for our priority queue? Express your answer as a function of n, and use $O(\cdot)$ notation.

6 Midterm Prep: Dijkstra Tiebreaking

We are given a directed graph G with positive weights on its edges. We wish to find a shortest path from s to t, and, among all shortest paths, we want the one in which the longest edge is as short as possible. How would you modify Dijkstra's algorithm to this end?

7 Midterm Prep: Divide-and-Conquer

How many lines does this algorithm print? Write a recurrence and solve it. Give your answer in $\Theta(\cdot)$ notation.
Bonus: Give the exact solution (no asymptotic notation).

```
function printalot(n:  an integer power of 2)
    if $n > 1$ {
        printalot($n/2$)
        printalot($n/2$)
        printalot($n/2$)
        for $i = 1$ to $n^4$ do
            println("are we done yet?")
    }
```