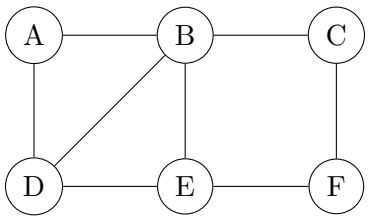
CS 170 DIS 03

Released on 2018-09-17

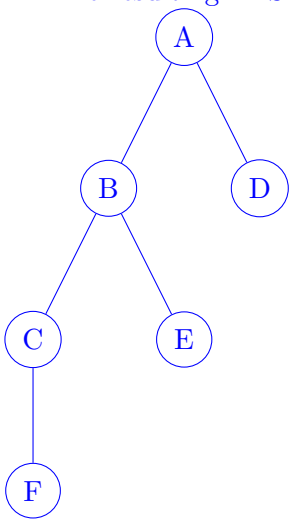
1 Breadth-First Search



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.

Solution:

The resulting BFS tree:



The node processed at each step and the resulting queue:

Node Processed	Queue
	A
A	B, D
B	D, C, E
D	C, E
C	E, F
E	F
F	{}

In this implementation of BFS, we ignore the technicality of pushing duplicate vertices into the queue. Note that we can deal with this by checking if the vertex at the head of the queue has been visited already and discarding it if so.

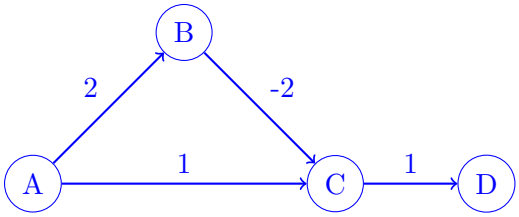
2 Dijkstra's Algorithm Fails on Negative Edges

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.

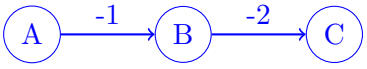
Solution:

1. Here's one example:



   Dijkstra's algorithm from source A will give the distance to D as 2 rather than 1, because it visits C before B.

2. Dijkstra's algorithm always works on directed paths. For example:



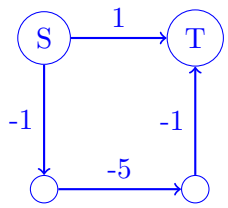
3 Fixing Dijsktra's Algorithm with Negative Weights

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.

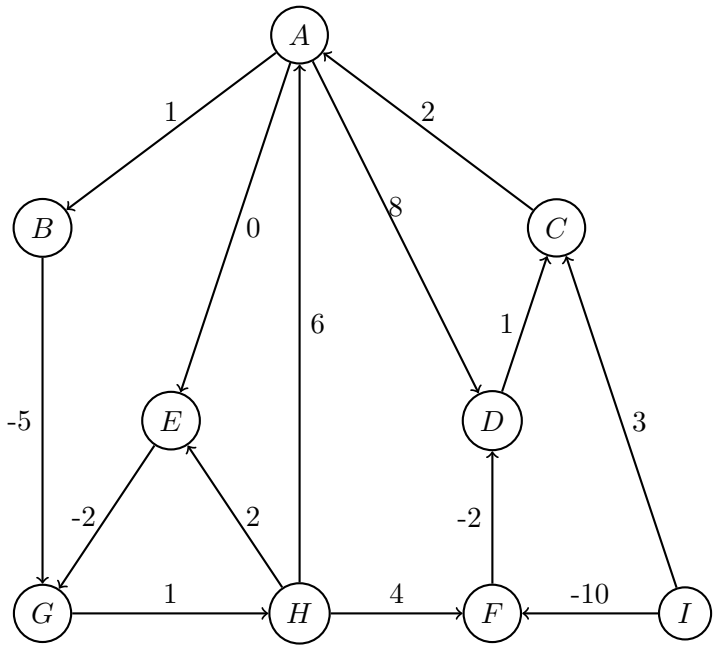
Solution: The above algorithm doesn't work when the actual shortest path has more edges than other potential shortest paths. In this case, the paths with more edges have their weights increased more than the path with fewer edges. We can see this in the following counterexample:



The shortest path is "down-right-up" (weight -7). After adding $M = 5$ to each edge, we increase the actual shortest path by fifteen 15. The path "right" only increases by 5 and so the algorithm returns this path as the shortest path.

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.



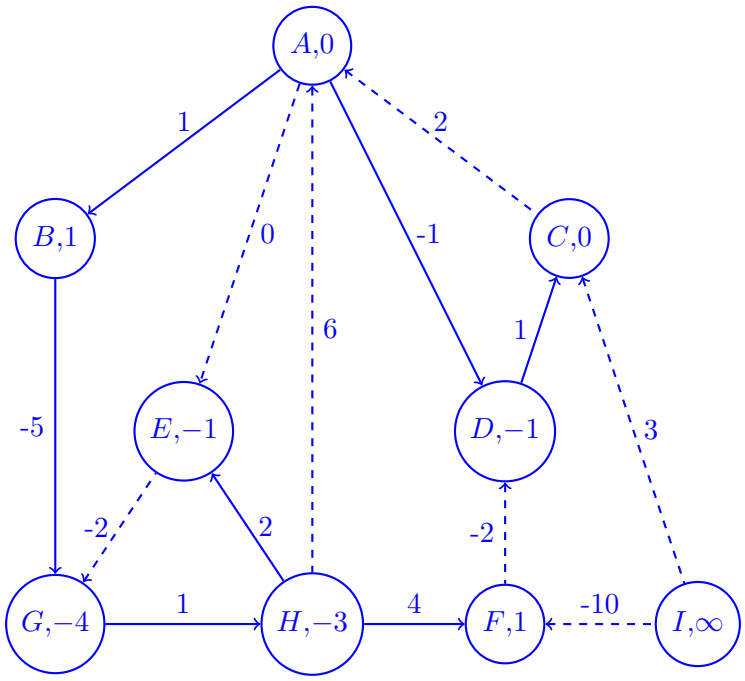
(b) What problem occurs when we change the weight of edge (H, A) to 1? How can we detect this problem when running Bellman-Ford? Why does this work?

(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?

Solution:

(a) The resulting shortest path tree (dashed edges are non-tree edges):

Remember that you can terminate the Bellman-Ford algorithm as soon as a relaxation step does not change any distances.



(b) Changing the weight of (H, A) to 1 introduces a negative cycle. We can detect this by checking whether, after making $|V| - 1 = 9$ passes over the edges (relaxation steps), we can still update the distances. Try it out! The reason it works is that after k passes, we have found the length of all shortest paths from s with at most k edges. Since a simple path can only have at most $|V| - 1$ edges, if we can update the distances on the $|V|$-th pass, the new shortest path contains a cycle. A cycle can only be part of a shortest path if it is of negative weight.

(c) If and only if the shortest path tree rooted at s is unique, i.e. if for each $v \in V$ there exists a unique shortest path from s to v. Note that this is not the case in the example: there are two paths of length -1 from A to D.

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?

Solution: Either: Use DFS, check for back-edges.

Or: Decompose into strongly connected components, check for a SCC with more than one vertex.

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

Solution: $O(|V| + |E|)$.

(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?

Solution: 700.

Comment: Each vertex in the metagraph corresponds to a strongly connected component in G, so the number of vertices in the metagraph is the same as the number of SCCs in G.

(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.

Solution: 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.

Solution: $V = \{a, b, c, d\}$, $E = \{(a, b), (b, c), (c, d), (a, d)\}$, $w(e) = 1$ for $e \in E$. **Comment:** Distances are $a : 0, b : 1, c : 2, d : 1$ but $c < d$ in any topological ordering.

(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.

Solution: $O(n(\lg n)^2)$.

Comment: $|V| = n$, $|E| = O(n \lg n)$, and Dijkstra's runs in $O((|V| + |E|) \lg |V|)$ time, which is $O((n + O(n \lg n)) \lg n) = O((n \lg n) \times \lg n) = O(n(\lg n)^2)$.

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?

Solution: Modify Dijkstra's algorithm to keep a map $\ell(v)$ which holds the longest edge on the current shortest path to v. Initially $\ell(s) := 0$ for source s and $\ell(v) := \infty$ for all $v \in V \setminus \{s\}$. When we consider a vertex u and its neighbour v, if $dist(u) + w(u, v) < dist(v)$, then we set $\ell(v) := \max \ell(u), w(u, v)$ in addition to the standard Dijkstra's steps. If there is a neighbour v of u such that $dist(v) = dist(u) + w(u, v)$, *and* $\ell(v) > \max \ell(u), w(u, v)$, we set v's predecessor to u and update $\ell(v) := \max \ell(u), w(u, v)$.

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
            printline("are we done yet?")
    }
```

Solution: $T(n)$ is the number of lines printed for input n, then $T(n) = 3T(n/2) + n^4$ with a base case of $T(1) = 0$.

For an asymptotic solution, we can apply the master theorem with $n^{\log_b a} = n^{\log_2 3}$ and $f(n) = n^4$. Since $f(n)$ polynomially dominates $n^{\log_b a}$ (see handout on website), $T(n) = \Theta(f(n)) = \Theta(n^4)$ by the master theorem.

For an exact solution, we expand the recurrence as follows:

$$T(n) = n^4 + 3(n/2)^4 + 9(n/4)^4 + \ldots + 3^k(2)^4,$$

where $k = \log_2 n - 1$ since $T(1) = 0$.

This gives $T(n) = \sum_{i=0}^{k} 3^i(n/2^i)^4 = n^4 \sum_{i=0}^{k}(3/16)^i = n^4((\frac{3}{16}^{\log_2 n} - 1)/(\frac{3}{16} - 1))$ (by the geometric series formula and using $k = \log_2 n - 1$). This can be further simplified to $T(n) = \frac{16}{13}n^4(1 - n^{\log_2(3/16)}) = \frac{16}{13}(n^4 - n^{\log_2 3})$.