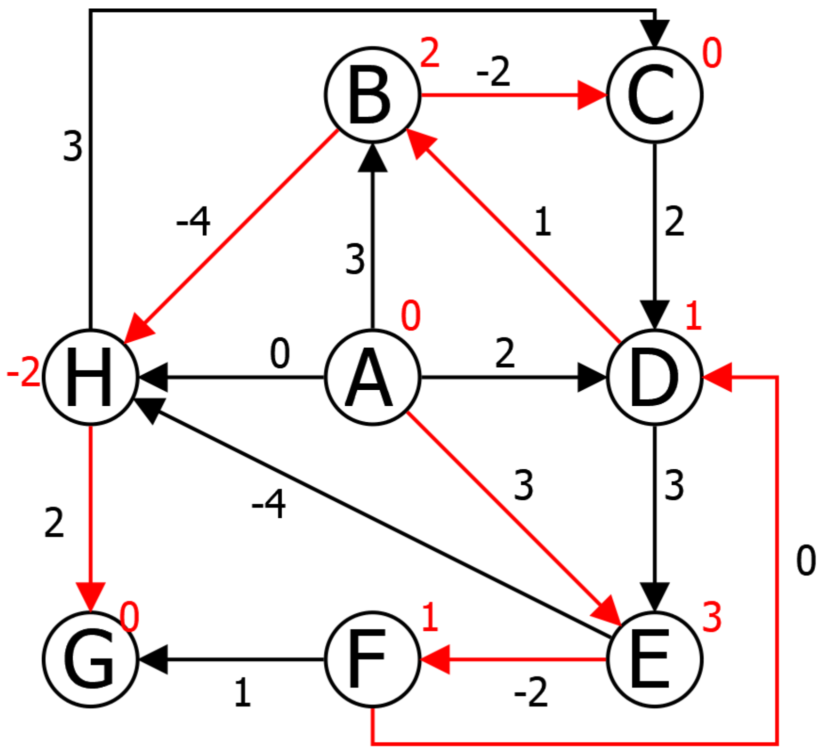
Question 1 (Shortest Path Computation, 30 points)**.** *What are the lengths of the shortest paths from A to each other vertex in the graph below?*



The answers are marked in red along with the shortest path tree. We can obtain this by running Bellman-Ford on the graph, getting the following output:

k	A	B	C	D	E	F	G	H
0	0	∞	∞	∞	∞	∞	∞	∞
1	0	3	∞	2	3	∞	∞	0
2	0	3	1	2	3	1	2	-1
3	0	3	1	1	3	1	1	-1
4	0	2	1	1	3	1	1	-1
5	0	2	0	1	3	1	1	-2
6	0	2	0	1	3	1	0	-2

After which it stabilizes.

Question 2 (Reaches Everything, 35 points)**.** *Let G be a finite DAG and v one of its vertices. Show that every vertex of G is reachable from v if and only if v is the only source in G.*

On the one hand, if there is a source $w \neq v$, then since w has no in-edges, it cannot be reached from v.

On the other hand, every vertex is reachable from a source. This is because you can follow edges backwards until you hit a source. This process must terminate because otherwise, it would eventually hit a cycle contradicting G being a DAG. If v is the only source, this means that every vertex is reachable from v.

Question 3 (Public Transit Navigation, 35 points). *Graphopolis' subway system is extremely regular. The layout of the system is given by a directed graph G with each edge corresponding to a train route that leaves at regular intervals and always takes exactly the same amount of time. Jade arrives at Source Station precisely at noon and wants to make it to Sink Junction as quickly as possible. Her subway map tells her for each edge of G when the trains leave along that route (specified by the time the first train leaves and the time between successive trains) and how long it takes the train to reach the next station.*

Provide an algorithm that given this information as well as the locations of the source and sink vertices determines the earliest time that Jade can make it to Sink Junction. For full credit, your algorithm should run in time $O(|V|\log|V| + |E|)$ or better.

Firstly, we define a function `NextArrival(e,t)` that determines if Jade is at the starting point of edge e at time t, waits for the next train at e determines when she arrives at the other end of e. This can be computed by using the schedule to find the next time after t that a train departs along e and adding the transit time. We can then produce a full algorithm modelled in Dijkstra's algorithm:

```
BestTimes(G,schedule)
  create priority queue Q
  For v vertex in G
    time(v) = infinity
    Q.insert(v)
  time(source) = noon
  Q.deckey(source)
  While(Q is not empty)
    v <- Q.deletemin()
    For (v,w) edge in G
      Tend <- NextArrival((v,w),time(v))
      If Tend < time(w)
        time(w) <- Tend
        Q.deckey(w)
  Return time(sink)
```

The runtime analysis here is identical to Dijkstra, and thus is $O(|V|\log|V| + |E|)$. The analysis is similar as well. We claim that whenever a vertex v is removed from Q that `time(v)` is the earliest time that Jade can reach v, and also claim that throughout the algorithm `time(w)` (for any w not equal to the source s) is the earliest time that Jade can reach w by first going to a vertex v that has been removed from Q and then taking an edge from v to w. These claims are easily shown by induction.

When s is removed, `time(s)` is noon, which is the earliest Jade can reach s. Initially each `time(w)` is infinity because no vertex has been removed from Q yet, but once any vertex is removed, `time(w)` is updated to be the minimum of its current value and `NextArrival((v,w),time(v))`, which is the time Jade would arrive at w if she took the fastest route to v followed by the edge (v, w). Finally, whenever a v is removed from Q, it has the smallest value of `time(v)` of any vertex still in Q. This means that `time(v)` is the earliest possible time that Jade could reach a vertex still in Q by taking one edge away from a vertex already removed from Q. But this must therefore be the fastest that Jade could reach any vertex currently in Q, and therefore the correct fastest time to v.