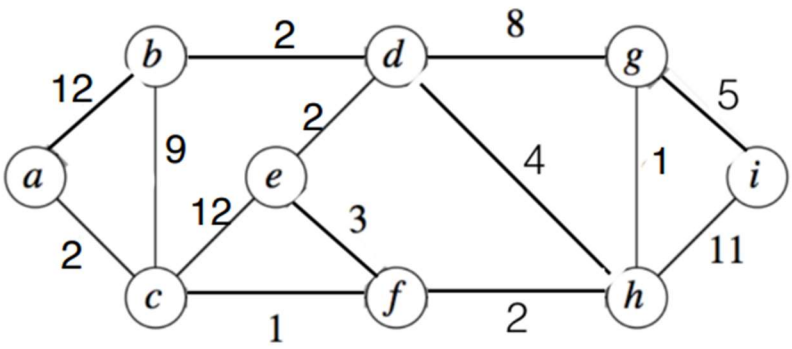
Brandon Vo

Homework 12

1. Run Dijkstra's algorithm on the following graph where the source vertex is a.



	A	B	C	D	E	F	G	H	I
1 (A)	0	12	2	∞	∞	∞	∞	∞	∞
2 (C)	0	11 (A)	**2 (A)**	∞	14 (C)	3 (C)	∞	∞	∞
3 (F)	0	11 (C)	**2 (A)**	∞	6 (F)	**3** (C)	∞	5 (F)	∞
4 (H)	0	11 (C)	**2 (A)**	9 (H)	6 (F)	**3** (C)	6 (H)	**5** (F)	16 (H)
5 (E)	0	11 (C)	**2 (A)**	8 (E)	**6** (F)	**3** (C)	6 (H)	**5** (F)	16 (H)
6 (G)	0	11 (C)	**2 (A)**	8 (E)	**6** (F)	**3** (C)	**6** (H)	**5** (F)	11 (G)
7 (D)	0	11 (C)	**2 (A)**	**8 (E)**	**6** (F)	**3** (C)	**6** (H)	**5** (F)	11 (G)
8 (B)	0	**10 (D)**	**2 (A)**	**8 (E)**	**6** (F)	**3** (C)	**6** (H)	**5** (F)	11 (G)
8 (i)	0	**10 (D)**	**2 (A)**	**8 (E)**	**6** (F)	**3** (C)	**6** (H)	**5** (F)	**11 (G)**

i has no adjacencies to explore, so the algorithm stops here. The rest of the graph has been explored after i has been reached.

Brandon Vo

2. Question 25.1-1 in CLRS on page 691.

Run SLOW-ALL-PAIRS-SHORTEST-PATHS on the weighted, directed graph of Figure 25.2, showing the matrices that result for each iteration of the loop. Then do the same for FASTER-ALL-PAIRS-SHORTEST-PATHS.

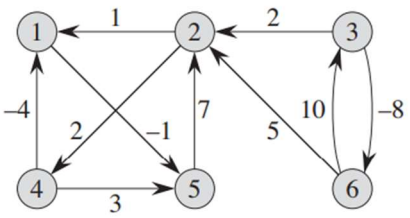


Figure 25.2 A weighted, directed graph for use in Exercises 25.1-1, 25.2-1, and 25.3-1.

Beginning iteration

	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	∞	∞
3	∞	2	0	∞	∞	-8
4	-4	∞	∞	0	3	∞
5	1	7	∞	∞	0	∞
6	∞	5	10	∞	∞	0

SLOW-ALL-PAIRS-SHORTEST-PATHS Variant:

First iteration: Starting at 1

L^0:

- [1, 2] = MIN{[1, 2], [1, 5] + [5, 2]}
  - ∞ > -1 + 7
  - [1, 2] = 6
- [2, 1] = MIN{[2, 1], [2, 4] + [4, 1]}
  - 1 > 2 – 4
  - [2, 1] = -2
- [2, 5] = MIN{[2,5], [2, 4] + [4, 5], [2, 1] + [1, 5]}
  - ∞ > 2 + 3
  - 5 > 1 - 1
  - [2, 5] = 0
- [3, 1] = MIN{[3, 1], [3, 2] + [2, 1]}
  - ∞ > 2 + 1

    - [3, 1] = 3
- [3, 2] = MIN{[3, 2], [3, 6] + [6, 2]}
    - 2 > -8 + 5
    - [3, 2] = -3
- [3, 4] = MIN{[3, 4], [3, 2] + [2, 4]}
    - ∞ > 2 + 2
    - [3, 4] = 4
- [4, 2] = MIN{[4, 2], [4, 5] + [5, 2]}
    - ∞ > 3 + 7
    - [4, 2] = 10
- [4, 5] = MIN{[4, 5], [4, 1] + [1, 5]}
    - 3 > -4 - 1
    - [4, 5] = -5
- [5, 1] = MIN{[5, 1], [5, 2] + [2, 1]}
    - ∞ > 7 + 1
    - [5, 1] = 8
- [5, 4] = MIN{[5, 4], [5, 2] + [2, 4]}
    - ∞ > 7 + 2
    - [5, 4] = 9
- [6, 2] = MIN{[6, 2], [6, 3] + [3, 2]}
    - 5 < 10 + 2
- [6, 1] = MIN{[6, 1], [6, 2] + [2, 1]}
    - ∞ > 5 + 1
    - [6, 1] = 6
- [6, 4] + MIN{[6, 4], [6, 2] + [2, 4]}
    - ∞ > 5 + 2
    - [6, 4] = 7

	1	2	3	4	5	6
1	0	6	∞	∞	-1	∞
2	-2	0	∞	2	0	∞
3	3	-3	0	4	∞	-8
4	-4	10	∞	0	-5	∞
5	8	7	∞	9	0	∞
6	6	5	10	7	∞	0

Second iteration:

- [1, 1] = [1, 1] < [1, 5] + [5, 7] + [2, 1]
- [1, 4] = MIN{[1, 4], [1, 5] + [5, 2] + [2, 4]}

    - ∞ > -1 + 7 + 2
    - [1, 4] = 8
- [2, 2] = [2, 2] < [2, 4] + [4, 5] + [5, 2]
- [2, 5] = MIN{[2, 5], [2, 4] + [4, 1] + [1, 5]
    - 0 > 2 – 4 – 1
    - [2, 5] = -3
- [3, 1] = MIN{[3, 1], [3, 2] + [2, 4] + [4, 1], [3, 6] + [6, 2] + [2, 1]}
    - 3 > 2 + 2 – 4
    - 0 > -8 + 5 + 1
    - [3, 1] = -2
- [3, 4] = MIN{[3, 4], [3, 6] + [6, 2] + [2, 4]}
    - 4 > -8 + 5 + 2
    - [3, 4] = -1
- [3, 5] = MIN{[3, 5], [3, 2] + [2, 4] + [4, 5], [3, 2] + [2, 1] + [1, 5]}
    - ∞ > 2 + 2 + 3
    - 7 > 2 + 1 - 1
    - [3, 5] = 2
- [4, 1] = [4, 1] < [4, 5] + [5, 2] + [2, 1]
- [4, 2] = MIN{[4, 2], [4, 1] + [1, 5] + [5, 2]}
    - 10 > -4 – 1 + 7
    - [4, 2] = 2
- [4, 4] = [4, 4] < [4, 5] + [5, 2] + [2, 4]
- [5, 1] = MIN{[5, 1], [5, 2] + [2, 4] + [4, 1]}
    - 8 > 7 + 2 – 4
    - [5, 1] = 5
- [5, 5] = [5, 1] < [5, 2] + [2, 4] + [4, 5]
- [6, 1] = MIN{[6, 1], [6, 2] + [2, 4] + [4, 1]}
    - 6 > 5 + 2 – 4
    - [6, 1] = 3
- [6, 2] = [6, 2] < [6, 3] + [3, 6] + [6, 2]
- [6, 5] = MIN{[6, 5], [6, 2] + [2, 4] + [4, 5], [6, 2] + [2, 1] + [1, 5]}
    - ∞ > 5 + 2 + 3
    - 10 > 5 + 1 - 1
    - [6, 5] = 5

	1	2	3	4	5	6
1	0	6	∞	8	-1	∞
2	-2	0	∞	2	-3	∞
3	-2	-3	0	-1	2	-8
4	-4	2	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	5	10	7	5	0

Brandon Vo

Third iteration:

- [3, 1] = MIN{[3, 1], [3, 6] + [6, 2] + [2, 4] + [4, 1]}
  - -2 < -8 + 5 + 2 – 4
  - [3, 1] = -5
- [3, 5] = MIN{[3, 5], [3, 6] + [6, 2] + [2, 1] + [1, 5]}
  - 2 > -8 + 5 + 1 – 1
  - [3, 5] = -3
- [6, 5] = MIN{[6, 5], [6, 2] + [2, 4] + [1, 4] + [1, 5]}
  - 5 > 5 + 2 – 4 – 1
  - [6, 5] = 2

	1	2	3	4	5	6
1	0	6	∞	8	-1	∞
2	-2	0	∞	2	-3	∞
3	-5	-3	0	-1	-3	-8
4	-4	2	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	5	10	7	2	0

Fourth iteration

- [3, 5] = MIN{[3, 5], [3, 6] + [6, 2] + [2, 4] + [4, 1] + [1, 5]}
- -3 > -8 + 5 + 2 – 4 – 1
- [3, 5] = -6

No other changes could be found.

	1	2	3	4	5	6
1	0	6	∞	8	-1	∞
2	-2	0	∞	2	-3	∞
3	-5	-3	0	-1	-6	-8
4	-4	2	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	5	10	7	2	0

FASTER-ALL-PAIRS-SHORTEST-PATHS

First iteration uses two pairs:

- [1, 2] = MIN{[1, 2], [1, 5] + [5, 2]}
- [2, 1] = MIN{[2, 1], [2, 4] + [4, 1]}

Brandon Vo

- [2, 5] = MIN{[2, 5], [2, 1] + [1, 5]}
- [3, 1] = MIN{[3, 1], [3, 2] + [2, 1]}
- [3, 2] = MIN{[3, 2], [3, 6] + [6, 2]}
- [3, 4] = MIN{[3, 4], [3, 2] + [4, 2]}
- [4, 2] = MIN{[4, 2], [4, 5] + [5, 2]}
- [4, 5] = MIN{[4, 5], [4, 1] + [1, 5]}
- [5, 1] = MIN{[5, 1], [5, 2] + [2, 1]}
- [5, 4] = MIN{[5, 4], [5, 2] + [2, 4]}
- [6, 1] = MIN{[6, 1], [6, 2] + [2, 1]}
- [6, 4] = MIN{[6, 4], [6, 2] + [2, 4]}

	1	2	3	4	5	6
1	0	6	∞	∞	-1	∞
2	-2	0	∞	2	0	∞
3	3	-3	0	4	∞	-8
4	-4	10	∞	0	-5	∞
5	8	7	∞	9	0	∞
6	6	5	10	7	∞	0

Second iteration: Use 3 pairs

- [1, 4] = MIN{[1, 4], [1, 5] + [5, 2] + [2, 4]}
- [2, 5] = MIN{[2, 5], [2, 4] + [4, 1] + [1, 5]}
- [3, 1] = MIN{[3, 1], [3, 6] + [6, 2] + [2, 1]}
- [3, 4] = MIN{[3, 4], [3, 6] + [6, 2] + [2, 4]
- [3, 5] = MIN{[3, 5], [3, 6] + [6, 2] + [2, 5]}
- [4, 2] = MIN{[4, 2], [4, 1] + [1, 5] + [5, 2]}
- [5, 1] = MIN{[5, 1], [5, 2] + [2, 4] + [4, 1]}
- [6, 1] = MIN{[6, 1], [6, 2] + [2, 4] + [4, 1]}
- [6, 5] = MIN{[6, 5], [6, 2] + [2, 1] + [1, 5]}

	1	2	3	4	5	6
1	0	6	∞	8	-1	∞
2	-2	0	∞	2	-3	∞
3	-5	-3	0	-1	-3	-8
4	-4	2	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	5	10	7	2	0

Third iteration:

Brandon Vo

- [3, 5] = MIN{[3, 5], [3, 6] + [6, 2] + [2, 4] + [4, 1] + [1, 5]}

	1	2	3	4	5	6
1	0	6	∞	8	-1	∞
2	-2	0	∞	2	-3	∞
3	-5	-3	0	-1	-6	-8
4	-4	2	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	5	10	7	2	0

Brandon Vo

3. Problem 25.2-1 in CLRS on page 699.

Run the Floyd-Warshall algorithm on the weighted, directed graph of Figure 25.2. Show the matrix $D^{(k)}$ that results for each iteration of the outer loop.

Beginning iteration

	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	0	∞
3	∞	2	0	∞	∞	-8
4	-4	∞	∞	0	-5	∞
5	∞	7	∞	∞	0	∞
6	∞	5	10	∞	∞	0

- [3, 1] = MIN{[3, 1], 2 + 1}
- [3, 4] = MIN{[3, 4], 2 + 2}
- [5, 1] = MIN{[5, 1], 7 + 1}
- [5, 4] = MIN{[5, 4], 7 + 2}
- [6, 1] = MIN{[6, 1], 5 + 1}
- [6, 4] = MIN{[6, 4], 5 + 2}
- [6, 5] = MIN{[6, 5], 5 + 0}

Second iteration

	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	0	∞
3	3	2	0	4	2	-8
4	-4	∞	∞	0	-5	∞
5	8	7	∞	9	0	∞
6	6	5	10	7	5	0

No changes with the third iteration

Fourth iteration:

- [2, 1] = MIN{[2, 1], 2 – 4}
- [2, 5] = MIN{[2, 5], -2 – 1}
- [5, 1] = MIN{[5, 1], 7 – 2}
- [6, 1] = MIN{[6, 1], 5 – 2}
- [6, 5] = MIN{[6, 5], 5 – 3}

Brandon Vo

	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	-2	0	∞	2	-3	∞
3	0	2	0	4	-1	-8
4	-4	∞	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	5	10	7	2	0

Fifth iteration:

- [1, 2] = MIN{[1, 2], -1 + 7}
- [4, 2] = MIN{[4, 2], -4 + 6}

	1	2	3	4	5	6
1	0	6	∞	8	-1	∞
2	-2	0	∞	2	-3	∞
3	0	2	0	4	-1	-8
4	-4	2	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	5	10	7	2	0

Sixth iteration:

- [3, 1] = MIN{[3, 1], -8 + 3}
- [3, 2] = MIN{[3, 2], -8 + 5}
- [3, 4] = MIN{[3, 2], -8 + 7}
- [3, 5] = MIN{[3, 5], -8 + 2}

	1	2	3	4	5	6
1	0	6	∞	8	-1	∞
2	-2	0	∞	2	-3	∞
3	-5	-3	0	-1	-6	-8
4	-4	2	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	5	10	7	2	0

Brandon Vo

4. Suppose you are given a connected weighted undirected graph, G, with n vertices and m edges, such that the weight of each edge in G is an integer in the interval [1, c], for a fixed constant c > 0. Show how to solve the single-source shortest paths problem, for any given vertex v, in G, in time O(n + m). Hint: Think about how to exploit the fact that the distance from v to any other vertex in G can be at most O(cn) = O(n).

   This question is from Goodrich, Michael T.; Tamassia, Roberto. Algorithm Design and Applications

- With the integers being on a fixed constant, that means there are no negative weight edges in this graph
- We can treat this as a directed graph where every node points away from itself and run BFS search to observe for the optimal path without a loop.
- If we go from a given node to an adjacent node, it will always be the nearest path
- When applying BFS, we make sure the weight is 1
- Make sure the weight of each edge is 1 by converting it into a BFS graph

Starting with the source node, we can start at one node and run BFS when searching for adjacent nodes. Due to our rule for intervals, it means that the path from the source node to its adjacent nodes will be the fastest due to the lack of negative edges.

When searching the adjacent nodes, we can measure the path and reduce the edges to 1. With this approach, we would search all vertices n, giving us O(N) to look at each vertex. We also explore every edge shared by the vertices which go on the interval 1…C. giving us O(c*M) time which is O(M).

In total, this gives us O(N + M) time.

5. Dijkstra's algorithm is not used in graph's which contain negative weights since a negative edge can break the algorithm. A workaround to this is that when a graph contains negative weights, we increase all the edge weights by adding to each edge the absolute value of the lowest edge weight of the graph. This makes all the weights positive. For example, if the weights are $w_1 = -7$, $w_2 = -3$, $w_3 = 2$, we can ensure no edge has a negative weight by adding 7 to each edge. Thus $w_1 = 0$, $w_2 = 4$, $w_3 = 9$. As the weights are non-negative, we can now apply Dijkstra's.

The transformation can be written as, G = (V, E) → G' = (V', E') :

E' = E

V' = V

W'(u, v) = w(u, v) + |m| where m is the smallest weight of any edge in the graph

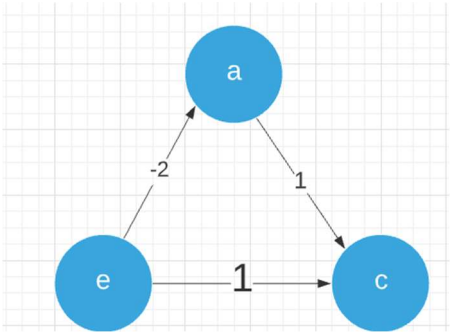
• Does this approach give correct results?

• If yes, what is the complexity of the transformation

• If no, provide reasoning or counter-example why it doesn't work

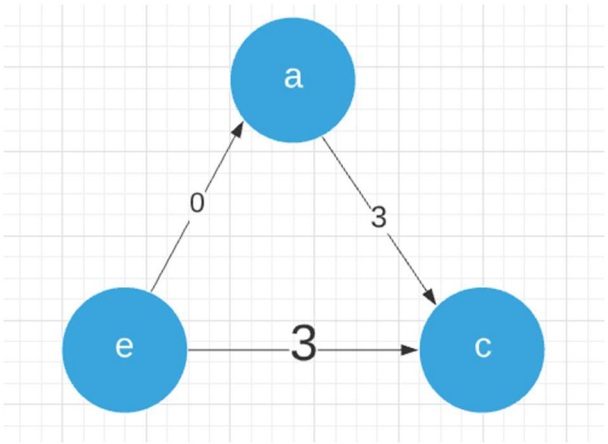
No, this does not work for the same reason why adding a constant to every edge will not lead to the same optimal path.

- When given a path, the constant amount being added will not be uniform because the cost of traveling a path will be dependent on the number of edges are crossing.
- If the smallest path requires a large number of edges to travel through, then the cost increase will be amplified due to all edges being used increasing which may cause the shortest path to change to a path that requires fewer edges.



As shown in the graph above, if we ran any algorithm where negative edges are allowed, it would use (e, a) and (a, c) as its optimal path, giving a cost of -1.

Brandon Vo



If we were to update the graph to increment the negative edges into positive edges, this would affect the entire graph, not just the negative edges. As a result, the graph above would show that (e, c) is the optimal path which would not be the case if the graph kept its negative edge.

Brandon Vo

6. Problem 25.2-6 in CLRS on page 700

How can we use the output of the Floyd-Warshall algorithm to detect the presence of a negative-weight cycle?

Beginning iteration:

	1	2	3	4
1	0	∞	-5	∞
2	1	0	∞	∞
3	∞	∞	0	-3
4	∞	-1	∞	0

First iteration:

	1	2	3	4
1	0	∞	-5	∞
2	1	0	-4	∞
3	∞	-4	0	-3
4	0	-1	∞	0

…

Fourth iteration:

	1	2	3	4
1	-8	-9	-5	-8
2	-1	-8	-4	-7
3	-3	-4	-8	3
4	0	-1	-5	-8

As shown, all nodes n in a negative edge cycle had the directions to themselves decrease by the n^{th} pairs of iteration. With every n^{th} iteration, the cost from every node to itself will always decrease by the same amount, revealing that there is a negative edge cycle.

In a negative edge cycle, the cost of all nodes will go down by a constant amount because with every n pairs used, the same loop will be used to continuously reduce the cost of every node infinitely. The cost for all nodes go down because these nodes are entering the negative edge cycle to reduce their cost before reaching their destination.

We can change the Floyd Warshal algorithm to detect negative edge cycles by checking to see if a node n_{ii} decreases by every n^{th} iteration.

Brandon Vo

7. Suppose you are given a timetable, which consists of the following:

• A set A of n airports, and for each airport a $\in$ A, a minimum connecting time c(a)

• A set F of m flights, and the following, for each flight f $\in$ F:

– Origin airport $a_1(f) \in A$

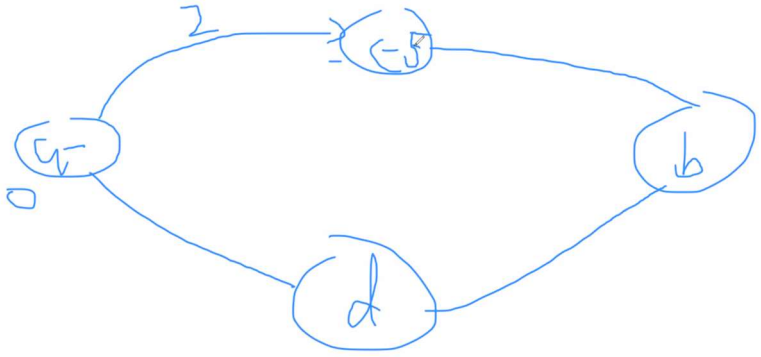
– Destination airport $a_2(f) \in A$

– Departure time $t_1(f)$

– Arrival time $t_2(f)$.

Describe an efficient algorithm for the flight scheduling problem. In this problem, we are given airports a and b, and a time t, and we wish to compute a sequence of flights that allows one to arrive at the earliest possible time in b when departing from a at or after time t. Minimum connecting times at intermediate airports should be observed. What is the running time of your algorithm as a function of n and m?

Problem from Goodrich, Michael T.; Tamassia, Roberto. Algorithm Design and Applications



- Given the arrival and departure time of every flight from an airport to another airport, this would be a graph where every airport is connected by a flight represented as an edge
- Will assume that no cycle will be found for this equation because this flight will never go back to the same airport
- When searching for a nearby airport to move to, the departure time > arrival time
- We can take a greedy approach where we choose the airports based on departure times
    - Our goal would be to find the path that leads to the lowest arrival time to airport b that can be reached.
    - We can search for l_{kj} where k is the intermediate airport with the lowest arrival time.
- As a result, we can use Dijkstra's algorithm where we look at an edge of (u, v) where the weight would be v.arrival_time - u.departure_time
    - If the arrival_time of v is before u's departure time, then this edge is unusable.
- We could use Dijkstra's algorithm and run DFS search from airport a to airport b which would give us O(E log V + V log V) time.

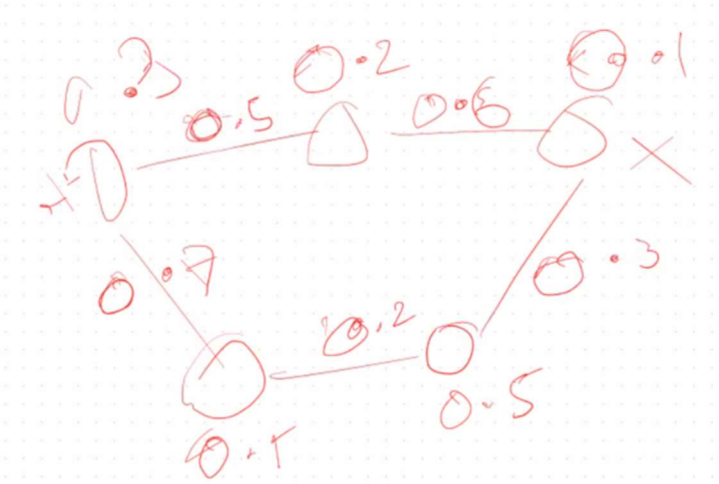
Brandon Vo

```
//Running modified Dijkstra algorithm
FLIGHT-SCHEDULE(G, a, b, n, m)
INIT-SINGLE-SOURCE(G, b)
 S = Ø
 Q = G.V
//Transit time represents the total amount of time it takes to travel from airport a to airport v
Q.transit_time = 0
 WHILE Q ≠ Ø
//Use arrival times as the weights for Q.
   u = EXTRACT-MIN(Q)  //Take the smallest path that can be used to airport B
   S = S ∪ {u}
   for each v ∈ G.Adj[u])
      RELAX(u, v, w)

INIT-SINGLE-SOURCE(G, s)
for each v∈G.V
    v.transit_time = ∞//A has yet to reach this airport
    v.π = NIL
    s.d = 0

RELAX(u, v, w)
//If the destination is earlier than the arrival time
   if v.arrival_time > u.departure_time    //If airport u departs and is able to arrive at airport v
      if v.transit_time > u.transit_time + (v.departure_time – u.arrival_time)
//Minimize connection time
       v.transit_time = u.transit_time + (v.departure_time – u.arrival_time)
       v.π = u
```

Brandon Vo

8. You are working for an ISP providing service in a relatively undeveloped country. A wealthy customer wants a highly reliable and stable connection from their house to the router that connects your network with the rest of the internet (the customer has been getting excessive ping in their online video games). In order to please this valued customer without spending money on expanding the ISP's infrastructure, your boss has tasked you with finding a dedicated path thru the network to route this customer's traffic. You, naturally, want this channel to have only the most reliable links. You have a directed graph, labeled with the chance of a given router (vertex) or link (edge), to go down on a given day. Design an algorithm so that you can quickly compute this and other reliable routes thru the network.

- This problem requires us to use a greedy algorithm where the weight of the edges represents the probability of that connection not failing.
- Need to choose the path that maximizes the probability of a path not failing.
- Because the routes can change and may need to be updated, a dynamic programming solution is required for an adaptable route
- Because the probability of all paths can change, we would have to compute the paths for all nodes from u to j to be able to recompute
- For this graph, all probabilities of failure will be converted into probabilities of not crashing on that day
- Floyd-Warshal's algorithm will be used and changed so that we focus on finding the path from I to J while factoring the probability of not failing for routers and the paths.
- Running Floyd-Warshal's algorithm gives us a time complexity of $O(n^3)$



OPTIMAL_ROUTE(G, n, R, j)

//Where J is the destination of the customer

let $D^{(0)} = W$

Let R[1…n] be the array containing the list of probabilities of a router node on the graph failing

Brandon Vo

for k = 1 to n

R[k] = 1 – R[k] //Convert the probability of failure into the probability of not crashing

let $D^{(k)} = (d^{(k)}_{ij})$ be a new n×n matrix where d_{ij} represents the probability of the route from I to J failing

for i = 1 to n

//Since we have only one destination, we do not need to find the shortest path between all nodes, we only need to find the shortest path to j

For j = 1 to n

//Still must use j despite needing only one route to the customer because we need to find all possible reliable routes

$d^{(k)}_{ij} = \max((1 - d^{(k-1)}_{ij}) * R[i] * R[j], (1 - d^{(k-1)}_{ik}) * (1 - d^{(k-1)}_{kj}) * R[i] * R[k] * R[j])$

//Find the path that has the maximum probability of not failing

//Because the routers have a probability of failing, the routers themselves must be included as well

return $D^{(n)}$

Brandon Vo

9. Think of a good exam question for the topic in lecture 12.

The god-emperor of mankind has hired you, his loyal guardsman to pilot a dreadnought class star ship and use the exterminatus cannons to wipe out a galaxy located on the other side of the Milky Way galaxy. However, your Gellar shields are malfunctioning, so you have to fly your ship to the local cluster in order to wipe them out.

However, this traitorous galaxy is being protected by several nearby planets with varying levels of military defense levels. Some planets have varying levels of military defense power, equipment, and men at their disposal, and as a result, will yield more casualties. What path would you need to take to wipe out your target while maximizing casualties?