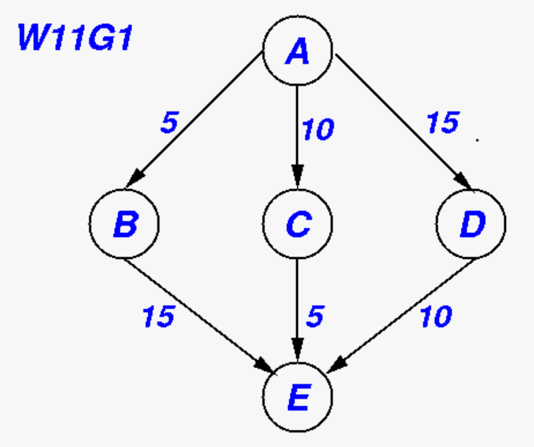
7SENG010W Data Structures and Algorithms
Week 10 Tutorial Exercises
Topics: Graph Shortest Paths using Dijkstra's Shortest Path Algorithm

These exercises cover the *Shortest Path* problem for weighted edge digraphs using Edsger W. Dijkstra's Shortest Path Algorithm. You can use either the Adjacency Matrix or Adjacency Lists representation of a graph developed in the Week 9 Tutorial. However, for this exercise it is recommended that you use Adjacency Lists to represent the graph.

Exercise 2.
Using the following graph ***W11G1***



V = { *A*, *B*, *C*, *D*, *E* }

E = { (*A*, *B*, 5), (*A*, *C*, 10), (A, *D*, 15), (*B*, *E*, 15), (C, *E*, 5), (*D*, *E*, 10) }

Using ***A*** as the *source* vertex ***s*** perform Dijkstra's Shortest Path algorithm on the graph.

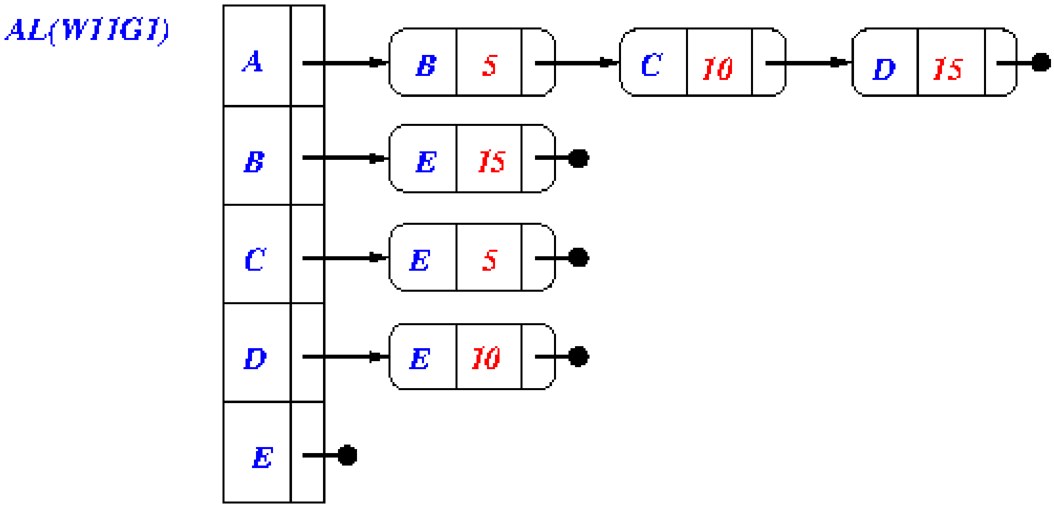
You should produce (pen and paper or an online tool) a "log" of the state of the search as it is performed, i.e. use a table similar to the one used in the lecture notes that records the: `edgeTo[v], pathTo[v], PriQueue` and `nearestVertex`.

After the algorithm has been completed reconstruct the shortest paths from ***A*** to each vertex from the `edgeTo[v]` edges for each vertex.

Compare your answers with other students in the tutorial. If there are any differences try to work why.

Exercise 2 Solution

(1) First construct the adjacency list representation for graph *W11G1*



(2) Now run *Dijkstra's Shortest Path Algorithm* with *A* as the source vertex *s*.

DSP Step 0: Call DijkstraShortestPath(A)

Initialise the data structures by:

- setting all edgeTo[v] to none,
- setting all distTo[v] to ∞,
- create an empty priority queue,

Start search from the start vertex A, by:

- setting its distance from itself to 0 &
- inserting it & its distance into the queue, i.e. (A, 0).

	A	B	C	D	E
edgeTo[v]	-	-	-	-	-
distTo[v]	**0**	∞	∞	∞	∞
PriQueue	< ***(A, 0)*** >				
nearestVertex	***A***				

DSP Step 1: Search from Nearest Vertex A

1) PriQueue equals < ***(A, 0)*** >, so enter WHILE-loop.

2) Get: nearestVertex <-- PriQueue.Dequeue()
   nearestVertex = A;

   then PriQueue = **<>**.

3) Find A's adjacent vertices: B, C & D, using edges: (A, B, 5), (A, C, 10), (A, D, 15).

4) ***Found shorter path to B***, via (A, B, 5):

   distTo(B) > distTo(A) + weight(A, B)
   ∞ > 0 + 5

   Update:
   edgeTo[B] <-- (A, B, 5)
   distTo[B] <-- distTo(A) + weight(A, B) = 0 + 5 = 5

   PriQueue.insert(**(B, 5)**)

5) ***Found shorter path to C***, via (A, C, 10):

   distTo(C) > distTo(A) + weight(A, C)
   ∞ > 0 + 10

   Update:
   edgeTo[C] <-- (A, C, 10)
   distTo[C] <-- distTo(A) + weight(A, C) = 0 + 10 = 10

   PriQueue.insert(**(C, 10)**)

6) ***Found shorter path to D***, via (A, D, 15):

   distTo(D) > distTo(A) + weight(A, D)
   ∞ > 0 + 15

   Update:
   edgeTo[D] <-- (A, D, 15)
   distTo[D] <-- distTo(A) + weight(A, D) = 0 + 15 = 15

   PriQueue.insert(**(D, 15)**)

So found shorter paths for B, C & D.

	A	B	C	D	E
edgeTo[v]	-	*(A, B, 5)*	*(A, C, 10)*	*(A, D, 15)*	-
distTo[v]	0	***5***	***10***	***15***	∞
PriQueue	< *(B, 5), (C, 10), (D, 15)* >				
nearestVertex	*A*				

DSP Step 2: Search from Nearest Vertex B

1) PriQueue equals < ***(B, 5), (C, 10), (D, 15)*** >, so enter WHILE-loop.

2) Get: nearestVertex <-- PriQueue.Dequeue()
   nearestVertex = B;

   then PriQueue = < ***(C, 10), (D, 15)*** >

3) Find B's adjacent vertices: E, using edges: (B, E, 15).

4) ***Found shorter path to E***, via (B, E, 15):

   distTo(E) > distTo(B) + weight(B, E)
   ∞ > 5 + 15 = 20

   Update:
   edgeTo[B] <-- (B, E, 15)
   distTo[B] <-- distTo(B) + weight(B, E) = 5 + 15 = 20

   PriQueue.insert(**(E, 20)**)

So found a shorter path for E of 20 via vertex B.

	A	B	C	D	E
edgeTo[v]	-	***(A, B, 5)***	***(A, C, 10)***	***(A, D, 15)***	*(B, E, 15)*
distTo[v]	0	***5***	***10***	***15***	**20**
PriQueue	< **(C, 10), (D, 15),** *(E, 20)* >				
nearestVertex	*B*				

DSP Step 3: Search from Nearest Vertex C

1) PriQueue equals < ***(C, 10), (D, 15), (E, 20)*** >, so enter WHILE-loop.

2) Get: nearestVertex <-- PriQueue.Dequeue()
   nearestVertex = C;

   then PriQueue = < ***(D, 15), (E, 20)*** >

3) Find C's adjacent vertices: E, using edges: (C, E, 5).

4) ***Found shorter path to E***, via (C, E, 5):

   distTo(E) > distTo(C) + weight(C, E)
   20 > 10 + 5 = 15

   Update:
   edgeTo[E] <-- (C, E, 5)
   distTo[E] <-- distTo(C) + weight(C, E) = 10 + 5 = 15

   PriQueue.insert(**(E, 15)**)

So found a new shorter path for E of 15 via C, which is shorter than the previous shortest path for E of 20 via B.

	A	*B*	*C*	*D*	*E*
edgeTo[v]	-	***(A, B, 5)***	***(A, C, 10)***	***(A, D, 15)***	***(C, E, 5)***
distTo[v]	0	***5***	***10***	***15***	**15**
PriQueue	< ***(D, 15)***, ***(E, 15)*** >				
nearestVertex	*C*				

DSP Step 4: Search from Nearest Vertex D

1) PriQueue equals < ***(D, 15), (E, 15)*** >, so enter WHILE-loop.

2) Get: nearestVertex <-- PriQueue.Dequeue()
   nearestVertex = D;

   then PriQueue = < ***(E, 15)*** >

3) Find D's adjacent vertices: E, using edge: (D, E, 10).

4) ***Have not found a shorter path to E***, via (D, E, 10):

   distTo(E) > distTo(D) + weight(D, E)
   15 > 15 + 10 = 25

   So E's shortest path does not need to be updated or inserted into the queue.

So only found a new path for E of 25 via D, which is longer than the previous shortest path for E of 15 via C.

	A	*B*	*C*	*D*	*E*
edgeTo[v]	-	***(A, B, 5)***	***(A, C, 10)***	***(A, D, 15)***	***(C, E, 5)***
distTo[v]	0	***5***	***10***	***15***	**15**
PriQueue	< ***(E, 15)*** >				
nearestVertex	*D*				

DSP Step 5: Search from Nearest Vertex E

1) PriQueue equals < ***(E, 15)*** >, so enter WHILE-loop.

2) Get: nearestVertex <-- PriQueue.Dequeue()
   nearestVertex = E;

   then PriQueue = **< >**

3) Find E's adjacent vertices: **there are none.**

So nothing to update & the queue is empty so the while-loop terminates & the algorithm terminates. The final state is:

	A	*B*	*C*	*D*	*E*
edgeTo[v]	-	***(A, B, 5)***	***(A, C, 10)***	***(A, D, 15)***	***(C, E, 5)***
distTo[v]	0	***5***	***10***	***15***	**15**
PriQueue	< >				
nearestVertex	*E*				

DSP Step 6: Final Shortest Paths from A

The final state of the shortest path edges, distances & paths from *A* to all the other vertices is:

Vertex	edgeTo**[v]**	**distTo[v]**	**Path Edges**	**Path**
A	-	*0*	< >	< *A* >
B	*(A, B, 5)*	*5*	< *(A, B, 5)* >	< *A, B* >
C	*(A, C, 10)*	*10*	< *(A, C, 10)* >	< *A, C* >
D	*(A, D, 15)*	*15*	< *(A, D, 15)* >	< *A, D* >
E	*(C, E, 5)*	*15*	< *(A, C, 10), (C, E, 5)* >	< *A, C, E* >