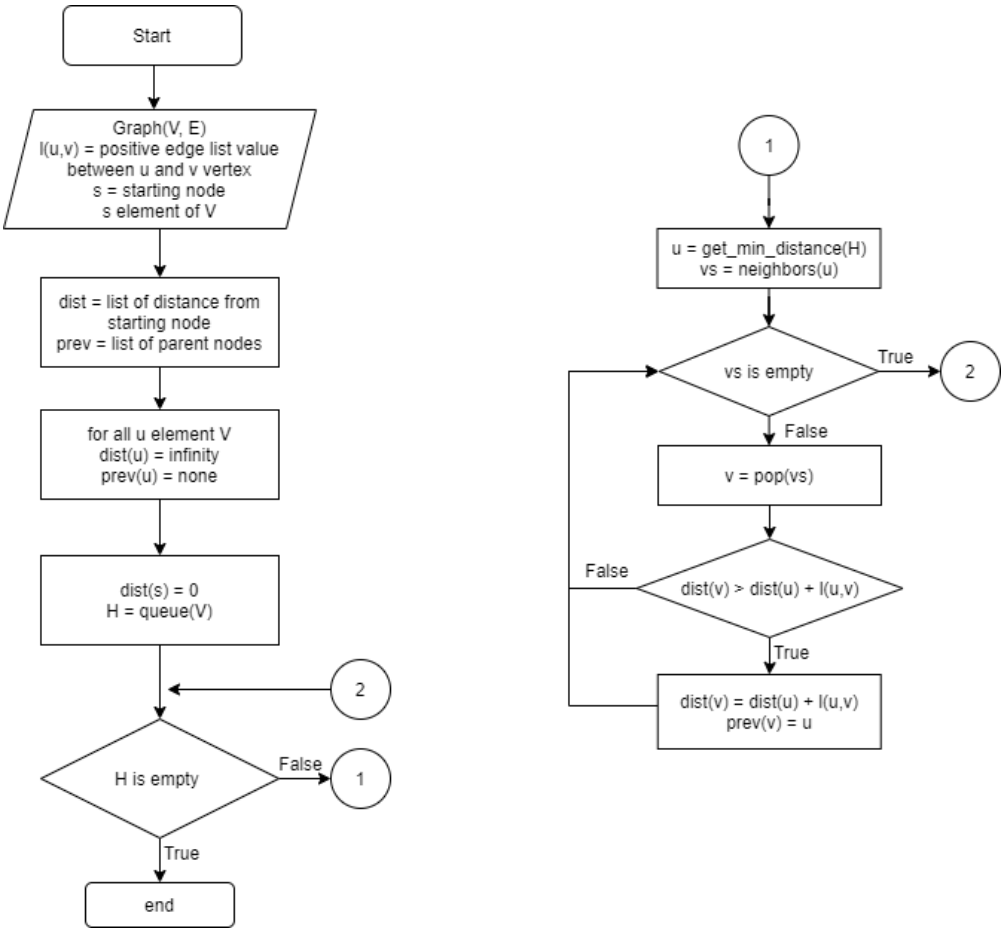
Assignment – Week 5 & 6

Azhar Harisandi

22320011

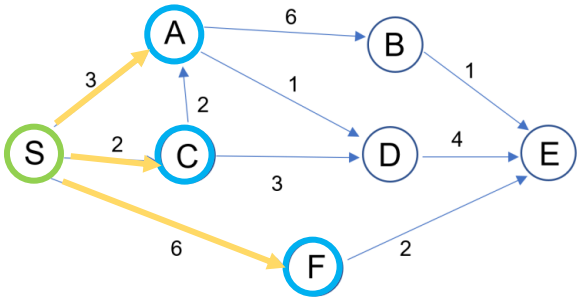
1. Dijkstra's Algorithm



First we initialize all distances with infinity, and starting node distance as 0, then store none for each parent node

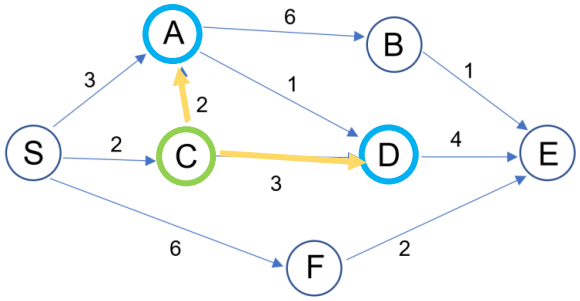
Node	Status	Shortest distance from Source	Parent Node
S	Nonactive	~	None
A	Nonactive	~	None
B	Nonactive	~	None
C	Nonactive	~	None
D	Nonactive	~	None
E	Nonactive	~	None
F	Nonactive	~	None

Choose starting node (S), and explore its adjacent nodes. If the distance to starting node is smaller to its current stored distance, update its distance and assign current node as parent node for currently active nodes.



Node	Status	Shortest distance from Source	Parent Node
S	Current	0	None
A	Active	3	S
B	Nonactive	~	None
C	Active	2	S
D	Nonactive	~	None
E	Nonactive	~	None
F	Active	6	S

Next step : choose the closest node to the source (C) and choose that node as the new current node, then again explore its adjacent nodes, then mark the previous node as done so we don't have to explore it again. Calculate the cumulative distance of nodes that are adjacent to current node with respect to the starting node by summing previous distance accumulated in the current node to its adjacent distance

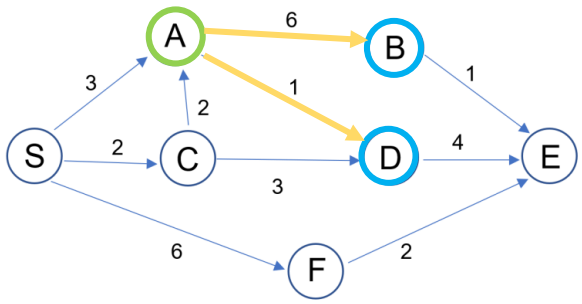


Node	Status	Shortest distance from Source	Parent Node
S	Done	0	None
A	Active	prev : 3, cur : 4	prev : S, cur : C
B	Nonactive	~	None
C	Current	2	S
D	Active	2 + 3 = 5	C
E	Nonactive	~	None
F	Nonactive	6	S

With C as the current node, distance of node A from starting point will be 2 + 2 = 4, but A already has a distance value from previous step. If the new distance is larger, do not update the distance and parent node, otherwise update the distance and parent node.

Node	Status	Shortest distance from Source	Parent Node
S	Done	0	None
A	Active	3 (4)	S
B	Nonactive	~	None
C	Current	2	S
D	Active	5	C
E	Nonactive	~	None
F	Nonactive	6	S

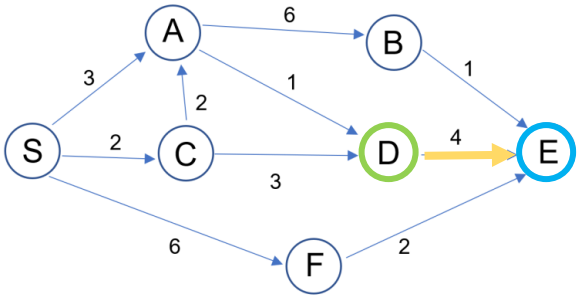
Next, choose the minimum cumulative distance from starting node to current node to each of its adjacent node. We will choose A since the distance of A is 4 and the distance of D is 5 (A < D). Then start exploring A's neighbors



Node	Status	Shortest distance from Source	Parent Node
S	Done	0	None
A	Current	3	S
B	Active	3+6 = 9	None
C	Done	2	S
D	Active	prev : 5, cur : 3 + 1	prev : C, cur : A
E	Nonactive	~	None
F	Nonactive	6	S

Node	Status	Shortest distance from Source	Parent Node
S	Done	0	None
A	Current	3	S
B	Active	9	A
C	Done	2	S
D	Active	4	A
E	Nonactive	~	None
F	Nonactive	6	S

Next choose D as the current node since cumulative distance from starting node to current node and to D node is the shortest, and mark A with Done, then start exploring D.



Node	Status	Shortest distance from Source	Parent Node
S	Done	0	None
A	Done	3	S
B	Nonactive	9	A
C	Done	2	S
D	Current	4	A
E	Active	8	D
F	Nonactive	6	S

Now we already have the shortest distance from every node to starting node. To reconstruct the path, we can simply go to our target node, and go back to its parent and repeat until we get to the starting node.

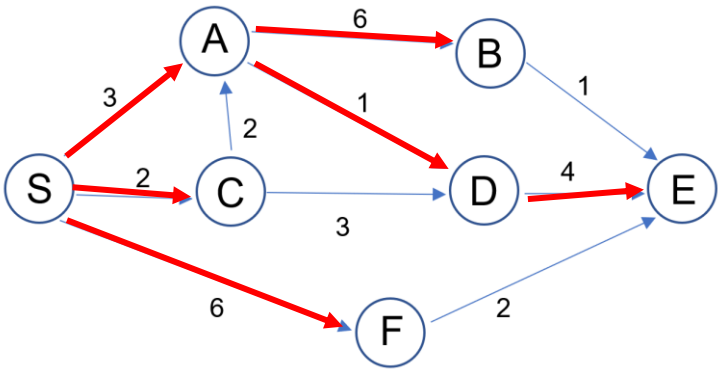
For example, path from start to B is

reverse(B -> A -> S)

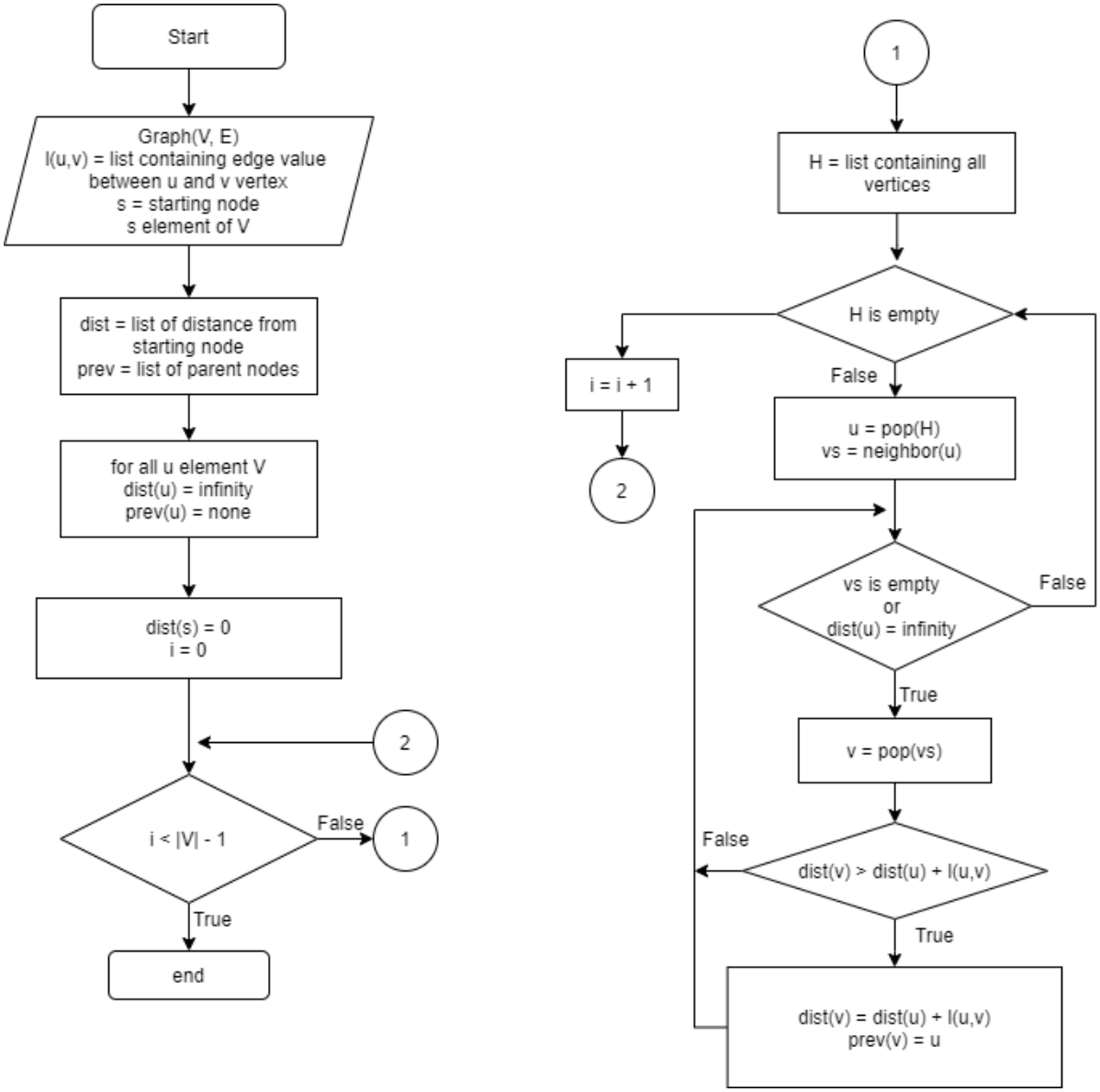
path from S to E is

reverse(E -> D -> A -> S)

Minimum distance tree from start node to all vertices.



2. Bellman-Ford Algorithm



Initialize all node distance as infinity, and parents as none. Then change the starting node distance as 0

Node	Status	Shortest distance from Source	Parent Node
S	Nonactive	0	None
A	Nonactive	~	None
B	Nonactive	~	None
C	Nonactive	~	None
D	Nonactive	~	None
E	Nonactive	~	None
F	Nonactive	~	None

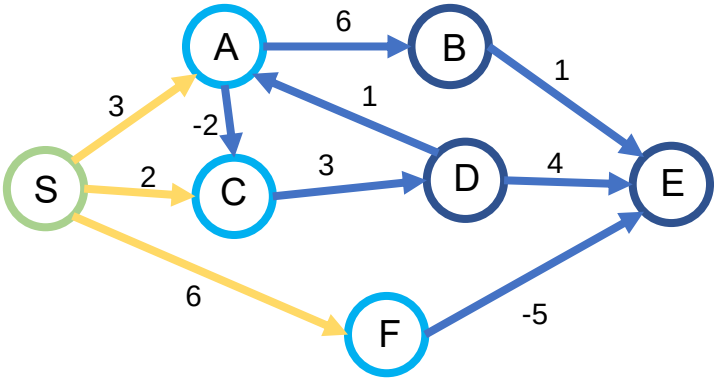
Iteration 1

V = [S, A, B, C, D, E, F]

Take one element from V as the current node

Current = S

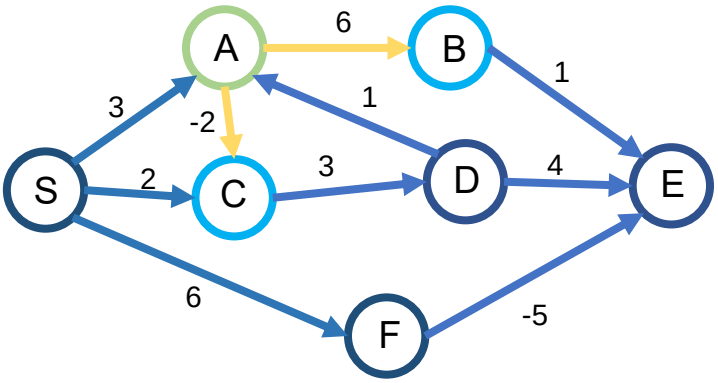
V = [A, B, C, D, E, F]



Node	Status	Shortest distance from Source	Parent Node
S	Current	0	None
A	Active	3	S
B	Nonactive	~	None
C	Active	2	S
D	Nonactive	~	None
E	Nonactive	~	None
F	Active	6	S

Current = A

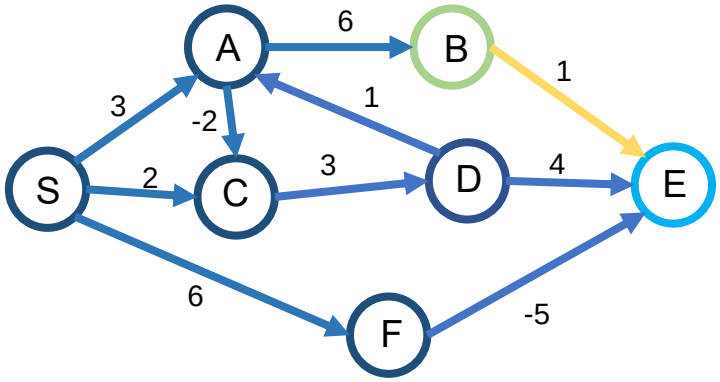
V = [B, C, D, E, F]



Node	Status	Shortest distance from Source	Parent Node
S	Nonactive	0	None
A	Current	3	S
B	Active	9	A
C	Active	1	A
D	Nonactive	~	None
E	Nonactive	~	None
F	Nonactive	6	S

Current = B

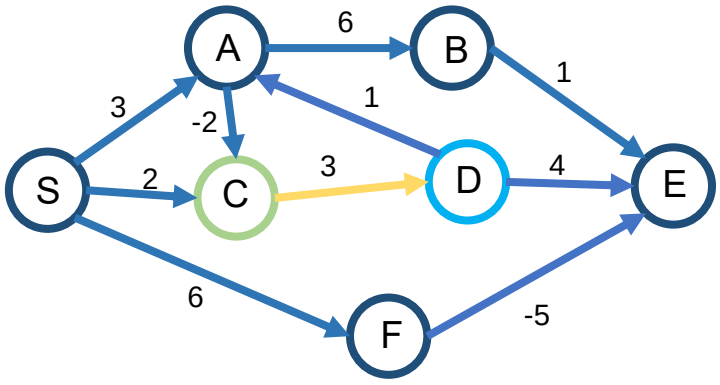
V = [C, D, E, F]



Node	Status	Shortest distance from Source	Parent Node
S	Nonactive	0	None
A	Nonactive	3	S
B	Current	9	A
C	Nonactive	1	A
D	Nonactive	~	None
E	Active	10	B
F	Nonactive	6	S

Current = C

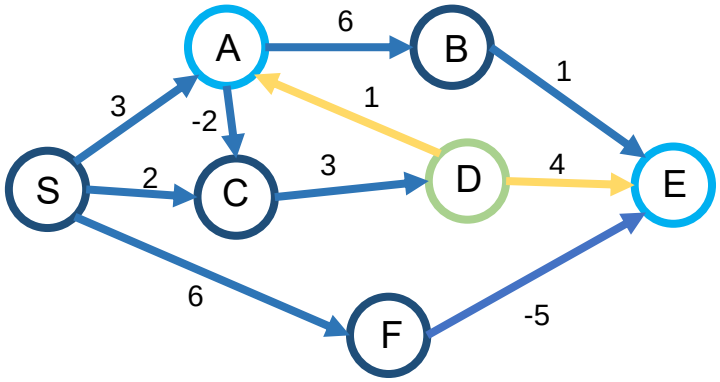
V = [D, E, F]



Node	Status	Shortest distance from Source	Parent Node
S	Nonactive	0	None
A	Nonactive	3	S
B	Nonactive	9	A
C	Current	1	A
D	Active	4	C
E	Nonactive	10	B
F	Nonactive	6	S

Current = D

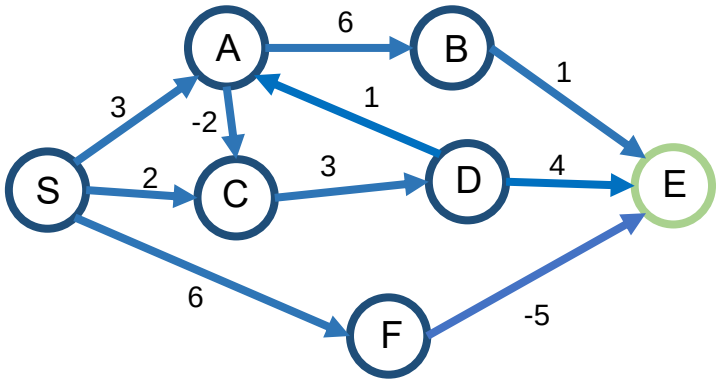
V = [E, F]



Node	Status	Shortest distance from Source	Parent Node
S	Nonactive	0	None
A	Active	3	S
B	Nonactive	9	A
C	Nonactive	1	A
D	Current	4	C
E	Active	8	D
F	Nonactive	6	S

Current = E

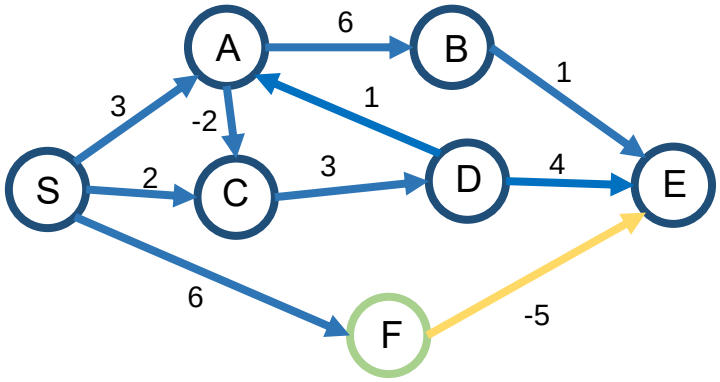
V = [F]



Node	Status	Shortest distance from Source	Parent Node
S	Nonactive	0	None
A	Nonactive	3	S
B	Nonactive	9	A
C	Nonactive	1	A
D	Nonactive	4	C
E	Current	8	D
F	Nonactive	6	S

Current = F

V = []



Node	Status	Shortest distance from Source	Parent Node
S	Nonactive	0	None
A	Nonactive	3	S
B	Nonactive	9	A
C	Nonactive	1	A
D	Nonactive	4	C
E	Active	1	F
F	Current	6	S

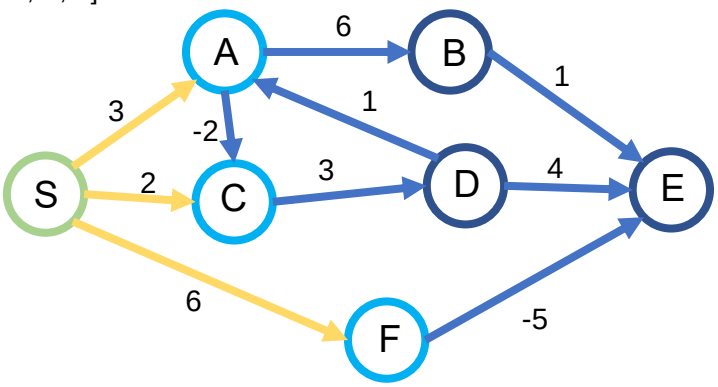
Iteration 2

V = [S, A, B, C, D, E, F]

Take one element from V as the current node, we use distance table from previous iteration.

Current = S

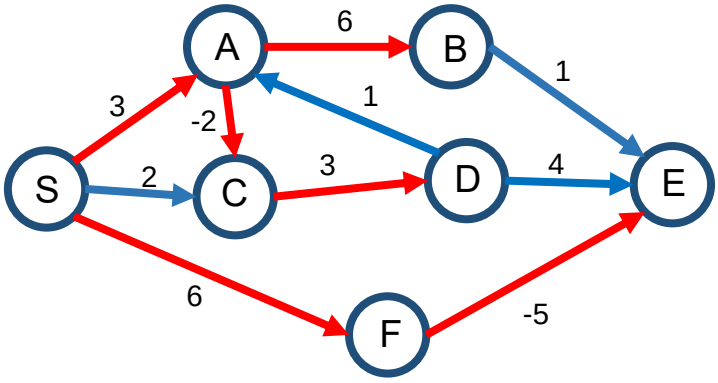
V = [A, B, C, D, E, F]



Node	Status	Shortest distance from Source	Parent Node
S	Current	0	None
A	Active	3	S
B	Nonactive	9	A
C	Active	1	A
D	Nonactive	4	C
E	Nonactive	1	F
F	Active	6	S

We can see that nothing gets updated in the beginning of 2nd iteration. If we no longer get update for the distance, we can break the loop and we already have the shortest distance from starting node to each node in the graph.

Minimum distance tree from starting node to each node :



3. A* Algorithm Heuristic function comparison