CSC 226
Zach Serviss
V00950002

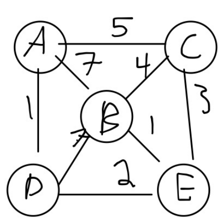
Assignment 3

1.a. (A,D,1),(D,E,2),(E,B,1),(E,C,3)

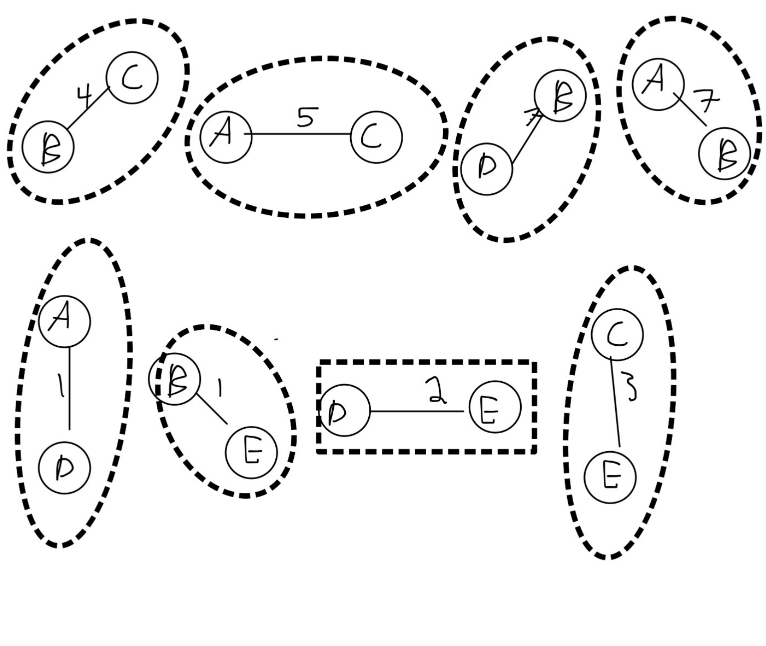
	A	B	C	D	E
A	(Start)0	7	5	1	∞
D	0	7	5	1	2
E	0	1	3	1	2
B	0	1	3	1	2

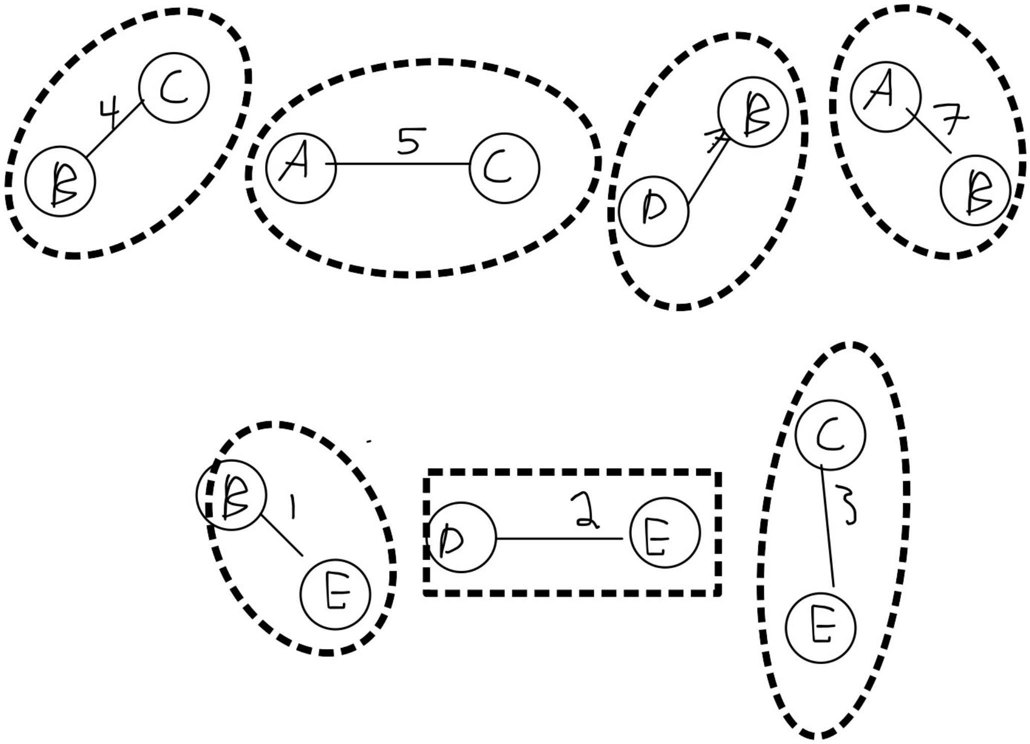
From the table above we can see that from A we we add A->D as that as that edge as a weight of 1, the shortest. Once we have the vertex D in our forest, we can add E to visible edges and give it the correct weight from D->E which is 2. From E we can update the value of B as it has the weight of 1 from E->B, we also update the value of C as E->C has a weight of 3. From here no more vertex values are updated and we just keep going through grabbing the minimum edge weight from the table of values not yet added to the tree (highlighted in green).

b.i.(A,D,1),(E,B,1),(D,E,2),(E,C,3)
ii .

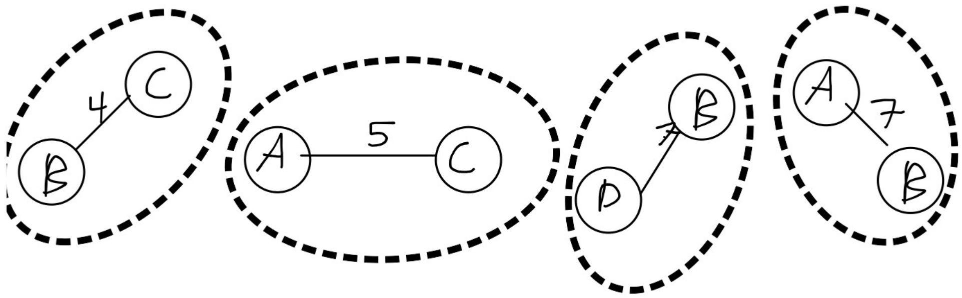


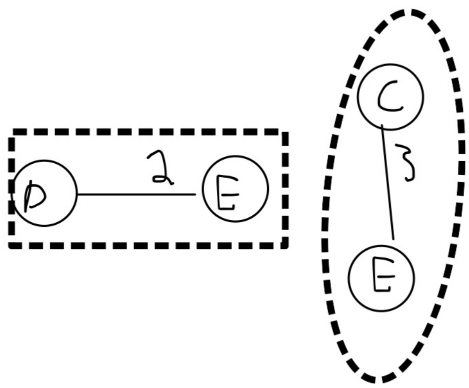
From the original graph, create the list of edges in order from smallest to largest

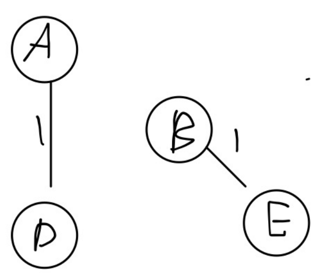




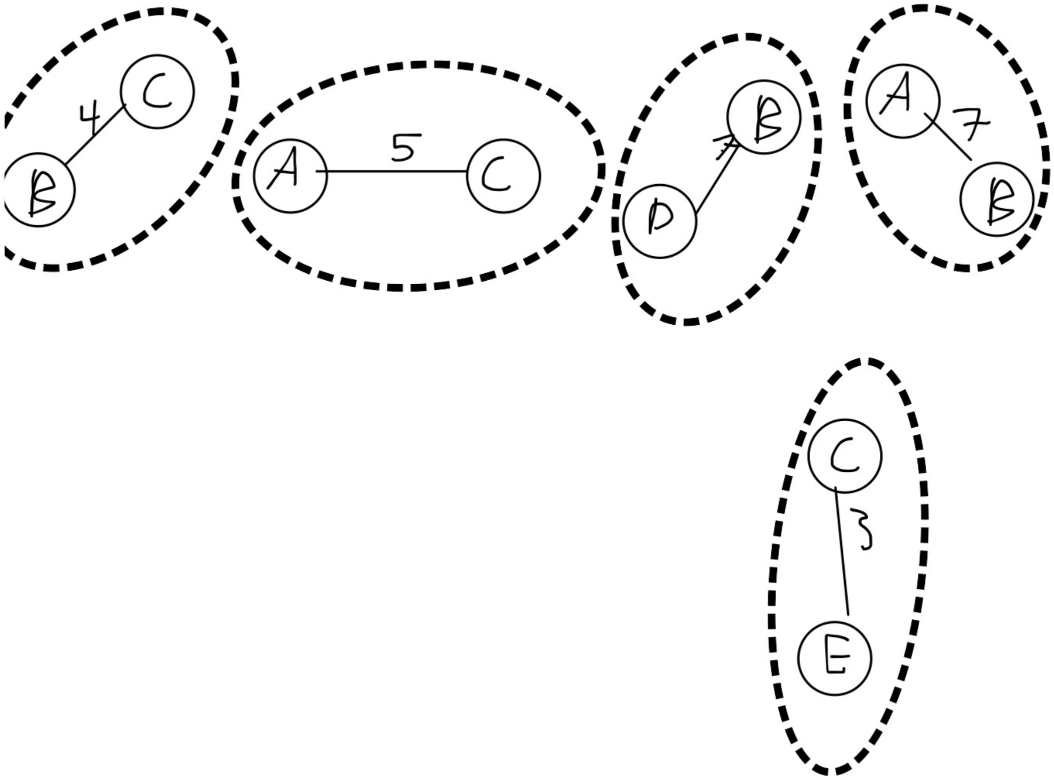
We then start the algorithm and bring down the lowest weighted edge and union it with the tree (initially the tree is empty.

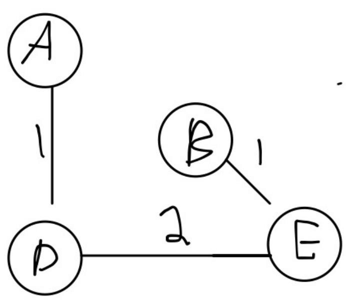




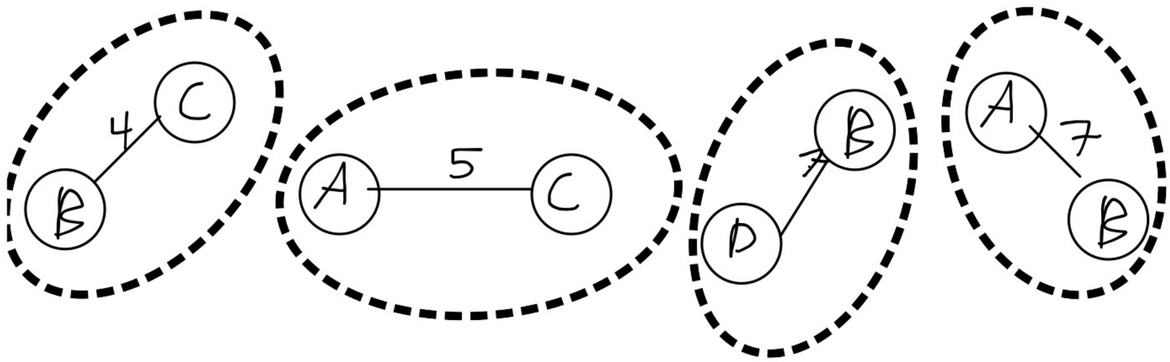


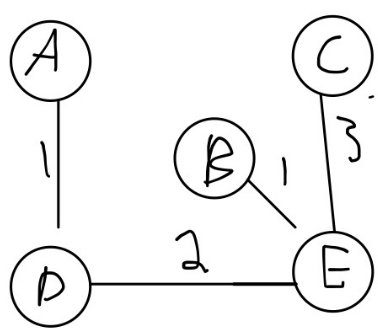
Second iteration.





Third iteration we already have nodes D and E so we union the new edge with existing vertices





On the fourth iteration we have reached our minimum spanning tree. Adding anymore edges will create a cycle so all other edges will not be included in the tree

2.Yes both algorithms will work correctly with negative edges. The algorithms are simply comparing values within the graph, meaning the algorithms really only care about the difference between values, not the actual values themselves. They will always find the correct MST for a given graph.

3. By using bottom up heap to sort the edge weights, we will get a run time of O(E) time where E is the number of edges. Then using Kruskals algorithm with the sorted edges, we will run in O((V + E) log V) which can be simplified to O(V log V) where E= edges and V= vertex. Adding the time to sort the edges will bring our final run time to O(E + V log V)

4. Edges added to the cloud are highlighted in cyan. Value changes are highlighted in green.

	Init									
A	∞	2								
B	∞	∞	3	3						
C	∞	4	3	3	3					
D	∞	∞	∞	∞	8	6	6			
E	∞	∞	∞	∞	∞	∞	∞	17	17	17
F	∞	9	9	5	5	5				
G	0									
H	∞	2	2							
I	∞	∞	∞	∞	∞	∞	∞	∞	10	
J	∞	7	7	7	7	7	7	7		

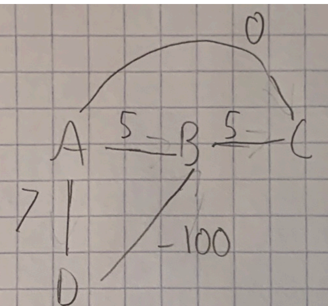
5.Below is the table representing the vertices and their values at the end of the first for loop in the Bellman-Ford algorithm

	Init	1	2	3	4	5	6	7	8	9
A	∞	11	11	11	11	11	11	11	11	11
B	0	0	0	0	0	0	0	0	0	0
C	∞	4	4	4	4	4	4	4	4	4
D	∞	4	4	4	4	4	4	4	4	4
E	∞	7	6	6	6	6	6	6	6	6
F	∞	∞	2	2	2	2	2	2	2	2
G	∞	∞	5	5	5	5	5	5	5	5
H	∞	∞	7	1	1	1	1	1	1	1
I	∞	∞	4	-2	-2	-2	-2	-2	-2	-2
V	∞	3	3	3	3	3	3	3	3	3

This is the table representing the vertex value changes while in the loop going in alphabetical order. Vertex values can change multiple times in this algorithm.

	i=1	I=2	I=3	I=4
(B->A)	A=11	NO CHANGE	NO CHANGE	NO CHANGE
(B->V)	V=3	NO CHANGE	NO CHANGE	NO CHANGE
(C->B)	∞	NO CHANGE	NO CHANGE	NO CHANGE
(C->D)	∞	NO CHANGE	NO CHANGE	NO CHANGE
(D->E)	∞	E=6	NO CHANGE	NO CHANGE
(D->G)	∞	NO CHANGE	NO CHANGE	NO CHANGE
(E->F)	∞	F=8	NO CHANGE	NO CHANGE
(E->G)	∞	G=5	NO CHANGE	NO CHANGE
(E->H)	∞	H=13	NO CHANGE	NO CHANGE
(F->I)	∞	I=4	I=-2	NO CHANGE
(G->F)	∞	F=2	NO CHANGE	NO CHANGE
(I->H)	∞	H=7	H=1	NO CHANGE
(V->C)	C=4	NO CHANGE	NO CHANGE	NO CHANGE
(V->D)	D=4	NO CHANGE	NO CHANGE	NO CHANGE
(V->E)	E=7	NO CHANGE	NO CHANGE	NO CHANGE

6.



Start @ A: (A->B, 5), (A->C, 0), (A->D, 7)

A then pull C into the cloud, then B, no change

then when finally pulling in D, A->B changes to (A->B, -93)

A->C is still 0 when the shortest path is actually -88