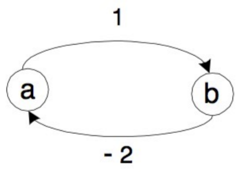
DAA Assignment V

1.

A. Give an example of a digraph with negative weights for which Floyd's algorithm does not yield the correct results.



Floyd's algorithm will yield:

$$D^{(0)} = \begin{bmatrix} 0 & 1 \\ -2 & 0 \end{bmatrix} \qquad D^{(1)} = \begin{bmatrix} 0 & 1 \\ -2 & -1 \end{bmatrix} \qquad D^{(2)} = \begin{bmatrix} -1 & 0 \\ -3 & -2 \end{bmatrix}$$

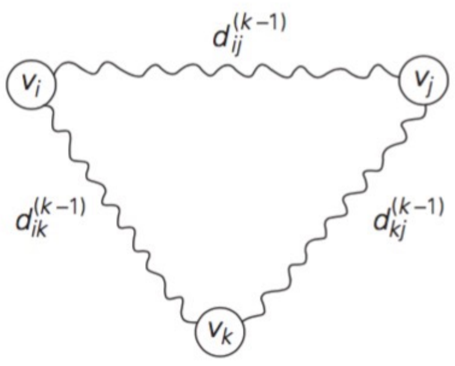
None of the four elements of the last matrix gives the correct value of the shortest path, which is, in fact, $-\infty$ because repeating the cycle enough times makes the length of a path arbitrarily small.

B. Considering an abstract example with three nodes V_i, V_j and V_k discuss in detail how Floyd's algorithm works.

We compute all the elements of each matrix $D^{(k)}$ from its immediate predecessor $D^{(k-1)}$ in series.

Let $d_{ij}^{(k)}$ be the element in ith row and jth column.

That means, $d_{ij}^{(k)}$ = length of the shortest path among all paths from the ith vertex v_i to the jth vertex v_j with their intermediate vertices numbered not higher than k.



All paths are made up of paths from v_i to v_k with each intermediate vertex not higher than k - 1 and a path from v_k to v_j with each intermediate vertex not higher than k - 1.

That gives the minimum length between two vertices v_i and v_j,

$$d_{ij}^{(k)} = \text{MIN}\,(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) \qquad \text{for } k \geq 1$$

C. Solve the all-pairs shortest path problem for the digraph with the following weight matrix:

$$\begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix}$$

$$D^{(0)} = \begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix} \qquad D^{(1)} = \begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \mathbf{14} \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \mathbf{5} & \infty & \mathbf{4} & 0 \end{bmatrix}$$

$$D^{(2)} = \begin{bmatrix} 0 & 2 & \mathbf{5} & 1 & 8 \\ 6 & 0 & 3 & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 5 & \mathbf{8} & 4 & 0 \end{bmatrix} \qquad D^{(3)} = \begin{bmatrix} 0 & 2 & 5 & 1 & 8 \\ 6 & 0 & 3 & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 5 & 8 & 4 & 0 \end{bmatrix}$$

$$D^{(4)} = \begin{bmatrix} 0 & 2 & \mathbf{3} & 1 & \mathbf{4} \\ 6 & 0 & 3 & 2 & \mathbf{5} \\ \infty & \infty & 0 & 4 & \mathbf{7} \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 5 & \mathbf{6} & 4 & 0 \end{bmatrix} \qquad D^{(5)} = \begin{bmatrix} 0 & 2 & 3 & 1 & 4 \\ 6 & 0 & 3 & 2 & 5 \\ \mathbf{10} & \mathbf{12} & 0 & 4 & 7 \\ \mathbf{6} & \mathbf{8} & 2 & 0 & 3 \\ 3 & 5 & 6 & 4 & 0 \end{bmatrix} = D$$

2. **Apply the bottom-up dynamic programming algorithm to the following instance of the knapsack problem and find the optimal subset. Capacity W=6. (Neatly show all the steps).**

Item	Weight	Value
1	3	25
2	2	20
3	1	15
4	4	40
5	5	50

ITEMS

CAPACITY	0 (0, 0)	1 (3, 25)	2 (2, 20)	3 (1, 15)	4 (4, 40)	5 (5, 50)
0	0	0	0	0	0	0
1	0	0	0	15	15	15
2	0	0	20	20	20	20
3	0	25	25	35	35	35
4	0	25	25	40	40	40
5	0	25	45	45	55	55
6	0	25	45	60	60	65

Maximum value of the optimal subset is 65, taking items **3** and **5**.

3.

A. **With a neat drawing, explain the correctness proof of Prim's algorithm. For the inductive step, use proof by contradiction to prove that the sub tree Ti generated by Prim's algorithm is a sub graph of some minimum spanning tree.**

T_0 consists of a single vertex and hence must be a part of any minimum spanning tree.

Let us assume T_{i-1} is is part of some minimum spanning tree T.

To prove : T_i generated from T_{i-1} by Prim's algorithm, is also a part of a minimum spanning tree.

Let no minimum spanning tree of the graph can contain T_i.

Let e_i = (v, u) be the minimum weight edge from a vertex T_{i-1} to a vertex not in T_{i-1} used by Prim's algorithm to expand T_{i-1} to T_i.

By our assumption e_i cannot belong to any minimum spanning tree, including T, hence if we add e_i to T, a cycle will be formed.

This cycle must contain another edge (v', u') connecting a vertex $v' \in T_{i-1}$ to a vertex u' that is not in T_{i-1}.

If we now delete the edge (v', u') from this cycle, we will obtain another spanning tree of the entire graph whose weight is less than or equal to the weight of T since the weight of e_i is less than or equal to the weight of (v', u').

Hence, this spanning tree is a minimum spanning tree, which contradicts the assumption that no minimum spanning tree contains T_i. This completes the correctness proof of Prim's algorithm.

B. **Explain greedy technique illustrating the property of choices made at each step.**

A **greedy algorithm** is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.

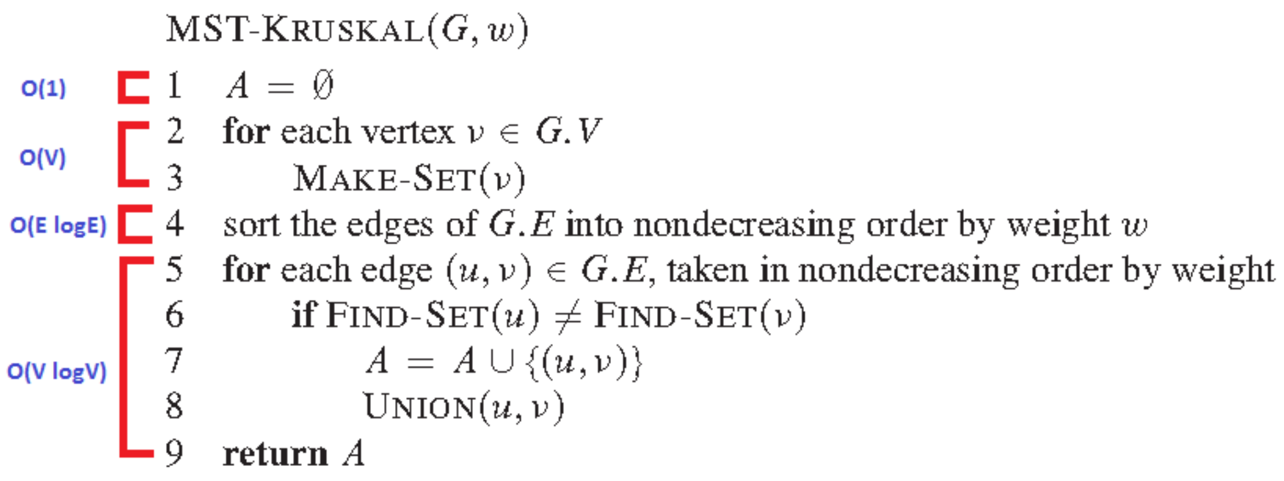
In general, greedy algorithms have five components: (Optional)

- A candidate set, from which a solution is created
- A selection function, which chooses the best candidate to be added to the solution
- A feasibility function, that is used to determine if a candidate can be used to contribute to a solution
- An objective function, which assigns a value to a solution, or a partial solution, and
- A solution function, which will indicate when we have discovered a complete solution

On each step the choice made must be:

- *feasible*, i.e., it has to satisfy the problem's constraints.
- *locally optimal*, i.e., it has to be the best local choice among all feasible choices available on that step.
- *irrevocable*, i.e., once made, it cannot be changed on subsequent steps of the algorithm.

C. Analyze the efficiency of Kruskal's algorithm.



```
MST-KRUSKAL($G, w$)
1  $A = \emptyset$
2  for each vertex $v \in G.V$
3      MAKE-SET($v$)
4  sort the edges of $G.E$ into nondecreasing order by weight $w$
5  for each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
6      if FIND-SET($u$) $\neq$ FIND-SET($v$)
7          $A = A \cup \{(u, v)\}$
8          UNION($u, v$)
9  return $A$
```

O(1)
O(V)
O(E logE)
O(V logV)

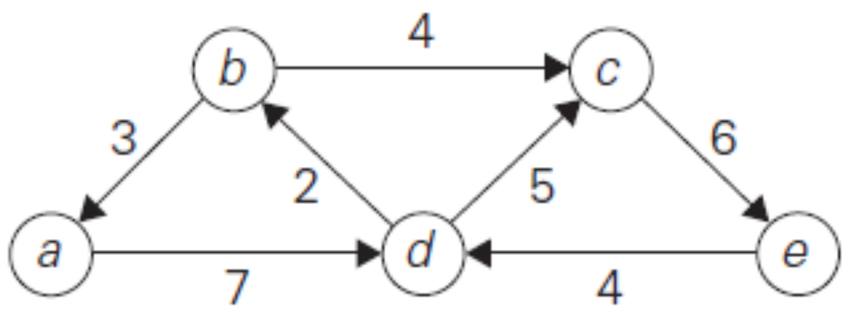
```
T(n) = O(1) + O(V) + O(E log E) + O(V log V)
     = O(E log E) + O(V log V)

as |E| >= |V| - 1

T(n) = E log E + E log E
     = E log E
```

4.

A. **Using Dijkstra's algorithm solve the following instance of single source shortest path problem considering 'a' as the source vertex:**



Tree vertices	Remaining vertices
a(-,0)	b(-,∞) c(−,∞) **d(a,7)** e(-,∞)
d(a,7)	**b(d,7+2)** c(d,7+5) e(-,∞)
b(d,9)	**c(d,12)** e(-,∞)
c(d,12)	**e(c,12+6)**
e(c,18)	

The shortest paths (identified by following nonnumeric labels backwards from a destination vertex to the source) and their lengths are:

from a to d:	$a - d$	of length 7
from a to b:	$a - d - b$	of length 9
from a to c:	$a - d - c$	of length 12
from a to e:	$a - d - c - e$	of length 18

B. **If the given graph is a complete graph then which graph representation (weight matrix or adjacency list) is more suitable to implement Dijkstra's algorithm? Justify your answer. (Assume that most suitable priority queue is used)**

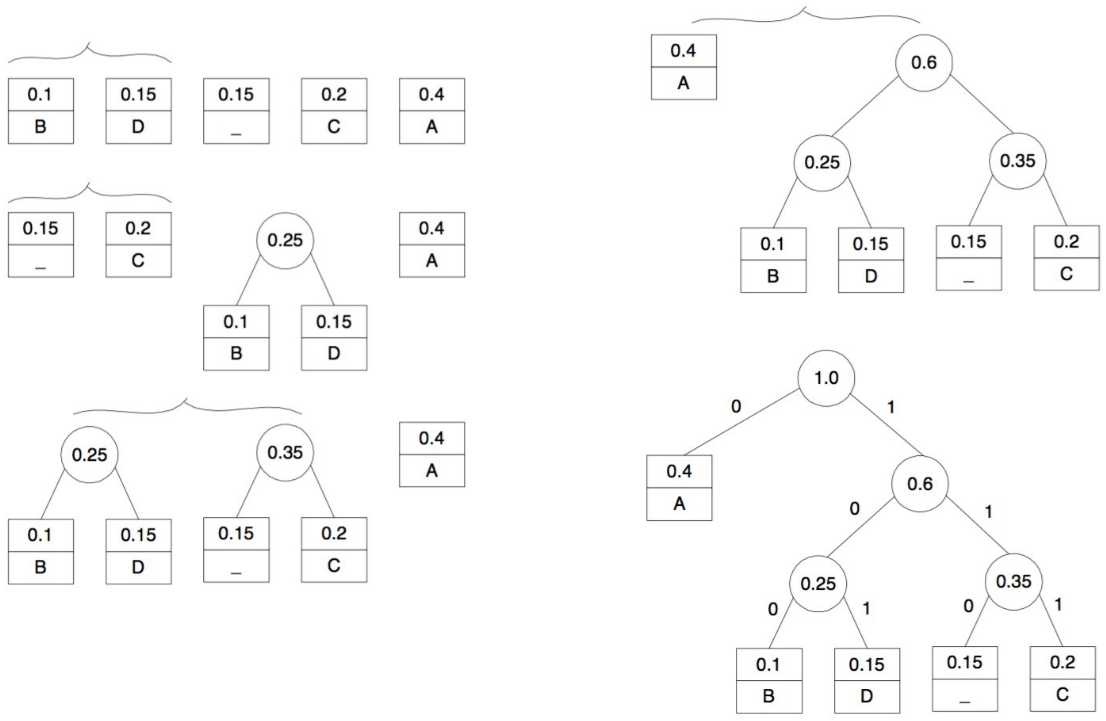
Dijkstra's algorithm finds the shortest paths from one source to all other nodes in the graph, for weighted graphs. It operates on dense graphs (matrix representation) in $O(V^2)$ time, and on sparse graphs (adjacency list representation) in O(E log(V)) time.

Matrix representation would be more suitable to implement since for dense graphs, cause in terms of running time, Adjacency Matrix would almost always outperform lists. The List implementation would use less memory (proportional to number of edges) to store the Graph, for dense graphs both representations would use the same amount of memory.

5.

A. Construct a Huffman code for the following data:

Character	A	B	C	D	_
Probability	0.4	0.1	0.2	0.15	0.15



Character	A	B	C	D	_
Probability	0.4	0.1	0.2	0.15	0.15
Codeword	0	100	111	101	110

B. Encode the text ABACABAD using the code of question 5(A).

ABACABAD = 0100011101000101

C. Decode the text whose encoding is 100010111001010 in the code of question 5(A).

100 – 0 – 101 – 110 – 0 – 101 – 0

B – A – D – _ – A – D – A

= B A D _ A D A