CSYE 7374 – Autonomous Learning in Games
Practice Problems & Solutions

Student Name: ____
Professor: Nik Bear Brown

This document to intended to present some examples of how to specify solutions for the classical game AI assignment.

Q Give a brief definition for the following:

i. Connected Component
ii. Strong Connectivity

Solution:

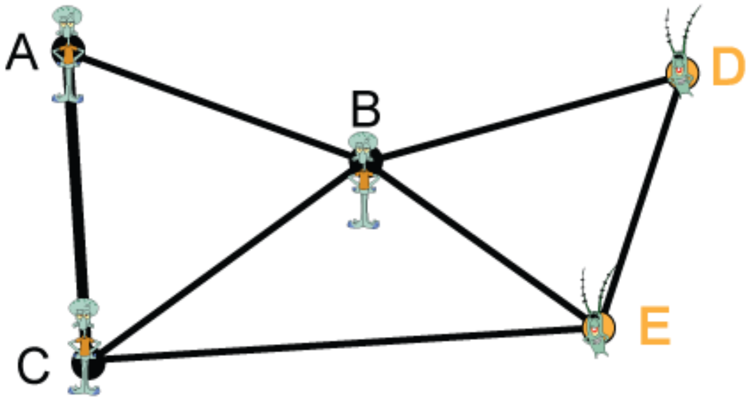
i. Connected Component

“In graph theory, a connected component of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph.” - Wikipedia

ii. Strong connectivity

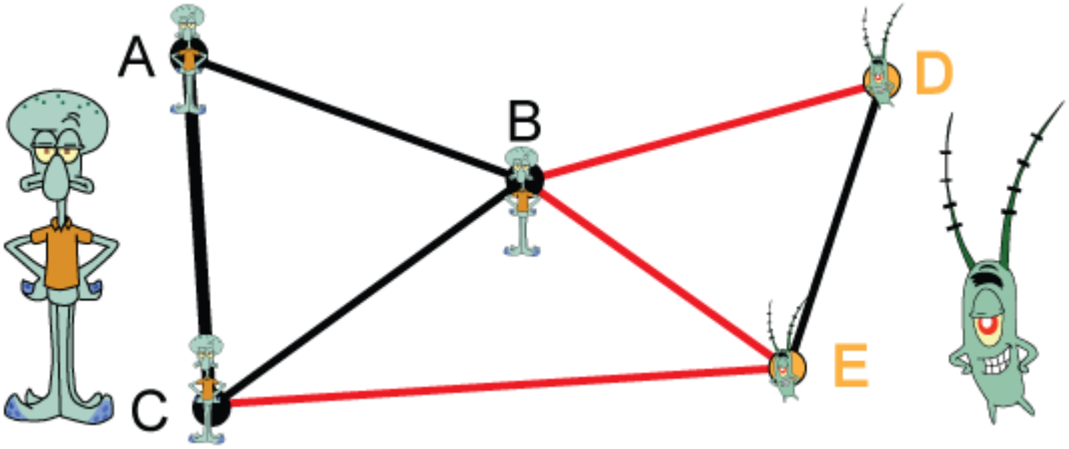
A graph or sub-graph is said to be strongly connected if every vertex is reachable from every other vertex.

Q The graph below is composed of two independent sets (A,B,C) (i.e. those vertices with Squidward) and (D,E) (i.e. those with Sheldon James Plankton). What is the cut set of the graph?



Solution:

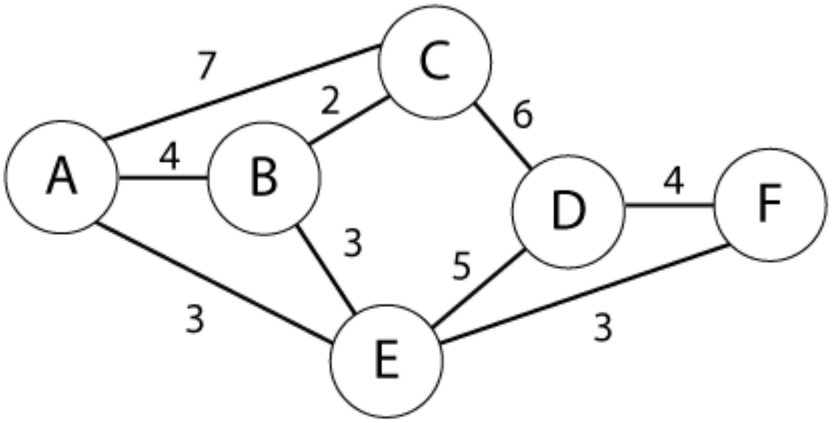
The cut set is (B-D, B-E, C-E)



A cut set is a set of edges whose end points are in different subset of the partition. Edges are said to be crossing the cut if they are in its cut-set. A set of edges of a graph, which, if removed (or "cut"), disconnects the graph (i.e., forms a disconnected graph).

Q

Use Prim's algorithm to find a minimum spanning tree for the connected weighted graph below. *Show your work.*



What is the Time Complexity of Prim's algorithm?

Solution:

Step 1: Choose any starting vertex. Look at all edges connecting to the vertex and choose the one with the lowest weight and add this to the tree.

Step 2: Look at all edges connected to the tree. Choose the one with the lowest weight and add to the tree.

Step 3: Repeat step 2 until all vertices are in the tree (n-1 edges).

```
Prim(G, c) {
        foreach (v in V) a[v] <- ∞
         Initialize an empty priority queue Q
        foreach (v in V) insert v onto Q
        Initialize set of explored nodes S = {}

        while (Q is not empty) {
                u ⍰ delete min element from Q
                S ⍰ S (union symbol){u}
                foreach (edge e = (u, v) incident to u)
                if ((v (is not an element symbol) (S) and (ce <a[v]))
                        decrease priority a[v] to ce
}
```

In other words

Take the minimum edge of the cut-set each time.

0: A S = {A}

1: A-E (3) take A-D S = {A,E}

2: B-E (3) or E-F (3) take B-E S = {A,E,B}

3: B-C (2) take B-C S = {A,E,B,C}

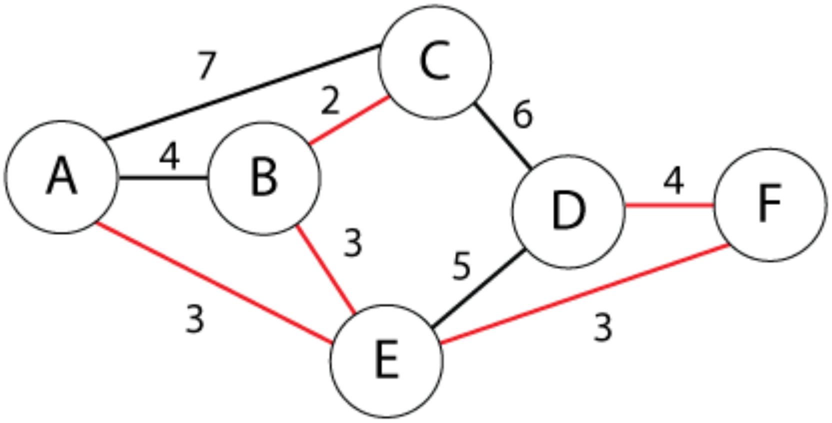
4: E-F (3) take E-F S = {A,E,B,C,F}

5: D-F (4) take D-F S = {A,E,B,C,F,D}

Done n-1 edges.

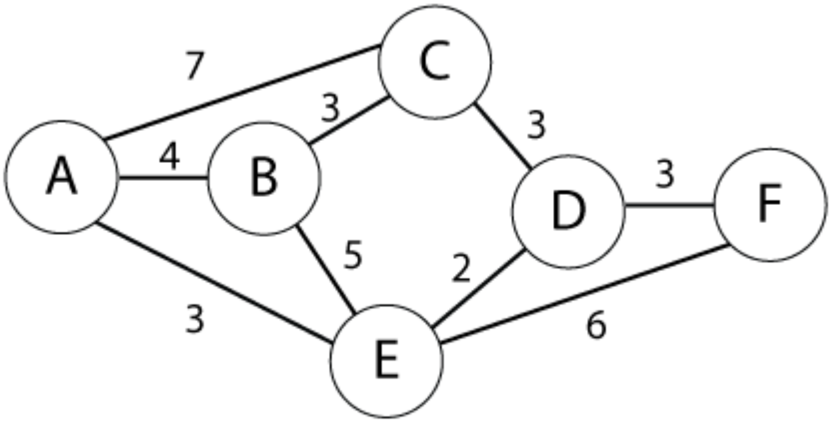
MST = {A-E , B-E, B-C, E-F, D-F }

MST weight = 3+3+2+3+4



Q

Find shortest path from A to F in the graph below using Dijkstra's algorithm. *Show your steps.*



Solution:

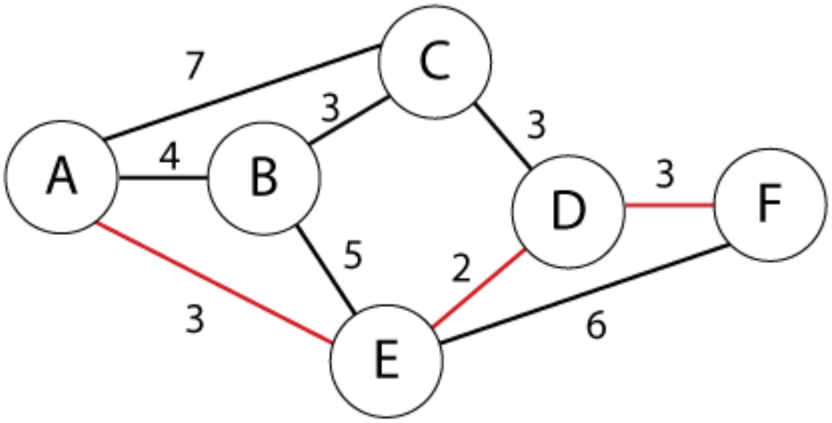
```
Given a graph, G, with edges E of the form (v1, v2) and vertices V, and a
source vertex, s

dist : array of distances from the source to each vertex
prev : array of pointers to preceding vertices
i    : loop index
F    : list of finished vertices
U    : list or heap unfinished vertices

/* Initialization: set every distance to INFINITY until we discover a path */
for i = 0 to |V| - 1
    dist[i] = INFINITY
    prev[i] = NULL
end
```

```
while(F is missing a vertex)
   pick the vertex, v, in U with the shortest path to s
   add v to F
   for each edge of v, (v1, v2)
        /* The next step is sometimes given the confusing name "relaxation"
        if(dist[v1] + length(v1, v2) < dist[v2])
            dist[v2] = dist[v1] + length(v1, v2)
            prev[v2] = v1
            possibly update U, depending on implementation
        end if
    end for
end while
```

Source A

		A	B	C	D	E	F
1: A {A}	A	0	(4,A)	(7,A)	INF	(3, A)**	INF
2: E {A,E}	E	0	(4,A)**	(7,A)	(5,E)	(3, A)	(9,E)
3: B {A,E,B}	B	0	(4,A)	(7,A)	(5,E)**	(3, A)	(9,E)
4: D {A,E,B,D}	D	0	(4,A)	(7,A) **	(5,E)	(3, A)	(8,D)
5: C {A,E,B,D,C}	C	0	(4,A)	(7,A)	(5,E)	(3, A)	(8,D) **
4: F {A,E,B,D,C,F}	F	0	(4,A)	(7,A)	(5,E)	(3, A)	(8,D)

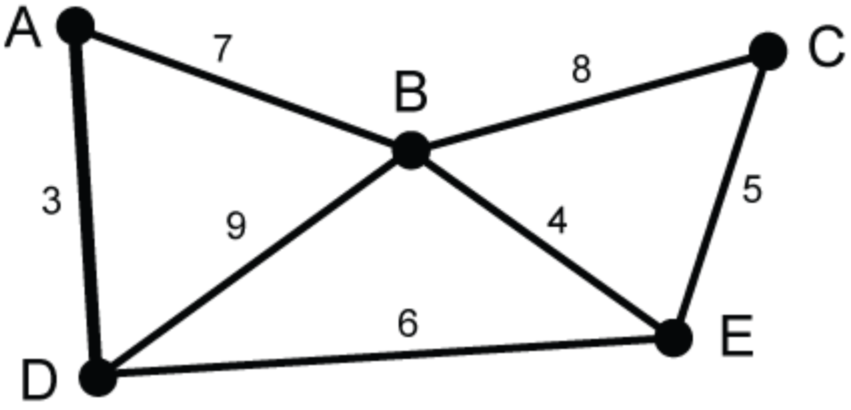
`** U with the shortest path to s`
`Red indicates a vertex has been moved from U to F`



Now we return our final shortest path, which is: A → E → D → F Cost (3+2+3 = 8)

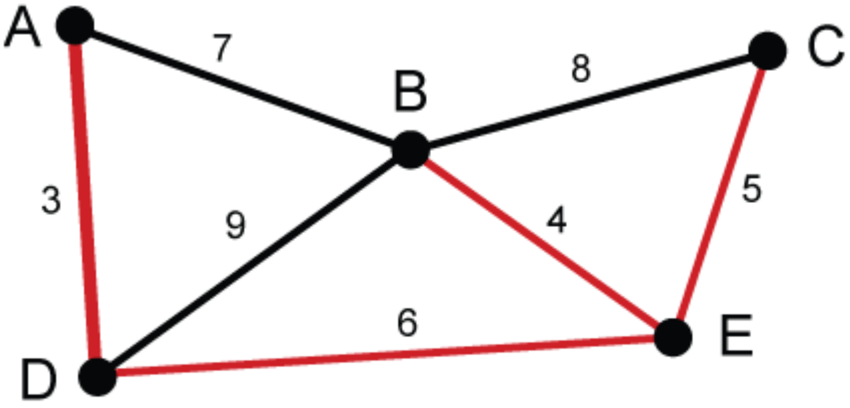
Q

Use Prim's algorithm to find a minimum spanning tree for the connected weighted graph below. *Show your work.*



What is the Time Complexity of Prim's algorithm?

Solution:



*Note: Using Kruskal's algorithm is NOT correct. It says: "*Use Prim's algorithm to find a minimum spanning tree*".*

Prim's algorithm (Queue edges for Minimum Spanning Tree) – is a greedy algorithm that finds a minimum spanning tree for a connected weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. Runs in O(n^2) with an array; O(m log n) with a binary heap

Step 1: Choose any starting vertex. Look at all edges connecting to the vertex and choose the one with the lowest weight and add this to the tree.

Step 2: Look at all edges connected to the tree. Choose the one with the lowest weight and add to the tree.

Step 3: Repeat step 2 until all vertices are in the tree (n-1 edges).

```
Prim(G, c) {
       foreach (v in V) a[v] <- ∞
        Initialize an empty priority queue Q
       foreach (v in V) insert v onto Q
       Initialize set of explored nodes S = {}

       while (Q is not empty) {
              u ⍰ delete min element from Q
              S ⍰ S (union symbol){u}
              foreach (edge e = (u, v) incident to u)
              if ((v (is not an element symbol) (S) and (ce <a[v]))
                     decrease priority a[v] to ce
}
```

In other words

Take the minimum edge of the cut-set each time.

0: A `S = {A}`

1: A-D < A-B take A-D `S = {A,D}`

2: D-E < A-B or D-B take D-E   `S = {A,D,E}`

3: B-E < A-B or D-B or C-E take B-E   `S = {A,D,E,B}`

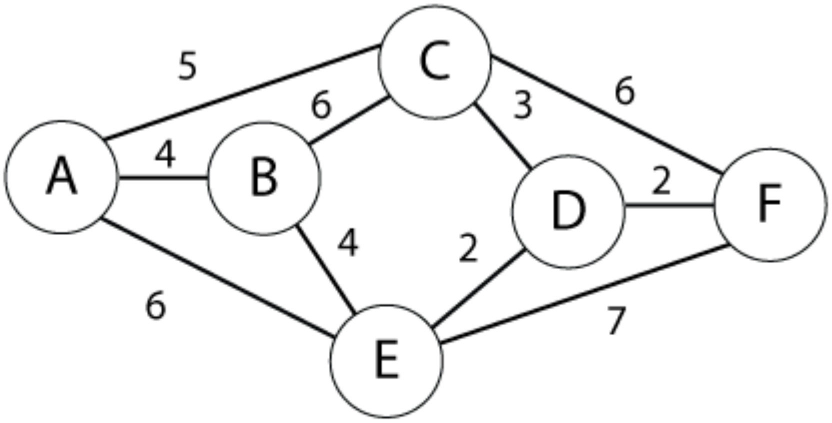
4: C-E   < A-B or D-B or B-C- take C-E   `S = {A,D,E,B,C}`

Done n-1 edges.

MST = { A-D , D-E , B-E, C-E }

Q

Use Kruskal's algorithm to find a minimum spanning tree for the connected weighted graph below:



Solution:

Kruskal's algorithm pseudocode:

A.	Create a forest T (a set of trees), where each vertex in the graph is a separate tree
B.	Create a set S containing all the edges in the graph
C.	Sort edges by weight

While S is nonempty and T is not yet spanning:

I. remove an edge with minimum weight from S
II. if that edge connects two different trees, then add it to the forest, combining two trees into a single tree, otherwise discard that edge.

Steps:

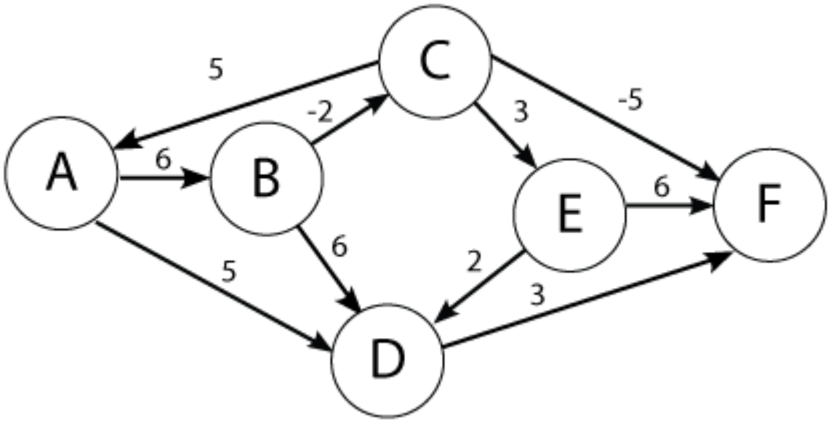
I. Connect D-E (2)
II. Connect D-F (2)
III. Connect C-D (3)
IV. Connect A-B (4)
V. Connect B-E (4)

Stop. MST formed (5 edges, 6 vertices)

MST = {A-B, B-E, E-D, C-D, D-F}

MST weight = 2+2+3+4+4=15

Q



Use the Bellman-Ford algorithm to find the shortest path from node A to F in the weighted directed graph above. *Show your work.*

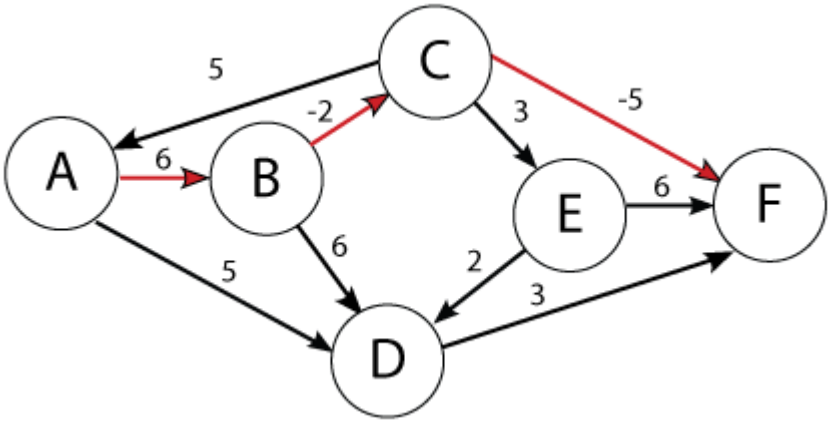
Solution:

Shortest path: A->B->C-->F at cost -1

	A	B	C	D	E	F
0	0	INF	INF	INF	INF	INF
1	0	6	INF	5	INF	INF
2	0	6	4	5	INF	8 D

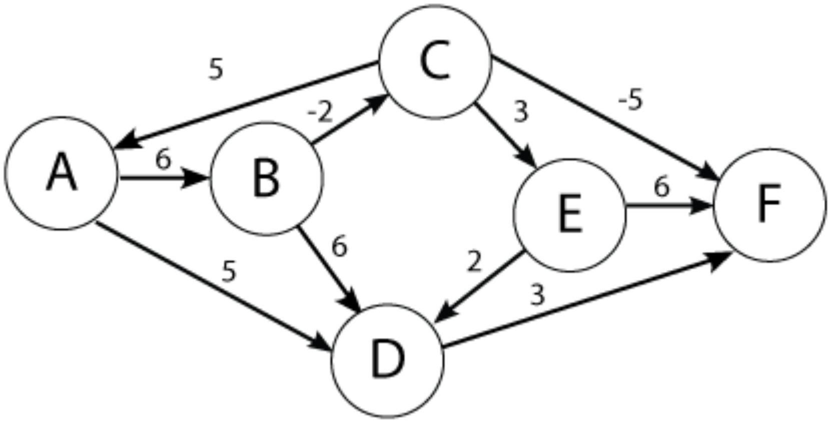
3	0	6	4	5	7	-1 C
4	0	6	4	5	7	-1
5	0	6	4	5	7	-1
6	0	6	4	5	7	-1

Many students calculate from F to A which is fine but the numbers in the table will be different even though the final path will be the same.



Another Bellman-Ford algorithm example is worked detail at
https://www.youtube.com/watch?v=hq3TZInZ5J4

Q



Use the Bellman-Ford algorithm to find the shortest path from node A to F in the weighted directed graph above. *Show your work.*

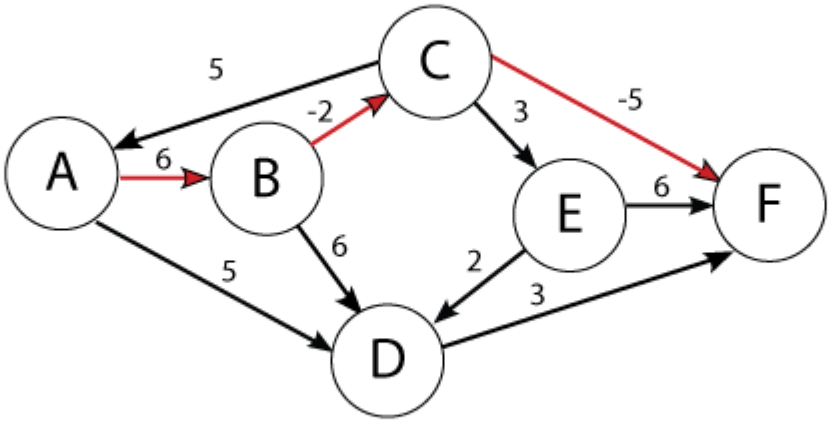
Solution:

Shortest path: A->B->C-->F at cost -1

	A	B	C	D	E	F
0	0	INF	INF	INF	INF	INF

1	0	6	INF	5	INF	INF
2	0	6	4	5	INF	8 D
3	0	6	4	5	7	-1 C
4	0	6	4	5	7	-1
5	0	6	4	5	7	-1
6	0	6	4	5	7	-1

Many students calculate from F to A which is fine but the numbers in the table will be different even though the final path will be the same.


5
C
-2
3
-5
A
6
B
E
6
F
6
2
5
3
D