CSYE 7374
Autonomous Learning in Games
Assignment 2 – Classical Game AI
Solutions

Professor: Nik Bear Brown
Due: February 15, 2020

TAs

Dikshant Rathi <rathi.d@husky.neu.edu>

Q1 (10 Points) Give a brief definition for the following:

i. Connected Component
ii. Strong Connectivity
iii. A* Search (include Pseudocode)
iv. Heuristic
v. Game Tree (include example)
vi. Constraint satisfaction problem (CSP)
vii. Dynamic Programming
viii. Breadth First Search (BFS) (include Pseudocode)
ix. Minimax algorithm (include example)
x. Alpha–beta pruning (include example)

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.

iii. A* Search

A* (pronounced "A-star") is a graph traversal and path search algorithm, which is often used in computer science due to its completeness, optimality, and optimal efficiency.[1] One major practical drawback is its space complexity, as it stores all generated nodes in memory.

Pseudocode
The following pseudocode describes the A* algorithm:

```
function reconstruct_path(cameFrom, current)
    total_path := {current}
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.prepend(current)
    return total_path

// A* finds a path from start to goal.
// h is the heuristic function. h(n) estimates the cost to reach goal from
node n.
function A_Star(start, goal, h)
    // The set of discovered nodes that may need to be (re-)expanded.
    // Initially, only the start node is known.
    openSet := {start}

    // For node n, cameFrom[n] is the node immediately preceding it on the
cheapest path from start to n currently known.
    cameFrom := an empty map

    // For node n, gScore[n] is the cost of the cheapest path from start to n
currently known.
    gScore := map with default value of Infinity
    gScore[start] := 0

    // For node n, fScore[n] := gScore[n] + h(n).
    fScore := map with default value of Infinity
    fScore[start] := h(start)

    while openSet is not empty
        current := the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, current)

        openSet.Remove(current)
        for each neighbor of current
            // d(current,neighbor) is the weight of the edge from current to
neighbor
            // tentative_gScore is the distance from start to the neighbor
through current
            tentative_gScore := gScore[current] + d(current, neighbor)
            if tentative_gScore < gScore[neighbor]
                // This path to neighbor is better than any previous one.
Record it->
                cameFrom[neighbor] := current
                gScore[neighbor] := tentative_gScore
```

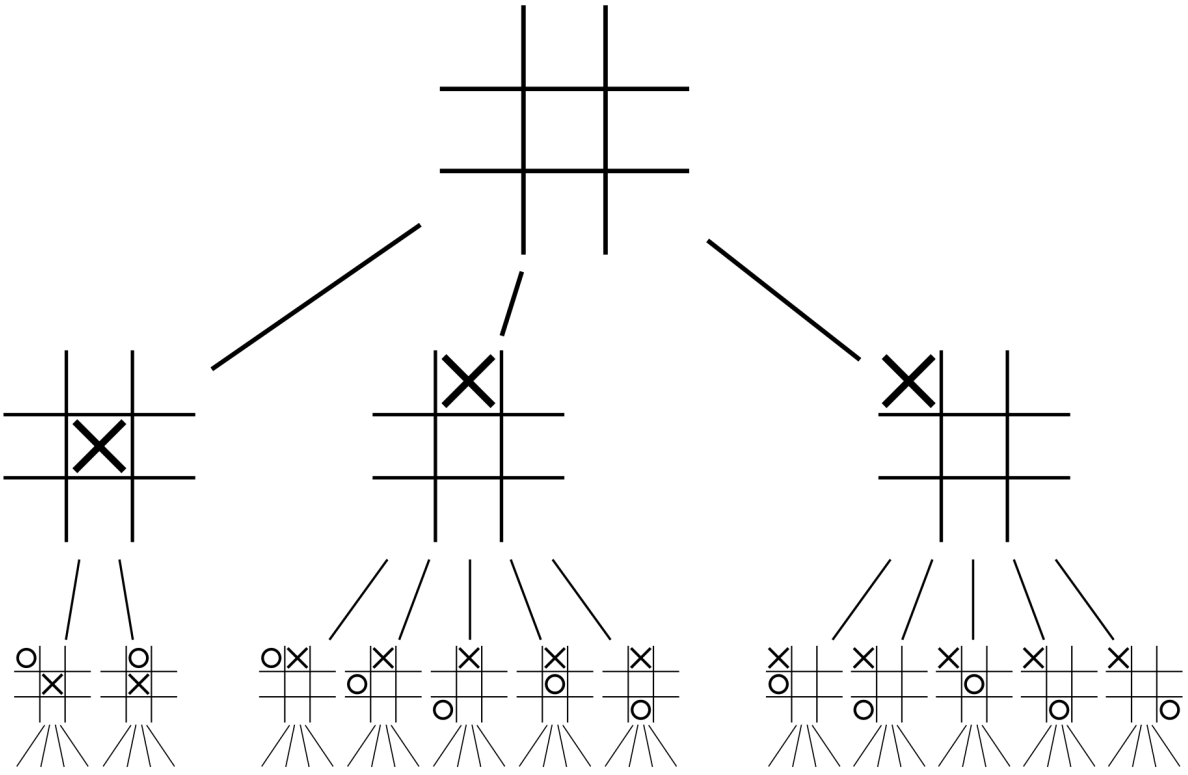
```
                fScore[neighbor] := gScore[neighbor] + h(neighbor)
                if neighbor not in openSet
                    openSet.add(neighbor)

    // Open set is empty but goal was never reached
    return failure
```

iv. Heuristic

A heuristic technique (/hjʊəˈrɪstɪk/; Ancient Greek: εὑρίσκω, "find" or "discover"), or a heuristic for short, is any approach to problem solving or self-discovery that employs a practical method that is not guaranteed to be optimal, perfect or rational, but which is nevertheless sufficient for reaching an immediate, short-term goal. Where finding an optimal solution is impossible or impractical, heuristic methods can be used to speed up the process of finding a satisfactory solution. Heuristics can be mental shortcuts that ease the cognitive load of making a decision

v. Game Tree

A game tree is a directed graph whose nodes are positions in a game and whose edges are moves. The complete game tree for a game is the game tree starting at the initial position and containing all possible moves from each position; the complete tree is the same tree as that obtained from the extensive-form game representation.



The first two plies of the game tree for tic-tac-toe.

vi. Constraint satisfaction problem (CSP)

Constraint satisfaction problems (CSPs) are mathematical questions defined as a set of objects whose state must satisfy a number of constraints or limitations. CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which is solved by constraint satisfaction methods.

vii. Dynamic Programming

Dynamic Programming is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, while combining the subsolutions into a solution for the whole problem.

viii. Breadth First Search (BFS)

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'[1]), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

ix. Minimax algorithm

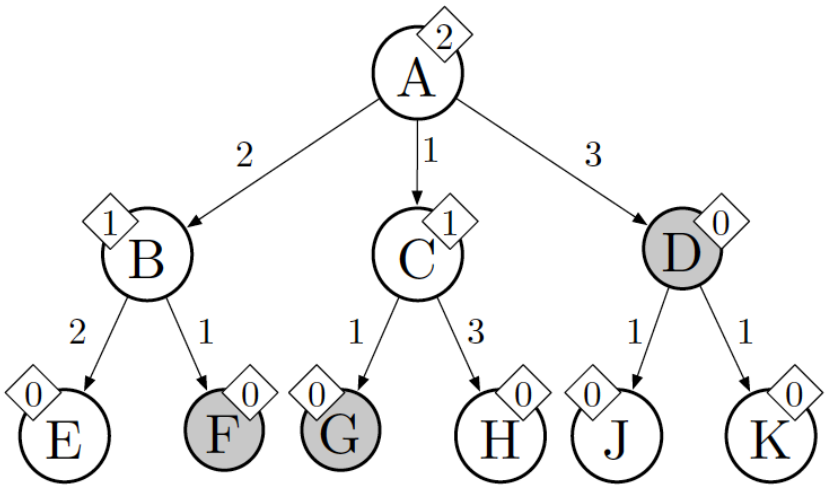
Minimax (sometimes MinMax, MM or saddle point) is a decision rule used in artificial intelligence, decision theory, game theory, statistics and philosophy for minimizing the possible loss for a worst case (maximum loss) scenario. When dealing with gains, it is referred to as "maximin"—to maximize the minimum gain. Originally formulated for two-player zero-sum game theory, covering both the cases where players take alternate moves and those where they make simultaneous moves, it has also been extended to more complex games and to general decision-making in the presence of uncertainty.

x. Alpha–beta pruning

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess, Go, etc.). It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision

Q2 (20 Points)

Consider the state space search problem shown in the graph. A is the start state and the shaded states are goals. Arrows encode possible state transitions, and numbers by the arrows represent action costs. Note that state transitions are directed; for example, A -> B is a valid transition, but B -> A is not.



Numbers shown in diamonds are heuristic values that estimate the optimal (minimal) cost from that node to a goal. For each of the following search algorithms, write down the nodes that are removed from fringe in the course of the search, as well as the final path returned. Because the original problem graph is a tree, the tree and graph versions of these algorithms will do the same thing, and you can use either version of the algorithms to compute your answer.

Assume that the data structure implementations and successor state orderings are all such that ties are broken alphabetically. For example, a partial plan S -> X -> A would be expanded before S -> X -> B; similarly, S -> A -> Z would be expanded before S -> B -> A.

(a) [4 pts] Depth-First Search

Nodes removed from fringe:
Path returned:

(b) [4 pts] Breadth-First Search

Nodes removed from fringe:
Path returned:

(c) [4 pts] Uniform-Cost Search

Nodes removed from fringe:
Path returned:

(d) [4 pts] Greedy Search

Nodes removed from fringe:
Path returned:

(e) [4 pts] A* Search

Nodes removed from fringe:
Path returned:

Solution:

(a) [4 pts] Depth-First Search (ignores costs)
Nodes removed from fringe: A, B, E, F
Path returned: A, B, F

(b) [4 pts] Breadth-First Search (ignores costs)
Nodes removed from fringe: A, B, C, D
Path returned: A, D

(c) [4 pts] Uniform-Cost Search
Nodes removed from fringe: A, C, B, G
Path returned: A, C, G

(d) [4 pts] Greedy Search
Nodes removed from fringe: A, D
Path returned: A, D

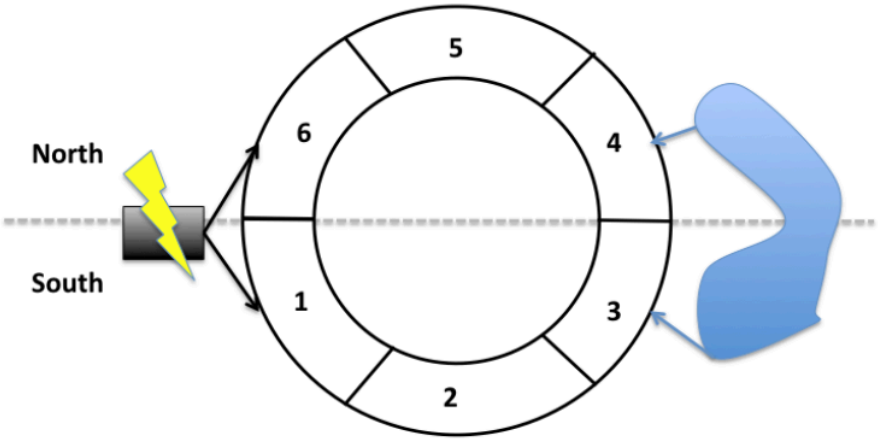
(e) [4 pts] A* Search
Nodes removed from fringe: A, C, G
Path returned: A, C, G

Q3 (20 Points)

CSPs: Apple's New Campus

Apple's new circular campus is nearing completion. Unfortunately, the chief architect on the project was using Google Maps to store the location of each individual department, and after upgrading to iOS 6, all the plans for the new campus were lost!

The following is an approximate map of the campus:



The campus has six Offices, labeled 1 through 6, and six departments:
_ Legal (L)
_ Maps Team (M)
_ Prototyping (P)
_ Engineering (E)
_ Tim Cook's office (T)
_ Secret Storage (S)

Offices can be next to one another, if they share a wall (for an instance, Offices 1-6). Offices can also be across from one another (specifically, Offices 1-4, 2-5, 3-6). The Electrical Grid is connected to Offices 1 and 6. The Lake is visible from Offices 3 and 4. There are two \halves" of the campus South (Offices 1-3) and North (Offices 4-6).

The constraints are as follows:
i. (L)egal wants a view of the lake to look for prior art examples.
ii. (T)im Cook's office must not be across from (M)aps.
iii. (P)rototyping must have an electrical connection.
iv. (S)ecret Storage must be next to (E)ngineering.
v. (E)ngineering must be across from (T)im Cook's office.
vi. (P)rototyping and (L)egal cannot be next to one another.
vii. (P)rototyping and (E)ngineering must be on opposite sides of the campus (if one is on the North side, the other must be on the South side).
viii. No two departments may occupy the same office.

Constraints. Note: There are multiple ways to model constraint viii. In your answers below, assume constraint viii is modeled as multiple pairwise constraints, not a large n-ary constraint.

(a) [4 pts] Which constraints are unary?

(b) [4 pts] In the constraint graph for this CSP, how many edges are there?

(c) [4 pts] Write out the explicit form of constraint iii.

(d) [4 pts] When enforcing arc consistency in a CSP, does the set of values which remain when the algorithm terminates depend on the order in which arcs are processed from the queue.

(e) [4 pts] In a general CSP with n variables, each taking d possible values, what is the maximum number of times a backtracking search algorithm might have to backtrack (i.e. the number of the times it generates an assignment, partial or complete, that violates the constraints) before finding a solution or concluding that none exists?

Solution:

(a) [4 pts] Which constraints are unary?

i & iii

i ii iii iv v vi vii viii

(b) [4 pts] In the constraint graph for this CSP, how many edges are there?

Constraint vii connects each pair of variables; there are 6 choose 2 or 15 such pairs.

(c) [4 pts] Write out the explicit form of constraint iii.

P is an element of {1,6}

(d) [4 pts] When enforcing arc consistency in a CSP, does the set of values which remain when the algorithm terminates depend on the order in which arcs are processed from the queue.

No, this is false.

(e) [4 pts] In a general CSP with n variables, each taking d possible values, what is the maximum number of times a backtracking search algorithm might have to backtrack (i.e. the number of the times it

generates an assignment, partial or complete, that violates the constraints) before finding a solution or concluding that none exists?

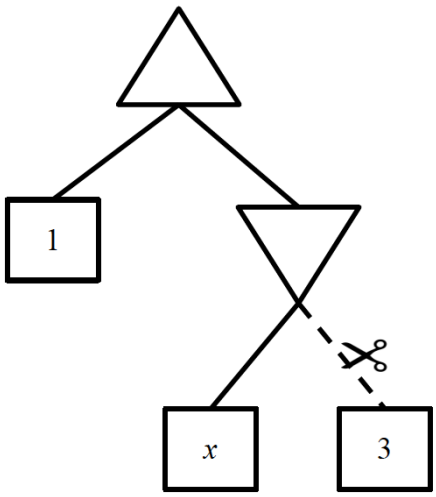
$O(d^n)$

In general, the search might have to examine all possible assignments.

Q6 (10 Points)

For each of the game-trees shown below, state for which values of x the dashed branch with the scissors will be pruned. If the pruning will not happen for any value of x write \none". If pruning will happen for all values of x write “all".

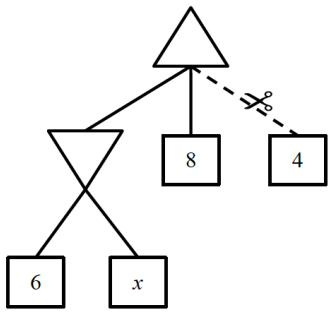
We are assuming that nodes are evaluated left to right and ties are broken in favor of the latter nodes. A different evaluation order would lead to different interval bounds, while a different tie breaking strategies could lead to strict inequalities (> instead of >=). Successor enumeration order and tie breaking rules typically impact the efficiency of alpha-beta pruning.



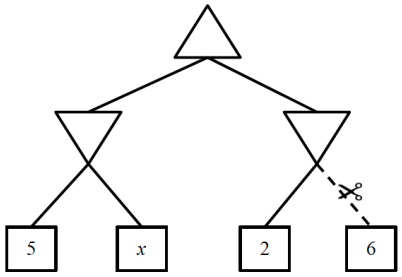
(a) Example Tree. Answer: $\mathbf{x \leq 1}$.

(a) [2 pts] Explain why the triangles point up and down in the tree and why values less than or equal to one can be pruned.

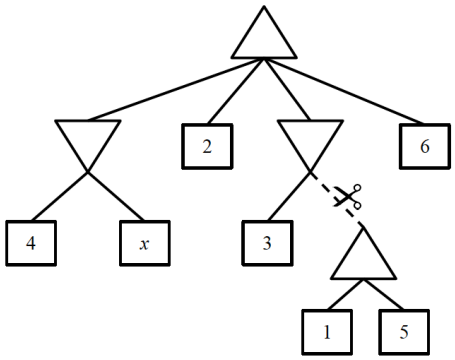
(b-e) [2 pts each]



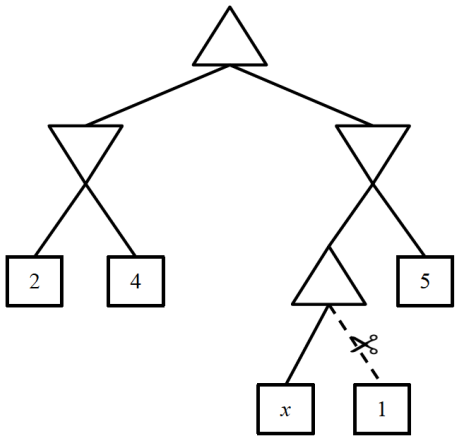
(b) Tree 1.



(c) Tree 2.



(d) Tree 3.



(e) Tree 4.

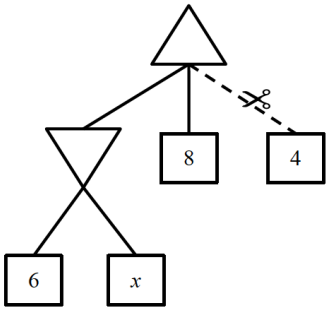
10

Solution:

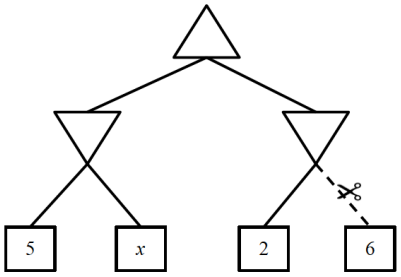
(a) [2 pts]

The downward **triangle** represents a location in the **tree** where **minimax** will minimize the opponent's advantage. Whereas, the upward **triangles** are the locations where **minimax** maximizes the agent's advantage.

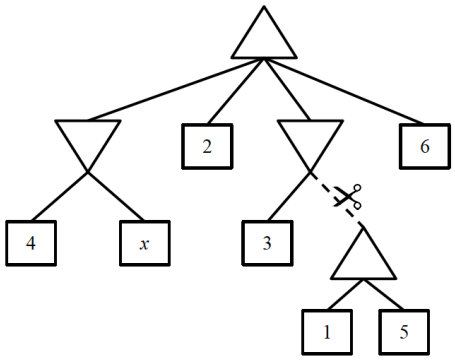
As we are minimizing the opponent's advantage **minimax** can ignore values greater than one.



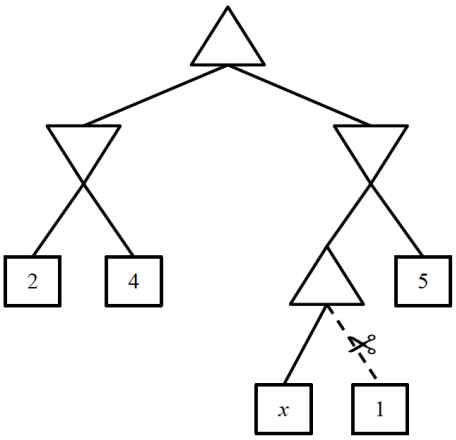
(b) Tree 1. Answer: None



(c) Tree 2. Answer: $x \geq 2$

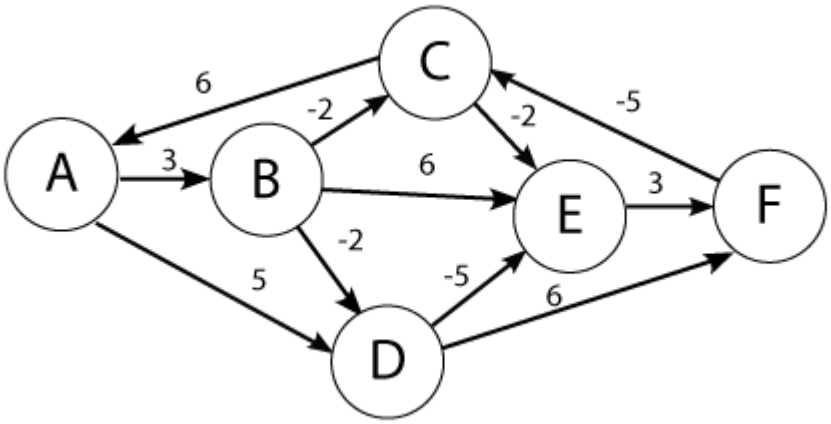


(d) Tree 3. Answer: $x \geq 3$,



(e) Tree 4. Answer: None

Q7 (10 Points)



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

Solution:

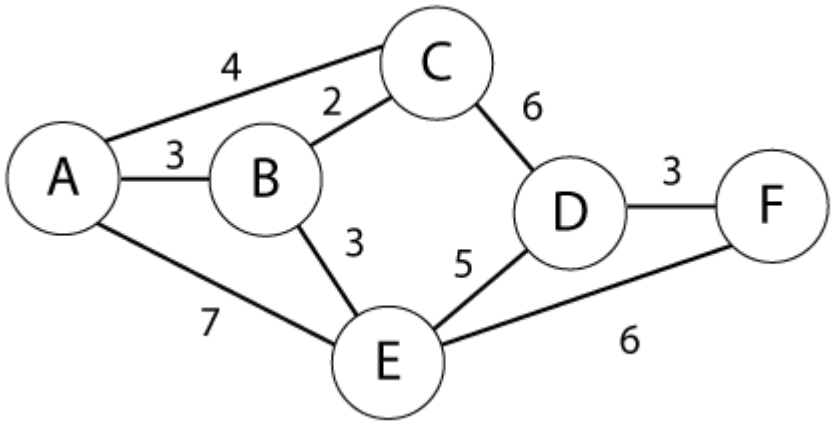
	A	B	C	D	E	F
0	0	INF	INF	INF	INF	INF
1	0	3 (A)	INF	5 (A)	INF	INF
2	0	3	1 (B)	1 (B)	0 (D)	11 (D)
3	0	3	1 (B)	1 (B)	-4 (D)	7 (D)
4	0	3	-3 (B)	1 (B)	-4 (D)	-1 (E)
5	0	3	-6 (F)	1 (B)	-4 (D)	-1 (E)
6	0	3	-6 (F)	1 (B)	-8 (C) (Negative - Stop)	-

Negative cycle: F->C->E at cost -4

Note: I DID NOT take off points for small arithmetic errors. 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.

Q8 (10 Points)

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



What is the Time Complexity of Kruskal's algorithm?

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

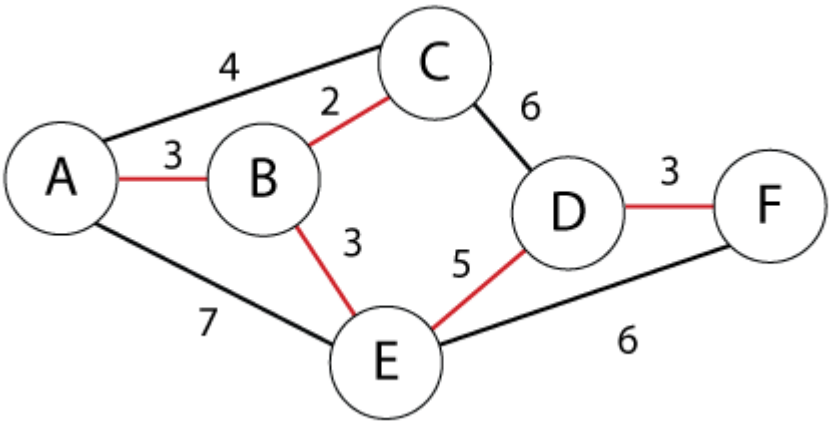
While S is nonempty and T is not yet spanning:

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 B-C (2)
II. Connect A-B (3)
III. Connect D-F (3)
IV. Connect B-E (3)
V. Skip A-C (4) (Forms cycle)
VI. Connect E-D (4)

Stop. MST formed (5 edges, 6 vertices)

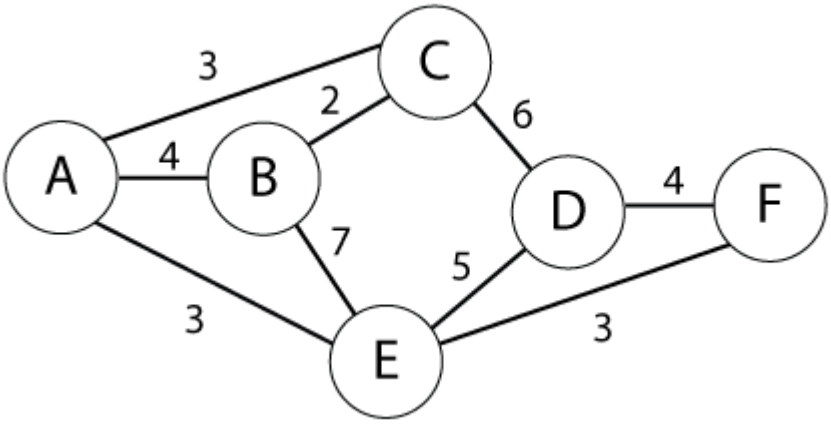


MST = {A-B , B-C , B-E, E-D, D-F}

Kruskal's algorithm is O(E log V) time. Where E is the set of edges and V is the set of vertices.

Q9 (10 Points)

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-C (3) or A-E (3) is min-cut take A-E `S = {A,E}`

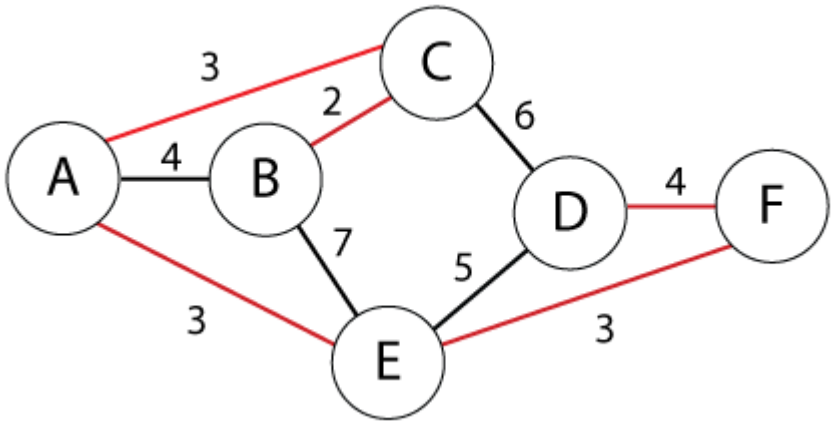
2: E-F (3) or A-C (3) is min-cut take E-F `S = {A,E,F}`

3: A-C (3) is min-cut take A-C `S = {A,E,F,C}`

4: B-C (2) is min-cut take B-C `S = {A,E,F,C,B}`

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

Done n-1 edges. (5)



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

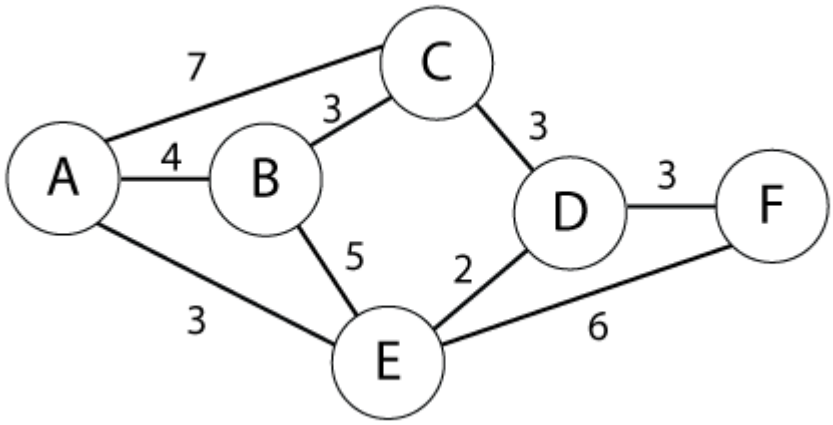
MST weight =3+3+3+2+4=15

Prim's algorithm time depends on the implementation. Any of the below are accepted. Where E is the set of edges and V is the set of vertices.

Minimum edge weight data structure	**Time complexity (total)**
adjacency matrix, searching	$O(\|V\|^2)$
binary heap and adjacency list	$O((\|V\| + \|E\|) \log \|V\|) = O(\|E\| \log \|V\|)$
Fibonacci heap and adjacency list	$O(\|E\| + \|V\| \log \|V\|)$

Q10 (10 Points)

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



Solution:

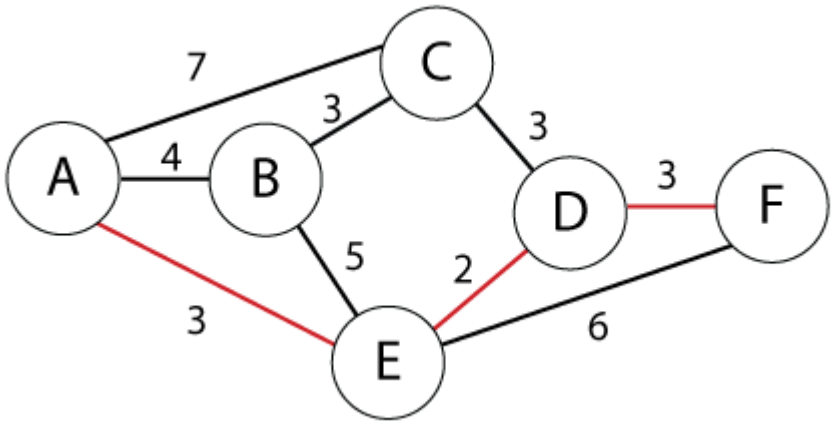
```
Given a graph, G, with edges E of the form (v1, v2) and vertices V, and a
source vertex, s

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)