Department of Computer Science
National University of Computer and Emerging Sciences, Lahore Campus

Mid-I	**CS 401: Artificial Intelligence**	**Fall 2016**
Date: 9/21/16	**Marks: 30**	**Time: 60 Minutes**

Student ID _____ **Name** _____ **Section** _____

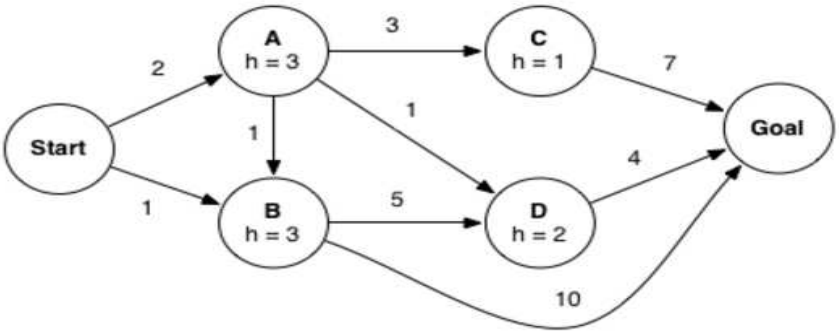
Notes:

This exam contains 7 pages (including this cover page) and 3 problems.
Check to see if any pages are missing. Enter all requested information on the top of this page, and put your ID on the top of every page, in case the pages become separated.
This a **closed Book, closed Notes** exam.

You are required to show your work on each problem on this exam.

1. (8 points) Consider A* graph search on the graph below. Arcs are labeled with action costs and states are labeled with heuristic values. Assume that ties are broken alphabetically (so a partial plan $S- > X- > A$ would be expanded before $S- > X- > B$ and $S- > A- > Z$ would be expanded before $S- > B- > A$.

   Note that to get full marks in each part you need to show complete algorithm working.



   (a) (2 points) In what order are states expanded by A* graph search?

(b) (2 points) What path does A* graph search return?

(c) (2 points) What is the final path returned by breadth-first graph search? Also specify the order of states expanded by BFS.

(d) (2 points) What is the final path returned by depth-first graph search? Also specify the order of states expanded by DFS.

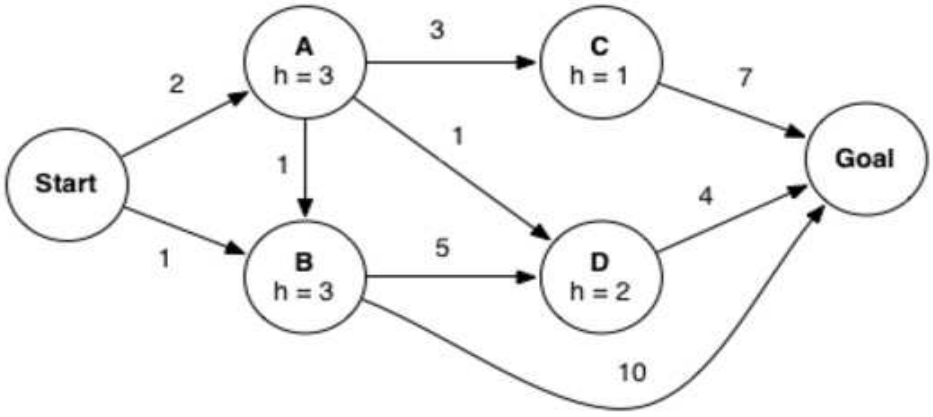
2. (10 points) The general algorithm for GRAPH-SEARCH as discussed in the class is given below.

```
function GRAPH-SEARCH(problem, fringe, strategy) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe, strategy)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

With the above implementation a node that reaches a goal state may sit on the fringe while the algorithm continues to search for a path that reaches a goal state. Let's consider altering the algorithm by testing whether a node reaches a goal state when inserting into the fringe. Concretely, we add the line of code highlighted below:

```
function EARLY-GOAL-CHECKING-GRAPH-SEARCH(problem, fringe, strategy) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe, strategy)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                if GOAL-TEST(problem, STATE[child-node]) then return child-node
                fringe ← INSERT(child-node, fringe)
            end
    end
```

The modified algorithm can find a solution faster. However, this change might have affected some properties of the algorithm. To explore the possible differences, consider the graph of problem on also given below.



(a) (2 points) If using EARLY-GOAL-CHECKING-GRAPH-SEARCH with a Uniform Cost node expansion strategy, which path, if any, will the algorithm return?

(b) (2 points) If using EARLY-GOAL-CHECKING-GRAPH-SEARCH with an A* node expansion strategy, which path, if any, will the algorithm return?

(c) (3 points) Assume you run EARLY-GOAL-CHECKING-GRAPH-SEARCH with the Uniform Cost node expansion strategy. Which of the following statements are true. Justify your answer.

- The algorithm is complete

- The algorithm will return an optimal solution.

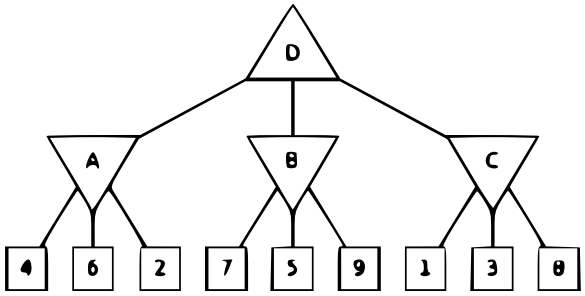
(d) (3 points) Assume you run EARLY-GOAL-CHECKING-GRAPH-SEARCH with the A* node expansion strategy and a consistent heuristic. Which of the following statements are true. Justify your answer.

- The algorithm is complete

- The algorithm will return an optimal solution.

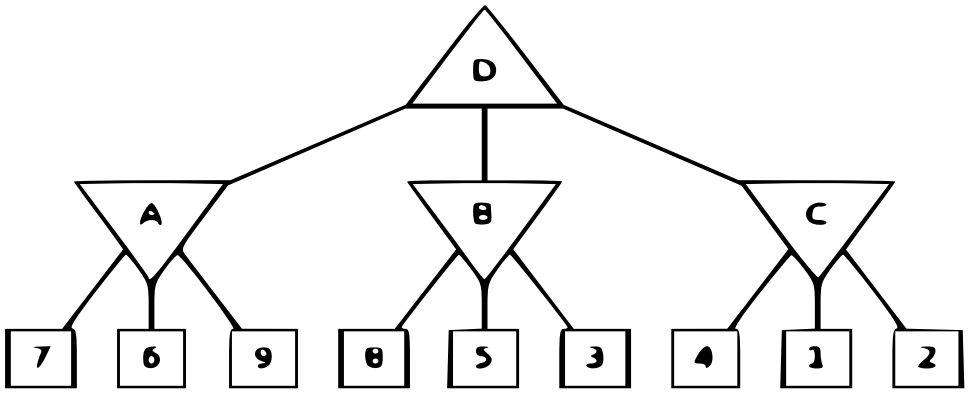
3. (12 points) **Minimax Search** For this problem you must use the given graphs to show your answers

   (a) (2 points) Consider the game tree shown below. Triangles that point up, such as at the top node (root), represent choices for the maximizing player; triangles that point down represent choices for the minimizing player. **Find the value of each node using minimax algorithm**.

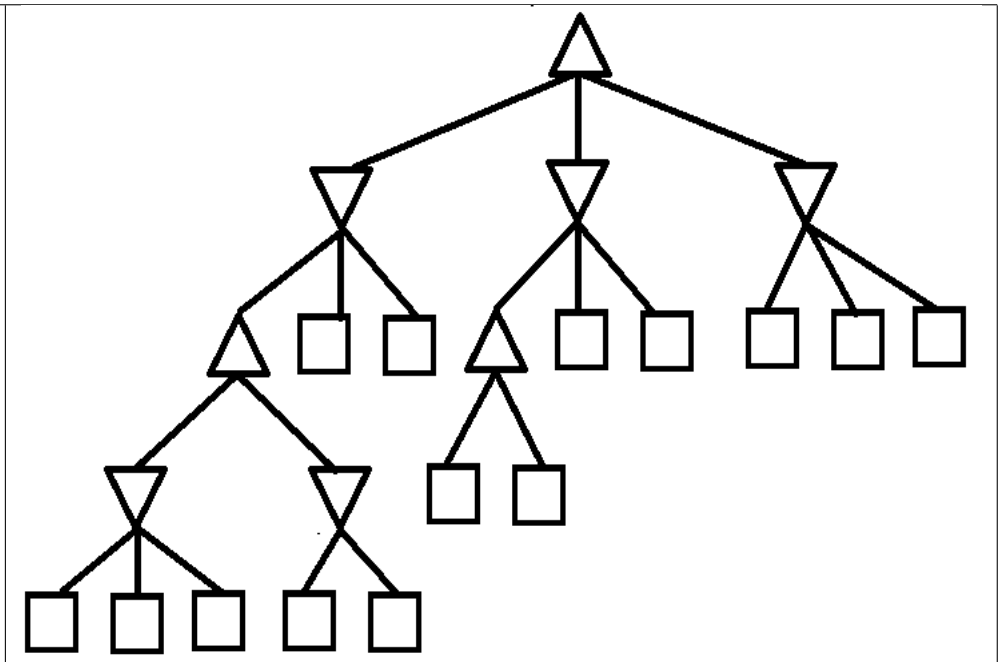


   (b) (4 points) Assuming both players in an adversarial game act optimally, use alpha-beta pruning to find the value of the root node with the game tree shown below.Assume that the search goes from left to right; when choosing which child to visit first, choose the left-most unvisited child.

   For each of the node specify the value computed by the algorithm. Also mark **an 'X'** for the nodes that don't get visited due to pruning. For



(c) (6 points) Now consider the two game trees shown below. For each of the graphs shown below specify the values of utility function for each of the terminal node (square nodes) that will cause the minimax algorithm prune the indicated branches. **Note that in this part every student will, typically, have a different answer**

No Pruning	
Pruning