Completed

NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

MIDTERM ASSESSMENT FOR
Semester 2 AY2016/17

CS3243: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

March 6, 2017 Time Allowed: 1 Hour 30 Minutes

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **THREE (3)** parts and comprises **NINE (9)** printed pages, including this page.
2. Answer **ALL** questions as indicated.
3. This is a **RESTRICTED OPEN BOOK** examination.
4. Please fill in your **Matriculation Number** below.

MATRICULATION NUMBER: _____

EXAMINER'S USE ONLY		
Part	Mark	Score
I	8	
II	19	
III	23	
TOTAL	50	

In Part I, II, and III, you will find a series of short essay questions. For each short essay question, give your answer in the reserved space in the script.

Part I
Constraint Satisfaction Problems

(8 points) Short essay questions. Answer in the space provided on the script.

Consider the following constraint satisfaction problem with the variables TOH, BMG, WN, CSW, and BCN, each of which can be colored either red, green, or blue. Furthermore, from the constraint graph shown in Fig. 1, each edge denotes a constraint such that its incident nodes must NOT have the same color.

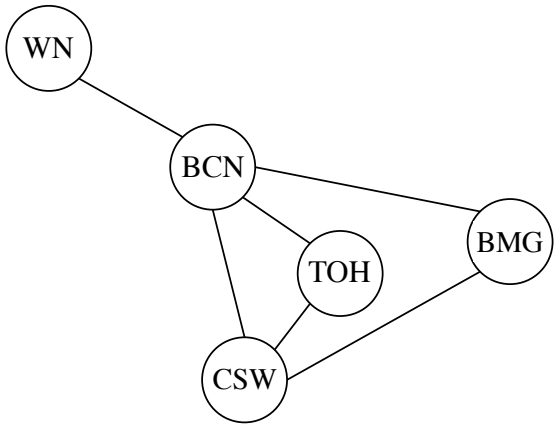


Figure 1: Constraint graph.

In this question, you will be asked to trace the backtracking search algorithm, given in Figure 6.5 of AIMA 3rd edition (reproduced in Figure 2), with FORWARD CHECKING on this map coloring problem. Assume that the domain values of each variable are tried in the following order: red, green, and blue.

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment then
          add {var = value} to assignment
          inferences ← INFERENCE(csp, var, value)
          if inferences ≠ failure then
              add inferences to assignment
              result ← BACKTRACK(assignment, csp)
              if result ≠ failure then
                  return result
      remove {var = value} and inferences from assignment
  return failure
```

Figure 2: Backtracking search algorithm.

1. (2 points) Use the degree (i.e., most constraining variable) heuristic to select the first unassigned variable. State the first variable that is selected as well as its assigned value. What are the domains of the remaining unassigned variables after forward checking?

Solution: BCN = red. Remaining unassigned values are {green, blue}

2. (2 points) Use the degree (i.e., most constraining variable) heuristic to select the second unassigned variable. State the second variable that is selected as well as its assigned value. What are the domains of the remaining unassigned variables after forward checking?

Solution: CSW = green. Remaining unassigned values for TOH and BMG are {blue}. WN = {green, blue}

3. (2 points) Use the minimum-remaining-values (i.e., most constrained variable) heuristic to select the third unassigned variable. State the third variable that is selected as well as its assigned value. What are the domains of the remaining unassigned variables after forward checking?

Solution:

TOH = blue.
BMG = blue. or

WN = {green, blue}

4. (2 points) Given the partial assignment obtained above and the revised domains of the remaining unassigned variables, what is a complete and consistent assignment that is eventually produced by the backtracking search algorithm?

Solution: Toh = blue, BMG = blue, WN = green, CSW = green, BCN = red.

Part II
Adversarial Search

(19 points) Short essay questions. Answer in the space provided on the script.

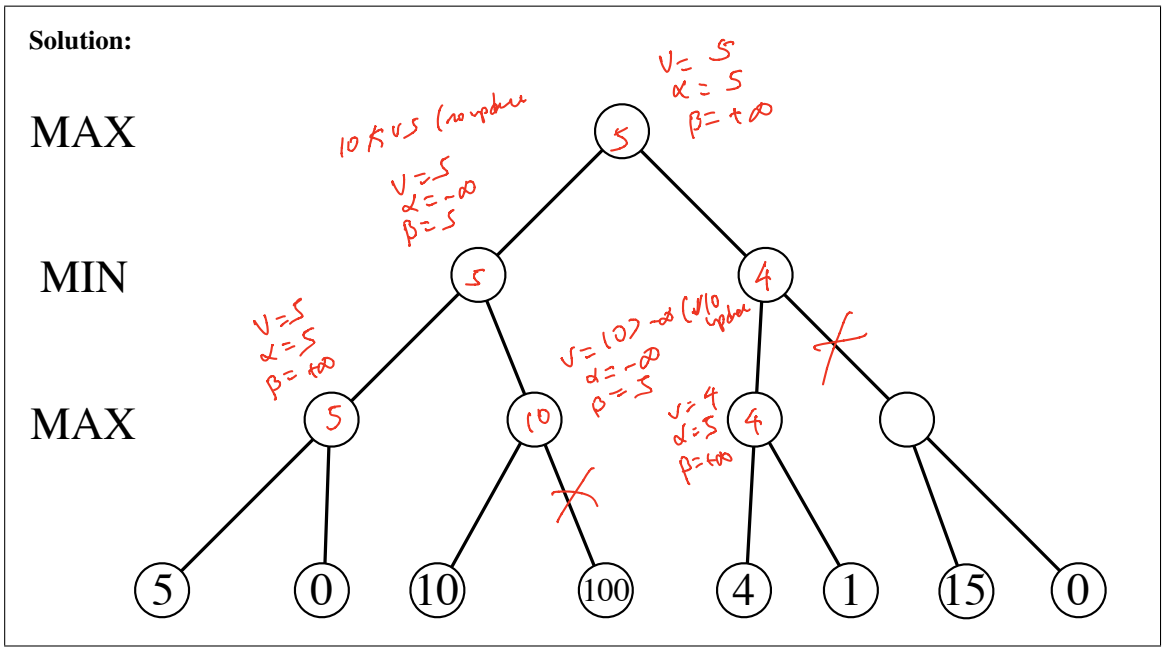
1. (6 points) Consider the minimax search tree shown in the solution space below; the utility function values are specified with respect to the MAX player and indicated at all the leaf (terminal) nodes. Suppose that we use alpha-beta pruning algorithm, given in Figure 5.7 of AIMA 3rd edition (reproduced in Figure 3), in the direction from left to right to prune the search tree.

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figure 3: Alpha-beta pruning algorithm (note that $s = state$).

Mark (with an 'X') all ARCS that are pruned by alpha-beta pruning, if any.

Solution:



2. (3 points) State the **EXACT** minimax value at the root node.

Solution: 5

3. (3 points) Suppose that the MIN player has decided before playing the game to perform action C in his turn, as shown in Figure 4 below. However, the MAX player does not know about this before playing the game and assumes that the MIN player is acting optimally. State MAX player's **EXACT** payoff value when starting from the root of the tree.

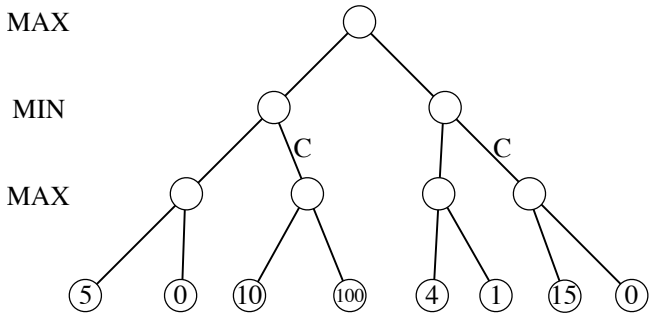


Figure 4: Minimax search tree.

Solution: 100

4. (7 points) Consider the minimax search tree shown in Figure 5 below; the utility function values are specified with respect to the MAX player and indicated at all the leaf (terminal) nodes. Suppose that we use alpha-beta pruning algorithm, given in Figure 5.7 of AIMA 3rd edition (reproduced in Figure 3), in the direction from left to right to prune the search tree. State the **largest possible integer value** for A and the **smallest possible integer value** for B such that **NO** arcs/nodes are pruned by alpha-beta pruning.

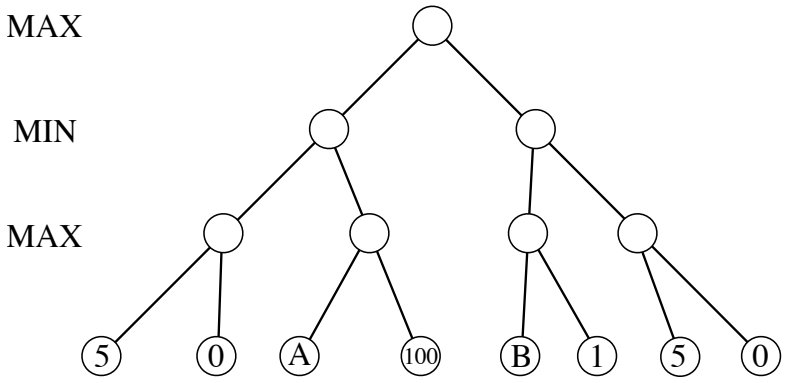


Figure 5: Minimax search tree.

Solution: A = 4 B = 6

Using the **largest possible integer value** for A and the **smallest possible integer value** for B in the minimax search tree shown in Figure 5, state the **EXACT** minimax value at the root node.

Solution: 5

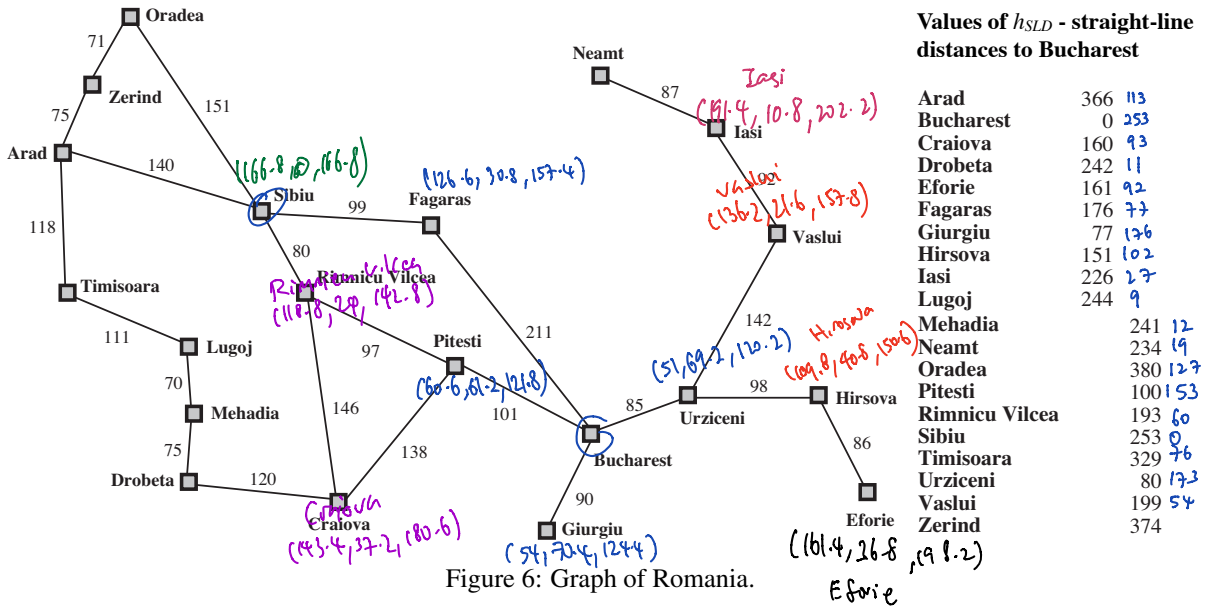
Part III
Uninformed and Informed Search

(23 points) Short essay questions. Answer in the space provided on the script.

Refer to Figure 6 below for questions 1 and 2 in Part III. Apply the graph search algorithms indicated below to find a path from **BUCHAREST** to **SIBIU** using the heuristic function (when necessary)

$$h(n) = |h_{SLD}(\text{Sibiu}) - h_{SLD}(n)|$$

where $h_{SLD}(n)$ is the straight-line distance from any city n to Bucharest given in Figure 3.22 of AIMA 3rd edition (reproduced in Figure 6).



Values of h_{SLD} - straight-line distances to Bucharest

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Figure 6: Graph of Romania.

1. (10 points) Trace the **best-first search algorithm using GRAPH SEARCH** with the evaluation function $f(n) = 0.6g(n) + 0.4h(n)$ by showing the nodes in the frontier at the end of each iteration of the outer loop. **Pay very careful attention to the following instructions when presenting your solution**:

   - This best-first search algorithm is identical to uniform-cost search (reproduced from Figure 3.14 of AIMA 3rd edition in Figure 7 below) except that best-first search uses f instead of g.
   - For each node n in the frontier, give the corresponding 3-tuple $(0.6g(n), 0.4h(n), f(n))$.
   - At the end of each iteration of outer loop, list the nodes in the frontier in nondecreasing order of f value.
   - AFTER the goal node is found (i.e., last iteration of outer loop), you must also list the nodes in frontier.

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

Figure 7: Uniform-cost search algorithm.

Solution: The node (denoting the initial state) in the frontier before entering the outer loop is provided.

FRONTIER:

Bucharest(0,101.2,101.2)

End of Iteration 1:

Urziceni (51, 69.2, 120.2), Pitesti (60.6, 61.2, 121.8), Giurgiu (54, 70.4, 124.4)
Fagaras (126.6, 20.8, 157.4)

End of Iteration 2:

Pitesti (60.6, 61.2, 121.8), Giurgiu (54, 70.4, 124.4), Hirsova (109.8, 40.8, 150.6)
Fagaras (126.6, 20.8, 157.4), Vaslui (136.2, 21.6, 157.8)

End of Iteration 3:

Giurgiu (54, 70.4, 124.4) Rimnicu Vilcea (118.8, 29, 142.8) Hirsova (109.8, 40.8, 150.6)
Fagaras (126.6, 20.8, 157.4), Vaslui (136.2, 21.6, 157.8) Craiova (143.4, 37.2, 180.6)

End of Iteration 4:

Rimnicu Vilcea (118.8, 29, 142.8) Hirsova (109.8, 40.8, 150.6)
Fagaras (126.6, 20.8, 157.4), Vaslui (136.2, 21.6, 157.8) Craiova (143.4, 37.2, 180.6)

End of Iteration 5:

Hirsova (109.8, 40.8, 150.6) Fagaras (126.6, 20.8, 157.4)
Vaslui (136.2, 21.6, 157.8), Sibiu (166.8, 0, 166.8) Craiova (143.4, 37.2, 180.6)

End of Iteration 6:

Fagaras (126.6, 20.8, 157.4), Vaslui (136.2, 21.6, 157.8), Sibiu (166.8, 0, 166.8)
Craiova (143.4, 37.2, 180.6) (161.4, 26.8, 191.2) Efori e

End of Iteration 7:

Vaslui (136.2, 21.6, 157.8), Sibiu (166.8, 0, 166.8)
Craiova (143.4, 37.2, 180.6) (161.4, 26.8, 191.2) Efori e

End of Iteration 8:

Sibiu (166.8, 0, 166.8) Craiova (143.4, 37.2, 180.6) (161.4, 26.8, 191.2) Efori e Iasi (191.4, 10.8, 202.2)

End of Iteration 9:

Craiova (143.4, 37.2, 180.6) (161.4, 26.8, 191.2) Efori e Iasi (191.4, 10.8, 202.2)

Give the solution path from Bucharest to Sibiu that is produced by the above best-first search algorithm using GRAPH SEARCH with the evaluation function $f(n) = 0.6g(n) + 0.4h(n)$.

Solution: Bucharest, Pitesti, RV, Sibiu

2. (5 points) (a) Give the solution path from Bucharest to Sibiu that is produced by the above best-first search algorithm using GRAPH SEARCH with the evaluation function $f(n) = 0.4g(n) + 0.6h(n)$.
(b) Does this best-first search algorithm incur lower, same, or higher cost for its solution path than that with the evaluation function $f(n) = 0.6g(n) + 0.4h(n)$?
(c) Does this best-first search algorithm incur lower, same, or higher search cost (i.e., number of EXPANDED nodes) that with the evaluation function $f(n) = 0.6g(n) + 0.4h(n)$? You don't have to explain your answer.

Solution:

a) Bucharest, Fagaras, Sibiu

b) higher solution path cost

c) lower search cost

3. (2 points) Consider the best-first search algorithm using GRAPH SEARCH with the evaluation function $f(n) = (1 - \omega)g(n) + \omega h(n)$ for **ANY CONSISTENT** heuristic function h where $\omega \in [0, 1]$.
(a) State the value of ω such that this best-first search algorithm reduces to greedy best-first search.
(b) State the value of ω such that this best-first search algorithm reduces to uniform-cost search given in Figure 3.14 of AIMA 3rd edition (reproduced in Figure 7)? You don't have to explain your answer.

Solution: a) $\omega = 1$ $f(n) = (1-1)g(n) + 1h(n) \equiv f(n) = h(n)$

b) $\omega = 0$ $f(n) = (1-0)g(n) + 0h(n) \equiv f(n) = g(n)$

4. (6 points) Consider the best-first search algorithm using GRAPH SEARCH with the evaluation function $f(n) = (1 - \omega)g(n) + \omega h(n)$ for **ANY CONSISTENT** heuristic function h where $\omega \in [0, 1]$.
(a) Prove that if $\omega \in [0, 0.5]$, then $f(n)$ is non-decreasing along any path.
(b) For any $\omega \in [0, 0.5]$, state whether this best-first search algorithm is optimal.

Solution:

$$f(n') = (1-\omega)g(n') + \omega h(n')$$
$$= (1-\omega)g(n) + (1-\omega)\,c(n, a, n') + \omega h(n')$$
$$\geq (1-\omega)g(n) + \omega\, c(n, a, n') + \omega h(n')$$
$$\geq (1-\omega)g(n) + \omega h(n)$$
$$= f(n)$$

Given $\omega \in [0, 0.5]$, implies $(1-\omega) \geq \omega$. Thus, second inequality is due to the consistency of $h(n)$

b) Yes.

Solution:

END OF PAPER