CS 6613
HW 2

Fall 2023
E. K. Wong

Total # questions = 7. Total # points = 120.

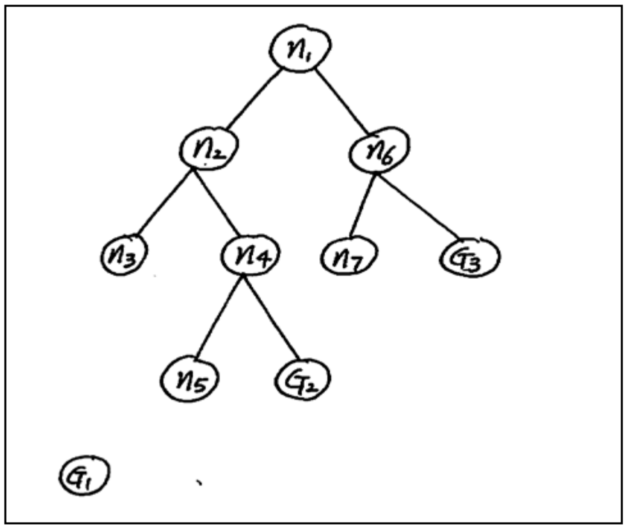
1. [10 points] Answer *True* or *False* to each of parts (a) to (d) below. No need to explain your answers.

(a) In order for Weighted A* search to generate fewer nodes than regular A* search, $W \times h(n)$ in the evaluation function $f(n) = g(n) + W \times h(n)$ must be admissible.
(b) A* search is *optimally efficient* means A* will always find the optimal solution for a problem if the heuristic function used is admissible.
(c) Compared with the *Minimax* algorithm, the *Alpha-beta* search algorithm is more efficient but the solution (best action) that it finds is not always optimal.
(d) In A* search, if the heuristic function used is *consistent*, the value of the evaluation function $f(n)$ always increases or stay the same along any path in the search tree from the root node to a leaf node.

2. [15 points] Answer *True or False* to each of parts (a) to (f) below. No need to explain your answers. (*Note*: for each part, an equation or inequality is false if it cannot be proven true based on the information given.)

A search* is used to find the optimal solution of a problem and the following tree is generated during the search. The heuristic function *h(n)* used in the *A** search is both *admissible* and *consistent*. The goal is to find a solution path from root node n_1 to optimal goal node G_1. In the following, *f* is the evaluation function used in the *A** search, *h* is the heuristic function, *g* is the path cost of a node and *c* is the step cost from a parent node to a child node. Nodes n_6 and n_7 are along the optimal path to the optimal goal node G_1. Leaf nodes G_2 and G_3 are suboptimal goal nodes.

(a) $g(n_7) + h(n_7) \geq f(G_1)$
(b) $g(n_5) > g(n_7)$
(c) $h(n_1) \leq h(n_2) + c(n_1, n_2)$
(d) $g(G_2) > f(G_1)$
(e) $g(n_5) \geq g(n_2)$
(f) $f(n_7) < f(G_3)$



3. [15 points] For the travelling problem in Figure 1 below (on page 4,) we would like to search for a travel path between the start city of *Arad* and the goal city of *Bucharest.* The estimated straight line distance from each city to the goal city *Bucharest* is given in Figure 2 (on page 4.) Assume that all roads on the map are toll roads and it costs 0.1 USD per kilometer to travel on them. Apply the A* search algorithm with *tree search* (allows repeated states) to find an optimal solution that will minimize the amount of toll we have to pay.

a) Propose a heuristic function for this problem.
b) Draw the search tree produced and indicate the order nodes are expanded by putting a number in the upper-right corner of the nodes and write the *f(n)* value to the left of each node generated. Highlight the solution path in your tree. When drawing the tree, you can use the first letter (in capital) of the city names to represent the nodes.

4. [20 points] For the same travelling problem in Question 3 above:

a) Apply the *weighted* A* algorithm with $W = 1.1$ to solve the problem. Let $h(n)$ be the heuristic function you proposed in 4(a) above. Draw the search tree produced and indicate the order nodes are expanded by putting a number in the upper-right corner of the nodes and write the *f(n)* value to the left of each node generated. Highlight the solution path in your tree. When drawing the tree, you can use the first letter (in capital) of the city names to represent the nodes.
b) Is $W \times h(n)$ with $W = 1.1$ an admissible heuristic function in this problem?
c) Is the solution found by weighted A* in part (a) above optimal?
d) Compared with the regular A* algorithm, is *weighted* A* with $W = 1.1$ more efficient (generates fewer nodes) in this example?

5. [20 points] Consider the 8-puzzle problem with start and goal states as shown below, where an adjacent tile can move *left, right, up* or *down* into the blank position.

a) Use *A** search with graph search (no repeated states) to find a solution. Use *total number of misplaced tiles* as heuristic function. Draw the search tree produced and indicate the order nodes are *expanded* by putting a number in the upper-right corner of each node expanded. Write the *f(n)* value to the left of each node *generated*. Highlight the solution path in the tree.

1	2	3
8		4
7	6	5

Start

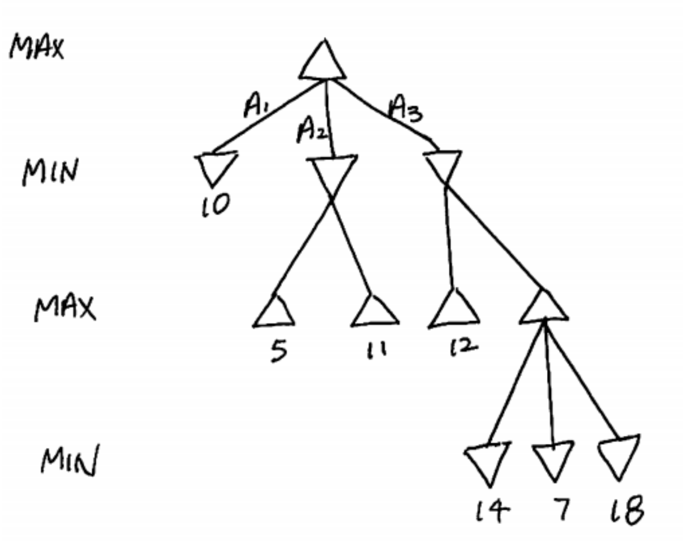
2	8	3
1		4
7	6	5

Goal

b) Assume that the Best-first search algorithm in Figure 3 below (on page 5) is used to implement *A** search in this problem. How many entries are in the table *reached* when the algorithm stops in part (a) above? (*Note*: an entry consists of a *key* and a *value*.)

6. [20 points] In A* search, prove that if a heuristic function *h(n)* is *consistent*, it must be *admissible*. [*Hint*: apply the definition of consistent functions repeatedly along the optimal path from node *n* to optimal goal node *G*.]

7. [20 points] The M*inimax* algorithm was applied to generate the game tree below for a two-player game. The utility values for the terminal nodes are as indicated in the tree. In the game tree, the *Max* player makes the first move. Show the game tree generated if the *Alpha-beta* search algorithm in Figure 4 (on page 5) is used. Write the final *alpha* value in the upper-left corner and the final *beta* value in the upper-right corner of each node generated in the tree. Indicate the order nodes are generated by putting a number inside the nodes. Also, put down the final *v* value to the right of each non-terminal node. What is the *best action* returned by the *Alpha-beta* search algorithm?


MAX
MIN
MAX
MIN
A1
A2
A3
10
5
11
12
14
7
18

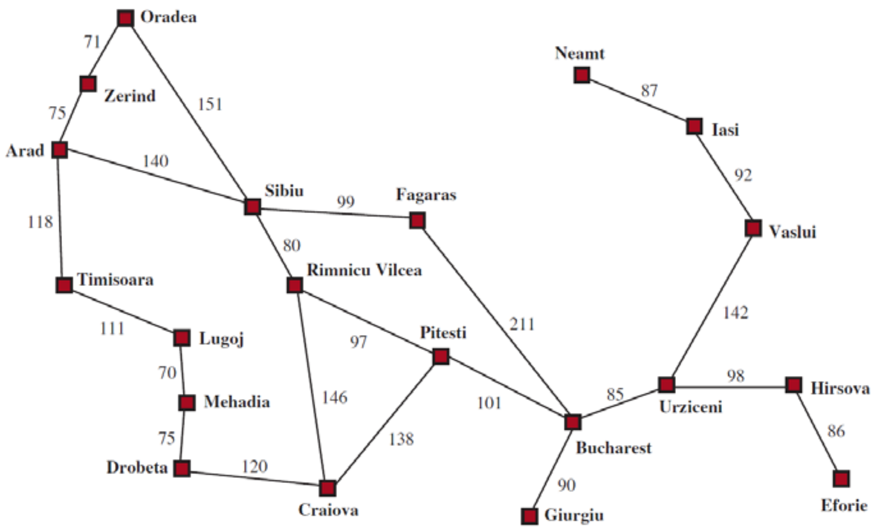
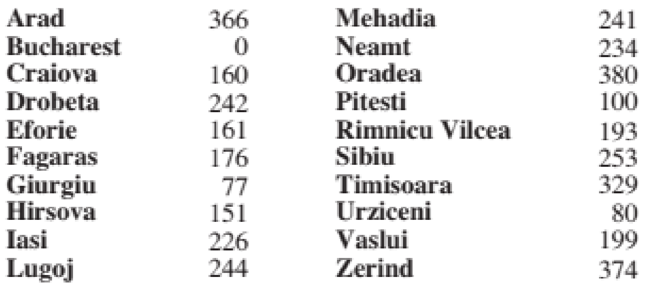


Figure 1. Map of Romania. Numbers on the map represent distances in kilometers.



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

Figure 2. Estimated straight line distances (in kilometers) to goal city Bucharest

```
function BEST-FIRST-SEARCH($problem, f$) returns a solution node or $failure$
  $node \leftarrow$ NODE(STATE=$problem$.INITIAL)
  $frontier \leftarrow$ a priority queue ordered by $f$, with $node$ as an element
  $reached \leftarrow$ a lookup table, with one entry with key $problem$.INITIAL and value $node$
  while not IS-EMPTY($frontier$) do
    $node \leftarrow$ POP($frontier$)
    if $problem$.IS-GOAL($node$.STATE) then return $node$
    for each $child$ in EXPAND($problem$, $node$) do
      $s \leftarrow child$.STATE
      if $s$ is not in $reached$ or $child$.PATH-COST $<$ $reached[s]$.PATH-COST then
        $reached[s] \leftarrow child$
        add $child$ to $frontier$
  return $failure$

function EXPAND($problem, node$) yields nodes
  $s \leftarrow node$.STATE
  for each $action$ in $problem$.ACTIONS($s$) do
    $s' \leftarrow problem$.RESULT($s, action$)
    $cost \leftarrow node$.PATH-COST + $problem$.ACTION-COST($s, action, s'$)
    yield NODE(STATE=$s'$, PARENT=$node$, ACTION=$action$, PATH-COST=$cost$)
```

Figure 3. Best-first search algorithm.

```
function ALPHA-BETA-SEARCH($game$, $state$) returns an action
  player $\leftarrow game$.TO-MOVE($state$)
  $value$, $move \leftarrow$ MAX-VALUE($game$, $state$, $-\infty, +\infty$)
  return $move$

function MAX-VALUE($game$, $state, \alpha, \beta$) returns a ($utility$, $move$) pair
  if $game$.IS-TERMINAL($state$) then return $game$.UTILITY($state$, $player$), $null$
  $v \leftarrow -\infty$
  for each $a$ in $game$.ACTIONS($state$) do
    $v2$, $a2 \leftarrow$ MIN-VALUE($game$, $game$.RESULT($state$, $a$), $\alpha, \beta$)
    if $v2 > v$ then
      $v$, $move \leftarrow v2$, $a$
      $\alpha \leftarrow$ MAX($\alpha$, $v$)
    if $v \geq \beta$ then return $v$, $move$
  return $v$, $move$

function MIN-VALUE($game$, $state, \alpha, \beta$) returns a ($utility$, $move$) pair
  if $game$.IS-TERMINAL($state$) then return $game$.UTILITY($state$, $player$), $null$
  $v \leftarrow +\infty$
  for each $a$ in $game$.ACTIONS($state$) do
    $v2$, $a2 \leftarrow$ MAX-VALUE($game$, $game$.RESULT($state$, $a$), $\alpha, \beta$)
    if $v2 < v$ then
      $v$, $move \leftarrow v2$, $a$
      $\beta \leftarrow$ MIN($\beta$, $v$)
    if $v \leq \alpha$ then return $v$, $move$
  return $v$, $move$
```

Figure 4. The Alpha-Beta Search Algorithm