CIS 471/571: Introduction to Artificial Intelligence, Winter 2019

MIDTERM SOLUTION

- You have approximately 80 minutes.
- The exam is open book.

First Name	
Last Name	
UID	

Q1. Search [32 points]

(a) True or False or Multiple Choices (mark your answer) [11 points]

1. **[1 pt]** Iterative deepening involves re-running breadth-first search repeatedly.
   (True False)
   False

2. **[1 pt]** Greedy search can return optimal solutions.
   (True False)
   True

3. **[2 pts]** Which of the following search algorithms returns the optimal path if the costs are all are a fixed cost $C > 0$? Mark all that apply.

   UCS BFS DFS Iterative Deepening

4. **[1 pt]** The sum of several admissible heuristics is still an admissible heuristic.
   (True False
   False

5. **[1 pt]** Simulated annealing ensures that you will always reach a global optimum.
   (True False)
   False

6. **[1 pt]** Admissibility of a heuristic for A^* search implies consistency as well.
   (True False)
   False

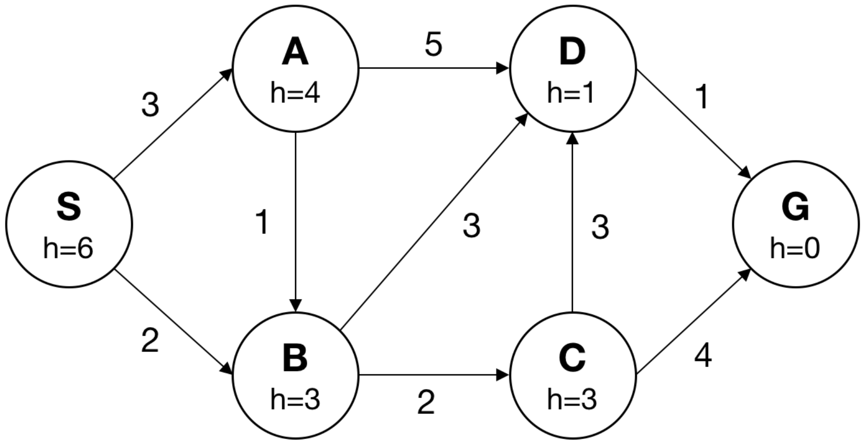
7. **[1 pt]** A^* graph search is guaranteed to expand no more nodes than depth-first graph search.
   (True False)
   False. Depth-first graph search could, for example, go directly to a sub-optimal solution.

8. **[1 pt]** A^* graph search is guaranteed to expand no more nodes than uniform-cost graph search.
   (True False)
   True. The heuristic could help to guide the search and reduce the number of nodes expanded. In the extreme case where the heuristic function returns zero for every state, A* and UCS will expand the same number of nodes. In any case, A* with a consistent heuristic will never expand more nodes than UCS.

9. **[1 pt]** Depth first search will always expand at-least as many nodes as A^* search with a consistent heuristic.
   (True False)
   False

10. **[1 pt]** If $h_1(s)$ is a consistent heuristic, and $h_2(s)$ is an admissible heuristic, then the minimum of the two must be consistent.
    (True False)
    False

(b) Search Algorithms [12 points] Consider the following graph. For the following sub-questions, ties are broken in alphabetical order.



1. **[1.5 pts]** What path would DFS return?
   S-A-B-C-D-G

2. **[1.5 pts]** What path would BFS return?
   S-A-D-G

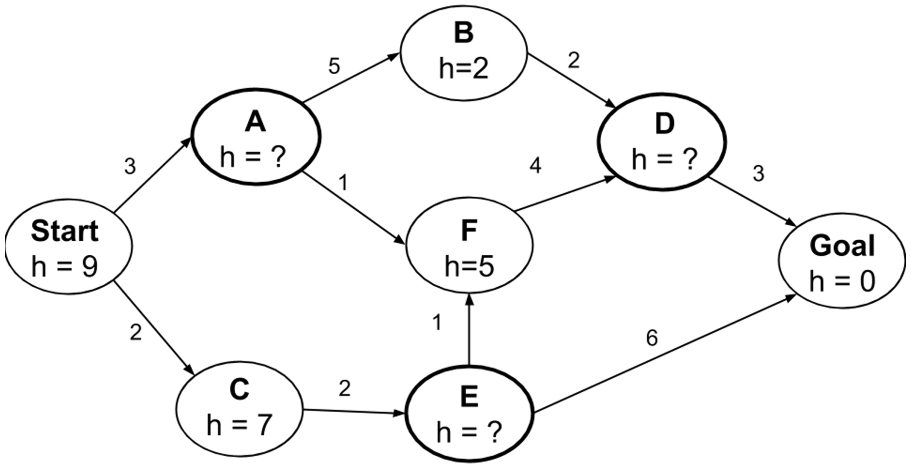
3. **[1.5 pts]** What path would UCS return?
   S-B-D-G

4. **[1.5 pts]** What path would Greedy Search return?
   S-B-D-G

5. **[3 pts]** Is the heuristic in the above graph admissible? If not, provide a minimal set of edges whose costs must be changed along with their new costs in order to make the heuristic admissible.
   Yes.

6. **[3 pts]** Is the heuristic in the above graph consistent? If not, provide a minimal set of edges whose costs must be changed along with their new costs in order to make the heuristic consistent.
   No. Change the cost of S-B to something equal to or above 3.

(c) Missing Heuristic Values [9 points]. Consider the state space graph shown below in which some of the states are missing a heuristic value. Determine the possible range for each missing heuristic value so that the heuristic is admissible and consistent. If this isn't possible, write so.



State	Range for $h(s)$
A	$6 \leq h(A) \leq 6$
D	$1 \leq h(D) \leq 3$
E	$5 \leq h(E) \leq 6$

We only need to check for consistency since admissibility is implied by consistency. A consistent heuristic is one such that $h(s) \leq c(s, s') + h(s')$. For the search graph above, this means that:

$$\begin{aligned}
&\text{For } s = A: \\
&\qquad h(Start) \leq c(Start, A) + h(A) \implies 6 \leq h(A) \\
&\qquad h(A) \leq c(A, B) + h(B) \implies h(A) \leq 7 \\
&\qquad h(A) \leq c(A, F) + h(F) \implies h(A) \leq 6 \\
&\text{For } s = D: \\
&\qquad h(D) \leq c(D, G) + h(G) \implies h(D) \leq 3 \\
&\qquad h(B) \leq c(B, D) + h(D) \implies 0 \leq h(D) \\
&\qquad h(F) \leq c(F, D) + h(D) \implies 1 \leq h(D) \\
&\text{For } s = E: \\
&\qquad h(E) \leq c(E, G) + h(G) \implies h(E) \leq 6 \\
&\qquad h(C) \leq c(C, E) + h(E) \implies 5 \leq h(E) \\
&\qquad h(E) \leq c(E, F) + h(F) \implies h(E) \leq 6.
\end{aligned}$$

Q2. CSP: Air Traffic Control [23 points]

We have five planes: A, B, C, D, and E and two runways: international and domestic. We would like to schedule a time slot and runway for each aircraft to **either** land or take off. We have four time slots: $\{1, 2, 3, 4\}$ for each runway, during which we can schedule a landing or take off of a plane. We must find an assignment that meets the following constraints:

- Plane B has lost an engine and must land in time slot 1.
- Plane D can only arrive at the airport to land during or after time slot 3.
- Plane A is running low on fuel but can last until at most time slot 2.
- Plane D must land before plane C takes off, because some passengers must transfer from D to C.
- No two aircrafts can reserve the same time slot for the same runway.

(a) [**5 pts**] Complete the formulation of this problem as a CSP in terms of variables, domains, and constraints (both unary and binary). Constraints should be expressed implicitly using mathematical or logical notation rather than with words.

Variables: A, B, C, D, E for each plane

Domains: a tuple *(runway type, time slot)* for runway type $\in \{international, domestic\}$ and time slot $\{1, 2, 3, 4\}$

Constraints:

$$B[1] = 1$$
$$D[1] \geq 3$$
$$A[1] \leq 2$$
$$D[1] < C[1]$$
$$A \neq B \neq C \neq D \neq E$$

(b) For the following subparts, we add the following two constraints:

- Planes A, B, and C cater to international fights and can only use the international runway.
- Planes D and E cater to domestic fights and can only use the domestic runway.

(i) [**3 pts**] With the addition of the two constraints above, we completely reformulate the CSP. You are given the variables and domains of the new formulation. Complete the constraint graph for this problem given the original constraints and the two added ones.

Variables: A, B, C, D, E for each plane.

Domains: $\{1, 2, 3, 4\}$

Explanation of Constraint Graph: We can now encode the runway information into the identity of the variable, since each runway has more than enough time slots for the planes it serves. We represent the non-colliding time slot constraint as a binary constraint between the planes that use the same runways.

$$\begin{tikzcd} A \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] & B \arrow[dl, leftrightarrow] \\ C \arrow[r, leftrightarrow] & D \arrow[d, leftrightarrow] \\ & E \end{tikzcd}$$

(ii) **[6 pts]** What are the domains of the variables after enforcing arc-consistency? Begin by enforcing unary constraints. (Cross out values that are no longer in the domain.)

A	1	2	3	4
B	1	2	3	4
C	1	2	3	4
D	1	2	3	4
E	1	2	3	4

A	1	2	3	4
B	1	2	3	4
C	1	2	3	4
D	1	2	3	4
E	1	2	3	4

(iii) **[6 pts]** Arc-consistency can be rather expensive to enforce, and we believe that we can obtain faster solutions using only **forward-checking** on our variable assignments. Using the Minimum Remaining Values heuristic, perform backtracking search on the graph, breaking ties by picking lower values and characters first. List the *(variable; assignment)* pairs in the order they occur (including the assignments that are reverted upon reaching a dead end). Enforce unary constraints before starting the search.

Answers. (B, 1), (A, 2), (C, 3), (C, 4), (D, 3), (E, 1)

(c) **[3 pts]** Suppose we have just one runway and n planes, where no two planes can use the runway at once. We are assured that the constraint graph will always be tree-structured and that a

A	1	2	3	4
B	1	2	3	4
C	1	2	3	4
D	1	2	3	4
E	1	2	3	4

(You don't have to use this table, it won't be graded.)

solution exists. What is the runtime complexity in terms of the number of planes, n, of a CSP solver that runs arc-consistency and then assigns variables in a topological ordering?

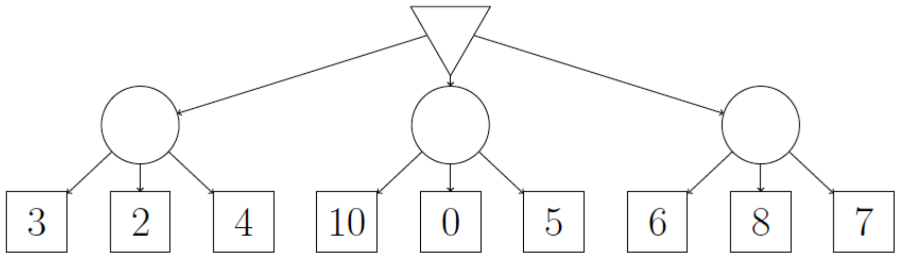
$O(1)$ $O(n)$ $O(n^2)$ $O(n^3)$ $O(n^n)$ None of the above

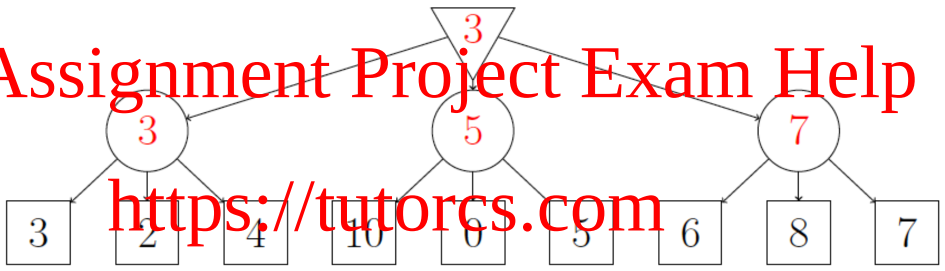
$O(n^3)$. Modified AC-3 for tree-structured CSPs runs arc-consistency backwards and then assigns variables in forward topological (linearized) ordering so we that we don't have to backtrack. The runtime complexity of modified AC-3 for tree-structured CSPs is $O(nd^2)$, but note that the domain of each variable must have a domain of size at least n since a solution exists.

Q3. Expectimin [21 points]

In this problem we model a game with a minimizing player and a random player. We call this combination "expectimin" to contrast it with expectimax with a maximizing and random player. Assume all children of expectation nodes have equal probability and sibling nodes are visited left to right for all parts of this question.

(a) **[3 pts]** Fill out the "expectimin" tree below.





The circles are expectation nodes and should be filled with the average of their children, and the triangle is the minimizer, which selects the minimum child value.

(b) **[4.5 pts]** Suppose that before solving the game we are given additional information that all values are non-negative and all nodes have exactly 3 children. Which leaf nodes in the tree above can be pruned with this information?

Leaf nodes 0, 5, 7. After the first expectation is determined to be 3, the minimizer is bounded to be ≤ 3. After the second expectation node sees 10 as its first child, it knows its value will $\geq 10/3$. Since these intervals do not overlap, we know the second expectation node will not be chosen by the minimizer and the rest of its children can be pruned. The third expectation node is bounded to $\geq 6/3 = 2$ after seeing 6, which still overlaps with ≤ 3, so we look at 8, after which the expectation is bounded to $\geq (6+8)/3 = 14/3$. Now the possible intervals are disjoint, so the last child can be pruned.

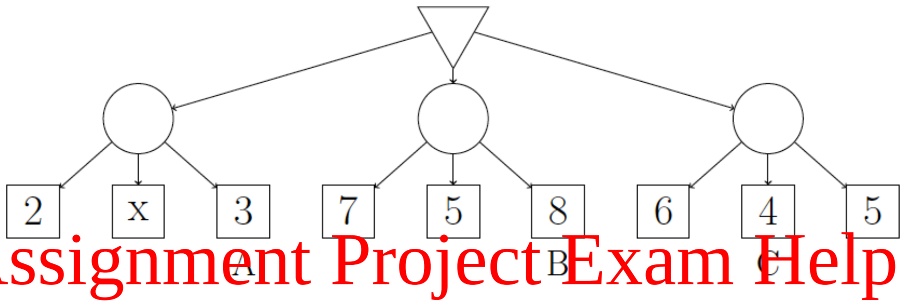
(c) **[4.5 pts]** In which of the following other games can we also use some form of pruning?

- Expectimax
- Expectimin
- Expectimax with all non-negative values and known number of children

- Expectimax with all non-positive values and known number of children
- Expectimin with all non-positive values and known number of children

Expectimax and expectimin can't be pruned in general since any single child of an expectation node can arbitrarily change its value. Expectimax has maximizer nodes that will accumulate lower bounds, so the value ranges must help us give upper bounds on the expectations, which means the values must be bounded from above. Expectimin with non-positive values does not allow pruning since both the minimizer and expectation nodes will accumulate upper bounds.

(d) **[9 pts]** For each of the leaves labeled A, B, and C in the tree below, determine which values of x will cause the leaf to be pruned, given the information that all values are non-negative and all nodes have 3 children. Assume we do not prune on equality.



- A: None. No value of x can cause A to be pruned since the minimizer does not have any bound until after A is visited.

- B: $x < 7$. To prune B, the value of the first expectation node must be less than the value of the second even if B were 0.

$$\frac{2+x+3}{3} < \frac{7+5}{3}$$
$$\implies x < 7$$

- C: $x < 1$. To prune C, the value of the first expectation node must have value less than the value of the third if both of the last two leaves had value 0.

$$\frac{2+x+3}{3} < \frac{6}{3}$$
$$\implies x < 1$$

Q4. One Wish Pacman [24 points]

(a) **Power Search.** Pacman has a special power: once in the entire game when a ghost is selecting an action, Pacman can make the ghost choose any desired action instead of the min-action which the ghost would normally take.

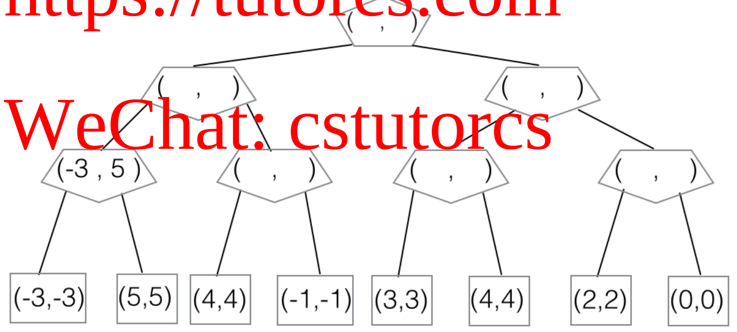
The ghosts know about this special power and act accordingly.

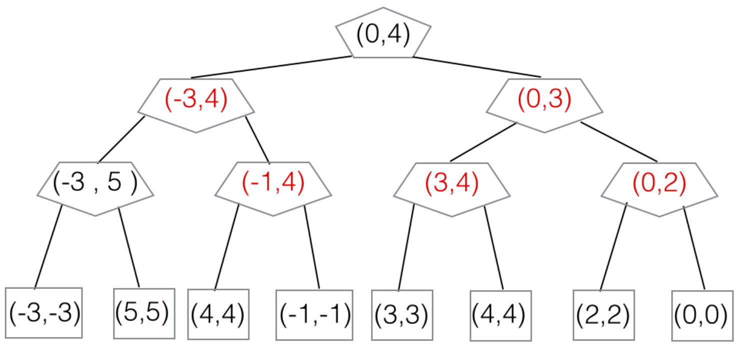
(i) [**4 pts**] Similar to the minimax algorithm, where the value of each node is determined by the game subtree hanging from that node, we define a value pair (u, v) for each node: u is the value of the subtree if the power is not used in that subtree; v is the value of the subtree if the power is used once in that subtree. For example, in the below subtree with values $(-3, 5)$, if Pacman does not use the power, the ghost acting as a minimizer would choose -3; however, with the special power, Pacman can make the ghost choose the value more desirable to Pacman, in this case 5.

Reminder: Being allowed to use the power once during the game is different from being allowed to use the power in only one node in the game tree below. For example, if Pacman's strategy was to always use the special power on the second ghost then that would only use the power once during execution of the game, but the power would be used in four possible different nodes in the game tree.

For the terminal states we set $u = v = Utility(State)$.

Fill in the (u, v) values in the modified minimax tree below. Pacman is the root and there are two ghosts.





(ii) **[8 pts]** Complete the algorithm below, which is a modification of the minimax algorithm, to work in the general case: Pacman can use the power at most once in the game but Pacman and ghosts can have multiple turns in the game.

```
function VALUE($state$)
    if $state$ is leaf then
        $u \leftarrow$ UTILITY($state$)
        $v \leftarrow$ UTILITY($state$)
        return $(u, v)$
    end if
    if $state$ is Max-Node then
        return MAX-VALUE($state$)
    else
        return MIN-VALUE($state$)
    end if
end function

function MAX-VALUE($state$)
    $uList \leftarrow [\,], vList \leftarrow [\,]$
    for $successor$ in SUCCESSORS($state$) do
        $(u', v') \leftarrow$ VALUE($successor$)
        $uList$.append($u'$)
        $vList$.append($v'$)
    end for
    $u \leftarrow \max(uList)$
    $v \leftarrow \max(vList)$
    return $(u, v)$
end function

function MIN-VALUE($state$)
    $uList \leftarrow [\,], vList \leftarrow [\,]$
    for $successor$ in SUCCESSORS($state$) do
        $(u', v') \leftarrow$ VALUE($successor$)
        $uList$.append($u'$)
        $vList$.append($v'$)
    end for

    $u \leftarrow$ ____________________

    $v \leftarrow$ ____________________

    return $(u, v)$
end function
```

$u \leftarrow \min(uList)$

$v \leftarrow \max(\max(uList), \min(vList))$

The u value of a min-node corresponds to the case if Pacman does not use his power in the game subtree hanging from the current min-node. Therefore, it is equal to the minimum of the u values of the children of the node.

The v value of the min-node corresponds to the case when pacman uses his power once in the subtree. Pacman has two choices here - a) To use the power on the current node, or b) To use the power further down in the subtree.

In case a), the value of the node corresponds to choosing the best among the children's u values = max(uList) (we consider u values of children as Pacman is using his power on this node and therefore, cannot use it in the subtrees of the node's children).

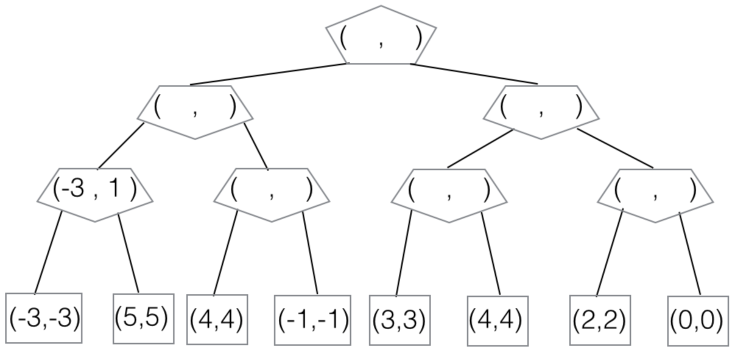
In case b), Pacman uses his power in one of the child subtrees so we consider the v values of the children, Since Pacman is not using his power on this node, the current node acts as a minimizer, making the value in case b) = min(vList)

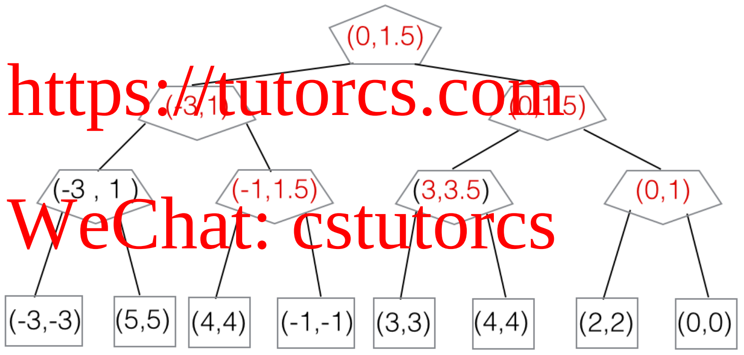
The v value at the current node is the best of the above two cases.

(b) **Weak-Power Search.** Now, rather than giving Pacman control over a ghost move once in the game, the special power allows Pacman to once make a ghost act randomly. The ghosts know about Pacman's power and act accordingly.

(i) [**4 pts**] The propagated values (u, v) are defined similarly as in the preceding question: u is the value of the subtree if the power is not used in that subtree; v is the value of the subtree if the power is used once in that subtree.

Fill in the (u, v) values in the modified minimax tree below, where there are two ghosts.





(ii) **[8 pts]** Complete the algorithm below, which is a modification of the minimax algorithm, to work in the general case: Pacman can use the weak power at most once in the game but Pacman and ghosts can have multiple turns in the game.

Hint: you can make use of a min, max, and average function.

```
function VALUE($state$)
    if $state$ is leaf then
        $u \leftarrow$ UTILITY($state$)
        $v \leftarrow$ UTILITY($state$)
        return $(u, v)$
    end if
    if $state$ is Max-Node then
        return MAX-VALUE($state$)
    else
        return MIN-VALUE($state$)
    end if
end function

function MAX-VALUE($state$)
    $uList \leftarrow [\,], vList \leftarrow [\,]$
    for $successor$ in SUCCESSORS($state$) do
        $(u', v') \leftarrow$ VALUE($successor$)
        $uList$.append($u'$)
        $vList$.append($v'$)
    end for
    $u \leftarrow \max(uList)$
    $v \leftarrow \max(vList)$
    return $(u, v)$
end function
```

```
function MIN-VALUE($state$)
    $uList \leftarrow [\,], vList \leftarrow [\,]$
    for $successor$ in SUCCESSORS($state$) do
        $(u', v') \leftarrow$ VALUE($successor$)
        $uList$.append($u'$)
        $vList$.append($v'$)
    end for

    $u \leftarrow$ ____________________

    $v \leftarrow$ ____________________

    return $(u, v)$
end function
```

$u \leftarrow \min(uList)$

$v \leftarrow \max(avg(uList), \min(vList))$

The solution to this scenario is same as before, except that when considering case a) for the v value of a min-node, the value of the node corresponds to choosing the average of the children's u values = avg(uList)

Q5. (Grads only) Graph Search [10 points]

You are trying to plan a road trip from city A to city B. You are given an **undirected graph** of roads of the entire country, together with the distance along each road between any city X and any city Y : $length(X, Y)$ (For the rest of this question, "shortest path" is always in terms of *length*, not number of edges). You would like to run a search algorithm to find the shortest way to get from A to B (assume no ties).

Suppose C is the capital, and thus you know the shortest paths from city C to every other city, and you would like to be able to use this information.

Let $path_{opt}(X \to Y)$ denote the shortest path from X to Y and let $cost(X, Y)$ denote the cost of the shortest path between cities X and Y. Let $[path(X \to Y), path(Y \to Z)]$ denote the concatenation.

(a) **[5 pts]** Suppose the distance along any edge is 1. You decide to initialize the queue with A, plus a list of all cities X, with $path(A \to X) = [path_{opt}(A \to C), path_{opt}(C \to X)]$. You run BFS with this initial queue (sorted in order of path length). Which of the following is correct? (Select all that apply)

- You always expand the exact same nodes as you would have if you ran standard BFS.
- You might expand a different set of nodes, but still find the shortest path.
- You might expand a different set of nodes, and find the sub-optimal path.

Consider a graph of 5 nodes: A,B,C,D,E and edges (A,C), (C,E), (E,B), (A,D),(D,B). Then our initial queue (in order) is

(a) C: A-C
(b) E: A-C-E
(c) B: A-C-E-B
(d) D: A-C-A-D

The path returned will be A-C-E-B

(b) **[5 pts]** You decide to initialize priority queue with A, plus a list of all cities X, with $path(A \to X) = [path_{opt}(A \to C), path_{opt}(C \to X)]$, and $cost(A, X) = cost(A, C) + cost(C, X)$. You run UCS with this initial priority queue. Which of the following is correct? (Select all that apply)

- You always expand the exact same nodes as you would have if you ran standard UCS.
- You might expand a different set of nodes, but still find the shortest path.
- You might expand a different set of nodes, and find the sub-optimal path.

Regardless of what is on the queue, UCS will explore nodes in order of their shortest-path distance to A, so the set of explored nodes is always {nodes X: dist(A,X) less than dist(A,B)}

INTENTIONALLY BLANK