CSE 415 Winter 2025 **Assignment 3: Solutions**

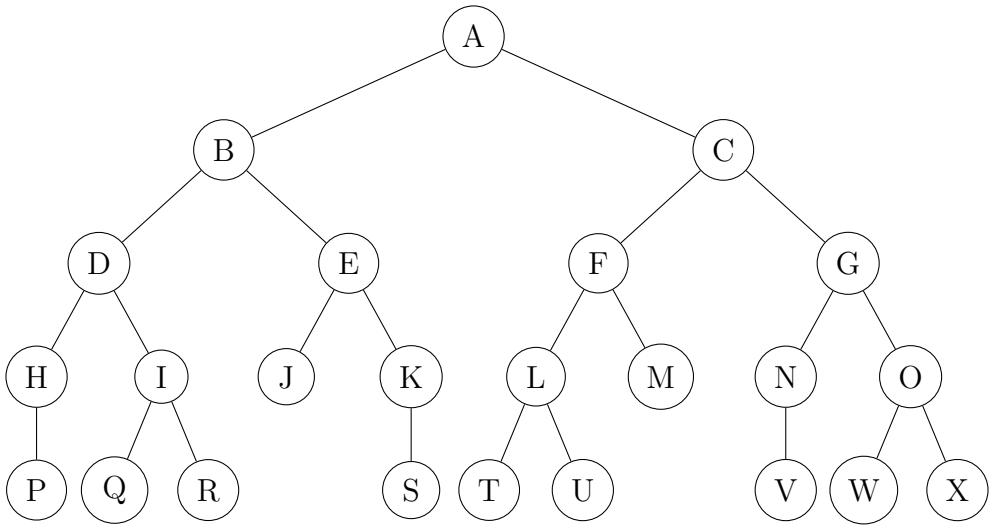
PLEASE DO NOT SHARE OUTSIDE OF CSE 415 IN WINTER 2025

(Version 250205)

---

1 Basic Search

Use the following tree to answer the questions below comparing Breadth First Search, Depth First Search, and Iterative-Deepening Depth First Search. Assume that children are visited from left to right.



(a) (2 points) Write out the order that the nodes are expanded in using Breadth First Search, starting from A and searching to N.

A, B, C, D, E, F, G, H, I, J, K, L, M, N

(b) (2 points) Write out the order that the nodes are expanded in using Depth First Search, starting from A and searching to N.

A, B, D, H, P, I, Q, R, E, J, K, S, C, F, L, T, U, M, G, N

(c) (2 points) Write out the order that the nodes are expanded in using Iterative-Deepening Depth First Search, starting from A and searching to N. If a node is repeated, make sure to include it each time it is expanded.

A, A, B, C, A, B, D, E, C, F, G, A, B, D, H, I, E, J, K, C, F, L, M, G, N

(d) (3 points) Which of the three search algorithms (BFS, DFS, IDDFS) has the smallest maximum size of the open list while searching from A to K? What is the maximum size of the open list for that algorithm?

IDDFS, 4 or DFS, 4
For partial credit:
BFS, 9

(e) (1 points) True or False: BFS, DFS, and IDDFS will each return the same path on this tree starting at A searching to K.

True

(f) (1 points) True or False: Given the same start and goal state, BFS, DFS, and IDDFS will always return the same path for any search graph (not necessarily a tree).

False

(g) (4 points) What are the advantages of using IDDFS over DFS and BFS for very large or infinite graphs in terms of (i) memory usage, (ii) completeness, and (iii) computation time?
Memory : Like DFS, it uses memory proportional to the depth of the search, $O(d)$, not the total number of nodes ($O(b^d)$ for BFS).
Completeness : Like BFS, it ensures that the shortest path is found first. DFS does not guarantee this since it may get stuck exploring a deep, incorrect branch. Computation time: In IDDFS, the time complexity is higher compared to BFS and DFS, matching BFS in terms of compelxity but with worse constant factors due to the repeated exploration of nodes at shallower levels. However, this tradeoff is acceptable in exchange for the advantages in memory and completeness for large or infinite graphs.

(h) (4 points) In terms of complexity, explain why IDDFS is considered to have the same time complexity as BFS even though it may revist nodes multiple times.
IDDFS revisits nodes multiple times, but most of that time is at the deepest level of the search, which contains the majority of the nodes. The repeated work of revisiting nodes at shallower depths is relatively small compared to the total number of nodes at the deepest level. Therefore, the time complexity of IDDFS remains $O(b^d)$ where b is the branching factor and d is the depth.

(i) (3 points) Suppose you are running your search algorithm on an embedded system device with highly limited memory available. Which of the three search algorithms (BFS, DFS, IDDFS) would you choose to run to minimize the memory used by the algorithm? Justify your answer with a one sentence explanation.

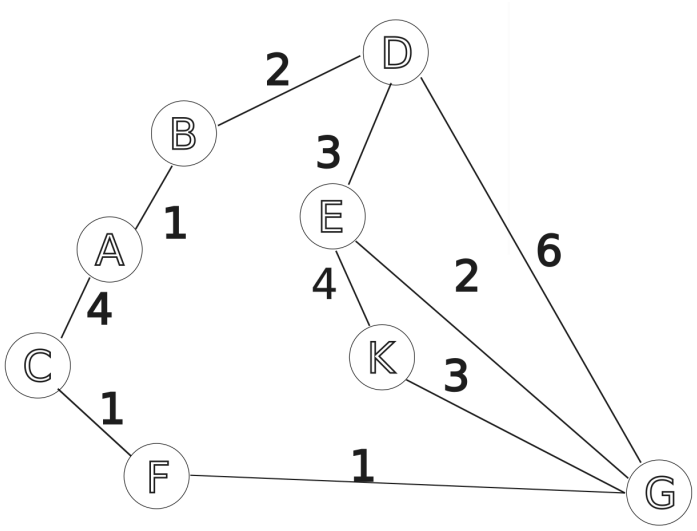
IDDFS, the maximum size of the open list will is at least as small as DFS and BFD so it will use the least memory of the options.

(j) (3 points) Suppose you need to run a search algorithm but you are confident that the vast majority of your search goals will be near the root of the tree (for example, G or earlier in the alphabet on this tree). Which of the three search algorithms (BFS, DFS, IDDFS) would you use to find the shortest path in the least amount of time given these conditions? Justify your answer with a one sentence explanation.

BFS, because it searches all the nodes level by level and will find those near the top first. IDDFS also accepted, because it will efficiently

2 A* search vs Uniform Cost

(25 points) This exercise is intended to reinforce your understanding of the mechanics of graph-search algorithms with and without heuristics. Consider the following graph, which represents cities and the distances between them. Each circle represents a city, and the edges between them represent roads from one city to another. The number next to an edge indicates the travel distance along that edge. For example, in the graph, the edge from city A to city B is labeled with 1, which means the travel distance from city A to city B is 1 unit.



The problem is to find a lowest-distance path from city A to city G. You'll solve this problem first using Uniform Cost Search (UCS) and then solve it again using A* Search. For the A* search, you are given a heuristic, which estimates the distance from each city to the goal, city G.

Heuristic (h(s)) Values for A* Search: These are the heuristic values, which estimate the distance from each city to city G. The values are used only in A* search:

city s:	A	B	C	D	E	F	G	K
heuristic $h(s)$:	7	6	2	3	1	1	0	2

(a) **Uniform Cost Search (UCS)**: Apply UCS to find a lowest-distance path from city A to city G.

- Show how the OPEN list (a.k.a. the "fringe") evolves during this search below, starting with the version before the beginning of the first iteration of the UCS main loop that is shown already with starting city A and its cumulative distance 0. Show the new version of the OPEN list at the end of each iteration. Be sure to include each city's cumulative distance (g-value). Within each list, show these pairs in order of increasing distance (even though a real priority queue doesn't have to use a sorted list in its implementation). (4 pts.)

  [(A,0)]
  After expanding City A, the following nodes are added to the OPEN list:
  OPEN list: [(B, 1), (C, 4)]
  After expanding City B:
  OPEN list: [(D, 3), (C, 4)]
  After expanding City D:
  OPEN list: [(C, 4), (E, 6), (G, 9)].
  After expanding City C:
  OPEN list: [(F, 5), (E, 6), (G, 9)].
  After expanding City F:
  OPEN list: [(E, 6), (G, 6)].
  After expanding City E: (this step can be skipped)
  OPEN list: [(G, 6), (K, 10)].
  Expand City G, finished.

- List the nodes in the order they are added to the CLOSED list (2 pt)
  CLOSED list: {A, B, D, C, F, E, G} or {A, B, D, C, F, G}

- Provide the final least-cost path and the total cost(2 pt)
  Final Path: A → C → F → G
  Total Cost: 6

(b) **A* Search**: Apply A* search with the given heuristic values to find a lowest-distance path from city A to city G.

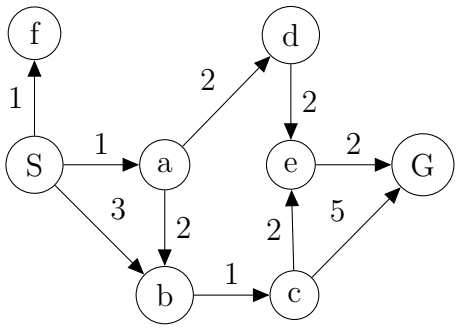
- Show how the OPEN list (a.k.a. the "fringe") evolves during this search, but use the cities' f values instead of their g values. As with UCS, show each snapshot of the OPEN list with cities ordered by associated values. (4 pts.)
  [(A,7)]
  After expanding City A, the following nodes are added to the OPEN list:
  OPEN list: [(C, 6), (B, 7)]
  After expanding City C:
  OPEN list: [(F, 6), (B, 7)]
  After expanding City F:
  OPEN list: [(G, 6), (B, 7)].
  Expand City G, finished.

- List the nodes in the order they are added to the CLOSED list (2 pts.)
  {A, C, F, G}
- Provide the final path and distance (2 pts.)
  Final Path: A → C → F → G
  Total Cost: 6

(c) **Node Expansion Comparison**: Count the number of nodes expanded in each of UCS and A* search.

- How many nodes did UCS expand? (1 pt.)
  Uniform Cost Search expanded seven nodes: A, B, D, C, F, E, G. or six nodes: A, B, D, C, F, G.
- How many did A* expand? (1 pt.)
  A* Search expanded 4 nodes: A, C, F, G.
- Which method expanded fewer nodes, and why? (1 pt.)
  A* Search expands fewer nodes because it uses the heuristic to guide the search towards the goal more efficiently, skipping unnecessary nodes like D.

3 Heuristic Search



For the following questions, consider three heuristics h_1, h_2, h_3. The table below indicates the estimated cost to goal, G, for each of the heuristics for each node in the search graph.

state (s)	S	a	b	c	d	e	f	G
heuristic $h_1(s)$	6	5	2	2	2	2	6	0
heuristic $h_2(s)$	6	6	4	5	4	1	5	0
heuristic $h_3(s)$	6	6	4	3	4	2	8	0

(a) (2 points) What does it mean for a heuristic to be "admissible"? A heuristic is admissible if it never overestimates the true cost to a nearest goal.

(b) (3 points) Which heuristics among $\{h_1, h_2, h_3\}$ shown above are admissible? For heuristics which are not admissible, identify the nodes where admissibility is violated. h_1 is admissible. h_3 is admissible. h_2 violates admissibility at node c.

(c) (2 points) What does it mean for a heuristic to be "consistent"? A heuristic is consistent if, when going from neighboring nodes a to b, the difference between the heuristics of the two nodes, never overestimates the actual cost of going from node a to node b.

(d) (3 points) Which heuristics are consistent? For heuristics which are not consistent, identify the edges where consistency is violated. h_3 is consistent. h_1 violates consistency on edges (a, b) where the weight is 2 but $h_1(a) - h_1(b) = 5 - 2 = 3 > 2$, (a, d) where the weight is 2 but $h_1(a) - h_1(d) = 5 - 2 = 3 > 2$, and (S, b) where the weight is 3 but $h_1(S) - h_1(b) = 6 - 2 = 4 > 3$. Since h_2 is not admissible, checking for consistency is arguably irrelevant, but it violates consistency on edges (d, e) and (c, e).

(e) (2 points) What does it mean for one heuristic to dominate another heuristic? For one admissible and consistent heuristic to dominate another, all of its values must be greater than or equal to the corresponding values of the other heuristic, without exceeding the actual cost to a nearest goal.

(f) (4 points) Of the heuristics provided above, which would you consider the best heuristic? Explain your reasoning in terms of admissibility, consistency, and dominance. Since the only heuristic that is both admissible and consistent is h_3, it is the best heuristic to use. In this case, the question of dominance is not relevant, as that would require comparing 2 heuristics that are admissible and consistent.

(g) (3 points) Explain why, when using the $A*$ search algorithm, it is important to continue the expansion process until the goal state is removed from the OPEN list and becomes the current state, rather than terminating as soon as the GOAL sate is found. Terminating as soon as a goal node is generated could lead to sub-optimal solution, as the goal node may have been reached by a path consisting of fewer nodes but with high-cost edges between some of those nodes. By placing the goal node on OPEN, it is removed according to its priority, as determined by its f value, thus ensuring that it is reached through the optimal path.

(h) (3 points) In A* search, if the heuristic overestimates the true distance (i.e., $h(n) \geq$ actual distance to the goal), what is the potential effect on the search? How could this affect the optimality of the solution? (2 pts.)

If the heuristic is overestimated ($h(n) \geq$ actual cost), A* may prioritize suboptimal paths, leading to a failure in finding the least-cost solution. This would violate the

optimality guarantee of A*, and the search may return a more costly path than necessary.

(i) (3 points) Explain why A* search is guaranteed to find the optimal solution when the heuristic is admissible (i.e., $h(s)$ never overestimates the actual distance to the goal). Does this apply to UCS as well?

A* is guaranteed to find the optimal solution when the heuristic is admissible (h(n) $\leq$ actual cost to the goal) because it never overestimates the true cost, ensuring the search expands the least-cost paths.
UCS, which doesn't use a heuristic (although one could think of UCS as a special case of A* where the trivial heuristic h(n) = 0 is used), also guarantees the optimal solution because it always expands the node with the least total cost from the initial node (thus, when it reaches the goal, had a cheaper solution existed, it would have already been found).

4 Adversarial Search and Alpha-Beta Pruning

Consider the Tic-Tac-Toe game tree shown in the figure.

(a) (8 points) Compute the static values of the leaf nodes in the given tree using the function $v(s)$, where v is defined below, as a weighted sum of three features.

$$v(s) = 1 \cdot f_1(s) + 10 \cdot f_2(s) + 100 \cdot f_3(s)$$

Here $f_1(s)$ counts the number times that there is a line containing an X alone minus the number of times there is a line containing an O alone. Also, $f_2(s)$ counts the number of unblocked instances of two Xs in a row, minus the number of unblocked instances of two Os in a row. Finally, $f_3(s)$ is the number of 3-in-a-row instances of Xs, minus the similar number of Os. (This is equivalent to the static evaluation function used in lecture and a worksheet.)

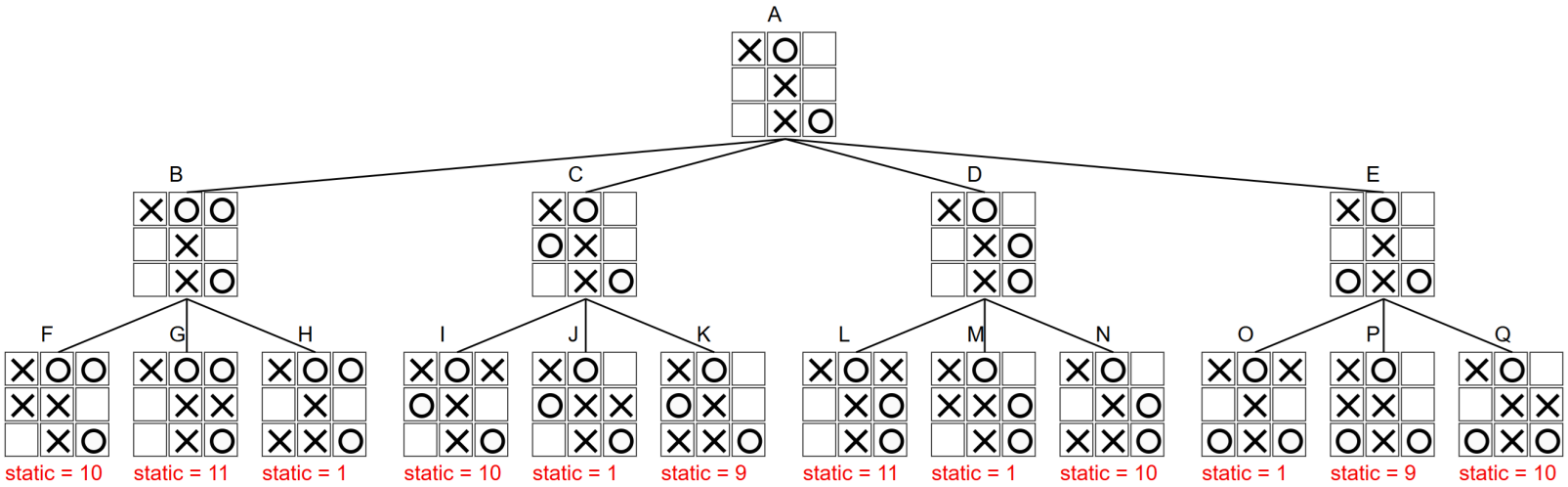
Show these values underneath states F through Q in the diagram. Hint: $v(F) = 10$.

(b) (5 points) Use straight minimax to determine the values of the internal nodes in the tree, and write those values in the table below.

node	Minimax value
A	10
B	11
C	10
D	11
E	10

Table 4.1: Minimax values for internal nodes

(c) (8 points) Next, redo the adversarial search using alpha-beta pruning on the same tree. Fill in the values in Table 4.2 below as you go. For α and β values, write the values that are passed from the state's parent to that state. For example, the value of state F will not be shown in the $\alpha - \beta$ values of state B, but would be reflected in the $\alpha - \beta$ values of state G. If a state does not have to be evaluated, do not write any values for it in the table.


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
static = 10
static = 11
static = 1
static = 10
static = 1
static = 9
static = 11
static = 1
static = 10
static = 1
static = 9
static = 10

state	value	α	β
A	10	N/A	N/A
B	11	$-\infty$	∞
C	10	$-\infty$	11
D	11	$-\infty$	10
E	10	$-\infty$	10
F	10	$-\infty$	∞
G	11	10	∞
H	1	11	∞
I	10	$-\infty$	11
J	1	10	11
K	9	10	11
L	11	$-\infty$	10
M	1		
N	10		
O	1	$-\infty$	10
P	9	1	10
Q	10	9	10

Table 4.2: State values and parameter values for α and β.

(d) (4 points) Suppose that O knows that X moves randomly choosing among the legal moves with equal probability for each choice. In order for O to choose the best move, O decides to choose the move corresponding to the minimum of the "expected values" of the resulting states. Redo the computation of backed up values of the internal nodes from (b) by using the statistical expectation of the values of states where it is X's turn to move, instead of the maximum value of its children. For each internal node B through E, show how you compute the expected value from its children's values. Assume the uniform probability distribution for three items: $[1/3, 1/3, 1/3]$. Put your expressions and values in Table 4.3.

node	expression	expected value
B	$(10+11+1)/3$	$22/3$
C	$(10+1+9)/3$	$20/3$
D	$(11+1+10)/3$	$22/3$
E	$(1+9+10)/3$	$20/3$

Table 4.3: Expected values for chance nodes.