Option in red is the correct answer

Q1. Given two admissible heuristic functions h1 and h2, h2 dominates h1 if for any state s, h1(s) <= h2(s) <= h*(s), where h*(s) is the true cost to the goal from a state s. For use in A* search, why do we prefer the dominating heuristic function h2 over h1?

A. We don't. In A* we always want to be optimistic, so a dominating heuristic is less optimistic and so may not work better. (We would like to be optimistic but not too optimistic)

B. A dominating heuristic is easier to compute since it is closer to the true heuristic. (Not necessary. Usually, a dominating heuristic is more complicated and harder to compute.)

C. A dominating heuristic is closer to the true heuristic and thus gives more accurate estimation of the costs. (h2 is closer to the true cost by definition.)

D. A dominating heuristic must have a simpler form than the other heuristic. (Not necessary. Usually, a dominating heuristic is more complicated and harder to compute.)

Q2. What is the main difference between Uninformed Search (e.g., uniform cost search) and Informed Search (e.g., A* search)?

A. Informed Search does not know the edge costs but has a guess for path costs called a heuristic. (It also knows the edge costs and knows the path costs from the initial state to the visited states.)

B. Uninformed Search does not know the edge costs but Informed does. (Uninformed search also knows the edge costs and knows the path costs from the initial state to the visited states.)

C. Informed Search has the same information as Uniformed Search but also a guess for the distance to the goal called a heuristic. (correct)

D. Informed Search has access to a probability distribution to inform most likely paths (Informed search doesn't have that in general)

Q3. Which of the following algorithm would be most likely to get stuck in a poor local optimum?

A. Hill climbing without random restart (greedy algorithm. Easy to get stuck)

B. Hill climbing with random restart (Random restart allows to explore more locations and may get better solutions)

C. Simulated annealing (has certain probability to pick random neighbors, so may escape poor local optima)

D. Genetic algorithm (keeps a population, has crossover/mutation, and thus may escape poor local optima)

Q4. Which of the following mechanism in genetic algorithm allows it to favor states with better fitness value?

A. Cross-over (No. Cross over is done without considering fitness.)

B. Mutation (No. Mutation doesn't give higher probabilities to better fitness values. The mutation is done without considering fitness.)

C. Natural selection (correct. Gives higher probabilities to those individuals with higher fitness values.)

D. Random restart (Not part of the standard Genetic algorithm. Even when used with genetic algorithms, random restart itself doesn't bias toward better fitness values.)

Q5. Consider a value function f on a finite state space. f only has one local optimum that is also the global optimum. And for any states s1 and s2, f(s1) is not equal to f(s2). When starting from the same initial state, which of the following algorithms will always reach the global optimum of f? Multiple answers are possible.

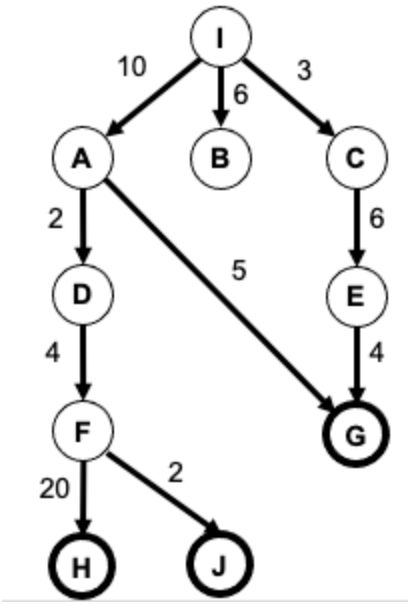
A. Hill climbing without random restart. (Yes. It only stops when reaching the global optimum.)

B. Hill climbing with random restart. (Yes. Each run will stop at the global optimum.)

C. Simulated Annealing. (Not necessary. Towards the end, the algorithm still allows some probability to go to a neighbor with worse value than the current solution. So it can escape the global optimum.)

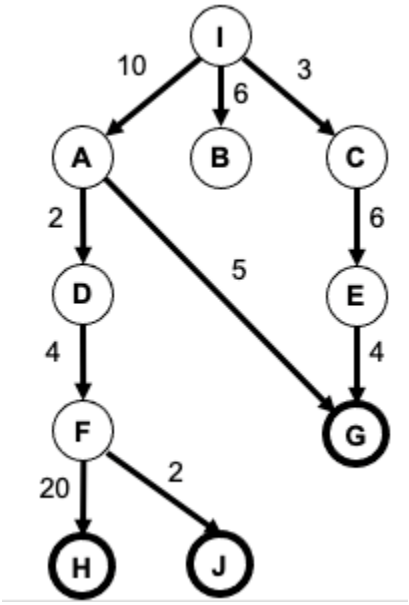
D. Genetic algorithm. (This algorithm is not guaranteed to get to even the local optimum.)

Q6. Consider the state space graph below. The initial state is I, and goal states have bold borders. Nodes are expanded left to right when there are ties. Suppose we run BFS. Write down for each iteration, the node expanded, and the fringe at the end of the iteration.



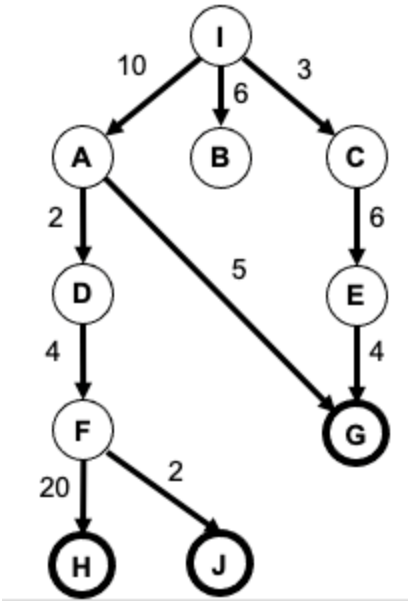
Answer:
Iteration: expanded node, fringe
1: I, [A, B, C]
2: A, [B, C, D, G]
3: B, [C, D, G]
4: C, [D, G, E]
5: D, [G, E, F]
6: G, [E, F]

Q7. Consider the state space graph below. The initial state is I, and goal states have bold borders. Nodes are expanded left to right when there are ties. Suppose we run DFS. Write down for each iteration, the node expanded, and the fringe at the end of the iteration.



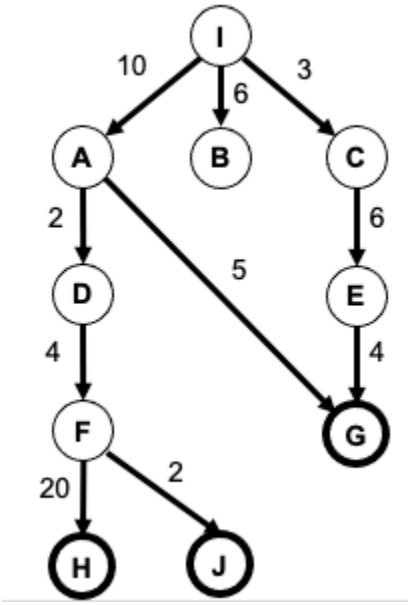
Answer:
Iteration: expanded node, fringe
1: I, [A, B, C]
2: A, [D, G, B, C]
3: D, [F, G, B, C]
4: F, [H, J, G, B, C]
5: H, [G, B, C]

Q8. Consider the state space graph below. The initial state is I, and goal states have bold borders. Nodes are expanded left to right when there are ties. Suppose we run UCS. Write down for each iteration, the node expanded, and the fringe at the end of the iteration. The fringe need not be sorted. The expanded node, and the nodes in the fringe should be in the format of (node id, cost). When there are multiple copies of a node encountered, keep the copy with the smaller cost.



Answer:
Iteration: expanded node, fringe
1: (I,0), [(A,10), (B,6), (C,3)]
2: (C,3), [(A,10), (B,6), (E,9)]
3: (B,6), [(A,10), (E,9)]
4: (E,9), [(A,10), (G, 13)]
5: (A,10), [(G,13), (D,12)]
6: (D,12), [(G,13), (F,16)]
7: (G,13), [(F,16)]

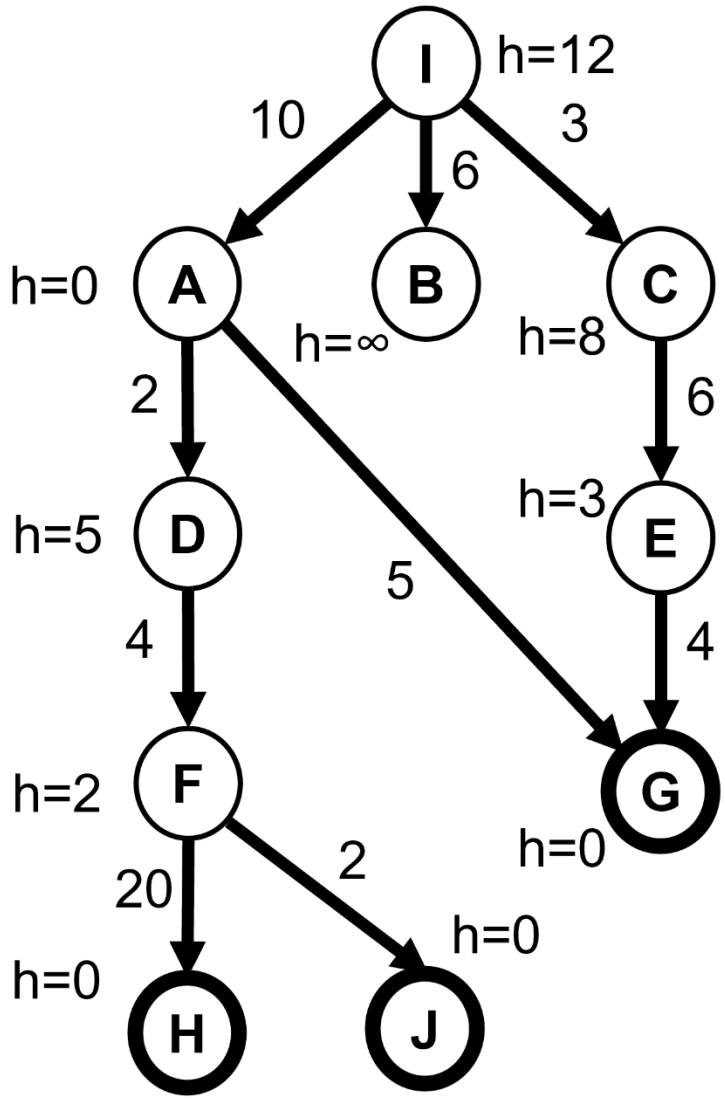
Q9. Consider the state space graph below. The initial state is I, and goal states have bold borders. Nodes are expanded left to right when there are ties. Suppose we run IDS (Iterative Deepening Search). Write down for each iteration in each stage, the node expanded, and the fringe at the end of the iteration.



Answer:
Stage 1
Iteration: expanded node, fringe
1: I, [A, B, C]
2: A, [B,C]
3: B, [C]
4: C, []

Stage 2
1: I, [A, B, C]
2: A, [D, G, B, C]
3: D, [G, B, C]
4: G, [B, C]

Q10. Consider the state space graph below, with heuristic h next to each state. The initial state is I, and goal states have bold borders. Nodes are expanded left to right when there are ties. Suppose we run A*. Write down for each iteration in each stage, the node expanded, and the fringe at the end of the iteration. The fringe need not be sorted. The expanded node, and the nodes in the fringe should be in the format of (node id, g+h value). When there are multiple copies of a node encountered, keep the copy with the smaller cost.



Answer:

Iteration: expanded node, fringe

1: (I,12), [(A, 10), (B, ∞), (C, 11)]

2: (A,10), [(B, ∞), (C, 11), (D,17), (G, 15)]

3: (C, 11), [(B, ∞), (E,12), (D,17), (G, 15)]

4: (E, 12), [(B, ∞), (D,17), (G, 13)]

5: (G,13), [(B, ∞), (D,17)]

Q11. Consider a state space with only five states and their corresponding scores (1, 3), (2, 5), (3, 2), (4, 3), (5, 1), for which the first element is the state and the second element is the score. Thus, (1, 3) indicates state 1 with a score 3. Each state i has two neighbors (i+1) and (i-1), except state 1 has only one neighbor state 2, and state 5 only has one neighbor state 4. Consider running hill-climbing with random restart to find the state with the maximum score. If we

always start or restart from a state sampled from all states with equal probability, what is the probability of finding the best state in two runs?

Answer:
The optimum state is state 2, with score 5.

If the initial state is state 1: we will reach state 2 in one step.
If the initial state is state 2: we will reach state 2 in 0 step.
If the initial state is state 3: we will reach state 2 in one step.
If the initial state is state 4: we will get stuck in the local optimum state 4, and return state 4 as the solution.
If the initial state is state 5: we will reach state 4 in one step, and then return state 4 as the solution.

So if the initial state is in {1,2,3}, then the algorithm will return the best state (state 2); otherwise not.

Since we are sample the initial state uniformly at random, each run will successfully find the best state with probability 3/5, and fail with probability 2/5.

The probability of finding the best state in two runs is $1 - 2/5*2/5 = 21/25$.