DQ2 (L2)

Due 22 Jan at 23:59 **Points** 25 **Questions** 13
Available after 16 Jan at 12:00 **Time limit** None
Allowed attempts Unlimited

Instructions

- This quiz is NOT GRADED. However, it is HIGHLY RECOMMENDED that you use these questions to complement your review of the lecture content.
- The questions are based on content from the Lecture 2 and from part of Chapter 3 the AIMA (4th Ed.) textbook (i.e., 3.1-3.4).

Take the quiz again

Attempt history

	Attempt	Time	Score
KEPT	Attempt 2	18 minutes	24.33 out of 25
LATEST	Attempt 2	18 minutes	24.33 out of 25
	Attempt 1	20 minutes	20.17 out of 25

Submitted 16 Jan at 14:40

Question 1 2 / 2 pts

Which one of the following are properties used to define a search problem?

Correct! ✓ Actions

Correct! ✓ Cost function

☐ Evaluation function

☐ Nodes

Correct! ✓ State representation

☐ Terminal states

Correct! ✓ Transition model

A search problem is defined by:

- State representation
- Initial state
- Goal test (goal states)
- Actions
- Cost function (action costs)
- Transition model

Question 2

2 / 2 pts

Determine the following search tree characteristics for an ***efficient formulation*** of Sudoku puzzle (refer to Tutorial Worksheet 1 for a description of a Sudoku puzzle).

Assume that each action corresponds to filling-in a single cell. Also, assume that the puzzle begins with 0 digits filled-in, and that the root of the search tree is depth 0.

Maximum Branching Factor (*b*): 9

Maximum Depth (*m*): 81

Depth of the shallowest goal (*d*): 81

Note: you may use ***infinite*** if applicable.

Answer 1:

Correct! 9

Answer 2:

Correct! 81

Answer 3:

Correct! 81

Since we are to assume that each action corresponds to filling-in a digit into any blank location, we have a branching factor 9*81 at the first level.

However, this is only required if the sequence of Sudoku cells being filled is important (i.e., it is important to distinguish the same assignments if those assignments were made in a different order). This is not the case for a Sudoku.

Consequently, we may design the search problem such that at each level of the search tree, we only consider filling-in a single cell of the Sudoku. This would reduce the branching factor to 9.

Finally, since 0 digits have been filled, we would get a tree of depth 81. All goal states would also be found at that level.

Can you think of alternate state representations that would improve the search complexity?

Question 3

2 / 2 pts

A state is a representation of a “physical” configuration. It represents a snapshot of the environment.

A node is not a state. A node instead corresponds to the composite data type.

Which of the following should be included in a node?

Hint: recall from the lecture the pieces of information that were necessary to improve the performance of Breadth-First and Depth-First Search (as well as the variants for Depth-First Search).

Correct! [✓ State (i.e., an abstraction or identifier for a particular state)]

Correct! [✓ Parent node]

Correct! [✓ Action (that was taken to reach this node)]

Path (that was taken to reach this node)

Correct! [✓ Depth (of this node within the search tree)]

Correct! [✓ Path cost (to reach this node from the initial state)]

State: Each node represents one state

Parent node and **action**: Recall that in order to achieve O(***m***) space complexity with Depth-First Search (DFS), we needed to store information corresponding the current action, as well as the parent node. By doing so, when perform backtracking, we need only assume a static sequence over the possible actions to determine which, if any actions, have not yet been considered, and therefore, which to use next.

Path: If we store the entire path taken at each node, we increase the space complexity of DFS since nodes at depth k would each reference a sequence of *k* nodes.

Depth: We should store the depth since this would more easily allow us to determine if the depth threshold has been reached during Depth-Limited and Iterative-Deepening Search.

Path cost: To efficiently update the path cost when extending upon the current path

Question 4

2 / 2 pts

When implementing graph search, we use an explored set.

To ensure optimal performance, the data structure used to implement the explored set is a hash table .

Answer 1:

Correct!	hash table
orrect answer	Hash Table
orrect answer	hashtable
orrect answer	Hashtable
orrect answer	hash-table
orrect answer	Hash-Table
orrect answer	reached hash table
orrect answer	Reached Hash Table
orrect answer	hashmap
orrect answer	Hashmap
orrect answer	hash map
orrect answer	Hash Map
orrect answer	hashset
orrect answer	Hashset
orrect answer	hash set
orrect answer	Hash Set

Hash table, to ensure O(1) costs.

Question 5

1 / 1 pts

Which of the following is ***true*** about graph and tree search implementations?

☐ A tree search implementation will avoid consideration of redundant paths.

Correct!

[✓ A graph search (version 2) implementation may revisit states.]

Option 1: False. A tree search implementations will explore all nodes - i.e., a check to see if that state represented by a node has been previously reached is not performed.

Option 2: True. While a graph search (version 2) implementation will avoid states that have already been reached, there is an exception. They will still revisit a state if the newer path has a cost that is less than the previous path (i.e., the new path is not redundant).

Additional Notes: The tree search and various graph search variants may have higher or lower time and space costs depending on the problem (i.e., search tree) in question and the actual search algorithm that is adopted. It would be a good exercise to find cases where one variant outperforms another for a particular uninformed search algorithm.

Question 6

2 / 2 pts

The execution of the Breadth-First Search algorithm is not influenced by action costs. It can handle problems with negative and zero costs.

Correct!

[✓ True]

False

True, since the algorithm does not reference action cost or path costs at all.

Question 7

2 / 2 pts

The **Breadth-First Search** algorithm is **complete** if ____.

Select all correct options.

Note: completeness is satisfied iff a solution is returned when a goal exists, or failure is indicted when a goal does not exist.

- [ ] the state space has finite depth and infinite branching factor
- [ ] the state space has infinite depth and finite branching factor
- [ ] the state space has infinite depth and infinite branching factor
- Correct! [✓] the state space has finite depth and finite branching factor

Option 1: False. Unless a goal can be found on the very first branching.

Option 2: False. This is only true if we may assume that a goal exists and is reachable.

Option 3: False. Once again, unless a goal can be found on the very first branching.

Option 4: True. If a state space is finite, Breadth-First Search (BFS) will eventually traverse the entire space. Do note that even if the state space is disconnected such that the initial state is disconnected from any goal, it will still be complete since BFS will rightly point out that there is no solution given the current state space (once it exhausts its search).

Question 8

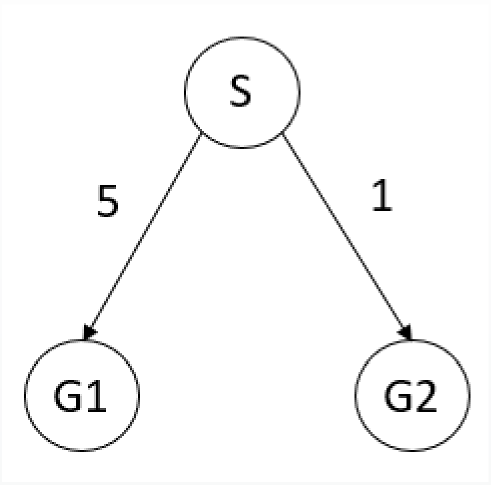
2 / 2 pts

Given that a goal exists within a finite state space, the Breadth-First Search (BFS) algorithm is optimal if all action costs from the initial state are non-decreasing in the depth of the search tree. That is, for any given level of the search tree, all action costs are greater than or equal to the action costs in the previous level.

True [✓ False]

Correct!

Consider the following counter example, where we assume node are explored from left to right. Here, S -> G1 is returned while S -> G2 is optimal.



For BFS to be optimal, the sequence of nodes explored must be monotonically increasing in terms of the action cost (i.e., the action cost taken to reach each node in question). This, or course, also includes the case when step costs are constant.

This in turn ensure that all paths are checked in the order of increasing path cost.

Question 9

2 / 2 pts

In terms of the Breadth-First Search algorithm, what is the benefit of applying the goal test when pushing to the frontier (as opposed to when popping from it)?

Assume that ***b*** denotes the number of successors for any node, and that ***d*** corresponds to the depth of the shallowest goal node (where the root is at depth 0).

- ○ Assuming the worst case, we would save on the space and runtime associated with b^(d-1) nodes.
- ○ Assuming the worst case, we would save on the space and runtime associated with b^d nodes.
- ○ Assuming the worst case, we would save on the space and runtime associated with b^(d+1) nodes.

Correct!

- ● None of the above.

At depth ***d*** of the search tree, there will be b^d nodes. However, if the goal test is done when popping from the frontier, we would potentially have to expand all but one of the nodes at depth ***d***, thus (b^d - 1) nodes are expanded. For each node expanded, we add ***b*** successors to the frontier. This results in a total of (b^{d+1} - ***b***) nodes being added to the frontier.

When early goal testing is done, we have found the goal after pushing all the nodes at depth d. This results in a saving of (b^{d+1} - ***b***) nodes.

Question 10

1.33 / 2 pts

Which of the following options about uninformed search algorithms are true?

rrect answer

☐ The Uniform-Cost Search algorithm is always complete and optimal.

Correct!

☑ The Uniform-Cost Search algorithm is equivalent to Breadth-First Search if all action costs are equal.

☐ A path cost, g(n), refers to the cost of going from the initial state to state n.

Correct!

☑ A action cost refers to the cost from state s to state s' via action a, which is denoted by c(s, a, s').

Option 1: False. The Uniform-Cost Search algorithm is only complete when the branching factor is finite and the state space is either has a solution or is finite. It also requires that all action costs are greater than 0 and greater than some small constant value ε.

Option 2: True. Based on the general assumption that action costs should be greater than 0. However, it should be noted that when action costs are 0 or less, UCS is incomplete, but BFS still is. This assumes that all other conditions for completeness under UCS/BFS are satisfied.

Option 3: False. Since there may be many possible paths from the initial state to state *n*, a specific path between the initial state and state *n* must be referenced.

Option 4: True. As defined.

Question 11

2 / 2 pts

Which of the following options about uninformed search algorithms are true?

Correct!

- [✓] A search algorithm is optimal if the solution it returns always has the minimal path cost among all solutions.

Correct!

- [✓] Breadth-First Search (BFS) and Iterative Deepening Search (IDS) are complete and optimal as long as b is finite and the search space either has a solution or is finite, and where action costs are identical.

Correct!

- [✓] Depth-First Search (DFS) and Depth-Limited Search (DLS) are not optimal.

- [ ] Iterative Deepening Search (IDS) will always be more expensive than Depth-First Search (DFS).

Option 1: True. As per the definition.

Option 2: True. If the branching factor is finite and the search space is finite (i.e., maximum depth is finite), these algorithms will eventually explore the entire search space, either returning failure if the solution does not exist, or the solution itself if it does. Further, given that action costs are identical, path costs are equal to the number of edges/actions in that path. Since BFS/IDS expand nodes level by level, they consider all paths with n actions before considering all paths with n+1 actions. In other words, they consider all paths in order of path cost and will thus ensure that when a goal is found, it is on a path with the lowest path cost — i.e., it is optimality.

Option 3: True. DFS/DLS do not account for step costs when searching through the state space. Hence, optimality is not guaranteed as the least-cost path to the goal node may not be found when a goal node is expanded.

Option 4: False. Consider a search tree with branching factor ***b*** and depth of all leaf nodes ***d***. Let ***d*** > ***b***. Now let the goal be the last node within the branching on the first level. In such a case, DFS would explore almost the entire tree whereas IDS would find the goal on the first iteration (at level 1).

Question 12

2 / 2 pts

Which search algorithm should you use (assuming you wish to minimise runtime) if the goal node is near the root, the branching factor and maximum depth are finite, and all action costs are equal?

- Depth-Limited Search (DLS)
- Depth-First Search (DFS)
- Correct! [✓ Breadth-First Search (BFS)]
- Iterative Deepening Search (IDS)

Option 1: Since we do not know the exact depth of the goal, DLS may not find a solution if the depth limit set is less than the goal depth. And even if we do hit upon the right depth, the solution found is not guaranteed to be optimal

Option 2: Since we do not know the maximum depth of the problem, DFS may explore a significant portion of the search tree before finding the goal. Also, the solution found is not guaranteed to be optimal

Option 3: BFS is the best option. Given the properties of the search tree, it will find the optimal solution.

Option 4: IDS is not a bad choice and will find an optimal solution. However, it will incur a highest cost as compared to BFS.

Question 13

2 / 2 pts

Which search algorithm should you use if all nodes at a certain depth are goal nodes and all action costs are equal?

You may not assume anything else apart from what is stated.

- ○ Uniform-Cost Search (UCS)
- Correct! [✓ Depth-First Search (DFS)]
- ○ Breadth-First Search (BFS)
- ○ Iterative-Deepening Search (IDS)

Option 1: Since action costs may be 0 or negative and still be equal, UCS may not work.

Option 2: DFS will find an optimal solution since any path leads to a goal, and all such solutions have the same path cost.

Option 3 & 4: BFS and IDS may not work since the branching factor may be infinite.