Quiz2

1. Compare tree-search versus graph-search in terms of one each of (a) implementational difference (b) Advantage of graph-search (c) disadvantage of graph search

Answer:

(a) The differences in implementation are:

1. In the case of a graph search, we use a list, called the closed list (also called explored set), to keep track of the nodes that have already been visited and expanded, so that they are not visited and expanded again.
2. In the case of a tree search, we do not keep this closed list. Consequently, the same node can be visited multiple (or even infinitely many) times, which means that the produced tree (by the tree search) may contain the same node multiple times.

(b) The advantage of graph search obviously is that, if we finish the search of a node, we will never search it again. On the other hand, the tree search can visit the same node multiple times.

(c) The disadvantage of graph search is that it uses more memory (which we may or may not have) than tree search. This matters because graph search actually has exponential memory requirements in the worst case, making it impractical without either a really good search heuristic or an extremely simple problem.

So, there is a trade-off between space and time when using graph search as opposed to tree search (or vice-versa).

1. What are the parameters on which a search algorithm is evaluated?

Answer:

* 1. Time complexity
  2. Space complexity
  3. Completeness
  4. Optimality

1. On what basis would you choose between DFS and BFS?

Answer: Some common properties of the graph based on which we can choose are:

* If you know a solution is not far from the root of the tree, BFS might be better.
* If the tree is very deep and solutions are rare, DFS might take an extremely long time, but BFS could be faster.
* DFS should not be used on infinitely deep graphs.
* If the tree is very wide (high branching factor), BFS might need too much memory, so it might be completely impractical.
* If solutions are frequent but located deep in the tree, BFS could be impractical.
* If the search tree is very deep, you will need to restrict the search depth for DFS.

1. The data structure used for DFS is stack/LIFO and for BFS is queue/FIFO
2. Under what condition is a DFS incomplete?
3. If there are many edges connected to one node
4. If the solution recurs at the same level
5. If the node can lead back to itself with a negligible cost
6. If there are loops in the connectivity graph

Answer: If a node leads to itself with a cheap edge weight, it will lead to an infinite loop and DFS will get stuck. Similarly, a sub graph which loops within itself will also cause DFS to get stuck.

1. What is the factor that makes A\* search optimal compared to greedy-first search?

Answer: Greedy-first search relies solely on the heuristic being correct for the cost, while A\* considers the history of the path into the cost.

In the case of the greedy BFS algorithm, the evaluation function is f(n) = h(n), that is, the greedy BFS algorithm first expands the node whose estimated distance to the goal is the smallest. So, greedy BFS does not use the "past knowledge", i.e., g(n).

In the case of the A\* algorithm, the evaluation function is f(n) = g(n) + h(n), where h is an admissible heuristic function.

A heuristic h(n) is admissible if for every node n, h(n) ≤ h\*(n), where h\*(n) is the true cost to reach the goal state from n.

An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic.

1. In tree-search algorithms, the cost function can be characterised as:
2. a state function
3. a path function
4. both
5. none of the above

Answer: the cost function takes into account the edge weights between two nodes to find the cost of the entire path between a start node and a goal node.

1. What is 'limited depth search algorithm' and its prime utility?
2. Searching until the goal is found at a certain finite depth; to make the search complete
3. Searching exhaustively; to find optimal solution
4. The search depth is prefixed; it prevents going down infinite length paths

Answer: The depth limited search is a variation of the DFS traversing algorithm. It takes care of an edge case problem with DFS by implementing a depth limit.

For a tree whose height is either very large or infinite, DFS would also go on infinitely, as there would always be more child nodes to push back into the stack. This is what the depth limited search addresses with a level limit variable.

This is more time efficient (O(bl)) where l is the set depth limit. A limitation of this algorithm is that if the goal node does not exist within the specified limit, it will not be discovered.

1. The bidirectional search algorithm reduces the space complexity of BFS algorithm to,
2. O(b^d/2)
3. O(b^d)
4. O((b/2)^d)
5. O(bd)

Answer: Bidirectional search is a search strategy that proceeds by expanding from both the initial and goal state simultaneously.

1. Each student must go through a series of courses to complete an educational program. Some courses are made easier by taking some other courses before. For e.g., going from course-A to course-B costs 4 while from course-A to course-D costs 16. There are a total of 20 courses, and they are deeply interdependent. Given a starting course, how can we find the ideal curriculum to reach a certain other course as soon as possible in the program? - (a) Name the algorithm you will use. (b) Estimate its worst-case time (in minutes) and space complexity (in MB). (Assume that each course on average connects to 3 courses; a computer performs 1M instructions in 1sec; 1 course information consumes 1Byte)
2. BFS/ breadth first search
3. WC Time complexity ~ 7 minutes
4. WC Space Complexity ~ 1162MB

Answer:

Since the solution might occur on shallow levels and not necessarily on the deepest level, it would be better to use BFS as it would find the solution quicker for shallow levels.

For complexity, BFS takes O(b^(d – 1)), since we can stop at generation of goal state. According to the problem, b = 3, d = 19. Time complexity will be calculated as 3^18/1M/60 = 7 minutes and space complexity will be 3^19 = 1162 MB.