**INF2D – Reasoning & Agents   
Coursework 1 written questions  
  
*3.4) UNINFORMED SEARCH***

Lexicographic

1. **BFS and DFS**
   1. A close up of text on a white background

      Description automatically generatedStep-by-step sequence of how Breadth First Search is applied to the graph in figure 2:
   2. A screenshot of a cell phone

      Description automatically generatedStep-by-step sequence of how Depth First Search is applied to the graph in figure 2:
   3. A graph that favours BFS over DFS: **(Let start node = A & goal node = F)**  
      A drawing of a face

      Description automatically generatedThis graph favours BFS as upon first expansion of the start node (A) it finds the goal node (F). Whereas DFS would expand the right branch first going all the way to node D, backtracking to node E, and then finally backtracking all the way to node F.  
        
      BFS would have to explore 3 nodes.  
      DFS would have to explore all 6 nodes.
   4. A graph that favours DFS over BFS: **(Let start node = A & goal node = D)**A drawing of a person

      Description automatically generatedThis graph favours DFS as it would expand the right branch first all the way to the goal node (D). Whereas BFS would first expand all adjacent, unvisited nodes of A (F,E, and B) and then expand B to find C, and then finally reach the goal node (D) upon expansion of node C.  
        
      DFS would have to explore 4 nodes.  
      BFS would have to explore all 6 nodes.

|  |  |
| --- | --- |
| **BFS** | **DFS** |
| Proceeds search level by level. | Proceeds search by following a path until it cannot expand the path no more and then backtracks to the last adjacent, unvisited node. |
| Uses queue data structure for storing nodes. | Uses stack data structure for storing nodes. |
| More suitable for finding nodes which are closer to the given source. | More suitable for finding nodes which are farther away from the given source. |
| Is not memory efficient. | Is memory efficient. |
| Optimal for finding the shortest path. | Not optimal for finding the shortest path. |
| Not suitable for decision making trees used in games. | More suitable for decision making trees used in games. |

* 1. Compare BFS & DFS

1. **Iterative Deepening Search**
   1. Optimal depth for the graph in figure 2:  
        
      Depth = # edges  
        
      We can define the optimal depth of an IDS as the most efficient (with regards to space and time) depth in order to find the goal/terminating node.  
        
      After analyzing the graph I found that the optimal depth must be 3, as this is the minimum possible path length to reach any node on the graph and thereby reduces the exploration of unnecessary branches, and the number of nodes needed to be stored on the stack.

|  |  |
| --- | --- |
| **DFS** | **IDS** |
| DFS may explore the entire graph before finding the goal node. | Iterative deepening will only explore the entire graph before finding the goal node if the distance between the start and end node is the maximum in the graph. |

* 1. Compare DFS & IDS  
       
     **When you would use IDS over DFS**
     1. Iterative deepening is useful for when the goal node is close to the start node (contrary to DFS) especially for large graphs with long branches.
     2. Iterative deepening can also be useful in order to find the optimal path in a graph (contrary to DFS).

***4.3) INFORMED SEARCH***

1. **Heuristics**
   1. Why straight-line distance is a valid heuristic

Straight-line distance is a valid heuristic as it informs the program which node is closer to the goal node. This is particularly useful and relevant as it makes node expansion and choosing an optimal path more efficient.

* 1. Problems that can be solved with A\* Search and their heuristics

|  |  |
| --- | --- |
| **Problems** | **Relevant heuristic** |
| Distance-based route-planning (maps) | Manhattan distance |
| Cost-based route-planning (ie. CityMapper) | Monetary cost |
| Game playing (ie. Chess) | Score (ie. Piece count) |

1. **Compare Best-First and A\* Search**
   1. Differences between Best-First and A\* search strategies

|  |  |
| --- | --- |
| **Best-First** | **A\*** |
| Visits next node based on heuristics function  f(n) = h with lowest heuristic value. It does not consider the cost of the path to that next node. | Visits next node based on overall cost function f(n) = h+g with the lowest sum of heuristic and path cost. |
| Not optimal | Optimal |
| Not complete | Complete |
| Uses less memory | Uses more memory |

* 1. How I would change my A\* implementation to be a Best-First search instead
     1. Base node expansions on the minimum respective heuristic values alone.
     2. If there are no adjacent, unvisited nodes from the current node to be expanded then the search terminates.

***5.4) CONNECT FOUR***

Quadrio assumptions:

* Can only place counters into the board from the top
* Possible moves to do in any given turn:   
  P, RP, PR, RPR  
    
  Where P = counter placement & R = 90° board rotation left/right

1. Number of possible actions a player can perform during 1 turn

**Possible moves:** P, RP, PR, RPR  
**Possible counter placements:** Col. 1, Col. 2, Col. 3, Col. 4  
**Possible rotations:** 90° left, 90° right

1. Main challenges for implementing Quadrio over Connect Four with a Twist using Alpha-Beta pruning  
     
   For Connect Four:

For any given turn in Connect Four with a Twist a player can do 8 possible actions, in contrast to Quadrio in which a player can do 36 possible actions (over 4 times more).

This thereby implies the total amount of possible games/outcomes in Quadrio is exponentially larger. In effect this means that implementing Quadrio with Minimax would have an exponentially larger space and time complexity. However, upon implementation of Alpha-Beta there would a significant reduction in the number of outcomes needed to traverse by Minimax.

So, in final, with regards to choosing an algorithm to implement Quadrio, I think using Alpha-Beta pruning would be a practical choice.