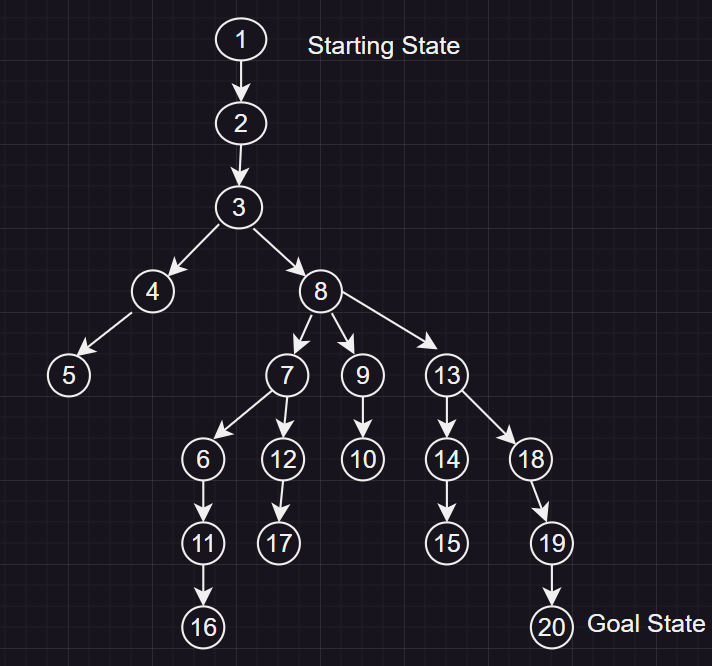
CSE4002 Assignment1 – Ben Smerd 22072922

* **State-space structure**



* **Depth-first search**

1. Open=[1];closed=[]
2. Open=[2];closed=[1]
3. Open=[3];closed=[2,1]
4. Open=[4,8];closed=[3,2,1]
5. Open=[5,8];closed=[4,3,2,1]
6. Open=[8];closed=[5,4,3,2,1]
7. Open=[7,9,13];closed=[8,5,4,3,2,1]
8. Open=[6,12,9,13];closed=[7,8,5,4,3,2,1]
9. Open=[11,12,9,13];closed=[6,7,8,5,4,3,2,1]
10. Open=[16,12,9,13];closed=[11,6,7,8,5,4,3,2,1]
11. Open=[12,9,13];closed=[16,11,6,7,8,5,4,3,2,1]
12. Open=[17,9,13];closed=[12,16,11,6,7,8,5,4,3,2,1]
13. Open=[9,13[;closed=[17,12,16,11,6,7,8,5,4,3,2,1]
14. Open=[10,13];closed=[9,17,12,16,11,6,7,8,5,4,3,2,1]
15. Open=[13];closed=[10,9,17,12,16,11,6,7,8,5,4,3,2,1]
16. Open=[14,18];closed=[13,10,9,17,12,16,11,6,7,8,5,4,3,2,1]
17. Open=[15,18];closed=[14,13,10,9,17,12,16,11,6,7,8,5,4,3,2,1]
18. Open=[18];closed=[15,14,13,10,9,17,12,16,11,6,7,8,5,4,3,2,1]
19. Open=[19];closed=[18,15,14,13,10,9,17,12,16,11,6,7,8,5,4,3,2,1]
20. Open=[20];closed=[19, 18,15,14,13,10,9,17,12,16,11,6,7,8,5,4,3,2,1]
21. Goal reached [20]
    * Path: 1->2->3->4->5->8->7->6->11->16->12->17->9->10->13->14->15->18->19->20
    * Discussion: Depth-first search will go down to the bottom of each branch before moving on to the next branch. It will iterate through all the descendants of a node before it moves on to the sibling branches of that node. It uses a stack structure (first in, last out) when processing nodes.

* **Breadth-first search (using ancestors to find path)**

1. Open=[1];closed=[]
2. Open=[(2,1)];closed=[1]
3. Open=[3];closed=[2,1]
4. Open=[4,8];closed=[3,2,1]
5. Open=[8,5];closed=[4,3,2,1]
6. Open=[5,7,9,13];closed=[8,4,3,2,1]
7. Open=[7,9,13];closed=[5,8,4,3,2,1]
8. Open=[9,13,6,12];closed=[7,5,8,4,3,2,1]
9. Open=[13,6,12,10];closed=[9,7,5,8,4,3,2,1]
10. Open=[6,12,10,14,18];closed=[13,9,7,5,8,4,3,2,1]
11. Open=[12,10,14,18,11];closed=[6,13,9,7,5,8,4,3,2,1]
12. Open=[10,14,18,11,17];closed=[12,6,13,9,7,5,8,4,3,2,1]
13. Open=[14,18,11,17];closed=[10,12,6,13,9,7,5,8,4,3,2,1]
14. Open=[18,11,17,15];closed=[14,10,12,6,13,9,7,5,8,4,3,2,1]
15. Open=[11,17,15,19];closed=[18,14,10,12,6,13,9,7,5,8,4,3,2,1]
16. Open=[17,15,19,16];closed=[11,18,14,10,12,6,13,9,7,5,8,4,3,2,1]
17. Open=[15,19,16,20];closed=[17,11,18,14,10,12,6,13,9,7,5,8,4,3,2,1]
18. Open=[19,16,20];closed=[15,17,11,18,14,10,12,6,13,9,7,5,8,4,3,2,1]
19. Open=[16,20];closed=[19,15,17,11,18,14,10,12,6,13,9,7,5,8,4,3,2,1]
20. Open=[20];closed=[16,19,15,17,11,18,14,10,12,6,13,9,7,5,8,4,3,2,1]
    1. Using ancestors for path Open=[(20,19)];closed=[(16,11),(19,18),(15,14),(17,12),(11,6),(18,13),(14,13),(10,9),(12,7),(6,7),(13,8),(9,8),(7,8),(5,4),(8,3),(4,3),(3,2),(2,1),(1,nil)]
21. Goal reached[20]
    * Path: 1->2->3->4->8->5->7->9->13->6->12->10->14->18->11->17->15->19->16->20
    * Parent path-
    * Discussion: Breadth first uses a queue when processing nodes (first in, first out order). Breadth first goes level-by-level through the tree and will not move to the next level until all nodes on the current level have been examined. Each node on the level will be processed independent from one another.

* **A\* search**
  + **Heuristic Evaluation Function f(n)-**
    - **Manhattan Distance** :coordinates of grid https://www.datacamp.com/tutorial/manhattan-distance
    - *H(n)*- the coordinates of the node on the grid using the Manhattan heuristic function

|  |  |  |
| --- | --- | --- |
| **State n** | **coordinates (x,y)** | **H(n) = (xgoal​ - xn) + (ygoal - yn)** |
| 1 | (1,1) | H(1)=(4-1)+(5-1)=3+4=7 |
| 2 | (1,2) | H(2)=(4–1)+(5–2)=3+3=6 |
| 3 | (1,3) | H(3)=(4–1)+(5–3)=3+2=5 |
| 4 | (1,4) | H(4)=(4–1)+(5-4)=3+1=4 |
| 5 | (1,5) | H(5)=(4-1)+(5-5)=3+0=3 |
| 6 | (2,1) | H(6)=(4-2)+(5-1)=2+4=6 |
| 7 | (2,2) | H(7)=(4-2)+(5-2)=2+3=5 |
| 8 | (2,3) | H(8)=(4-2)+(5-3)=2+2=4 |
| 9 | (2,4) | H(9)=(4-2)+(5-4)=2+1=3 |
| 10 | (2,5) | H(10)=(4-2)+(5-5)=2+0=2 |
| 11 | (3,1) | H(11)=(4-3)+(5-1)=1+4=5 |
| 12 | (3,2) | H(12)=(4-3)+(5-2)=1+3=4 |
| 13 | (3,3) | H(13)=(4-3)+(5-3)=1+2=3 |
| 14 | (3,4) | H(14)=(4-3)+(5-4)=1+1=2 |
| 15 | (3,5) | H(15)=(4-3)+(5-5)=1+0=1 |
| 16 | (4,1) | H(16)=(4-4)+(5-1)=0+4=4 |
| 17 | (4,2) | H(17)=(4-4)+(5-2)=0+3=3 |
| 18 | (4,3) | H(18)=(4-4)+(5-3)=0+2=2 |
| 19 | (4,4) | H(19)=(4-4)+(5-4)=0+1=1 |
| 20 | (4,5) | H(20)=(4-4)+(5-5)=0+0=0 |

* + - *G(n)*- the distance for each node from the starting node

A diagram of a graph

Description automatically generated

* + - Process- *F(n) = G(n) + H(n)*

1. Open=[1];closed=[0]
2. Evaluate[1];open=[2(7)];closed=[1]
3. Evaluate[2];open=[3(7)];closed=[2,1]
4. Evaluate[3];open=[4(7),8(7)];closed=[3,2,1]
5. Evaluate[4];open=[5(7),8(7)];closed=[4,3,2,1]
6. Evaluate[5];open=[8(7)];closed=[5,4,3,2,1]
7. Evaluate[8];open=[9(7),13(7),7(9)];closed=[8,5,4,3,2,1]
8. Evaluate[9];open=[10(7),13(7),7(9)];closed=[9,8,5,4,3,2,1]
9. Evaluate[10];open=[13(7),7(9)];closed=[10,9,8,5,4,3,2,1]
10. Evaluate[13];open=[14(7),18(7),7(9)];closed=[13,10,9,8,5,4,3,2,1]
11. Evaluate[14];open=[15(7),18(7),7(9)];closed=[14,13,10,9,8,5,4,3,2,1]
12. Evaluate[15];open=[18(7),7(9)];closed=[15,14,13,10,9,8,5,4,3,2,1]
13. Evaluate[18];open=[19(7),7(9)];closed=[18,15,14,13,10,9,8,5,4,3,2,1]
14. Evaluate[19];open=[20(7),7(9)];closed=[19,18,15,14,13,10,9,8,5,4,3,2,1]
15. Path identification- 1->2->3->4->5->8->9->10->13->14->15->18->19->20
    * + Discussion- A\* search works by combining both greedy search and unform cost search into one. It looks at the closest node to the goal node (greedy search) while looking at the lowest cost path (uniform cost search). It uses these to find the most promising paths. Due to the nature of the maze being a grid, it is more optimal to use the Manhattan distance, this is because it is admissible and will never overestimate the true cost to reach the goal because it always provides equal or lower estimation of the path cost. It is also best because the Manhattan works best when navigating a grid with only horizontal and vertical movements allowed, so it fits in the constraints of this puzzle well. Because of the walls within the maze, using the G(n) will work well with the H(n) for any sort of obstacles.

* **Advantages/Disadvantages**
  + Depth-first
    - Advantages-
      * Can use depth limit and iteration to ensure an incorrect path isn’t taken on a larger grid
      * Simple to implement as it only stores the current path it is on from the start state to goal state.
      * Can be optimal in a problem where the goal state is within a deep branch, which can occur in a problem like this
    - Disadvantages-
      * Because the goal state was on an edge branch, it had to examine every other node on all the branches before it
      * Time complexity can become a big issue if there were many more branches as well as deep branches before the goal state
      * Depth first is not complete and it can get lost within a very deep branch/s.
  + Breadth-first
    - Advantages-
      * It is guaranteed to find the most optimal path as it checks all nodes at the current depth level before moving on, so if there is a shortest path, it will be found
      * Can be more optimal for a smaller graph with less obstacles
    - Disadvantages-
      * The open list becomes fairly large if many branches
      * If it’s a large graph then it has to explore every node which can take a very long time
      * Has to go through every branch on every level to reach the goal, which was at the very bottom of the last branch in this problem, meaning it had to go through every node
  + A\*
    - Advantages-
      * The heuristic function (Manhattan Distance) helps with prioritising nodes that are closer to the goal node, while looking for the lowest cost node.
      * It takes into consideration information about preferred states compared to BFS and DFS.
    - Disadvantages-
      * No guarantee that the heuristic function will return a value that ensures the node is closer to the goal state, as seen in the A\* path above.
      * A\* implementation quality is dependent on the heuristic function, so if its not the correct one it may produce an inefficient process where the A\* exploring may become worse than blind searches.
      * To implement a heuristic function, specific domain knowledge is needed for the problem to ensure that the heuristic function will be implemented correctly.
      * Because of a poorly made heuristic function, a solution may not be found