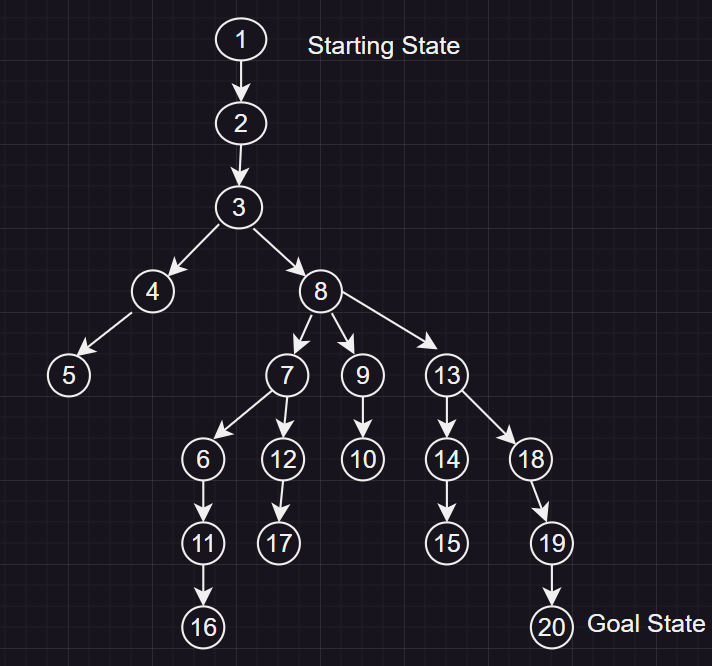
Assignment1 – Ben Smerd 22072922

* **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 of a child node of a parent, before moving on to a sibling branch of the same parent node. Depth-first uses the stack with first in, last out. Depth-first search is not complete and it can get lost within a very deep branch/s. Because the time complexity is *O(b^m)* this means that if there are many deep branches, it can take a long time to find the goal. A more optimal way using depth-first search would be to add in an iterative deepening on a depth-limit that does provide completeness. This will consistently get deeper with every iteration of a branch once it has reached the current depth-limit. The time complexity remains the same, however it does provide completeness.

* **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). Once the goal node has been dequeued from the open list then the goal has been reached. Breadth-first works by processing each node within a level independent of how each node on that level is related to one another, and it will only go to the next level once every node on the level has been processed. This can make sure that the search never goes deeper within the tree than where the goal state is.

* **A\* search**
  + TODO need to define the g(n) and h(n) as tiles moved (both have to be the same) and then make that the cost function so g(n) would be tiles moved to a certain position I think and h(n) is tiles left to the goal state--- pretty sure this is it--- MANHATTEN DISTANCE
  + TODO same as the lab where each tile will have a distance number- each is the amount of squares to the goal state from the start state, takes in to account the walls
  + **Heuristic Evaluation Function f(n)-**
    - **Manhattan Distance** :coordinates of grid https://www.datacamp.com/tutorial/manhattan-distance
    - f

|  |  |  |
| --- | --- | --- |
| **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 |