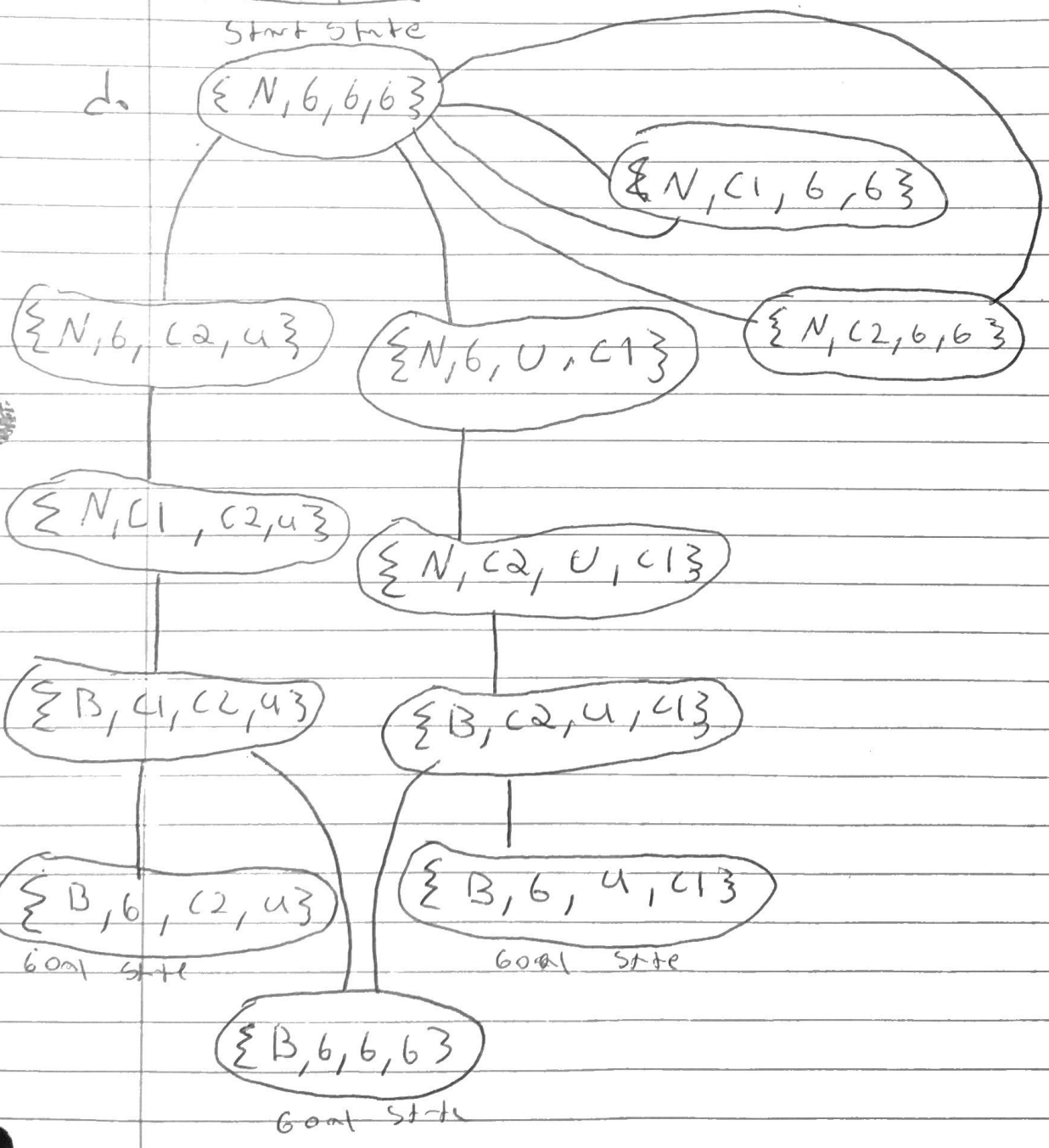
Hw2. Michael Chillemi

1

- each edge is an action, each node is the result of the action.

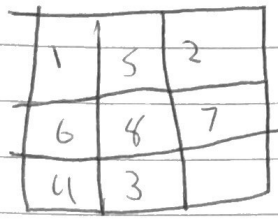
start state

d.

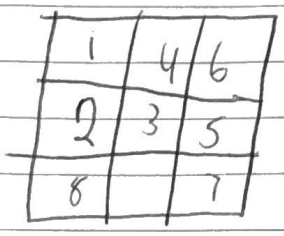


Problem 2.

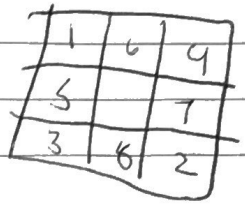
a. 8-puzzle



2 move
4 corners



3 moves
4 spots

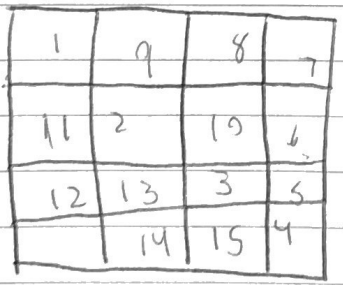


4 moves
1 middle

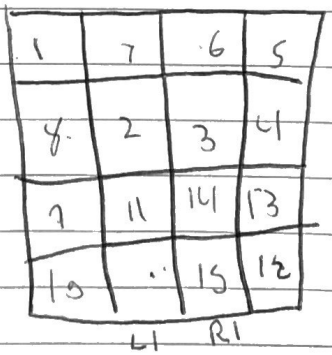
$$= \frac{(4 \cdot 2) + (3 \cdot 4) + (4 \cdot 1)}{9}$$

$$= \frac{8 + 12 + 4}{9} = \frac{24}{9} = 2.\overline{6}$$

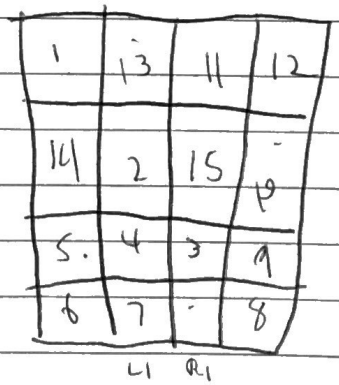
b.



2 moves
4 corners



3 moves
4 L1 spots



3 moves
4 R1 spots

1	12	11	10
2		13	9
3	15	14	8
4	5	6	7

4 moves
4 middle spots

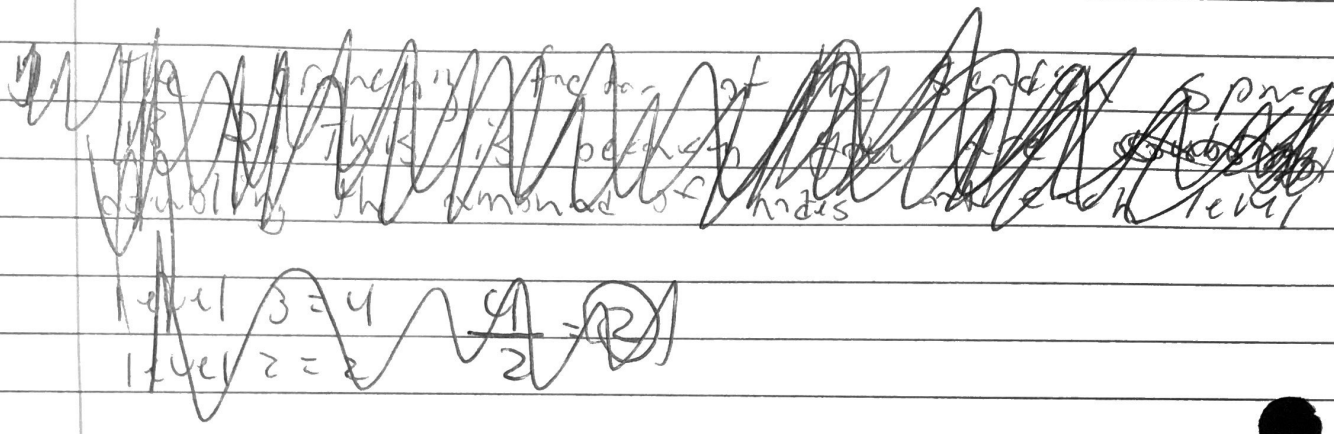
$$= \frac{(4 \cdot 2) + (4 \cdot 3) + (4 \cdot 3) + (4 \cdot 4)}{16}$$

$$= \frac{8 + 12 + 12 + 16}{16}$$

$$= \frac{48}{16} = 3$$

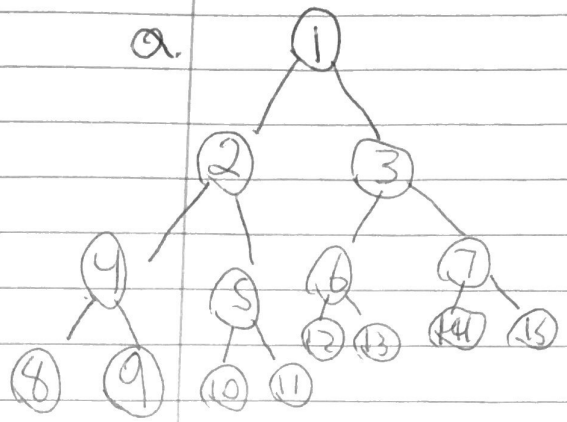
C. The branching factor of a rubik's cube is 12. You get this because there are 6 faces on the cube and you can move each face left or right.

$$\frac{6}{1} \cdot 2 = 12$$



3.

a.



b. BFS

Visited = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

c. Depth limited DFS limit = 3

{1, 2, 4, 8, 9, 5, 10, 11}

D IDS

Visited = {1, 2, 3, 1, 2, 4, 5, 3, 6, 7, 1, 2, 4, 8, 9, 5, 10, 11}

e Assuming ~~this~~ what was stated above that the space state was infinite in size. If you use DFS on this tree it will not terminate because you will continously move down the 2k side of the tree and never reach ~~anyone~~ ~~anyone~~ the end because the tree has an infinite size.

f. Going off the same assumptions as above you will not terminate because you will continuously move down the right side $(2k+1)$ side. Since the tree is infinite the algorithm will never terminate because you will not hit your goal state.

g. The branching factor of the Search space is 2. This is because you are doubling the amount of nodes at each level.

level 3 = 4 $\quad \frac{4}{2} = 2$
level 2 = 2

H. If the current node that you are at is even the function should divid by 2. If the node is odd then you must subtract 1 first and then divide by 2.

I. The predecessor function will be one because each child in the tree has one parent.

~~The Search algorithm would work the best is the depth limited. This is because it will go by level and if the goal is not then the level limit will increase by 1. This is only efficient if you know what level the goal is on, otherwise~~

v. The ~~Depth~~ Search algorithm that is most efficient is working backwards. This is because if you work backwards you are just moving up to the parent node ~~until~~ until you get to the root. Once you get to the root then the task is complete and you have an algorithm with the least amount of search steps.