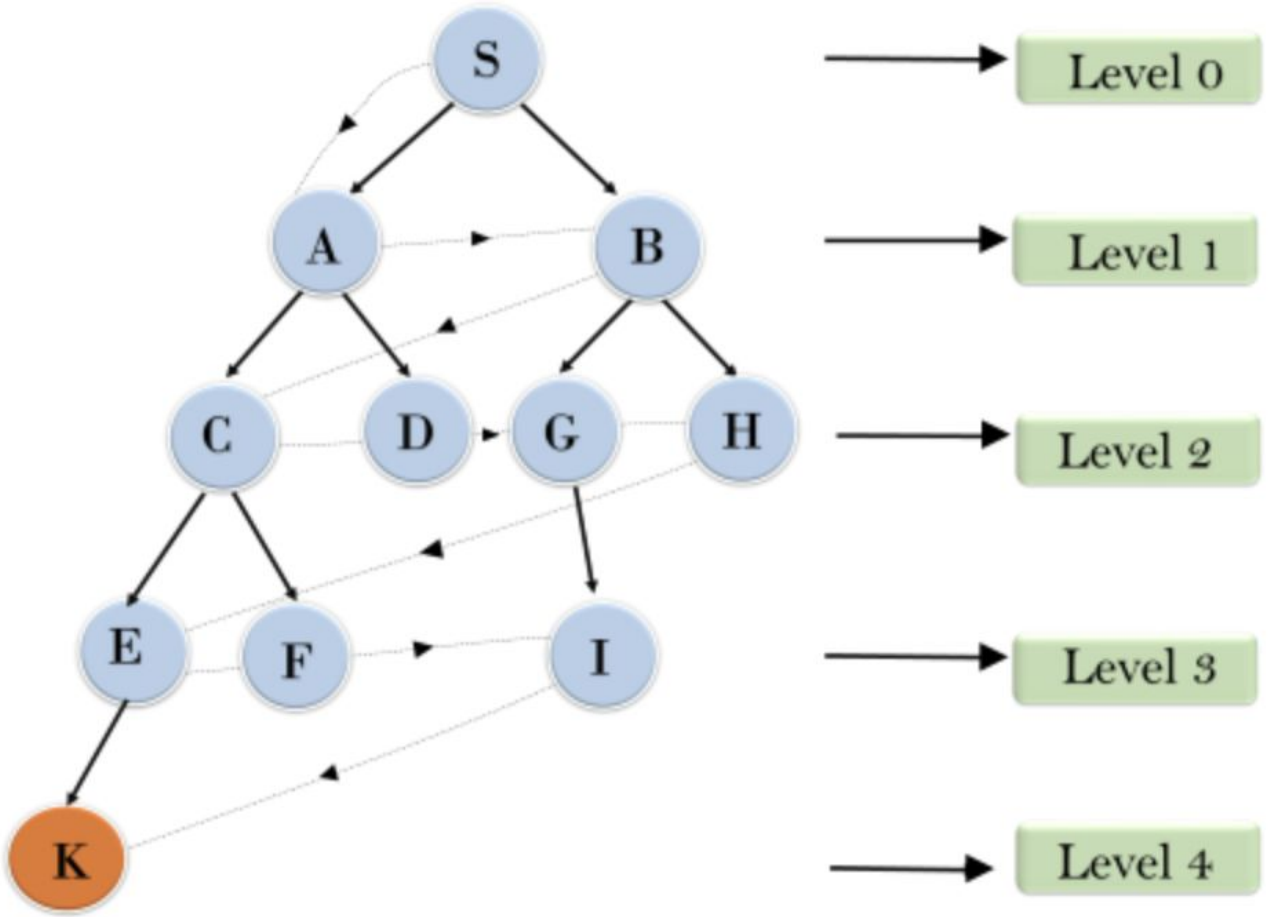
BFS

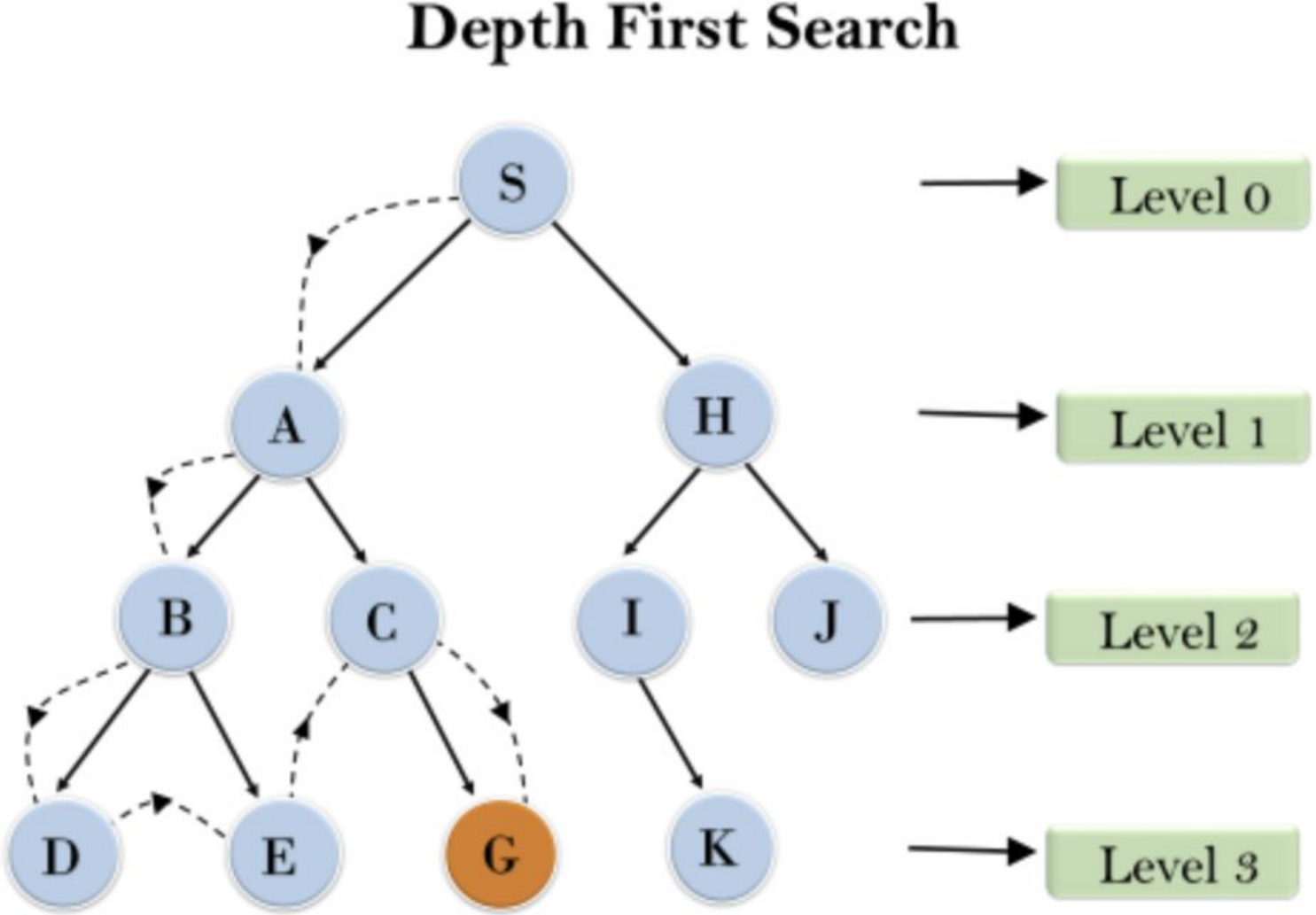

Breadth First Search
S
A
B
C
D
G
H
E
F
I
K
Level 0
Level 1
Level 2
Level 3
Level 4

Input: A graph G and a *starting vertex root* of G

Output: Goal state. The *parent* links trace the shortest path back to *root*[8]

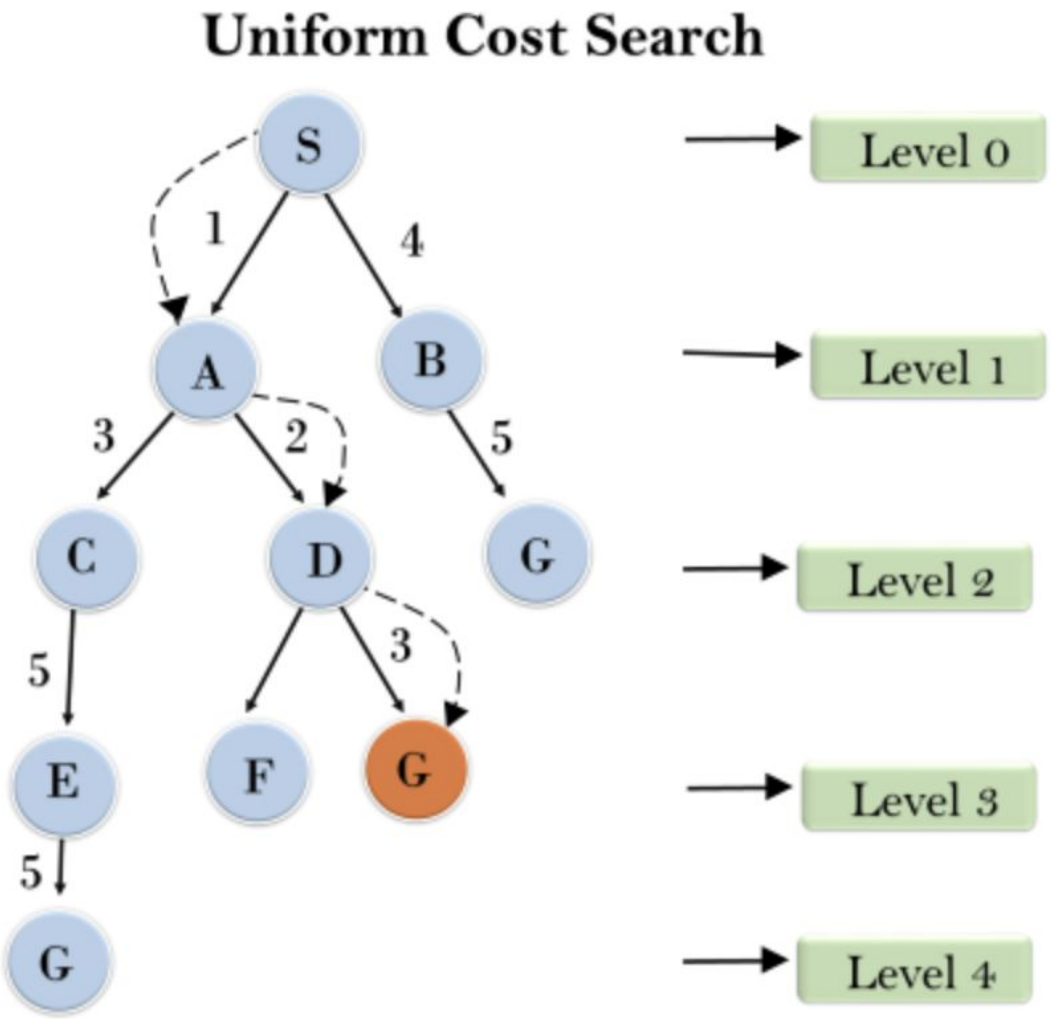
```
 1  procedure BFS(G, root) is
 2      let Q be a queue
 3      label root as explored
 4      Q.enqueue(root)
 5      while Q is not empty do
 6          v := Q.dequeue()
 7          if v is the goal then
 8              return v
 9          for all edges from v to w in G.adjacentEdges(v) do
10              if w is not labeled as explored then
11                  label w as explored
12                  Q.enqueue(w)
```

DFS


Depth First Search
S
A
H
B
C
I
J
D
E
G
K
Level 0
Level 1
Level 2
Level 3

```
procedure DFS_iterative(G, v) is
    let S be a stack
    S.push(v)
    while S is not empty do
        v = S.pop()
        if v is not labeled as discovered then
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```

UCS


Uniform Cost Search
S
A
B
C
D
G
E
F
G
G
1
4
3
2
5
5
3
5
Level 0
Level 1
Level 2
Level 3
Level 4

```
procedure uniform_cost_search(start) is
    node ← start
    frontier ← priority queue containing node only
    explored ← empty set
    do
        if frontier is empty then
            return failure
        node ← frontier.pop()
        if node is a goal state then
            return solution(node)
        explored.add(node)
        for each of node's neighbors n do
            if n is not in explored and not in frontier then
                frontier.add(n)
            else if n is in frontier with higher cost
                replace existing node with n
```

Q1

1. The space complexities of BFS and DFS are $O(b^d)$ and $O(bd)$ respectively, where b is the branching factor and d is the search depth. Why is one exponential with respect to d and the other not?

When searching at a particular depth, BFS maintains all nodes at that depth in its frontier while DFS only keeps those nodes corresponding to one branch.

Note that d is the search depth but not the tree depth.

Q2

3. What is the difference among BFS, DFS, and uniform-cost search (Dijkstra's algorithm) with respect to their implementations in the generic tree search algorithm?

From a high level view, the three algorithms only differ in the way the frontier is maintained. In particular, BFS keeps its frontier as a queue (FIFO), DFS uses a stack (LIFO), and Dijkstra utilizes a priority queue where nodes are ordered by their current path costs.

Q3

3. (a) What is the time complexity of uniform cost search, in terms of the branching factor and costs on edges? (b) What is the key assumption that guarantees the optimality of uniform cost search?

(a) $O(b^{1+\lfloor C/\epsilon \rfloor})$ where C is the cost of the shortest path to a goal and ϵ is the minimum edge cost.
(b) All edge costs are positive ($\epsilon > 0$)

Q4

4. Consider the two statements (a) BFS is a special case of uniform cost search, and (b) uniform cost search is a special case of A*. Under what conditions are they true?

(a) Uniform cost reduces to BFS when all edge costs are equal (and non-negative).
(b) A* reduces to uniform cost if the heuristic is constant for every node.

Q4

A heuristic function is said to be admissible if it never overestimates the cost of reaching the goal, i.e. the cost it estimates to reach the goal is not higher than the lowest possible cost from the current point in the path.

A*: f(n) = g(n) + h(n)

USC: f(n) = g(n)

If h(n) is a constant O(1), h() can be neglected in the evaluation function f(n).

Q5

5. Sudoku is a popular game in which the player tries to fill in all blank cells so that the resulting board contains 1 to 9 on each row, each column, and each 3×3 block.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

a. Formulate it as a tree search problem. What are the state space, goal state, successor function, and the path costs?
b. Assume we use uninformed search. Which method would you prefer? Why?
c. Is a good heuristic possible in this case? If so, provide one. If not, why not?

Q5

(a)State space is just the 9 x 9 matrix
Goal state is completed puzzle satisfying Sudoku requirements.
All rows, columns and 3x3 units are filled with non-repeating digits 1 to 9.
Successor function is to fill in a value. Path costs: 1?
(b)I would prefer depth first search as the solution must occur at a fixed depth
(c) No heuristic is possible because I would: i) Need to know the exact goal state (which is the aim of the problem) and ii) All paths are of the same length there is no shortest path.

Q5

Possible solution, equivalent to graph coloring problem.

- Construct a graph with 81 nodes (one for each position)
- Some nodes will have a value 1 through 9
- Others will need to be filled in
- Place an edge between the nodes if they are either:
  - 1) In the same 3 x 3 box
  - 2) In the same row
  - 3) In the same column
- Fill in the values of the ones with no value
  - No two nodes sharing an edge have the same value.

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Q6

6. Consider the classic farmer, fox, goose, and grain problem. The farmer wants to move himself, the fox, the goose, and the edible grain from the west side to the east side of the river. Only he can row his small boat across the river, and he can only take one of his items with him at a time. If the fox is left with the goose, the goose will be eaten. If the goose is left with the grain, the grain will be eaten. It turns out that you can pose this as a graph search problem.
   a. Describe a representation of the state space for this problem. What would the goal state look like in this representation.
   b. What are the possible actions at a particular state?
   c. Our goal is to get everything to the other side in one piece, so what are the constraints on the state space that we need for the actions in part b?
   d. Suppose we want to use A* here. Describe a non-trivial admissible heuristic that we could use.

Q6

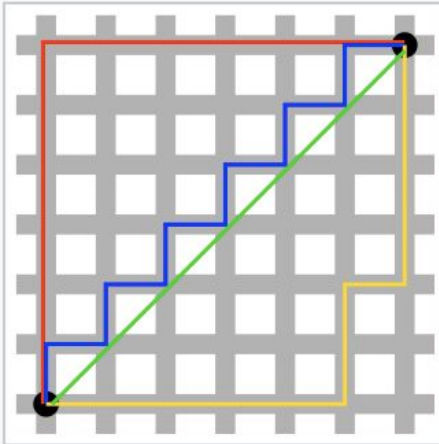
(a)Represent the state as four boolean variables: <G,GR,F,Fa>
Set to 0 if on West side, 1 for East side
Start State <0,0,0,0>, Goal state <1,1,1,1>
(b)Actions [1. Take Fa bit and flip it.
2. Take Fa bit and one other bit of same state and flip both.
(c) Constraints:
[G == Gr AND G != Fa, F == G AND G != Fa]
So branching factor is? 4? Why four, because its worst case.
(d)Hamming Distance is an admissible heuristic? No?
Counter example? <G=0,GR=1,F=1,Fa=0> -> <G=1,GR=1,F=1,Fa=1>
Fix? Hamming distance divided by 2,

Distances

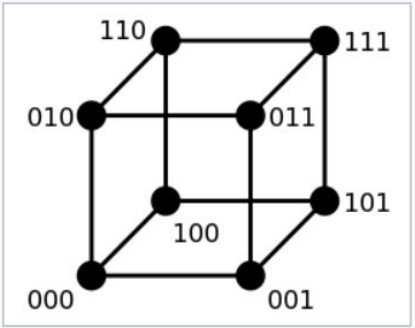
Manhattan:

|x1-x2| + |y1-y2|

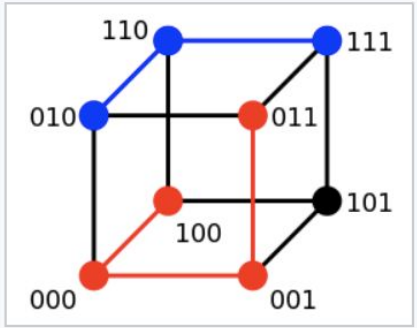
Hamming: the number of positions at which the corresponding symbols are different



Red: **Manhattan distance**. Green: diagonal, straight-line distance. Red, blue, yellow: equivalent Manhattan distances.



3-bit binary cube for finding Hamming distance



Two example distances: 100→011 has distance 3; 010→111 has distance 2

The minimum distance between any two vertices is the Hamming distance between the two binary strings.