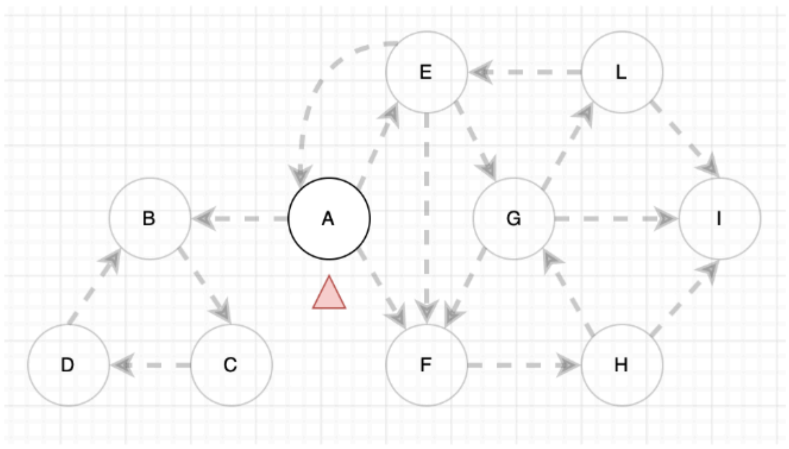
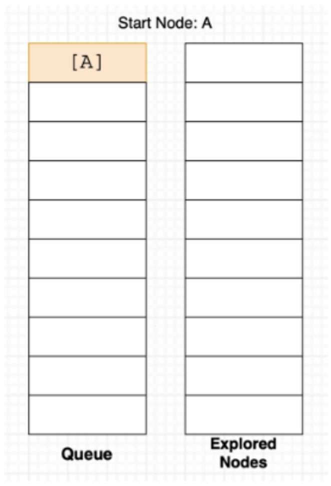
3.4 Uninformed Search — Questions

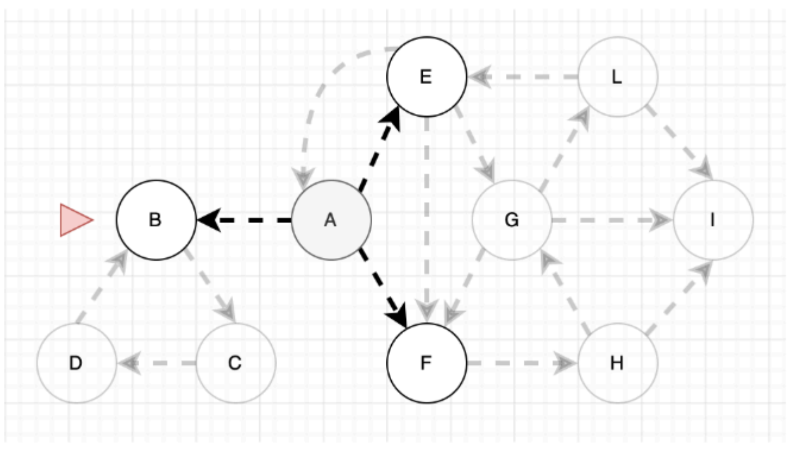
1. Breadth-first search and Depth-first search

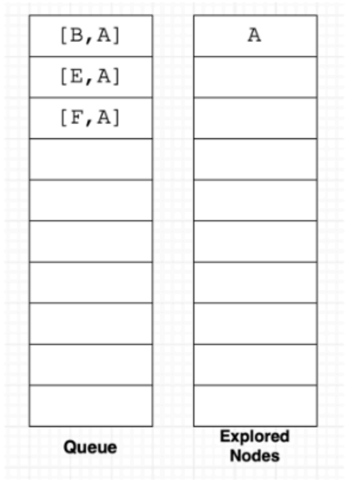
1. (a) Breadth - First Search on the provided graph where A is the start node and L is the destination node. Note that we do not go through nodes that have already been explored (elimination of repeated states), i.e, the branches with already explored first element are skipped (marked in red). Also I've omitted the cost as they're not useful for BFS.



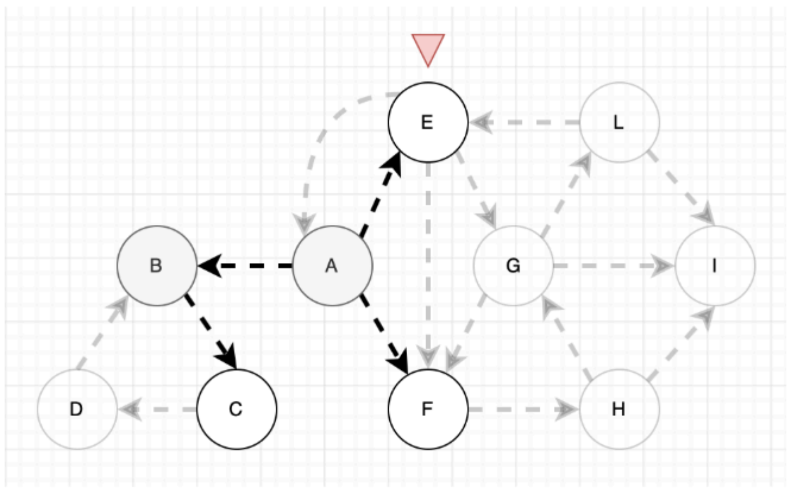

Start Node: A

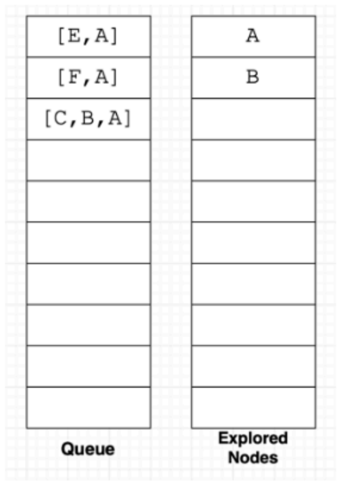
Queue	Explored Nodes
[A]	



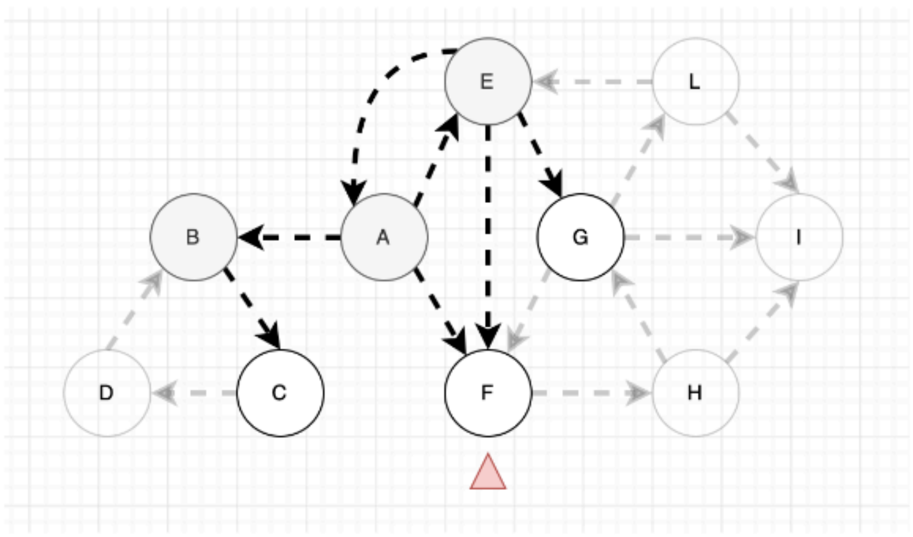


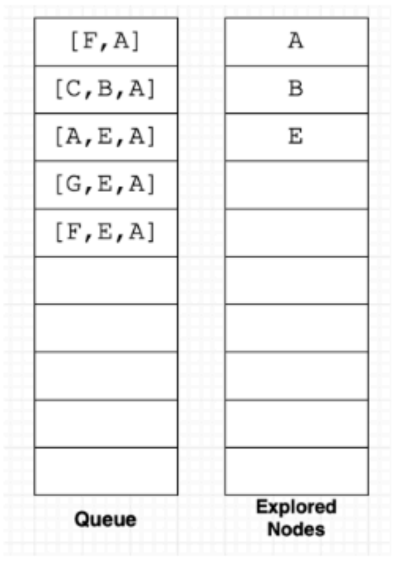
Queue	Explored Nodes
[B,A]	A
[E,A]	
[F,A]	



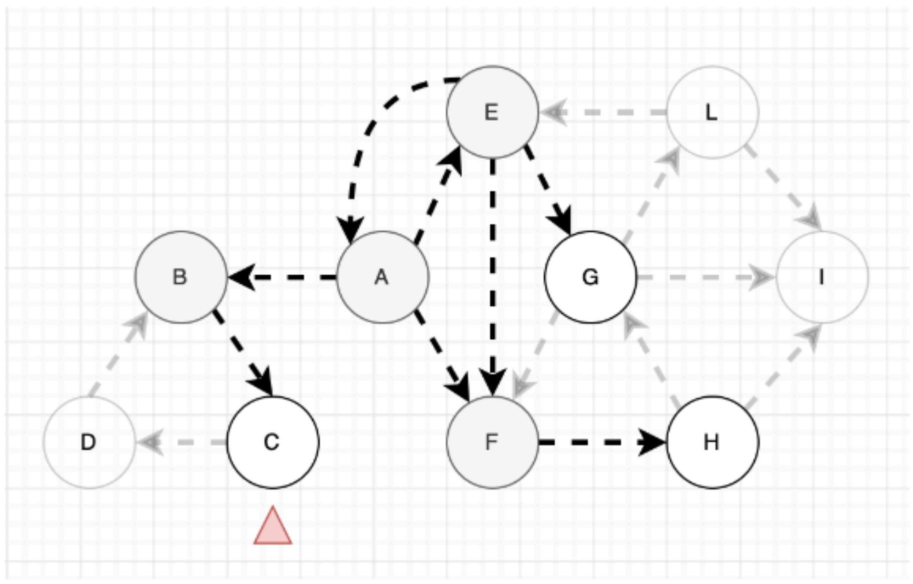


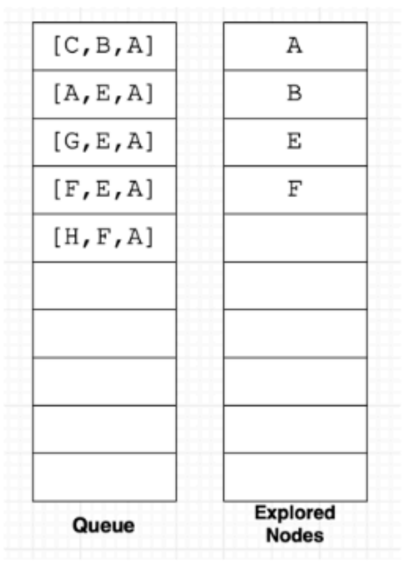
Queue	Explored Nodes
[E,A]	A
[F,A]	B
[C,B,A]	



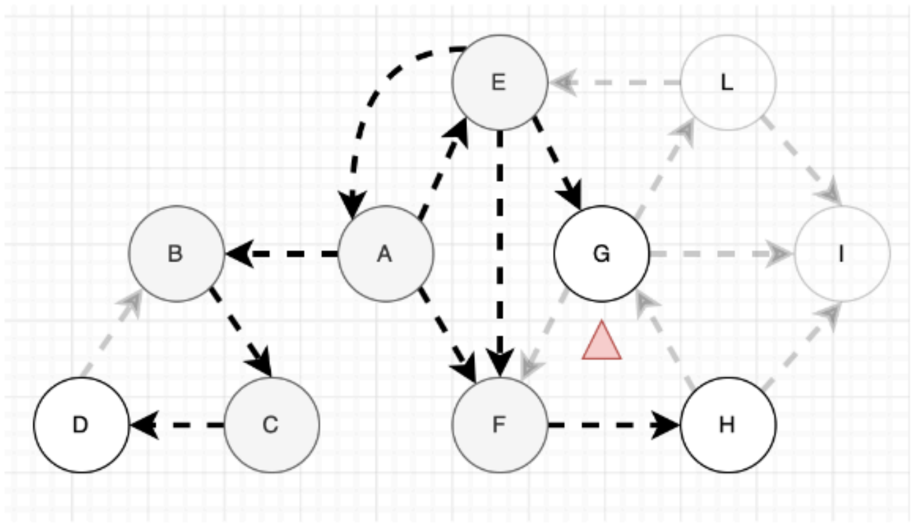


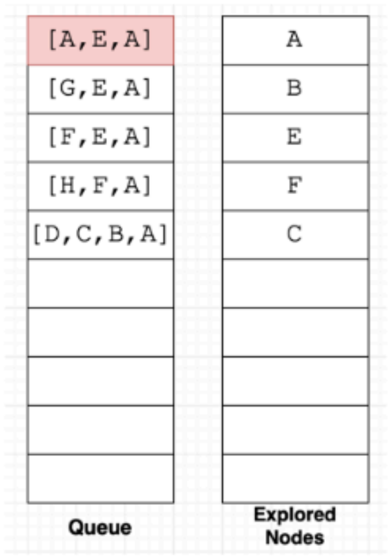
Queue	Explored Nodes
[F,A]	A
[C,B,A]	B
[A,E,A]	E
[G,E,A]	
[F,E,A]	



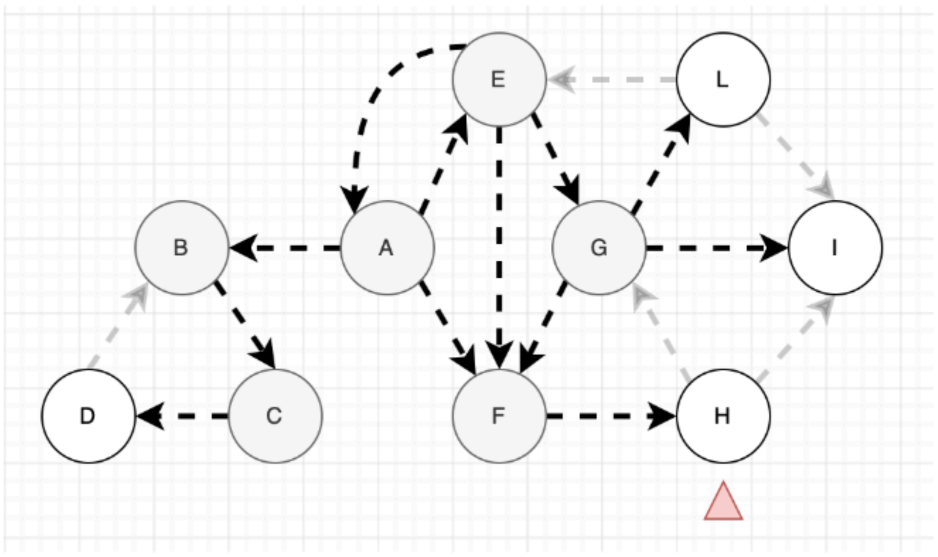


Queue	Explored Nodes
[C,B,A]	A
[A,E,A]	B
[G,E,A]	E
[F,E,A]	F
[H,F,A]	

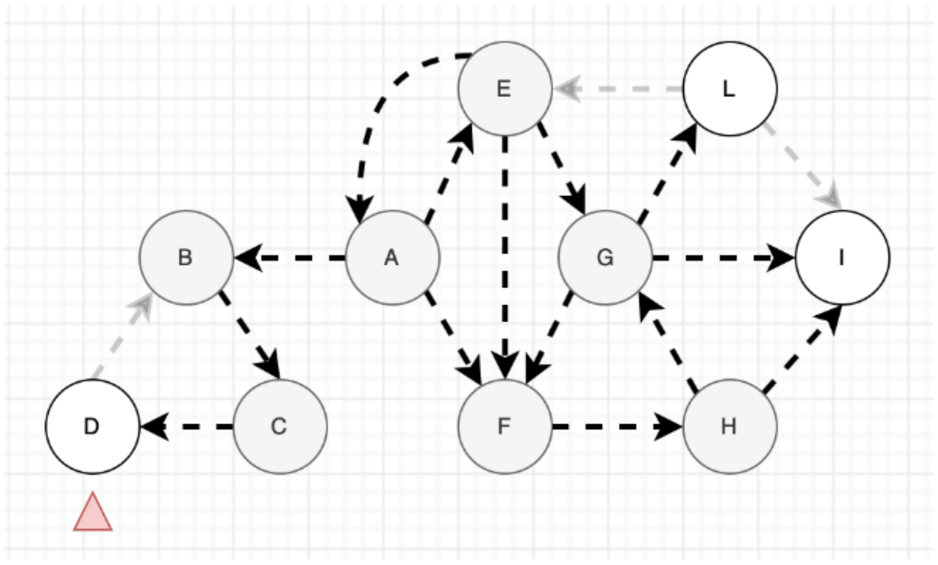




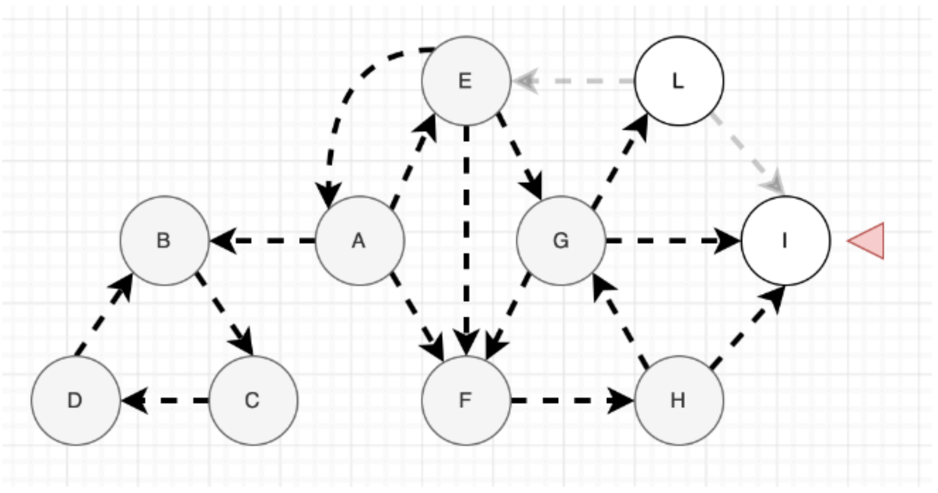
Queue	Explored Nodes
[A,E,A]	A
[G,E,A]	B
[F,E,A]	E
[H,F,A]	F
[D,C,B,A]	C



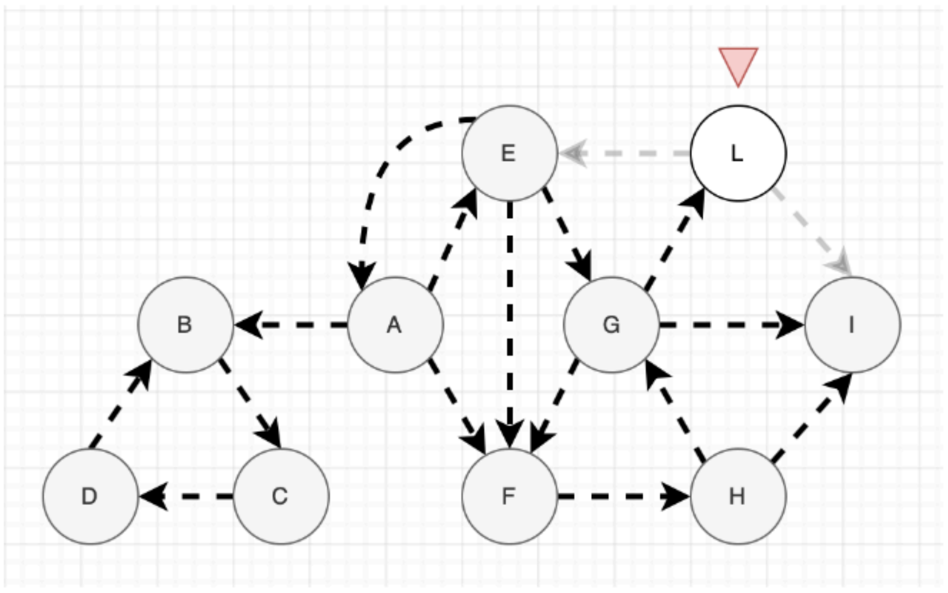
Queue	Explored Nodes
[F,E,A]	A
[H,F,A]	B
[D,C,B,A]	E
[F,G,E,A]	F
[I,G,E,A]	C
[L,G,E,A]	G

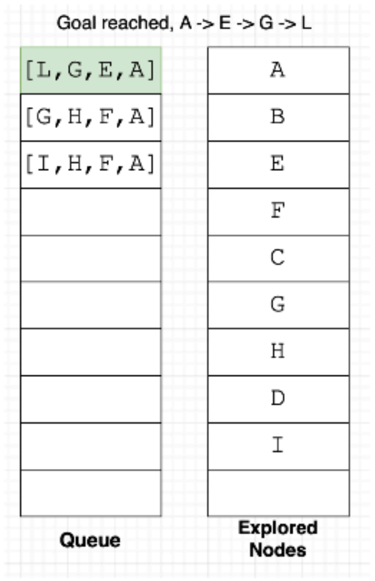


Queue	Explored Nodes
[D,C,B,A]	A
[F,G,E,A]	B
[I,G,E,A]	E
[L,G,E,A]	F
[G,H,F,A]	C
[I,H,F,A]	G
	H



Queue	Explored Nodes
[F,G,E,A]	A
[I,G,E,A]	B
[L,G,E,A]	E
[G,H,F,A]	F
[I,H,F,A]	C
	G
	H
	D

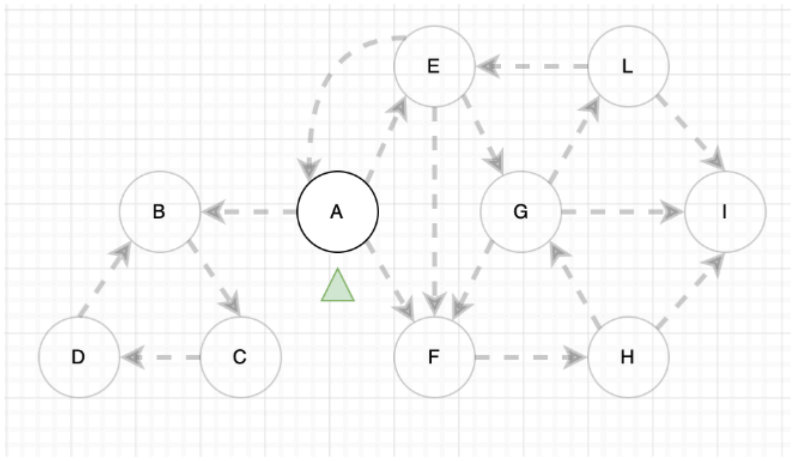


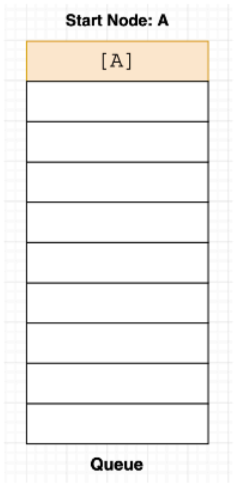

Goal reached, A -> E -> G -> L

Queue	Explored Nodes
[L,G,E,A]	A
[G,H,F,A]	B
[I,H,F,A]	E
	F
	C
	G
	H
	D
	I

Please see next page for DFS

1. (b) Depth-First Search on the provided graph where A is the start node and L is the destination node. Note that the branches with already explored first element are skipped (marked in red). I do not use an explored list for this search. Instead, I check if the head of the branch appears twice in the branch. Also I've omitted the cost as they're not useful for DFS.

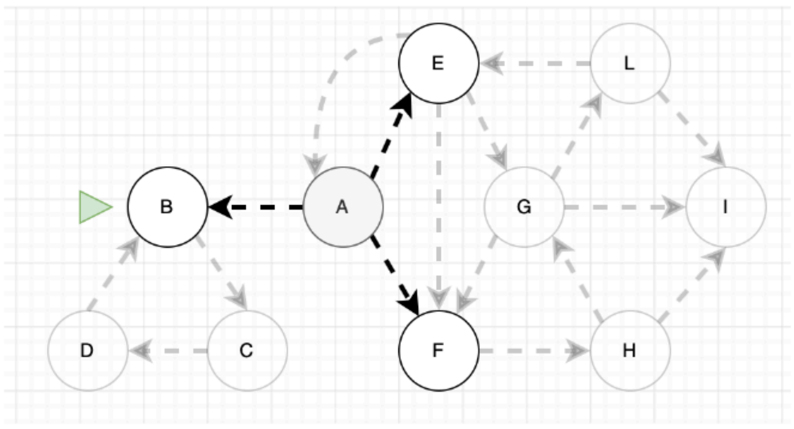


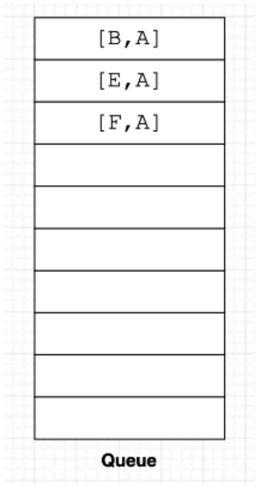


Start Node: A

[A]

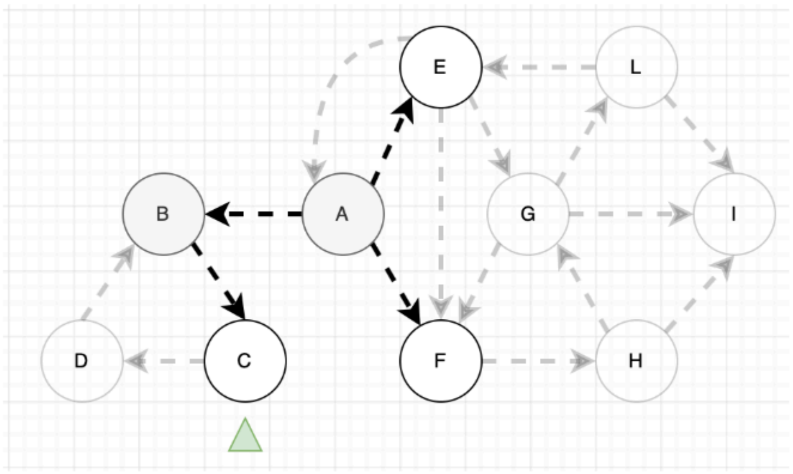
Queue

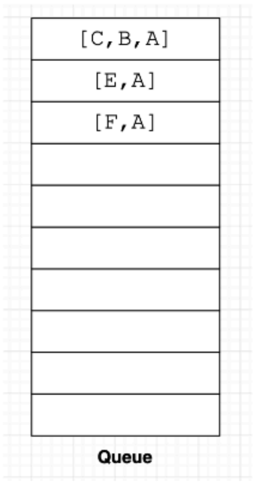




[B,A]
[E,A]
[F,A]

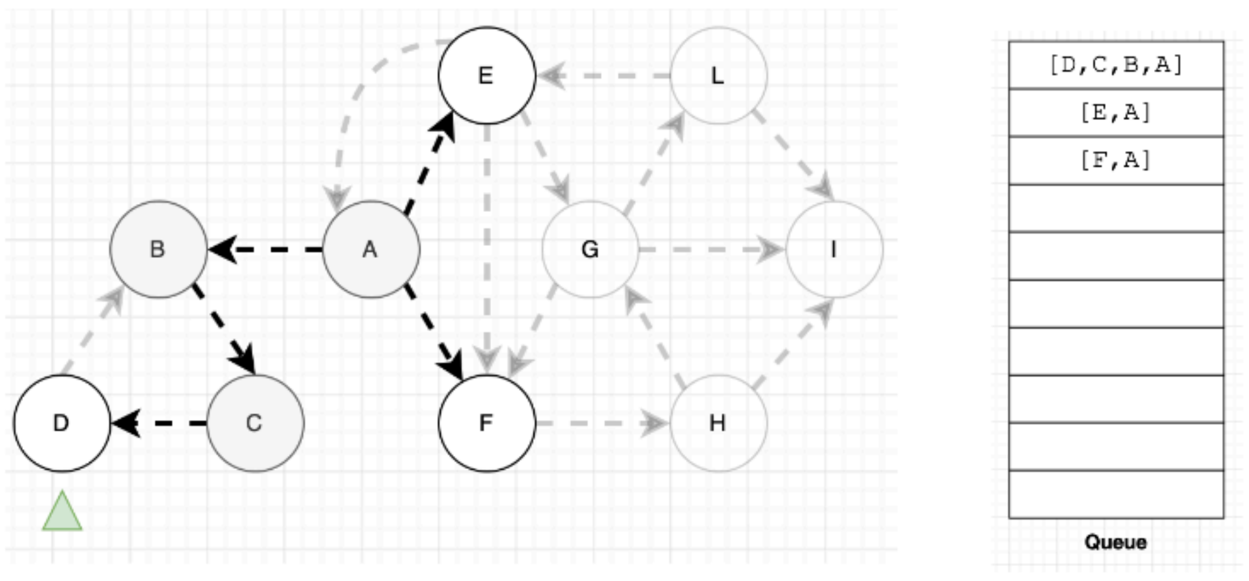
Queue



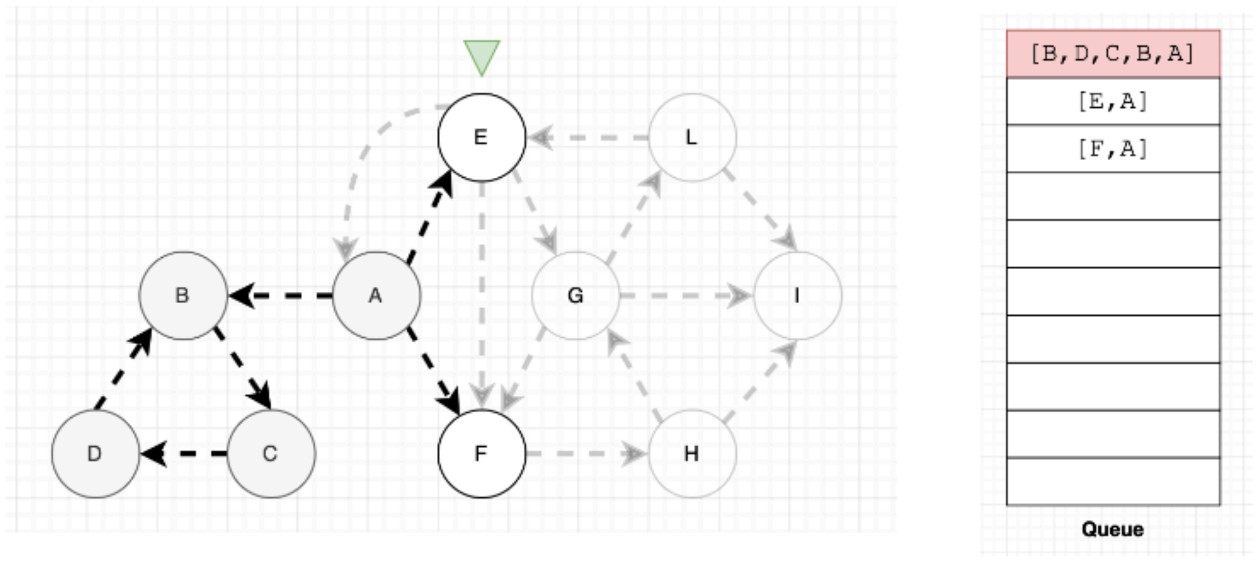


[C,B,A]
[E,A]
[F,A]

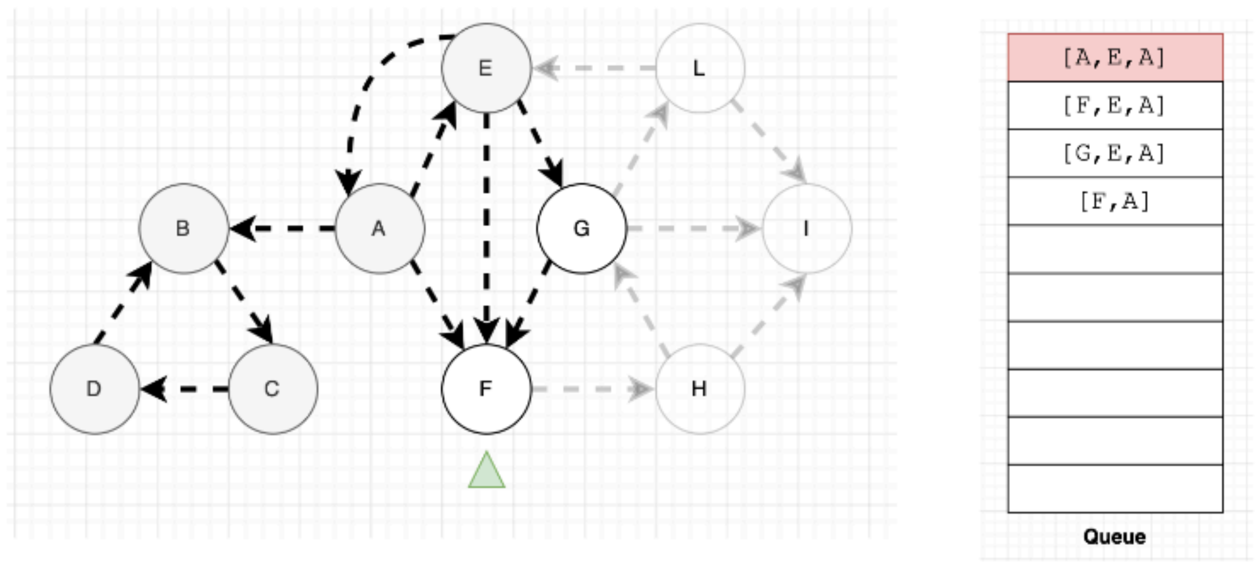
Queue



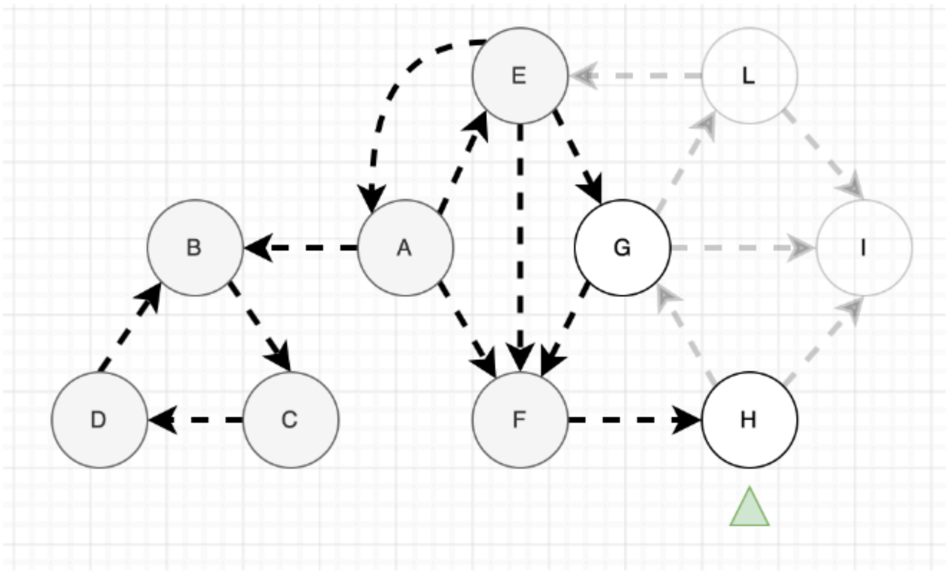
[D,C,B,A]
[E,A]
[F,A]



[B,D,C,B,A]
[E,A]
[F,A]

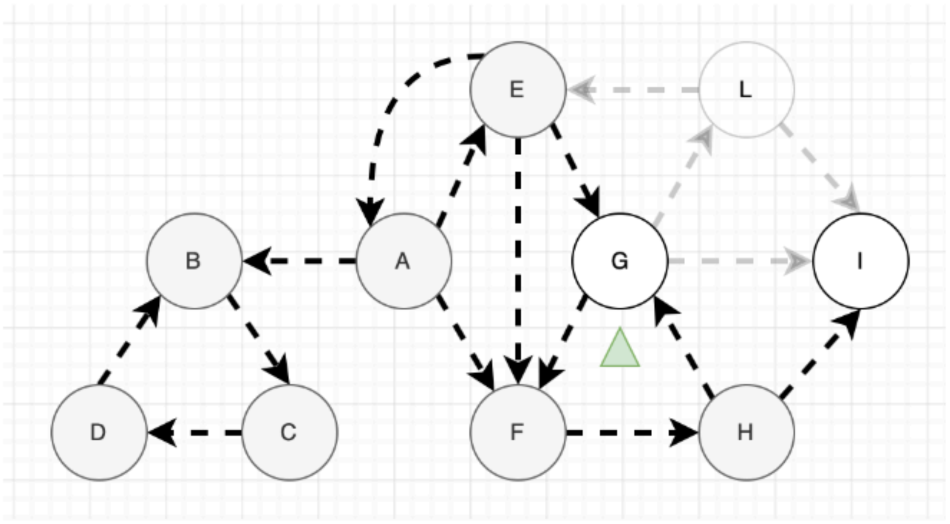


[A,E,A]
[F,E,A]
[G,E,A]
[F,A]



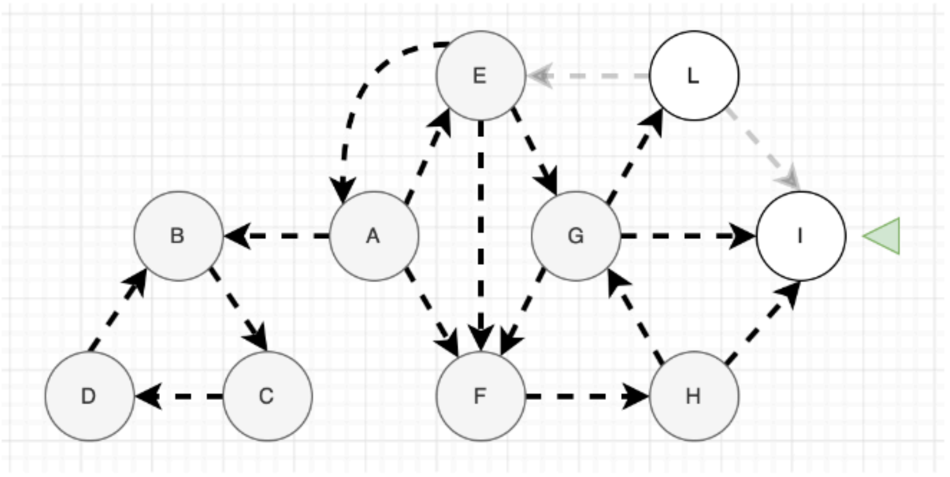
[H,F,E,A]
[G,E,A]
[F,A]

Queue



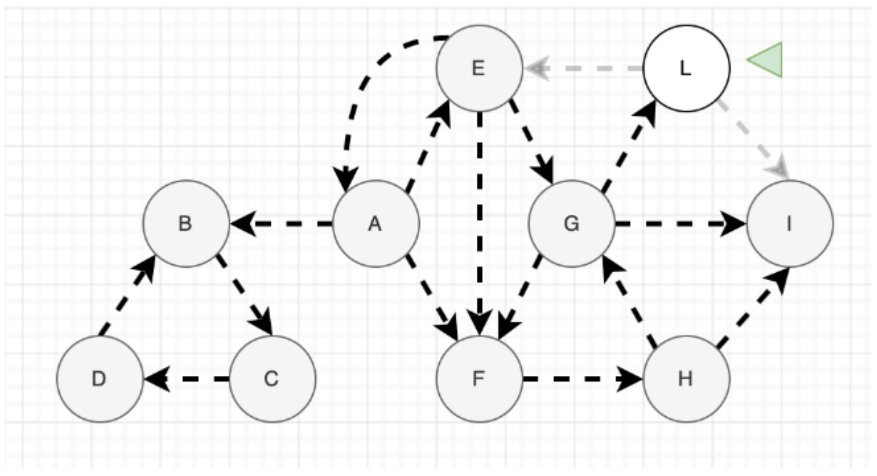
[G,H,F,E,A]
[I,H,F,E,A]
[G,E,A]
[F,A]

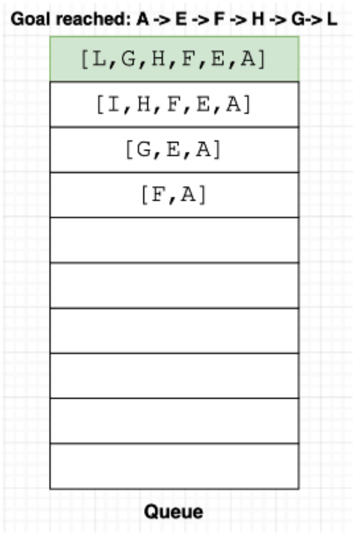
Queue



[F,G,H,F,E,A]
[I,G,H,F,E,A]
[L,G,H,F,E,A]
[I,H,F,E,A]
[G,E,A]
[F,A]

Queue



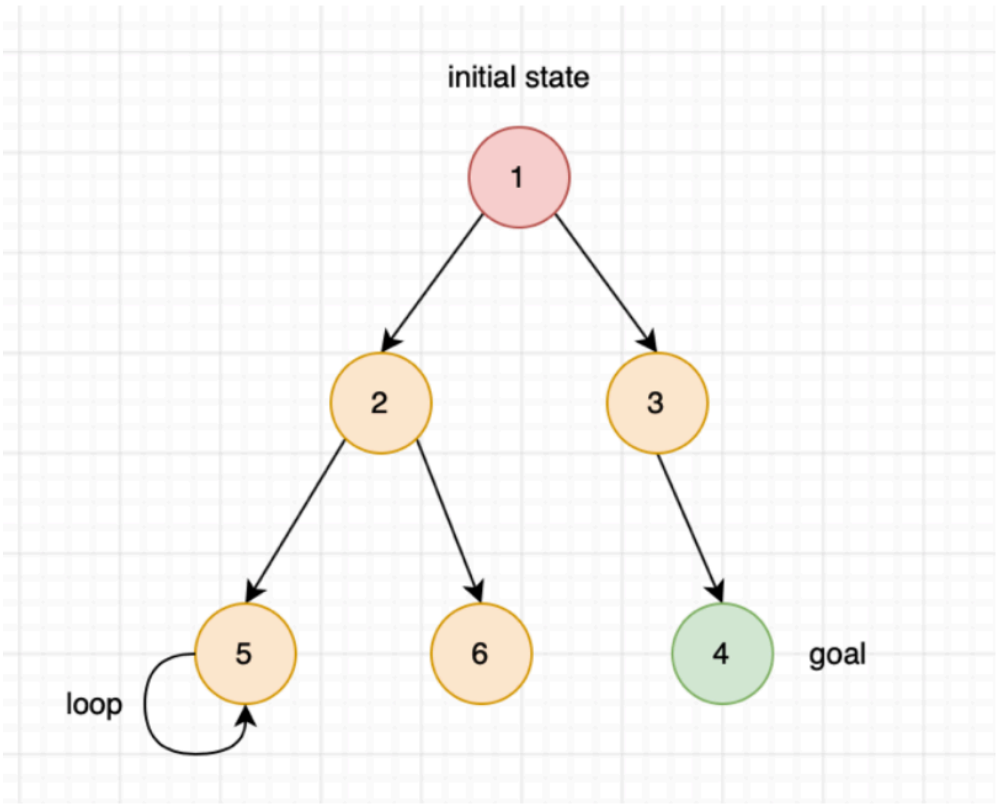


Goal reached: A -> E -> F -> H -> G-> L

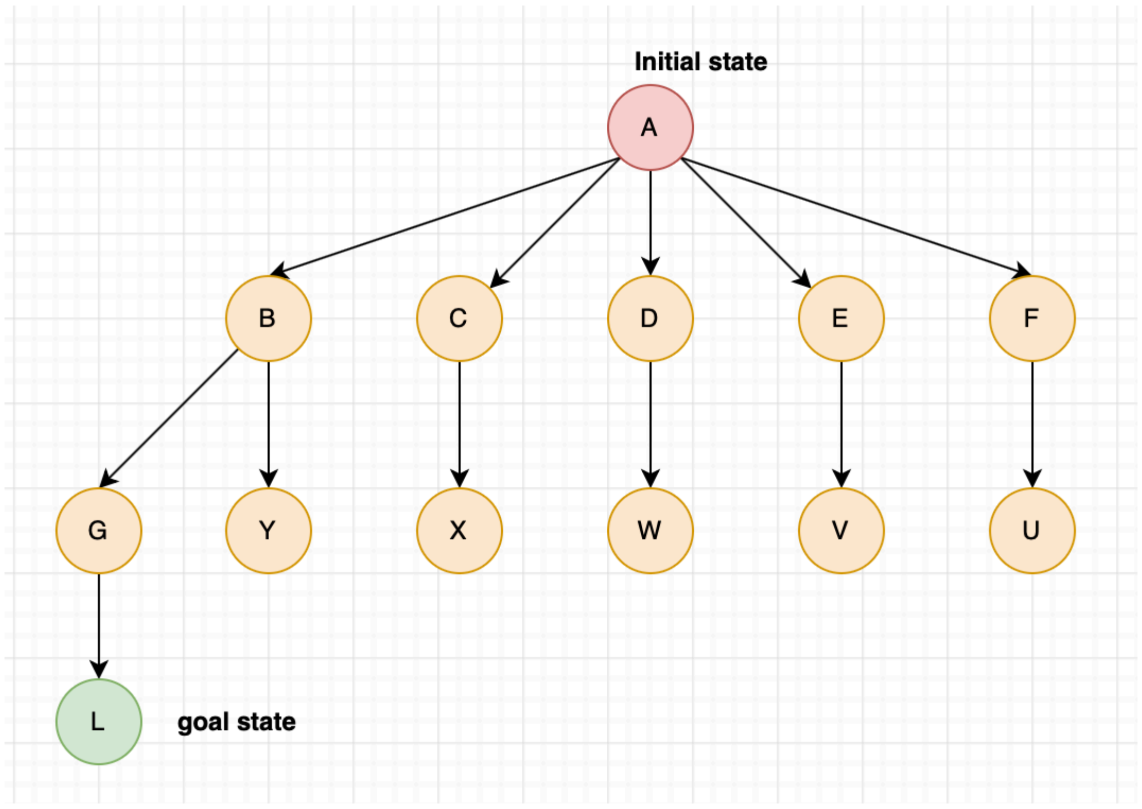
[L,G,H,F,E,A]
[I,H,F,E,A]
[G,E,A]
[F,A]

Queue

1. (c) Graph that favours BFS: I chose this graph because if we use Depth First Search instead, it will be stuck in an infinite loop and never reach the goal destination. It will first expand 1, than 2, then 5 and then get stuck in a loop after that and never search the other branches. If we use Breadth First search, we will expand 1 first, then 2, then 3, then 5, then 6 and then reach the destination.



1. (d) Graph that favours DFS: I chose this graph because if we use breadth first search instead, we will be expanding all the nodes before reaching the destination. We also notice there are no loops and only one way to reach the goal, so there are no optimal solutions to be found. Depth first search in this situation will only expand 3 nodes (A,B and G) before reaching the destination.



1. (e) Comparison between Depth-First-Search and Breadth-First - Search

- Frontier is a LIFO queue for DFS and a FIFO queue for BFS
- DFS has a better space complexity because it is linear O(bm) instead of O(b^d) for BFS (because it follows a single path).
- BFS has a better time complexity (O(b^d) instead of O(b^l) (DFS can easily get stuck in loops)
- DFS is better if the solution is dense and space is important
- BFS is better if we want to find the optimal solution (DFS is not optimal)
- DFS is not complete whereas BFS is complete (if b is finite)

2. Iterative Deepening Search

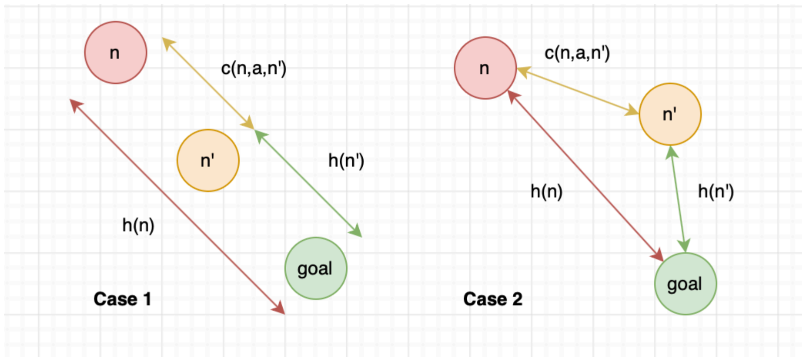
2. (a) Since our start node is A and our destination node is L, the optimal depth for the provided graph would be 3. If we use 3 as our depth limit then we will find the optimal path which is: $A \rightarrow E \rightarrow G \rightarrow L$. If we use 2 as our depth limit then our algorithm will never reach the destination as there is no such path and if we use 4 as our depth limit then the solution will not necessarily be optimal as we could obtain $A \rightarrow F \rightarrow H \rightarrow G \rightarrow L$ as a path.

2. (b) Comparison between Depth First Search and Iterative Deepening Search: Iterative Deepening Search combines Depth First search with Breadth first search. It allows to do a depth first search but with levels. The space complexity remains linear O(bd) but has an improved time complexity O(b^d). It solves the problem of depth first search being stuck in a loop. Because it gradually increments the depth limit, it gives us a complete an optimal result which is not the case of depth-first. So, if the solution is closer to the root, doing an Iterative Deepening Search is much more effective. So is it if there's a loop in the tree/graph.

4.3. Informed Search — Questions

1. Heuristics

1. (a) The heuristic needs to be consistent and admissible. An admissible heuristic never overestimates the true cost to reach the goal, it is optimistic. Since straight line distance is the shortest distance between two points then straight line distance is an admissible heuristic. A heuristic is consistent if for every node n, every successor n′ of n generated by any action a $h(n) \leq c(n, a, n') + h(n')$. Where, h(n) is the straight line distance from n to the goal, c the straight line distance between n and its successors, and h(n′) is the straight line distance between that successor and the goal. h(n) cannot be grater than the RHS because there are only two possible scenarios:



In the first case, $h(n) = c(n, a, n') + h(n')$ because the intermediate node is « on » the straight line. In the second case however, $h(n) < c(n, a, n') + h(n')$ as we are not on the straight line anymore. This also uses (again) the fact that straight line distance is the shortest distance between two points. Therefore straight line distance is a consistent heuristic. **Hence, since straight line distance satisfies both criteria, we can conclude that it is a valid heuristic.**

1. (b) Three problems that can be solved with A* search and their heuristic:

- Number of different bits in bit string of same length using Hamming distance or Lee Distance. (Shortest path from one sequence to the other)
- Number of different letters in a sequence of letters using Levenshtein distance or Jaro Winkler distance. (Shortest path from one sequence to the other)
- Shortest path between two locations along the curve of the earth using great-circle distance

2. Greedy best first and A* Search comparison

2. (a) Main differences between Greedy Best First Search and A* Search:

- The function it uses to evaluate. Greedy Best First search explores nodes that appear closer to the goal (i.e uses only the heuristic $f(n) = h(n)$) whereas A* search avoids paths that are already costing too much by considering the cost so far as well $f(n) = h(n) + g(n)$.
- If the heuristic is well chosen, A* search is complete (unless infinitely many nodes) and optimal whereas Best First isn't (that is there is a chance of taking a path that does not lead to the goal).
- A* has an exponential time complexity whereas Best First has time complexity O(b^m) which can be made a lot better if we use a very good heuristic.
- Greedy Best First can get stuck in loops

2. (b) Changes that need to be made to the code in order to make it a Greedy best first search:

- We do not need the cost function anymore.
- When we sort the branches, we do not need to sort all the branches but only the next ones (those generated by the next function) as greedy best first behaves like a depth first search (since it follows a single path all the way to the end).

5.4. Connect Four — Questions

2. You can either rotate Left or Right or Not Rotate and you can rotate before and/or after placing a piece. Hence there are 3^2 = 9 possibles ways to rotate. Since there are 16 openings, there are 16 possible ways to place a piece. So in total there are 9 * 16 = 144 possible actions a player can perform during one term.

3. In Connect Four with a twist you can only rotate left after placing a piece, that is, there are only 8 possible actions a player can perform during one term. So finding the next action is easier than having to go through 144 possible actions each time. One problem with that is that the time efficiency of alpha beta pruning on large trees is not really good. So we would end up with a very slow algorithm. However, alpha beta can still be a practical implementation if we add some changes to it. There are two ways of doing that:

- The first one is adding another evaluation faction that estimates if a given path is « interesting », That is we won't have to explore the whole tree, we just have to come up with an estimation of how desirable that branch is. (For example we could find the way to see how possible it is to have a succession of 3 pieces of the same color.)
- The other way is by combining alpha-beta with another algorithm like iterative deepening or by using Quiescence search which allows us to defer the evaluation of a state until it is stable enough to be evaluated. Hence helping a lot with the time efficiency of the algorithm.