REPORT Lab 2

*Group 2, Subgroup 7 (**prigu857** and **fabcr549**)*

Task 1

In the first part of the task, after a quick analysis of the context, we implemented a basic graph search algorithm based on the one in Figure 3.7 of the course's reference book (3rd edition). The basic search algorithm given a problem p tries to return a path to problem's p goal.
To achieve its search goals, the algorithm keeps two main data structures in memory:

- A "frontier" of node to be processed, stored as NodeQueue, which is a data structure that can act both as FIFO and LIFO queue.
- A HashSet of already explored states.

The algorithm from the problem's initial state starts to process the node in the frontier: if the state reached is the same as the goal one, it just returns the path to it, otherwise it will analyze all the states reachable from the current one and add them to the frontier. The HashSet of explored nodes is updated accordingly.

The program is structured such that by using OOP, specifically inheritance, two additional search algorithms are defined: Depth-First Search (DFS) and Breadth-First Search (BFS).

The two algorithms share the same implementation coming from the basic algorithm described above. The main difference between them is given by how the frontier queue is populated: BFS is implemented by using a FIFO frontier queue, while DFS uses a LIFO one. The way the frontier queue acts is defined in the basic implementation.

Task 2

1. In the vacuum cleaner domain in part 1, what were the states and actions? What is the branching factor?

- States: Agent's position (x,y).
- Actions: There are 5 actions available to agent – Go North, Go South, Go West, Go East and Suck Dirt
- Branching factor: It is defined as the number of successor nodes and as the agent can go in 4 directions, we have 4 successor nodes, Thus, Branching factor is 4.

2. What is the difference between Breadth First Search and Uniform Cost Search in a domain where the cost of each action is 1?

Uniform-Cost Search works by processing first the nodes with lower cost in the "frontier". Breadth-First search just process nodes based on the order they were inserted into it. Given this explanation, when the cost of each action is 1, no difference will be noticed during the execution of both algorithms.

3. Suppose that h1 and h2 are admissible heuristics (used in for example A*). Which of the following are also admissible?

a) (h1+h2)/2

b) 2h1

c) max (h1, h2)

An admissible heuristic is one that never overestimates the cost to reach the goal state.
As we know $h1(n) \leq cost(n)$ and $h2(n) \leq cost(n)$ (h1 and h2 are admissible)

a. $(h1+h2)/2 \leq cost(n)$ (a is admissible)
b. 2h1 may or may not be less than cost(n), so it is like overestimating the cost to reach the goal state, hence, it will not be admissible.
c. $Max(h1, h2) \leq cost(n)$ (c is also admissible)

4.If one uses A* to search for a path to one specific square in the vacuum domain, what could the heuristic function (h) be? What could the cost function (g) be? Is your choice of (h) an admissible heuristic? Explain why.

A heuristic function (h) that fits well in this context could approximate the distance between our current position and the goal position. This distance could be calculated as the measure of the straight line between the two coordinates.

A good cost function (g) should be monotonic and therefore always increase when going along a path. A cost function(g) could be based on the number of nodes that separate us from our initial position. H(), in this case, is an admissible heuristic because it always gives an optimistic approximation of the goal's cost.

5. Draw and explain. Choose your three favorite search algorithms and apply them to any problem domain (it might be a good idea to use a domain where you can identify a good heuristic function). Draw the search tree for them and explain how they proceed in the searching. Also include the memory usage. You can attach a hand-made drawing.

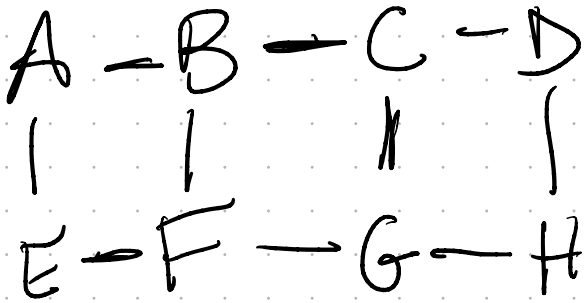
See attachment, memory usage can be deduced from how the queue gets populated during the various steps. DFS is the most memory efficient algorithm in this case since in a hypothetical implementation we could just traverse the whole graph by using recursion or a stack.

6. Look at all the offline search algorithms presented in chapter 3 plus A* search (i.e., Best-first search, Breadth-first search, Uniform-cost search, Depth-first search, Iterative deepening search, Bidirectional search, Greedy best-first search and A* search). Are they complete? Are they optimal? Explain why!

Algorithms	Is complete?	Is optimal?
Best-first search	No, can get stuck in infinite loop, if revisiting the already explored states is not taken into checked manually.	No, as it does not consider the cost of the path taken.
Breadth-first search	Yes, if the branching factor (max no. Of child nodes) is finite	Yes, if each step's cost is the same, as it explores nodes at the same level before going to the next.
Uniform-cost search	Yes, if the cost of reaching each node is positive, as it always explores the node with lowest cost.	Yes, as it always explores the nodes in increasing order of their path cost.
Depth-first search	Yes, if the state space is finite, as it explores the path at maximum depth.	No, as it does not guarantee that the path returned is the shortest one.
Iterative deepening search	Yes, if the state space is finite and the cost of reaching each node is positive.	No, as it explores in depth-first manner, so it may return the path which is not shortest.
Bidirectional search	It is complete if both the forward and backward search is complete, or else we will term it as not complete.	In general, this does not grantees optimality, but if it follows uniform cost bidirectional search then it can be termed as optimal.
Greedy best-first search	No, as it can get stuck in loop same as best-first search.	No, as it just considers the heuristic function without the cost of reaching the node.
A* search	Yes, if there is finite branching factor and heuristics function is admissible.	Yes, as it considers both heuristics function and cost of reaching the node to reach the goal efficiently.

7. Assume that you had to go back and do Lab 1/Task 2 once more (if you did not use search already). Remember that the agent did not have perfect knowledge of the environment but had to explore it incrementally. Which of the search algorithms you have learned would be most suited in this situation to guide the agent's execution? What would you search for? Give an example.

We did use search for the first lab. We chose Depth First Search because among the goals specified in the lab's instructions, the agent had to explore the entire world. Since the world was not known in advance, the goal chosen was to reach the first unexplored node. The process was repeated until all unexplored nodes were reached, and the search algorithm stopped returning valid paths. Since the cost for reaching every node in the world was the same, Depth First Search seemed a valid choice. DFS returns the first solution, even if it is not the cheapest (not relevant in this context) thus giving reliable performance in this context.

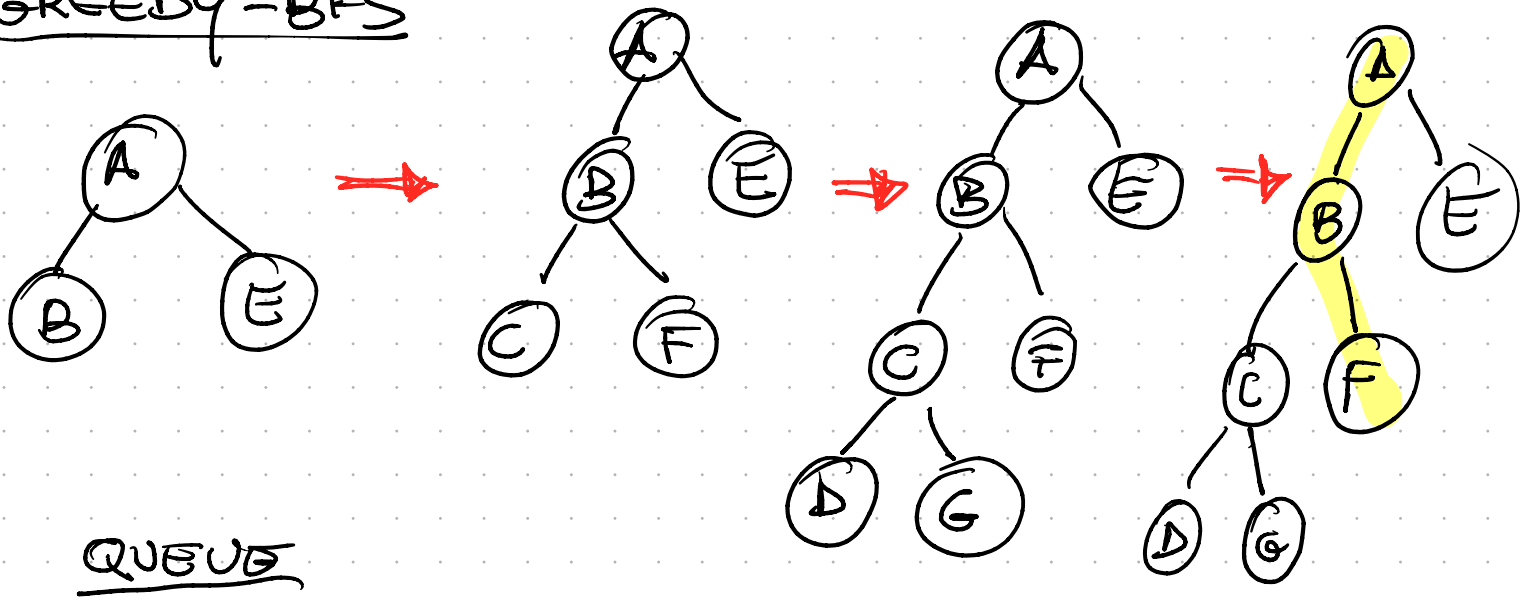


GOAL: FINDING SHORTEST PATH OF CONNECTION BETWEEN TWO USERS ON A SOCIAL NETWORK

H: NUMBER OF SHARED FRIENDS, (NUMBER OF EDGES)

Ex. SHORTEST PATH FROM A TO F.

GREEDY-BFS

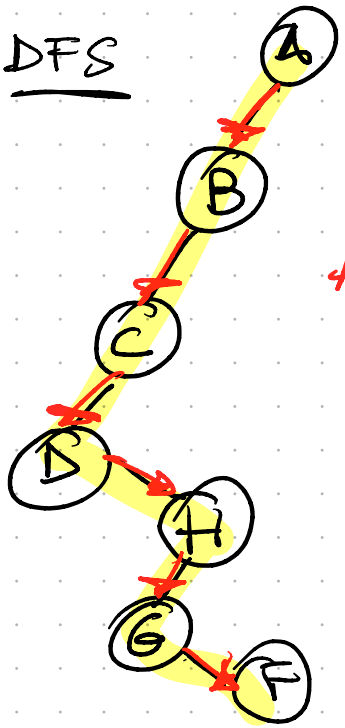


QUEUE

1. A
2. B, E
3. E, C, F
4. E, F, D, G
5. E, D, G

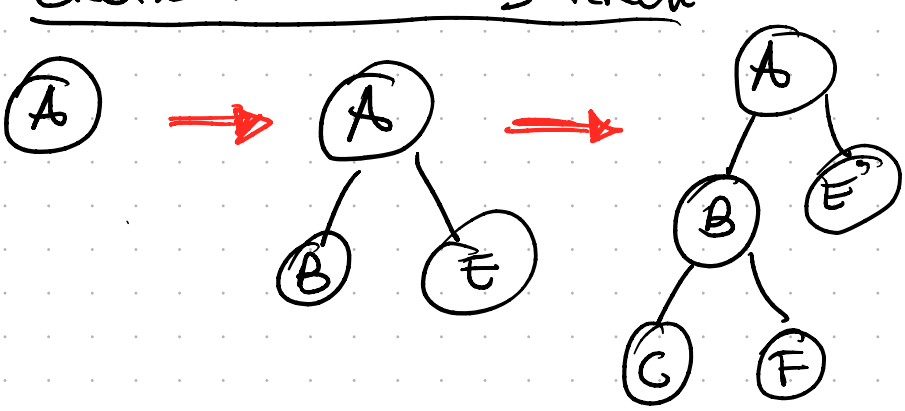
A → B → F

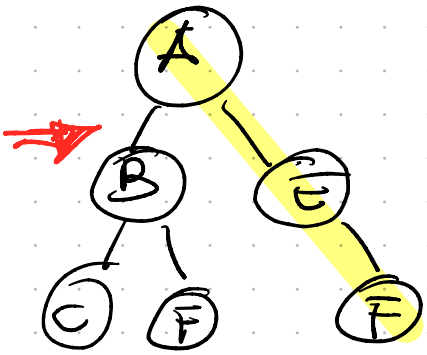
DFS



A → B → C → D → H → G → F

BREADTH-FIRST SEARCH





A → E → F

QUEUE

1. A
2. B, E
3. E, C, F
4. C, F