GT Lab Practical 02 - Search Algorithm (DFS, BFS) and Graph

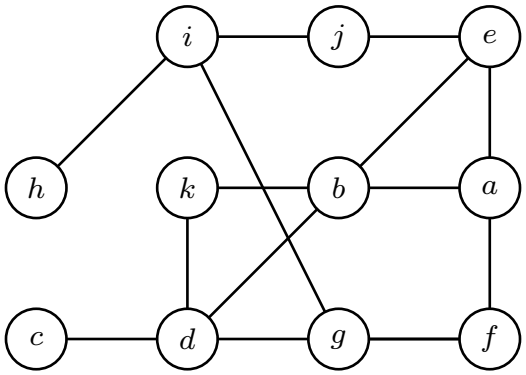
Lecturer: Pheak NEANG, Vinha CHHAENG

October 31, 2025

1. Introduction of search algorithm

A search algorithm is a method used to systematically explore alternative solutions in problem-solving, typically modeled as a search process within a problem-space graph, where nodes represent possible states and edges represent operators that transition between those states. In this section, we focus on two widely used algorithms, Depth First Search (DFS) and Breadth First Search (BFS), and demonstrate their operation through graph-based simulations. We will discuss the working principles of each algorithm and explore their respective applications in problem-solving.

Given an undirected graph G as illustrated below, we aim to explore the process of visiting each node based on this algorithm.



Question 1. Exploration of Graph Structure and Search Algorithms

(1.1). Let the graph $G = (X, U)$, where X denotes the set of vertices and U represents the set of edges in graph G. Determine the sets X and U.

(1.2). Given the neighborhood lists and degrees of vertices (a, b, c).

(1.3). Given a path from vertex c to vertex a, determine how many possible paths exist between c and a.

(1.4). Given the result of the Breadth-First Search (BFS) algorithm on the previous graph starting from vertex c to vertex a, determine how many possible Depth-First Search (DFS) traversals exist from c to a.

(1.5). Given the result of the Depth-First Search (DFS) algorithm on the previous graph starting from vertex c to vertex a, determine how many possible Breadth-First Search (BFS) traversals exist between c and a.

(1.6). Based on your results from questions (1.4) and (1.5), which algorithm performs better? Can this observation be generalized in theory?

Question 2. Time Complexity of Graph Algorithms

(2.1). Give a general definition of time complexity in algorithm analysis.

(2.2). Determine the time complexity of the Breadth-First Search (BFS) algorithm for a given graph $G = (X, U)$. Express the result in terms of the number of vertices $|X|$ and edges $|U|$.

(2.3). Determine the time complexity of the Depth-First Search (DFS) algorithm for the same graph $G = (X, U)$. Express the result in terms of $|X|$ and $|U|$.

Question 3. Graph Simulation and node exploration (DFS) and (BFS) in Python using `NetworkX`.

(3.1). Recall the network G by setting its name as `graph` using the function `nx.Graph()` to create an undirected graph.

(3.2). Create two lists, `ver` and `arc`, and a dictionary named `pos` to store all elements of the vertex set X, the edge set U, and the positions of the vertices in the graph network G, respectively.

(3.3). Build the graph network $G(X, U)$ by adding the vertices X and edges U obtained from variable `ver` and `arc` using the functions `graph.add_nodes_from()` and `graph.add_edges_from()`.

(3.4). To visualize the graph, first draw the nodes using the function `nx.draw_networkx_nodes()`, then add labels to the nodes with `nx.draw_networkx_labels()` then draw the edges using `nx.draw_networkx_edges()` and finally `plt.show()` to display this graph.

(3.5). List all possible paths from node c to node h using the function `all_simple_paths(graph, c, h)`, and determine how many such paths exist between c and h.

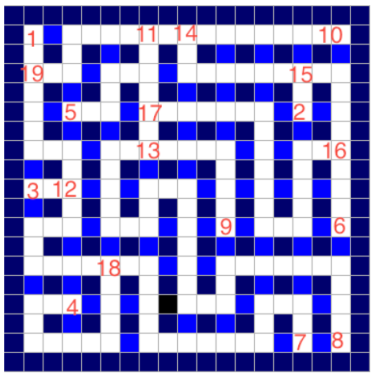
(3.6). Explore the Depth First Search (DFS) algorithm on the given graph G, starting from the node c, by using the function `nx.dfs_edges()`. Next step implement your own function named `explore_dfs()` to stop the DFS traversal to be reaching the goal node, and explore the path from node c to node h. Finally, draw the update graph by adding blue color on this path.

(3.7). Explore the Breadth First Search (BFS) algorithm on the given graph G, starting from the node c, by using the function `nx.bfs_edges()`. And then, implement your own function named `explore_bfs()` to stop the BFS traversal to be reaching the goal node, and explore the path from node c to node h. Finally, draw the update graph by adding red color on this path.

(3.8). Repeat the experiments from questions (3.6) and (3.7) several times. Do you observe any different paths from the starting node to the goal node? If not, modify both functions `explore_dfs()` and `explore_bfs()` to be more generalized for finding paths using the (DFS) and (BFS) algorithms. Run each modified algorithm (BFS and DFS) 10 times, and then provide a remark on which algorithm can explore better.

2. Application of DFS and BFS algorithm on Maze game

Given that a maze 19×19 grid is created as an environment, where the white color indicates the possible paths to travel, and the blue and black colors represent walls that cannot be crossed. The specific positions are denoted by numbers in the range $i, j \in [1, 19]$, as demonstrated in the figure below. This section aims to apply search strategies such as Depth First Search (DFS) and Breadth First Search (BFS) to find a path from the starting position $\textcircled{i}$ to the ending position $\textcircled{j}$.



Question 1. In the file name `graph.py`, define the nodes as values in the range $\in [1, 19]$, and the edges as available paths (e.g., $1 \rightarrow 19, \ldots$). Apply **BFS** and **DFS** in the graph environment to test the search from the starting node $i = 1$ to the ending node $j = 2$. Finally, visualize the resulting paths.

Question 2. In the file name `mask.py`, design a maze of size 19×19 where white cells represent available paths and blue or black cells represent walls. Represent the maze as a 2D array with 1 for paths and 0 for walls. Apply **BFS** and **DFS** in the maze environment to test the search from the starting node $i = 1$ to the ending node $j = 2$. Finally, visualize the resulting paths.

Note

- Summit only file `mask.py` and `graph.py`.
- For both files `mask.py` and `graph.py` you need created by yourself.