

School of Computer Science Engineering and Technology

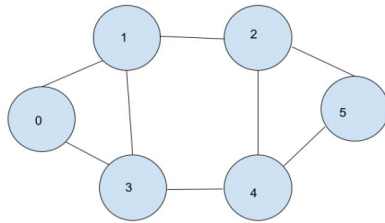
Assignment-02

Course- B.Tech Type- Core
Code-23CS106 Course Name- Artificial Intelligence & Machine Learning
Year- 2024-2025 Semester- Even
Date- 10-01-2024 Batch- AIML-A,B

HALL TICKET NUMBER : 2203A51192
NAME OF THE STUDENT : SRI VIKRAM VELDE
BATCH NUMBER : 17

Lab Exercise - Implement Breadth First, Depth First and A* Search Algorithms

Part 1 – Implement Breadth First Search Algorithm using a Queue



1. Given a graph with adjacency list and a starting vertex and we have to traverse the graph.
2. We will first print the value in the starting vertex,
3. Continue to print the value of neighbors of the starting vertex and
4. Next move on to the next level after completing the current level till all the vertices of the graph are printed.

```
graph = {
    '0': ['1', '3'],
    '1': ['2', '3'],
    '2': ['4', '5'],
    '3': [],
    '4': ['5'],
    '5': []
}
visited = []
queue = []
def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)
    while queue:
        m = queue.pop(0)
        print(m, end = " ")
        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)
print(" Breadth-First Search for the graph ")
bfs(visited, graph, '0')
```

Breadth-First Search for the graph
0 1 3 2 4 5

Release notes X

Please follow our [blog](#) to see more information about new features, tips and tricks, and featured notebooks such as [Analyzing a Bank Failure with Colab](#).

2024-01-29

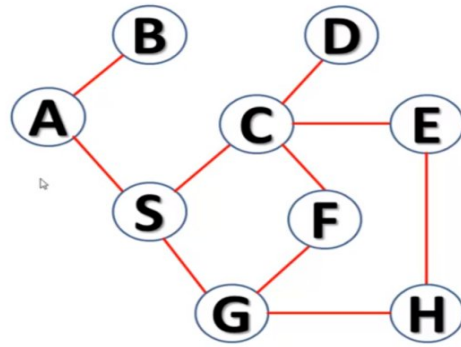
- New [Kaggle Notebooks](#) <- [Colab updates!](#) Now you can:
 - Import directly from Colab without having to download/re-upload
 - Upload via link, by pasting Google Drive or Colab URLs
 - Export & run Kaggle Notebooks on Colab with 1 click
- Try these notebooks that talk to Gemini:
 - [Gemini and Stable Diffusion](#)
 - [Learning with Gemini and ChatGPT](#)
 - [Talk to Gemini with Google's Speech to Text API](#)
 - [Sell Lemonade with Gemini and Sheets](#)
 - [Generate images with Gemini and Vertex](#)
- Python package upgrades
 - google-cloud-aiplatform 1.38.1 -> 1.39.0
 - bigframes 0.18.0 -> 0.19.2
 - polars 0.17.3 -> 0.20.2
 - gdown 4.6.6 -> 4.7.3 ([GitHub issue](#))
 - tensorflow-hub 0.15.0 -> 0.16.0
 - flax 0.7.5 -> 0.8.0
- Python package inclusions
 - sentencepiece 0.1.99

2024-01-08

- Avoid nested scrollbars for large outputs by using `google.colab.output.no_vertical_scroll()` [Example notebook](#)
- Fix [bug](#) where downloading models from Hugging Face could freeze
- Python package upgrades
 - huggingface-hub 0.19.4 -> 0.20.2

0s completed at 7:36 PM

Part 2 – Implement Depth First Search Algorithm using a Stack



0.1 DFS Stack implementations steps to be followed:

1. Start at the root node and push it onto the stack.
2. Check for any adjacent nodes of the tree and select one node.
3. Traverse the entire branch of the selected node and push all the nodes into the stack.
4. Upon reaching the end of a branch (no more adjacent nodes) ie nth leaf node, move back by a single step and look for adjacent nodes of the n-1th node.
5. If there are adjacent nodes for the n-1th node, traverse those branches and push nodes onto the stack.

```
graph1 = {
    'A': ['B', 'S'],
    'B': ['A'],
    'C': ['D', 'E', 'F', 'S'],
    'D': ['C'],
    'E': ['C', 'H'],
    'F': ['C', 'G'],
    'G': ['F', 'S'],
    'H': ['E', 'G'],
    'S': ['A', 'C', 'G']
}

def dfs(graph, node, visited):
    if node not in visited:
        visited.append(node)
        for k in graph[node]:
            dfs(graph, k, visited)
    return visited

visited = dfs(graph1, 'A', [])
print(visited)
```

['A', 'B', 'S', 'C', 'D', 'E', 'H', 'G', 'F']

Part 3 – Implement A* Algorithm using Numpy

1. A*Algorithm (pronounced as A-star) is a combination of 'branch and bound search algorithm' and 'best search algorithm' combined with the dynamic programming principle.
2. The A* Algorithm is well-known because it is used for locating path and graph traversals.
3. This algorithm is used in numerous online maps and games.
4. It uses a heuristic or evaluation function usually denoted by $f(X)$ to determine the order in which the search visits nodes in the tree.

5. The heuristic function for a node N is defined as follows:

$$f(x) = g(x) + h(x) \quad (1)$$

where $g(x)$ is the actual cost estimate, $h(x)$ is the heuristic cost estimate for the gives

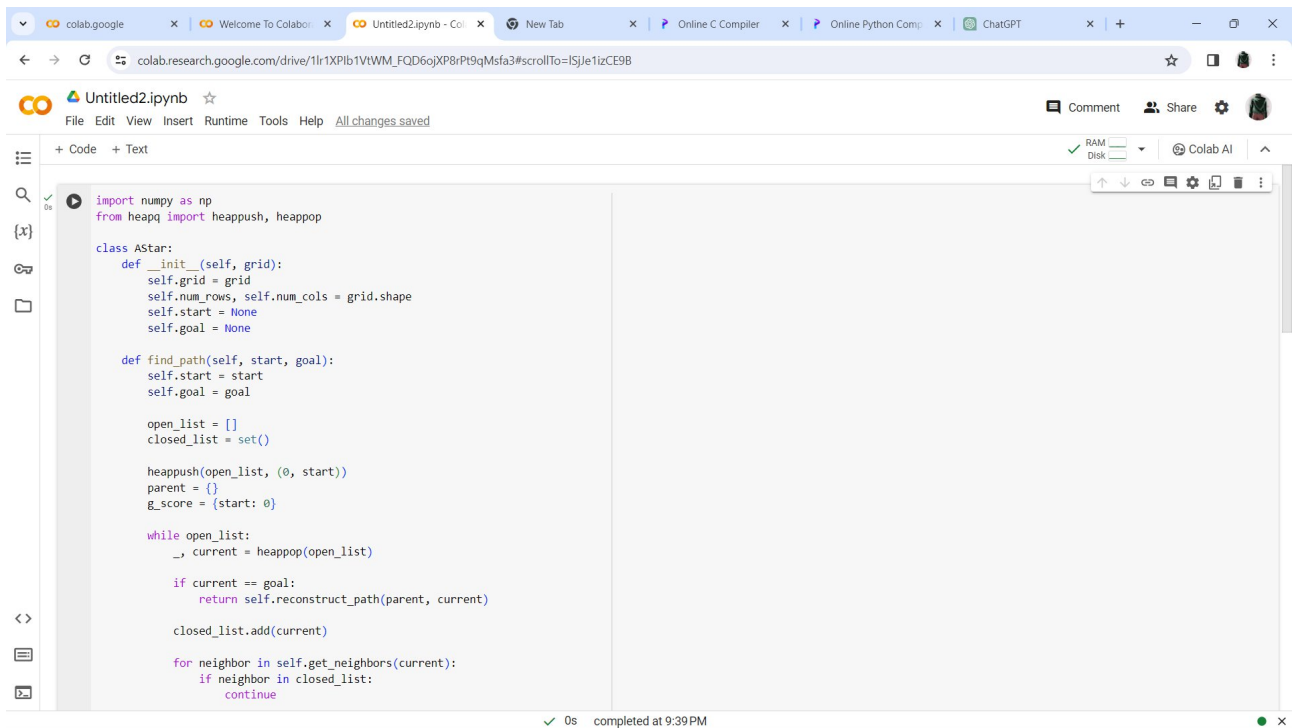
2	8	3
1	6	4
7		5

INITIAL STATE

1	2	3
8		4
7	6	5

FINAL STATE

states of the 8-puzzle problem.



```
import numpy as np
from heapq import heappush, heappop

class AStar:
    def __init__(self, grid):
        self.grid = grid
        self.num_rows, self.num_cols = grid.shape
        self.start = None
        self.goal = None

    def find_path(self, start, goal):
        self.start = start
        self.goal = goal

        open_list = []
        closed_list = set()

        heappush(open_list, (0, start))
        parent = {}
        g_score = {start: 0}

        while open_list:
            _, current = heappop(open_list)

            if current == goal:
                return self.reconstruct_path(parent, current)

            closed_list.add(current)

            for neighbor in self.get_neighbors(current):
                if neighbor in closed_list:
                    continue
```

colab.google x Welcome To Colaboratory x Untitled2.ipynb - Col x New Tab x Online C Compiler x Online Python Comp x ChatGPT x + -

colab.research.google.com/drive/1lr1XPib1VtWM_FQD6ojXP8rPt9qMsf3#scrollTo=ISJe1izCE9B

Untitled2.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
for neighbor in self.get_neighbors(current):
    if neighbor in closed_list:
        continue

    tentative_g_score = g_score[current] + 1

    if neighbor not in g_score or tentative_g_score < g_score[neighbor]:
        parent[neighbor] = current
        g_score[neighbor] = tentative_g_score
        f_score = tentative_g_score + self.heuristic(neighbor, goal)
        heappush(open_list, (f_score, neighbor))

return None

def get_neighbors(self, node):
    row, col = node
    neighbors = []
    for dr, dc in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
        new_row, new_col = row + dr, col + dc
        if 0 <= new_row < self.num_rows and 0 <= new_col < self.num_cols and self.grid[new_row][new_col] != 1:
            neighbors.append((new_row, new_col))
    return neighbors

def heuristic(self, a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def reconstruct_path(self, parent, current):
    path = []
    while current in parent:
        path.append(current)
        current = parent[current]
    path.append(self.start)
    path.reverse()
    return path
```

0s completed at 9:39 PM

colab.google x Welcome To Colaboratory x Untitled2.ipynb - Col x New Tab x Online C Compiler x Online Python Comp x ChatGPT x + -

colab.research.google.com/drive/1lr1XPib1VtWM_FQD6ojXP8rPt9qMsf3#scrollTo=ISJe1izCE9B

Untitled2.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
# Example usage:
grid = np.array([
    [0, 0, 0, 0, 0],
    [1, 1, 0, 1, 0],
    [0, 0, 0, 0, 0],
    [0, 1, 1, 1, 1],
    [0, 0, 0, 0, 0]
])

astar = AStar(grid)
start = (0, 0)
goal = (4, 4)
path = astar.find_path(start, goal)

if path:
    print("Path found:", path)
else:
    print("No path found")
```

Path found: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 1), (2, 0), (3, 0), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)]

0s completed at 9:39 PM