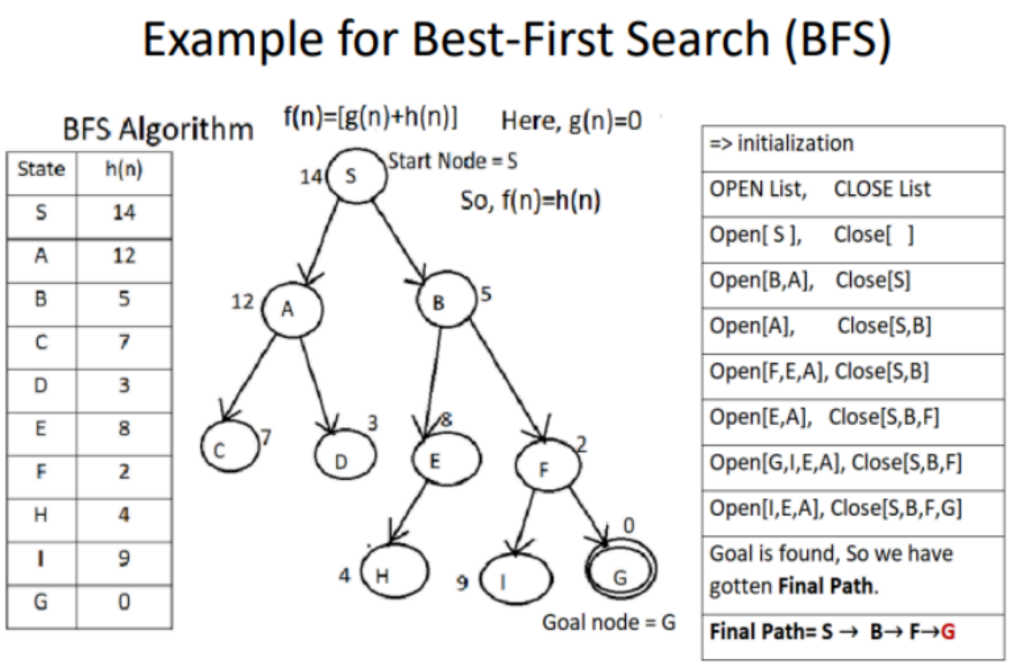
Artificial Intelligence Lab Assignment
Searching Algorithms

Best First Search:

Here is the **Google Colab** link to Best First Search: Best_First_Search_Algorithm.ipynb

We implemented the algorithm using Python and illustrated it with the following example.



Sample Code is given below and Code with detailed explanation can be found in Google Colab.

```
import heapq
def best_first_search(graph, start, goal, heuristic):
    # Priority queue to store (heuristic, node)
    open_list = []
    heapq.heappush(open_list, (heuristic[start], start))

    # Dictionary to keep track of visited nodes and their
parents
    close_list = []
```

```
    while open_list:
        # Get the node with the lowest heuristic value
        current_heuristic, current_node =
heapq.heappop(open_list)
        close_list.append(current_node)

        # Check if we've reached the goal
        if current_node == goal:
            break

        # Explore neighbors
        for neighbor in graph[current_node]:
            if neighbor not in close_list:
                heapq.heappush(open_list, (heuristic[neighbor],
neighbor))

    if goal in close_list:
      return close_list
    else:
      return None
def best_first_search(graph, start, goal, heuristic):
    # Priority queue to store (heuristic, node)
    open_list = []
    heapq.heappush(open_list, (heuristic[start], start))

    # Dictionary to keep track of visited nodes and their
parents
    close_list = []

    while open_list:
        # Get the node with the lowest heuristic value
        current_heuristic, current_node =
heapq.heappop(open_list)
        close_list.append(current_node)
```

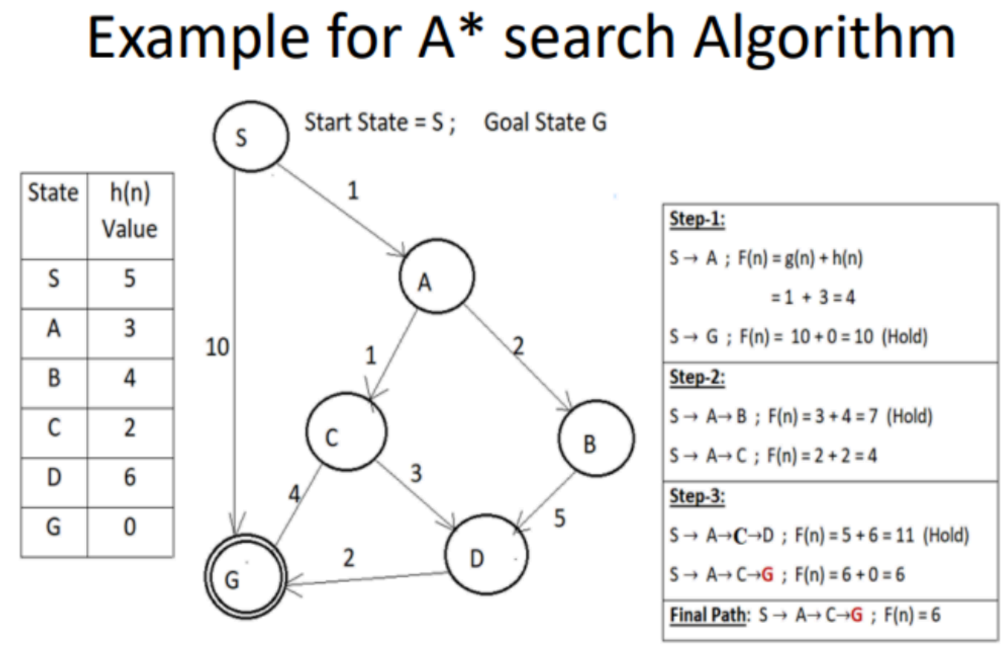
```
        # Check if we've reached the goal
        if current_node == goal:
            break

        # Explore neighbors
        for neighbor in graph[current_node]:
            if neighbor not in close_list:
                heapq.heappush(open_list, (heuristic[neighbor],
neighbor))

    if goal in close_list:
      return close_list
    else:
      return None
start_node = 'S'
goal_node = 'G'

path = best_first_search(graph, start_node, goal_node,
heuristic)
print("Best-First Search Path:", path)
```

Output: Best-First Search Path: ['S', 'B', 'F', 'G']

A* Search Algorithm

Here is the **Google Colab** link to A* Search Algorithm: A_Star_Search_Algorithm.ipynb

We implemented the algorithm using Python and illustrated it with the following example.



Example for A* search Algorithm

Step-1: Initialization; Start State = S; Goal State = G;	=> initialization OPEN List, CLOSE List Open[S], Close[ ] Open[A,G], Close[S] Open[G], Close[S,A] Open[C,B,G], Close[S,A] Open[G,D,B,G], Close[S,A,C] Open[D,B,G], Close[S,A,C,G] Lower cost goal is found. So, we have gotten **Final Path**. **Final Path= S → A→ C→ G**
Step-2: S → A ; F(n) = g(n) + h(n) = 1 + 3 = 4 S → G ; F(n) = 10 + 0 = 10 (Hold)	
Step-3: S → A→ B ; F(n) = 3 + 4 = 7 (Hold) S → A→ C ; F(n) = 2 + 2 = 4	
Step-4: S → A→C →D ; F(n) = 5 + 6 = 11 (Hold) S → A→ C→**G** ; F(n) = 6 + 0 = 6	
Final Path: S → A→ C→ **G** ; F(n) = 6	

Sample Code is given below and Code with detailed explanation can be found in Google Colab.

```
import heapq
# Define a graph as an adjacency list
graph = {
  'S': [('A', 1), ('G', 10)],
  'A': [('B', 2), ('C', 1)],
  'B': [('D', 5)],
  'C': [('D', 3), ('G', 4)],
  'D': [('G', 2)],
  'G': []
}

# Define heuristic values for each node
heuristic = {
  'S': 5,
  'A': 3,
  'B': 4,
  'C': 2,
  'D': 6,
  'G': 0
}
def a_star_search(graph, start, goal, heuristic):
    open_list = []
    heapq.heappush(open_list, (heuristic[start], start))

    close_list = []
    g_cost = {node: float('inf') for node in graph}
    g_cost[start] = 0

    while open_list:
        current_f, current_node = heapq.heappop(open_list)
        close_list.append(current_node)

        if current_node == goal:
```

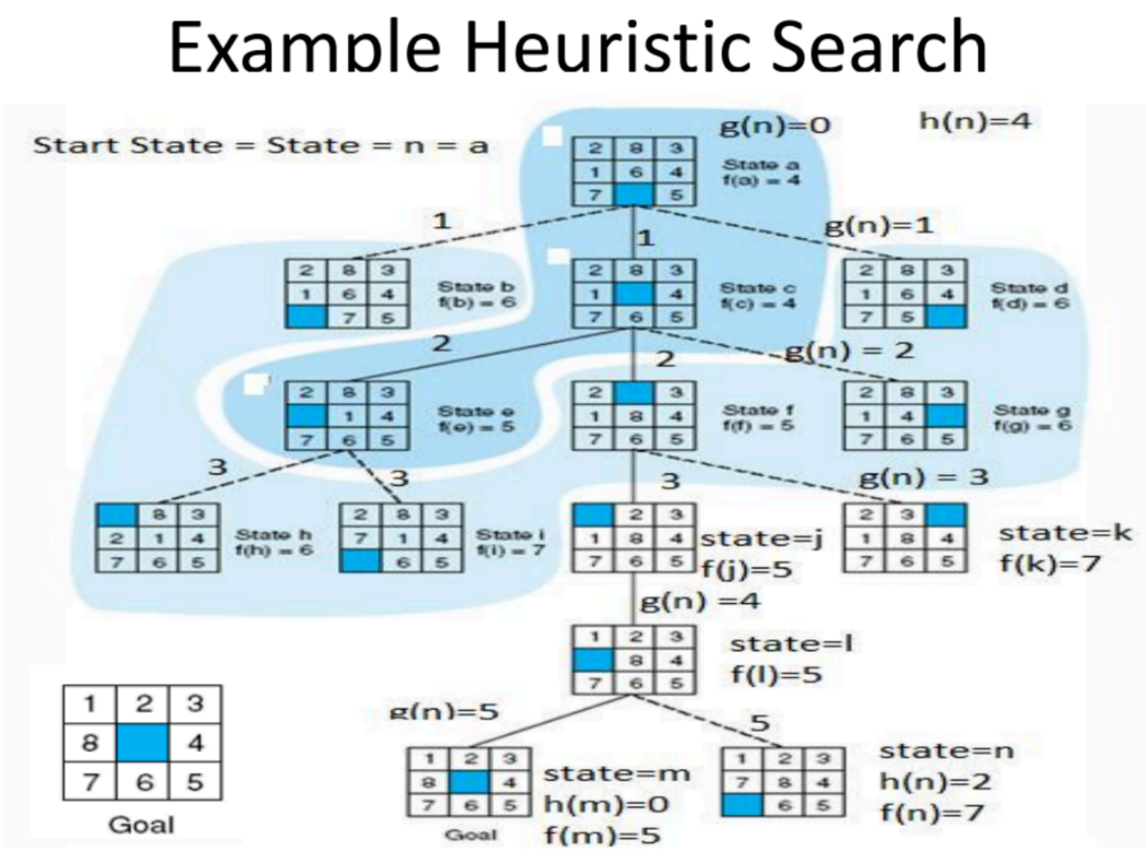
```
            break

        for neighbor, step_cost in graph[current_node]:
            if neighbor not in close_list:
                temp_g = g_cost[current_node] + step_cost
                if temp_g < g_cost[neighbor]:
                    g_cost[neighbor] = temp_g
                    f_cost = temp_g + heuristic[neighbor]
                    heapq.heappush(open_list, (f_cost,
neighbor))

    if goal in close_list:
      return close_list
    else:
      return None
    start_node = 'S'
    goal_node = 'G'

    path = a_star_search(graph, start_node, goal_node,
heuristic)
    print("A* Search Path:", path)
```

Output: A* Search Path: ['S', 'A', 'C', 'G']

Heuristic Search (8 puzzle problem)

Here is the Google Colab link to the Heuristic Search Algorithm:
Heuristic_Seach(8_Puzzle_Problem).ipynb

We implemented the algorithm using Python and illustrated it with the following example.



Sample Code is given below and Code with detailed explanation can be found in Google Colab.

```
from heapq import heappush, heappop
import numpy as np
# Start state of the 8-puzzle
START_STATE = [
        [2, 8, 3],
        [1, 6, 4],
        [7, 0, 5]
```

```python
    ]

# Goal state of the 8-puzzle
GOAL_STATE = [
    [1, 2, 3],
    [8, 0, 4],
    [7, 6, 5]
]

# Directions for moving the blank tile (up, down, left, right)
MOVES = [(0, -1), (0, 1), (-1, 0), (1, 0)]

# Helper function to calculate heuristic value
def heuristic(state):
    distance = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0 and abs(state[i][j] -
GOAL_STATE[i][j]) != 0:
                distance += 1
    return distance
# Find the position of the blank tile (0)
def find_blank(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return (i, j)
    return None
# Generate possible moves from the current state
def generate_moves(state):
    blank_row, blank_col = find_blank(state)
    moves = []
    for dr, dc in MOVES:
        new_row, new_col = blank_row + dr, blank_col + dc
        if 0 <= new_row < 3 and 0 <= new_col < 3:
```

```
            new_state = [row[:] for row in state]
            new_state[blank_row][blank_col],
new_state[new_row][new_col] = new_state[new_row][new_col],
new_state[blank_row][blank_col]
            moves.append(new_state)
    return moves
# Check if the current state is the goal state
def is_goal(state):
    return state == GOAL_STATE
# A* search algorithm
# A* search algorithm
def a_star_search(start_state):
    for row in start_state:
      print(row)
    print('\nExpanding the Start State\n')
    open_list = []
    heappush(open_list, (heuristic(start_state), 0, start_state,
[]))  # (f(n), g(n), state, path)
    visited = set()

    while open_list:
        _, g, current_state, path = heappop(open_list)
        if is_goal(current_state):
            return path + [current_state]  # Return the solution
path

        if tuple(map(tuple, current_state)) in visited:
            continue
        visited.add(tuple(map(tuple, current_state)))

        print('------------------------------------------')

        idx = 0
        heuristic_values = []
        for move in generate_moves(current_state):
```

```python
            if tuple(map(tuple, move)) not in visited:
              idx = idx + 1
              print(idx, ":")
              for row in move:
                print(row)
              print()
              print(f"f(n) = {heuristic(move) + g + 1}")
              heuristic_values.append(heuristic(move) + g + 1)
              print()
              heappush(open_list, (g + 1 + heuristic(move), g + 1, move, path + [current_state]))

        print(f"Expanding Matrix: {np.argmin(heuristic_values) + 1} (lowest heuristic value)")
        print()


    return None

solution = a_star_search(START_STATE)

if solution:
  print("Solution found! Steps:")
  for step in solution:
    for row in step:
      print(row)
    print()

else:
  print("No solution found.")
```

Output of the 8 puzzle problem with visualization:

[2, 8, 3]
[1, 6, 4]
[7, 0, 5]

Expanding the Start State

\--
1 :
[2, 8, 3]
[1, 6, 4]
[0, 7, 5]

f(n) = 6

2 :
[2, 8, 3]
[1, 6, 4]
[7, 5, 0]

f(n) = 6

3 :
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]

f(n) = 4

Expanding Matrix: 3 (lowest heuristic value)

\--
1 :
[2, 8, 3]
[0, 1, 4]
[7, 6, 5]

f(n) = 5

2 :
[2, 8, 3]
[1, 4, 0]
[7, 6, 5]
f(n) = 6

3 :
[2, 0, 3]
[1, 8, 4]
[7, 6, 5]

f(n) = 5

Expanding Matrix: 1 (lowest heuristic value)

----
1 :
[0, 2, 3]
[1, 8, 4]
[7, 6, 5]

f(n) = 5

2 :
[2, 3, 0]
[1, 8, 4]
[7, 6, 5]

f(n) = 7

Expanding Matrix: 1 (lowest heuristic value)

----
1 :
[0, 8, 3]
[2, 1, 4]
[7, 6, 5]

f(n) = 6

2 :
[2, 8, 3]
[7, 1, 4]
[0, 6, 5]

f(n) = 7

Expanding Matrix: 1 (lowest heuristic value)

----
1 :
[1, 2, 3]
[0, 8, 4]
[7, 6, 5]

f(n) = 5

Expanding Matrix: 1 (lowest heuristic value)

----
1 :
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]

f(n) = 5

2 :
[1, 2, 3]
[7, 8, 4]
[0, 6, 5]

f(n) = 7

Expanding Matrix: 1 (lowest heuristic value)

Solution found! Steps:
[2, 8, 3]
[1, 6, 4]
[7, 0, 5]

[2, 8, 3]
[1, 0, 4]
[7, 6, 5]

[2, 0, 3]
[1, 8, 4]
[7, 6, 5]

[0, 2, 3]
[1, 8, 4]
[7, 6, 5]

[1, 2, 3]
[0, 8, 4]
[7, 6, 5]

[1, 2, 3]
[8, 0, 4]
[7, 6, 5]