untitled2

January 31, 2024

[2]:
```python
from collections import deque

# Define the graph as an adjacency list
graph = {
    0: [1, 2],
    1: [0, 3, 4],
    2: [0, 5],
    3: [1],
    4: [1],
    5: [2]
}

# Define the BFS function
def bfs(graph, start_vertex):
    # Initialize the visited set to keep track of visited vertices
    visited = set()

    # Initialize the queue with the starting vertex
    queue = deque([start_vertex])

    # Loop until the queue is empty
    while queue:
        # Dequeue the next vertex from the queue
        current_vertex = queue.popleft()

        # If the current vertex has not been visited yet, print it and mark it␣
 ↪as visited
        if current_vertex not in visited:
            print(current_vertex)
            visited.add(current_vertex)

            # Enqueue the neighbors of the current vertex that have not been␣
 ↪visited yet
            for neighbor in graph[current_vertex]:
                if neighbor not in visited:
                    queue.append(neighbor)
```

```
# Call the BFS function with the graph and a starting vertex
bfs(graph,0)
```

```
0
1
2
3
4
5
```

[3]:
```
def dfs(graph, start):
    visited = set()
    stack = [start]

    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            visited.add(vertex)
            print(vertex)
            stack.extend(neighbor for neighbor in graph[vertex] if neighbor not␣
 ↪in visited)

# Example usage:
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}

dfs(graph,'A')
```

```
A
C
F
E
B
D
```

[4]:
```
import heapq

class PuzzleNode:
    def _init_(self, state, g_value, heuristic):
        self.state = state
        self.g_value = g_value
```

```python
        self.heuristic = heuristic

    def __lt__(self, other):
        return (self.g_value + self.heuristic) < (other.g_value + other.heuristic)

class EightPuzzleSolver:
    def __init__(self, initial_state, goal_state):
        self.initial_state = initial_state
        self.goal_state = goal_state
        self.moves = [(1, 0), (-1, 0), (0, 1), (0, -1)]  # Possible moves: right, left, down, up

    def calculate_heuristic(self, state):
        # Your heuristic function h(x) implementation for the 8-puzzle problem
        # You can use various heuristics such as Manhattan distance, misplaced tiles, etc.
        # For simplicity, let's assume the heuristic is the count of misplaced tiles.
        misplaced_tiles = sum([1 for i, j in zip(state, self.goal_state) if i != j])
        return misplaced_tiles

    def is_valid_move(self, x, y):
        return 0 <= x < 3 and 0 <= y < 3

    def generate_next_states(self, current_state):
        zero_index = current_state.index(0)
        zero_x, zero_y = zero_index % 3, zero_index // 3
        next_states = []

        for dx, dy in self.moves:
            new_x, new_y = zero_x + dx, zero_y + dy

            if self.is_valid_move(new_x, new_y):
                new_state = current_state[:]
                new_index = new_y * 3 + new_x
                new_state[zero_index], new_state[new_index] = new_state[new_index], new_state[zero_index]

                next_states.append(new_state)

        return next_states

    def solve_puzzle(self):
```

```
        initial_node = PuzzleNode(self.initial_state, 0, self.
↪calculate_heuristic(self.initial_state))
        priority_queue = [initial_node]
        visited_states = set()

        while priority_queue:
            current_node = heapq.heappop(priority_queue)

            if current_node.state == self.goal_state:
                return current_node.g_value  # Return the cost to reach the goal

            visited_states.add(tuple(current_node.state))

            for next_state in self.generate_next_states(current_node.state):
                if tuple(next_state) not in visited_states:
                    next_g_value = current_node.g_value + 1
                    next_heuristic = self.calculate_heuristic(next_state)
                    next_node = PuzzleNode(next_state, next_g_value,␣
↪next_heuristic)
```