

# PRINCIPLES OF ARTIFICIAL INTELLIGENCE

## LABORATORY PROGRAMS

### BEST FIRST SEARCH ALGORITHM PYTHON PROGRAM

#### SOURCE CODE:

```
class Node:
    def __init__(self, state, parent=None, cost=0, heuristic=0):
        self.state = state
        self.parent = parent
        self.cost = cost
        self.heuristic = heuristic
        self.f = cost + heuristic

def is_goal(state, goal):
    return state == goal

def generate_successors(node, goal):
    successors = []
    if node.state < goal:
        successors.append(Node(node.state + 1, node, node.cost + 1, heuristic(node.state +
1, goal)))
    return successors

def heuristic(state, goal):
    return abs(goal - state)

def rbfs(node, f_limit, goal):
    if is_goal(node.state, goal):
        return node

    successors = generate_successors(node, goal)
    if not successors:
        return None

    while True:
        successors.sort(key=lambda x: x.f)
```

```

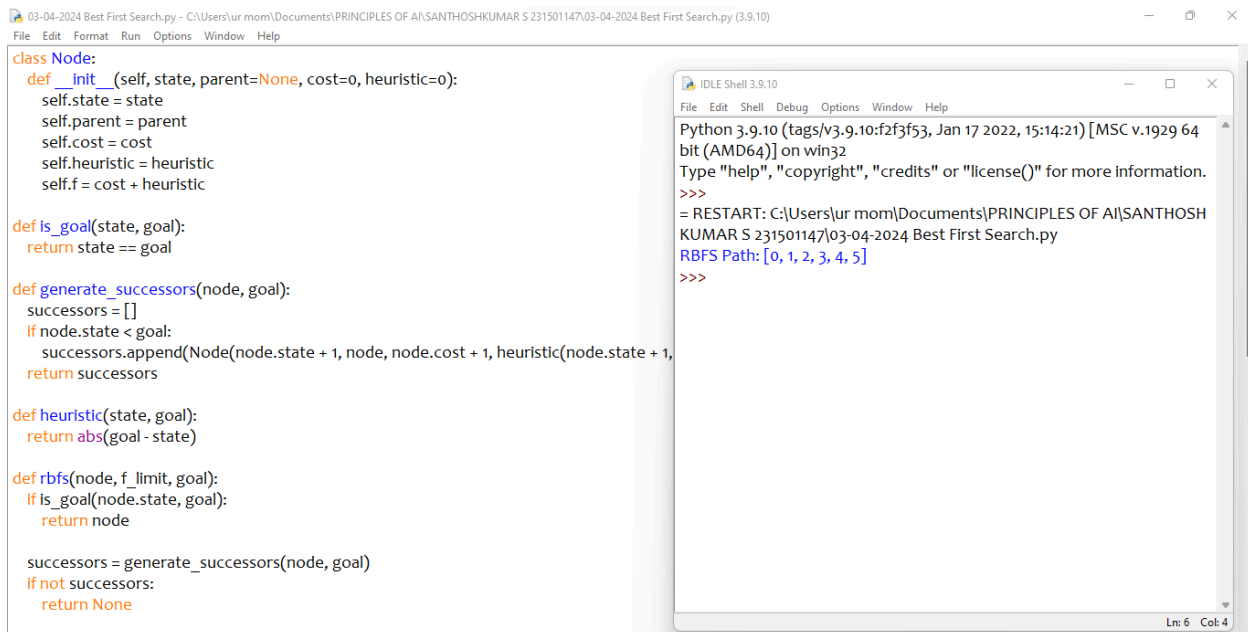
    best = successors[0]
    if best.f > f_limit:
        return None
    if len(successors) > 1:
        alternative = successors[1].f
    else:
        alternative = float('inf')
    result = rbfs(best, min(f_limit, alternative), goal)
    if result is not None:
        return result

initial_state = 0
goal_state = 5
initial_node = Node(initial_state, None, 0, heuristic(initial_state, goal_state))
solution = rbfs(initial_node, float('inf'), goal_state)

if solution is not None:
    path = []
    while solution is not None:
        path.append(solution.state)
        solution = solution.parent
    path.reverse()
    print("RBFS Path:", path)
else:
    print("No solution found.")

```

## OUTPUT:



The screenshot shows a Python IDE with two windows. The left window displays the code for a Node class and search functions. The right window shows the output of the program, which includes the Python version, the file path, and the RBFS Path: [0, 1, 2, 3, 4, 5].

```

class Node:
    def __init__(self, state, parent=None, cost=0, heuristic=0):
        self.state = state
        self.parent = parent
        self.cost = cost
        self.heuristic = heuristic
        self.f = cost + heuristic

def is_goal(state, goal):
    return state == goal

def generate_successors(node, goal):
    successors = []
    if node.state < goal:
        successors.append(Node(node.state + 1, node, node.cost + 1, heuristic(node.state + 1, goal)))
    return successors

def heuristic(state, goal):
    return abs(goal - state)

def rbfs(node, f_limit, goal):
    if is_goal(node.state, goal):
        return node
    successors = generate_successors(node, goal)
    if not successors:
        return None

```

```

Python 3.9.10 (tags/v3.9.10:f2f3f53, Jan 17 2022, 15:14:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\ur mom\Documents\PRINCIPLES OF AI\SANTHOSH KUMAR S 231501147\03-04-2024 Best First Search.py
RBFS Path: [0, 1, 2, 3, 4, 5]
>>>

```