# **Recursive BFS**

**Program:**

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:**

**RBFS Path: [0, 1, 2, 3, 4, 5]**