



ARTIFICIAL
INTELLIGENCE

BFS/DFS.ipynb - Colab

SHANZA

FA23-BSE-101

SUBMISSION TO: MA 'AM ZEENAT ZULFIQAR



```

1)
from collections
import deque
tree_graph = {
"A": ["B", "C",
"D"], "B": ["E"],
"C": ["F", "G"],
"D": ["H"],
"E": ["I", "J"],
"F": [],
"G": ["K", "L"],
"H": ["M"],
"I": [],
"J": [],
"K": [],
"L": [],
"M": []
}
def bfs(graph,start):
    queue=deque([start])
    visited=set([start])
    while queue:
        node=queue.popleft()
        print(node,
end=" ")

for neighbour in graph[node]:
    if neighbour not in visited:
        queue.append(neighbour)
        visited.add(neighbour)
print("BFS traversal")
bfs(tree_graph,"A")

def
dfs(graph,start,visited):
    if start not in visited:
        print(start,end=" ")
        visited.add(start)
        for neighbour in graph[start]:
            dfs(graph,neighbour,visited)

visited=set()
print("\nDFS traversal")
dfs(tree_graph,"A",visited)

⌘ BFS traversal
A B C D E F G H I J K L M
DFS traversal
A B E I J C F G K L D H M

```

2)

```

from collections import
deque def
bfs(graph,start,goal):
    queue=deque([(start,
[start])]) visited=set([start])
    while queue:
        node, path =
        queue.popleft() if node
        == goal:
            return path
    for neighbour in graph[node]:
        if neighbour not in visited:
            queue.append((neighbour, path + [neighbour]))
            visited.add(neighbour)

city_graph = {
    "Islamabad": ["Rawalpindi", "Lahore",
    "Peshawar"], "Rawalpindi": ["Islamabad",
    "Peshawar", "Quetta"], "Peshawar":
    ["Islamabad", "Rawalpindi", "Quetta"],
    "Lahore": ["Islamabad", "Multan",
    "Quetta"],
    "Multan": ["Lahore", "Karachi", "Quetta"],
    "Quetta": ["Rawalpindi", "Peshawar", "Multan",
    "Karachi"], "Karachi": ["Multan", "Quetta"]
}
start_city = "Islamabad"
goal_city = "Karachi"
path = bfs(city_graph, start_city,
goal_city) if path:
    print("Path from", start_city, "to", goal_city, ":", " ->
".join(path)) else:
    print("No path found from", start_city, "to", goal_city)

```

➡ Path from Islamabad to Karachi : Islamabad -> Rawalpindi -> Quetta -> Karachi

3)

```

import random
import time

```

```
import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Generate Random Unique Numbers
def generate_unique_numbers(size, range_start, range_end):
    return random.sample(range(range_start, range_end), size)

# Generate sets
sets = {
    1000: generate_unique_numbers(1000, 1, 10000),
    40000: generate_unique_numbers(40000, 1, 1000000),
    80000: generate_unique_numbers(80000, 1, 1000000),
    200000: generate_unique_numbers(200000, 1, 1000000),
    1000000: generate_unique_numbers(1000000, 1, 10000000)
}
```

Step 2: Build a Binary Search

Tree class TreeNode:

```
def __init__(self, key):
    self.left = None
    self.right = None
    self.val = key

def insert(root,
key): if root is
None:
    return
TreeNode(key) else:
    if root.val < key:
        root.right = insert(root.right, key)
    else:
        root.left = insert(root.left,
key) return root

def
    build_tree(numbers)
: root = None
for number in numbers:
    root = insert(root,
number) return root
```

Step 3: Implement BFS

and DFS

```
def bfs(root,
goal):
    if root is
        None:
            return
        False
    queue = [root]
    while queue:
        node =
        queue.pop(0) if
        node.val == goal:
            return
        True if
        node.left:
```

```
        queue.append(node.left)
    if node.right:
        queue.append(node.right)
    return False

def dfs(root, goal):
    if root is None:
        return False
    if root.val == goal:
        return True
    return dfs(root.left, goal) or dfs(root.right, goal)
```

Step 4: Measure Execution

Time results = []

```
for size, numbers in sets.items():
    tree = build_tree(numbers)
    goal = numbers[len(numbers) - 220]
```

Measure BFS

```
start_time = time.time()
bfs(tree, goal)
bfs_time = time.time() - start_time
```

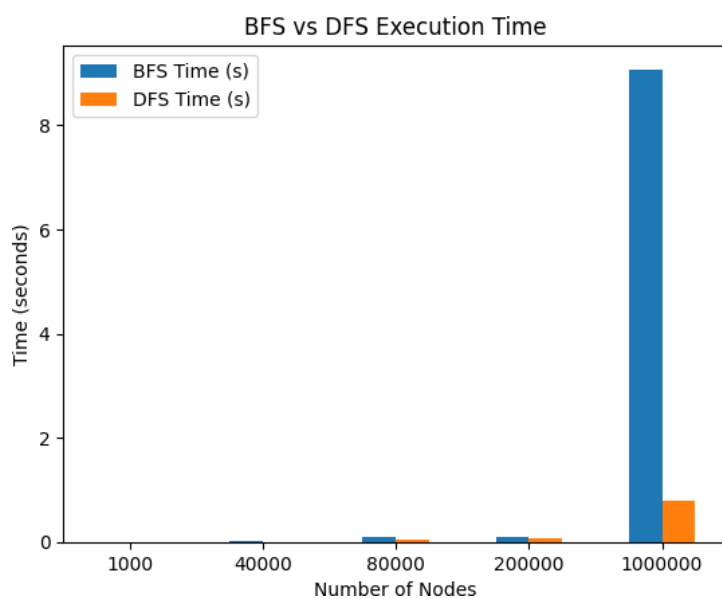
Measure DFS

```
start_time = time.time()
dfs(tree, goal)
dfs_time = time.time() - start_time
```

```
results.append(
    { 'Size': size,
      'BFS Time (s)': bfs_time,
      'DFS Time (s)': dfs_time
    })
```

```
# Step 5: Create a  
DataFrame df =  
pd.DataFrame(results)
```

```
# Step 6: Plot the Results  
df.set_index('Size')[['BFS Time (s)', 'DFS Time  
(s)']].plot(kind='bar') plt.title('BFS vs DFS Execution Time')  
plt.ylabel('Time  
(seconds)')  
plt.xlabel('Number of  
Nodes')  
plt.xticks(rotation=0)  
plt.show()
```



[Code](#) [Text](#)

[GitHub Link](#)

<https://github.com/Shanza-Rafique/DFS-BFS.git>

[YouTube Link](#)

https://youtu.be/V2M_FzMAIWI?si=ReHqdE-2aIWmuIm