

Tugas Pendahuluan Praktikum Dasar Kecerdasan Artifisial

Modul 7 : Implementasi Depth-First Search (DFS)

INSTALLASI LIBRARY MK DKA

Jika belum terinstal, jalankan perintah berikut pada terminal atau notebook:

```
pip install jupyter ipykernel numpy matplotlib networkx sympy scikit-fuzzy scipy
```

Petunjuk Umum

1. Jangan ubah struktur kode yang diberikan, cukup isi bagian yang kosong.
2. Jalankan setiap sel satu per satu agar tidak ada error.
3. Gunakan modul praktikum atau dokumentasi NetworkX (<https://networkx.org/documentation/stable/>) jika membutuhkan referensi tambahan.
4. Kerjakan soal yang ada secara berurutan, karena soal saling berkesinambungan.

```
In [1]: print("Jason Emmanuel") # Tuliskan nama Lengkap Anda
print("10312300115") # Tuliskan NIM Anda
print("IF-47-06") # Tuliskan kelas Anda
print("ENS") # Tuliskan kode asprak anda
```

```
Jason Emmanuel
10312300115
IF-47-06
ENS
```

Pengantar Depth-First Search (BFS)

Depth-first search (DFS) adalah algoritma yang mencari grafik atau struktur data pohon dengan menjelajahi sejauh mungkin ke bawah cabang sebelum melakukan backtracking. Ini adalah algoritma rekursif yang menggunakan tumpukan untuk melacak node.

```
In [2]: import networkx as nx # Library untuk membuat graf (nx)
import matplotlib.pyplot as plt # Library untuk plot grafik
```

Contoh fungsi pendukung untuk mencetak graf

!! Tidak perlu dimodifikasi !!

```
In [3]: # Support function to print the graph
def show_graph(G, pos=None, title='') :
    # If position is not specified from the parameter it will generate a new position
    if pos == None :
        # The pos variable that stores the coordinates of the node placement
        pos = nx.spring_layout(G)

    # Plot the graph G given in the parameter
    nx.draw(G, pos, with_labels=True, node_color='red', node_size=2500,
            font_color="white", font_weight="bold", width=5)

    # Retrieve each weight of each edge in graph G
    edge_labels = nx.get_edge_attributes(G, "weight")

    # Draw the weight of the edge in the visualization
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='blue',
                                font_weight="bold", font_size=12)

    # Add margins on each side
    plt.margins(0.2)
    # Add a title as given by the parameter
    plt.title(title)
    # Display the graph visualization on the terminal
    plt.show()
```

Soal 1

Rina sedang bermain di taman bermain yang memiliki banyak jalur seperti labirin. Ia ingin keluar dari taman tersebut.

Taman itu bisa digambarkan dalam bentuk graf, dan kamu akan membantu Rina mencari jalan keluar menggunakan metode pencarian **Depth-First Search (DFS)**.

!! Tidak usah dimodifikasi !!

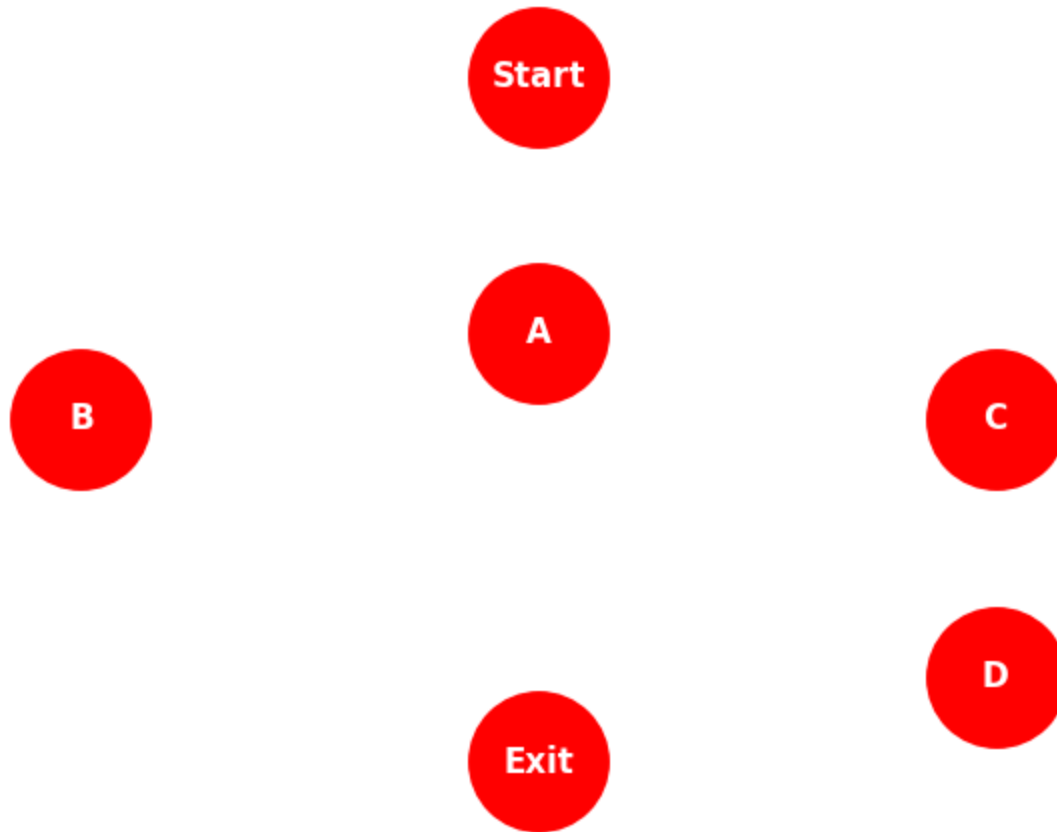
```
In [4]: # Posisi untuk undirected graph agar sesuai dengan contoh soal
pos = {
    "Start": (0, 5),
    "A": (0, 2),
    "B": (-1, 1),
    "C": (1, 1),
    "D": (1, -2),
    "Exit": (0, -3)
}
```

```
In [7]: # Inisialisasi graf berarah
G = nx.DiGraph()


# Tambahkan edge di sini
nodes = ["Start", "A", "B", "C", "D", "Exit"]
G.add_nodes_from(nodes)

# Tampilkan graf
show_graph(G, pos, title = "Kerangka graf")
```

Kerangka graf



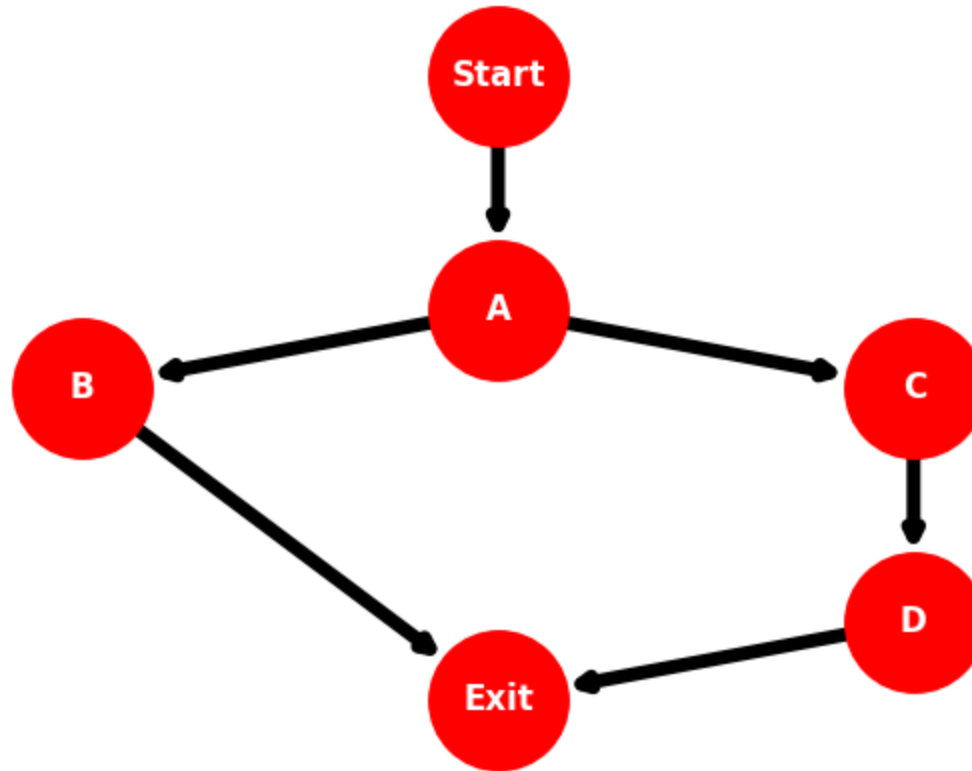
Contoh *output*:

 tp7_kerangka_graf


```
In [8]: # hubungkan node node yang ada sesuai dengan contoh outpt yang telah diberikan
edges = [("Start", "A"), ("A", "B"), ("A", "C"), ("B", "Exit"), ("C", "D"), ("D", "Exit"), ]
G.add_edges_from(edges)
```

```
# menampilkan graf  
show_graph(G, pos, title = "Graf berarah")
```

Graf berarah



Contoh output:

 tp7_graf_berarah

Soal 2

Sekarang Rina sudah memiliki peta labirin yang ada, tetapi Rina sebagai programmer penasaran memecahkan labirin menggunakan algoritma metode komputasi.

Bantu Rina menyelesaikan permasalahan ini menggunakan algoritma **Depth First Search (DFS)**

```
In [9]: def dfs(graph, start, goal, path=None, visited=None): ### jangan dirubah
        if visited is None: ### jangan rubah
            visited = set() # ubah dengan menginisialisasi library set
        if path is None: ### jangan rubah
            path = [] # inisialisasi dengan list kosong

        visited.add(start) # lakukan penambahan data pada `visited` dari node awal
        path = path + [start] ### jangan diubah

        if start == goal: # jika node awal sama dengan node yang dicari
            return path ### jangan diubah

        for neighbor in graph.neighbors(start):
            if neighbor not in visited: # jika tetangga dari node sekarang belum dikunjungi
                new_path = dfs(graph, neighbor, goal, path, visited)
                if new_path:
                    return new_path
        return None ### jangan dirubah
```

Soal 3

Bantu Rina membuat program sederhana untuk mengecek jalur yang dilalui dari **lokasi awal** ke **lokasi tujuan** pada labirin menggunakan algoritma yang telah dibuat.

```
In [11]: path = dfs(G, "Start", "Exit") # isikan dengan menyesuaikan parameter wajib yang dibutuhkan berdasarkan kebutuhan soal
        print(f"Jalur yang ditemukan DFS: {path}") # isikan agar sesuai dengan test case yang telah disediakan
```

Jalur yang ditemukan DFS: ['Start', 'A', 'B', 'Exit']

Test case Program:

input

output

Jalur yang ditemukan DFS: ['Start', 'A', 'B', 'Exit']

