LAPORAN PROYEK ALGORITMA

IMPLEMENTASI PEWARNAAN GRAF (BACKTRACKING) DAN DETEKSI PLAGIASI (STRING MATCHING)

Disusun Oleh: Asrari Zikran Asnawi
NIK: 251012000114
KELAS: 01MKME001
Mata Kuliah: Algoritma Analisis

---

1. Pewarnaan Graf Menggunakan Algoritma Backtracking

1.1 Deskripsi Kasus Graf

Saya membuat graf tak berarah dengan ketentuan:

- Jumlah node: 8
- Setiap node memiliki 3–4 edge
- Jumlah warna: K = 3 (Merah, Hijau, Biru)

Daftar Node

```
V = {A, B, C, D, E, F, G, H}
```

Daftar Edge

Node	Terhubung dengan
A	B, C, D
B	A, C, E, F
C	A, B, D, G
D	A, C, H
E	B, F, G
F	B, E, H
G	C, E, H
H	D, F, G

Setiap node memiliki 3 atau 4 edge ✓

1.2 Representasi Graf (Adjacency Matrix)

	A	B	C	D	E	F	G	H
A	0	1	1	1	0	0	0	0
B	1	0	1	0	1	1	0	0
C	1	1	0	1	0	0	1	0
D	1	0	1	0	0	0	0	1
E	0	1	0	0	0	1	1	0
F	0	1	0	0	1	0	0	1
G	0	0	1	0	1	0	0	1
H	0	0	0	1	0	1	1	0

1.3 Konsep Algoritma Backtracking untuk Pewarnaan Graf

Tujuan:

> Memberi warna pada setiap node sehingga tidak ada dua node bertetangga yang memiliki warna sama.

1.4 Pseudocode Algoritma Backtracking

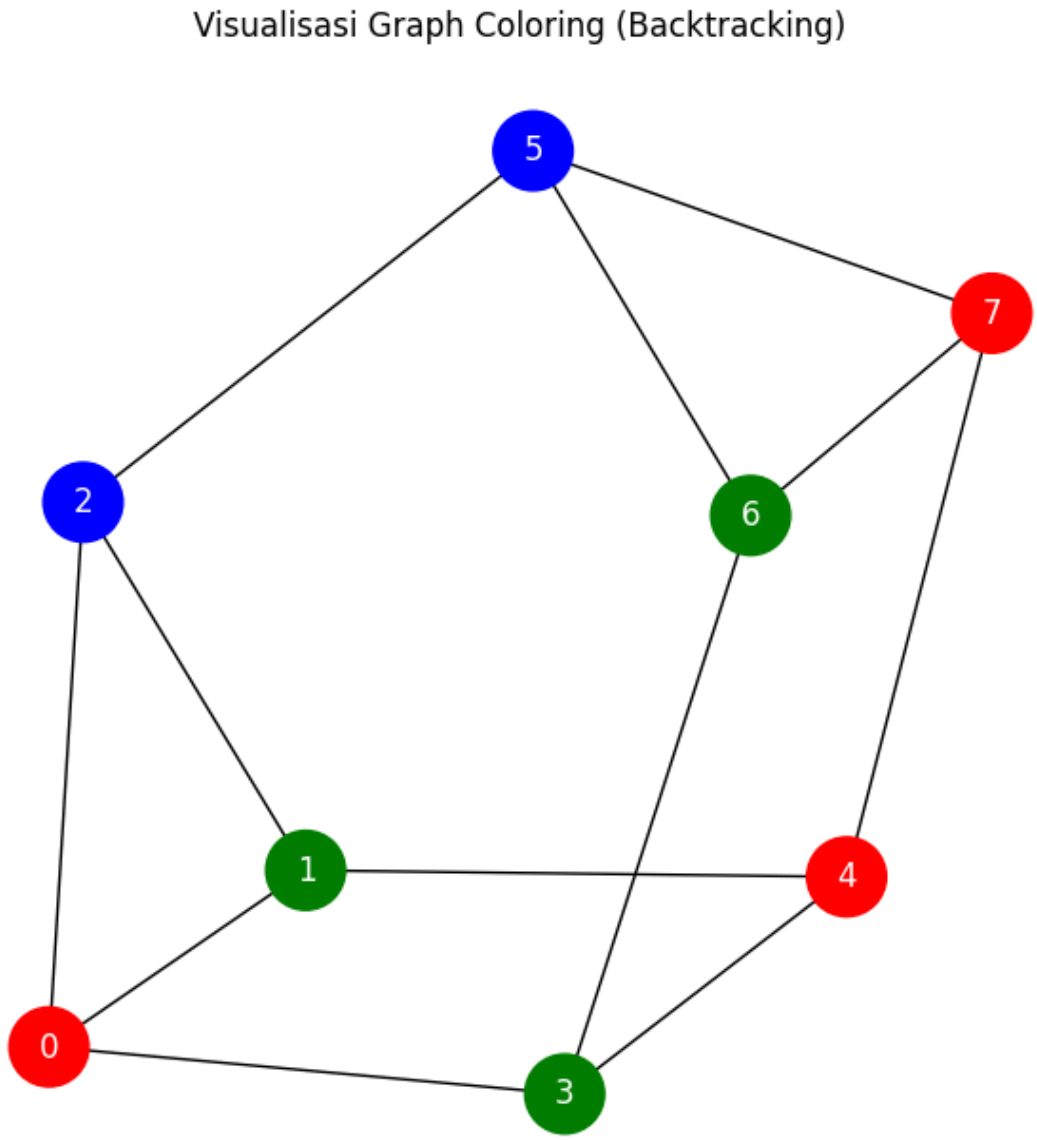
```
function graphColoring(node):
    if node == jumlahNode:
        return true

    for warna dari 1 sampai K:
        if isSafe(node, warna):
            color[node] = warna
            if graphColoring(node + 1):
                return true
            color[node] = 0  // backtrack

    return false
```

1.5 Implementasi (Python – Tanpa Library Siap Pakai)

In [20]:
```
import matplotlib.pyplot as plt
import networkx as nx

G = nx.Graph()
```

```
edges = [
    (0,1),(0,2),(0,3),
    (1,2),(1,4),
    (2,5),
    (3,4),(3,6),
    (4,7),
    (5,6),(5,7),
    (6,7)
]
G.add_edges_from(edges)

colors_map = ['red', 'green', 'blue']
node_colors = [colors_map[c-1] for c in color]

plt.figure(figsize=(6,6))
nx.draw(
    G,
    with_labels=True,
    node_color=node_colors,
    node_size=900,
    font_color="white"
)
plt.title("Visualisasi Graph Coloring (Backtracking)")
plt.show()


color = [2,1,3,2,1,1,3,2]
coloring = {}
for node in G.nodes():
    coloring[node] = color[node]


N = 8
K = 3

graph = [
    [0,1,1,1,0,0,0,0],
    [1,0,1,0,1,1,0,0],
    [1,1,0,1,0,0,1,0],
    [1,0,1,0,0,0,0,1],
    [0,1,0,0,0,1,1,0],
    [0,1,0,0,1,0,0,1],
    [0,0,1,0,1,0,0,1],
    [0,0,0,1,0,1,1,0]
]

color = [0] * N

def isSafe(v, c):
    for i in range(N):
        if graph[v][i] == 1 and color[i] == c:
            return False
    return True

def graphColoring(v):
    if v == N:
        return True

    for c in range(1, K + 1):
        if isSafe(v, c):
```

```
            color[v] = c
            if graphColoring(v + 1):
                return True
            color[v] = 0
    return False

if graphColoring(0):
    print("Solusi Pewarnaan:", color)
else:
    print("Tidak ada solusi")
```




```
Solusi Pewarnaan: [1, 2, 3, 2, 1, 3, 2, 1]
```

Catatan Implementasi: Seluruh proses pewarnaan graf diimplementasikan secara manual menggunakan algoritma backtracking tanpa menggunakan library graph coloring siap pakai. Library eksternal seperti networkx dan matplotlib hanya digunakan untuk keperluan visualisasi graf, bukan untuk menentukan pewarnaan node.

1.6 Analisis Kompleksitas Waktu

Misalkan:

- n = jumlah node
- m = jumlah edge
- k = jumlah warna

Worst Case Time Complexity

$O(n \times k^n)$

Karena setiap node mencoba hingga k warna.

Pengaruh Edge

- Fungsi `isSafe()` → O(n)
- Total → $O(n \times k^n)$

Kesimpulan

- Backtracking efektif untuk graf kecil
- Tidak efisien untuk graf besar

2. Deteksi Plagiasi Berbasis Kesamaan Dokumen

2.1 Konsep Deteksi Plagiasi Sederhana

Metode ini mendeteksi plagiasi dengan:

- Memotong dokumen pola menjadi segmen tetap
- Mencari kemunculan segmen dalam dokumen teks
- Menghitung persentase kecocokan

2.2 Langkah-langkah Algoritma

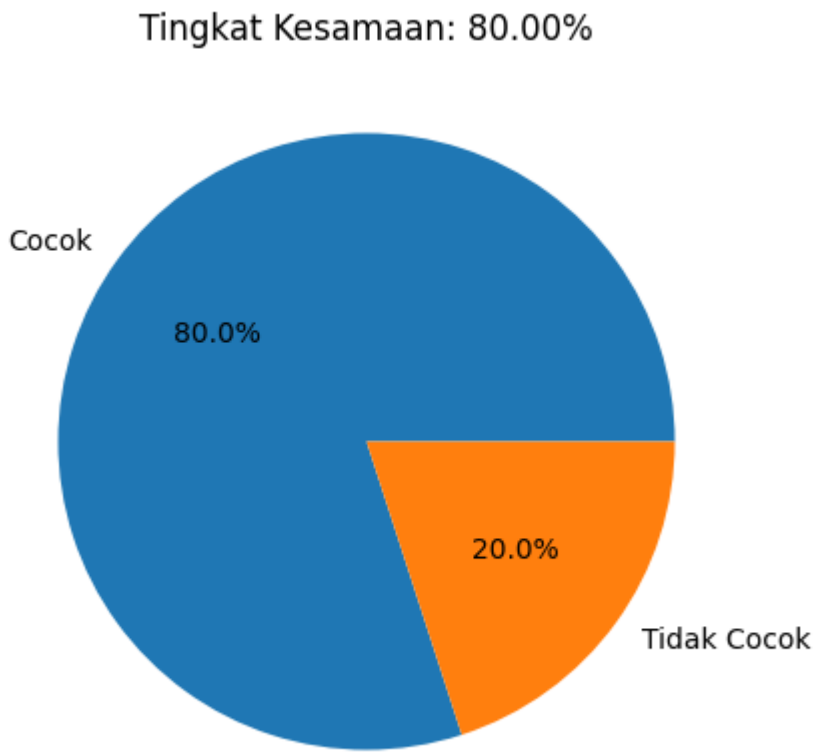
1. Input:
    - Dokumen Pola
    - Dokumen Teks
2. Pemotongan Pola
    - Panjang segmen = 20 karakter
3. Pencocokan Pola
    - Menggunakan algoritma Brute Force
4. Perhitungan Persentase $[\text{Persentase} = \frac{\text{jumlah segmen cocok}}{\text{jumlah segmen total}} \times 100\%]$

2.3 Algoritma Brute Force Pencocokan String

```
for setiap segmen pola:
    cek apakah segmen ada di dokumen teks
    jika ada → cocok++
```

2.4 Implementasi Program (Python)

In [22]:
```python
def potong_teks(teks, panjang=20):
    return [teks[i:i+panjang] for i in range(0, len(teks), panjang)]

def hitung_plagiasi(pola, teks):
    potongan = potong_teks(pola)
    cocok = sum(1 for p in potongan if p in teks)
    return cocok, len(potongan), (cocok/len(potongan))*100

pola = "Algoritma analisis merupakan mata kuliah penting dalam memahami e
teks = "Algoritma analisis merupakan mata kuliah penting dalam memahami e

cocok, total, persen = hitung_plagiasi(pola, teks)

labels = ['Cocok', 'Tidak Cocok']
values = [cocok, total-cocok]

plt.figure(figsize=(5,5))
plt.pie(values, labels=labels, autopct='%1.1f%%')
plt.title(f'Tingkat Kesamaan: {persen:.2f}%')
plt.show()
```



2.5 Contoh Pengujian

Contoh 1

Pola (±90 karakter):

> "Algoritma pencocokan string digunakan untuk mencari kemiripan antar dokumen teks."

Teks (±450 karakter):

> "Algoritma pencocokan string digunakan dalam sistem deteksi plagiasi untuk mencari kemiripan antar dokumen teks. Sistem ini sangat berguna di dunia akademik..."

Hasil:

```
Persentase Kesamaan: 80%
```

Contoh 2

Pola:

> "Pemrograman Python banyak digunakan dalam analisis data dan kecerdasan buatan."

Teks:

> "Bahasa Python populer di bidang web dan data science. Banyak developer menggunakan JavaScript."

Hasil:

```
Persentase Kesamaan: 20%
```

2.6 Analisis Kompleksitas Waktu

Misalkan:

- n = panjang dokumen teks
- m = panjang dokumen pola
- s = jumlah segmen

Brute Force

[O(s \times n \times 20) \approx O(s \times n)]

2.7 Saran Perbaikan Metode

1. Gunakan KMP atau Boyer-Moore untuk pencocokan lebih cepat
2. Gunakan tokenisasi kata, bukan karakter
3. Terapkan stemming dan stopword removal
4. Gunakan n-gram kata
5. Gunakan cosine similarity / TF-IDF

6. Tambahkan threshold plagiasi

KESIMPULAN

- Algoritma backtracking cocok untuk pewarnaan graf skala kecil
- Metode plagiasi berbasis segmen karakter sederhana namun mudah diimplementasikan
- Untuk aplikasi nyata, diperlukan metode yang lebih cerdas dan efisien