Optimalisasi Anagram *Solver* Menggunakan Pendekatan Graf

Heleni Gratia Meitrina Tampubolon - 13523107[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*helenigratia@gmail.com, 13523107@std.stei.itb.ac.id*

***Abstrak*— Anagram merupakan fenomena linguistik yang sering dijumpai dalam kehidupan sehari-hari dan memiliki berbagai aplikasi dalam linguistik, permainan kata, hingga teknologi. Penelitian ini mengkaji penggunaan pendekatan berbasis graf untuk mengidentifikasi dan mengelompokkan anagram secara efisien. Dengan memanfaatkan graf berbobot tak berarah, metode ini merepresentasikan hubungan antar kata berdasarkan aturan yang telah ditetapkan, memungkinkan identifikasi dan klasifikasi anagram secara sistematis. Struktur graf ini meningkatkan efisiensi proses pencarian, sehingga sangat berguna untuk menangani dataset besar, seperti frekuensi kata dalam bahasa Inggris.**

Kata Kunci*—Graf, Anagram, Pencarian, *Signature

I. PENDAHULUAN

Anagram adalah fenomena linguistik yang melibatkan pengaturan ulang huruf-huruf dalam sebuah kata atau frasa untuk membentuk kata atau frasa baru dengan susunan yang berbeda. Fenomena ini dapat menghasilkan kata atau frasa yang bermakna, seperti kata "part" yang dapat diubah menjadi "trap," maupun kata tanpa makna tertentu, tergantung pada kombinasi huruf yang terbentuk. Anagram tidak hanya menjadi bagian penting dalam permainan kata dan teka-teki, tetapi juga memiliki aplikasi signifikan di berbagai bidang, seperti kriptografi, pemrosesan bahasa alami (Natural Language Processing/NLP), hingga studi psikologi.

Bahasa Inggris merupakan salah satu bahasa yang paling produktif dalam menghasilkan anagram valid. Hal ini disebabkan oleh kekayaan kosakata bahasa Inggris, fleksibilitas struktur morfologinya, serta sifat fonetiknya yang memungkinkan berbagai kombinasi huruf untuk membentuk kata baru. Selain itu, banyaknya kata serapan dari bahasa asing dalam bahasa Inggris semakin memperluas kemungkinan variasi anagram yang dapat dihasilkan.

Sebaliknya, meskipun bahasa Indonesia juga memiliki potensi untuk membentuk anagram, variasi yang dihasilkan cenderung lebih terbatas. Hal ini terkait dengan struktur morfologi bahasa Indonesia yang lebih sederhana dan aturan fonetik yang lebih konsisten. Kata-kata dalam bahasa Indonesia sering kali mengikuti pola yang lebih teratur, sehingga peluang untuk menghasilkan kombinasi baru relatif lebih kecil dibandingkan dengan bahasa Inggris.

Makalah ini berfokus pada optimalisasi metode untuk menemukan dan mengelompokkan anagram dari kumpulan kosakata bahasa Inggris yang ada. Dalam upaya ini, akan dibahas penerapan pendekatan berbasis graf untuk meningkatkan efisiensi proses pencarian dan pengelompokan anagram. Graf adalah struktur data yang terdiri dari simpul (*node*) dan sisi (*edge*), yang memungkinkan representasi hubungan antar elemen. Dalam konteks anagram, setiap simpul merepresentasikan kata atau pola huruf tertentu (*signature*), sementara sisi menghubungkan simpul-simpul yang memiliki hubungan sebagai anagram.

Pendekatan graf memungkinkan proses pencarian anagram dilakukan dengan lebih sistematis, terstruktur, dan efisien. Dengan mengelompokkan kata-kata berdasarkan pola huruf yang sama, graf dapat mengurangi redundansi perhitungan dan mempercepat identifikasi anagram, terutama ketika bekerja dengan kumpulan kosakata dalam jumlah besar. Lebih dari itu, pendekatan ini juga memberikan fleksibilitas untuk memvisualisasikan hubungan antar kata dalam bentuk jaringan, yang tidak hanya membantu dalam analisis, tetapi juga memungkinkan eksplorasi lebih lanjut terhadap pola-pola linguistik yang tersembunyi.

Melalui penerapan metode ini, diharapkan makalah ini dapat memberikan kontribusi pada pengembangan teknik optimalisasi pencarian anagram, tidak hanya untuk aplikasi dalam bahasa Inggris tetapi juga dengan potensi adaptasi untuk bahasa lain, termasuk bahasa Indonesia. Dengan demikian, pendekatan ini tidak hanya relevan dalam konteks linguistik dan NLP, tetapi juga dapat diterapkan di berbagai bidang lain, seperti pengenalan pola dan analisis jaringan.

II. LANDASAN TEORI

A. Graf (*Graph*)

a. Defenisi Graf

Dalam ilmu matematika diskrit dan ilmu komputer, graf didefenisikan sebagai sebuah pasangan terurut yang digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antar objek di dalamnya. Elemen pertama dalam graf adalah himpunan simpul (*vertices*), sementara elemen kedua adalah himpunan sisi (*edges*) yang menghubungkan simpul-simpul terebut. Secara sistematis, graf dapat dinyatakan sebagai berikut:

$$G = (V, E) \tag{1}$$

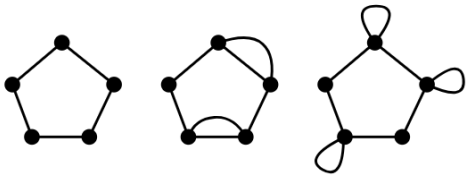
keterangan:
G = graf,
V = himpunan tiak kosong dari simpul-simpul (vertices), contoh: {v_1, v_2, …, v_3},
E = himpunan (boleh kosong) sisi-sisi (*edge*s) yang menghubungkan sepasang simpul, contoh: {e_1, e_2, …, e_3},

b. Jenis-Jenis Graf

Berdasarkan ada atau tidaknya sisi ganda (dua atau lebih sisi yang menghubungkan dua simpul yang sama) atau sisi gelang (sisi yang berawal dan berakhir pada simpul yang sama), graf dapat dibedakan menjadi dua kategori:

1. Graf Sederhana (*Simple Graph*)
   Graf sederhana adalah jenis graf yang tidak mengandung sisi ganda maupun sisi gelang.
   - Sisi Ganda, merupakan situasi terdapat dua atau lebih sisi yang menghubungkan dua simpul yang sama.
   - Sisi Gelang, merupakan sisi yang menghubungkan satu simpul dengan dirinya sendiri, yang juga disebut sebagai loop.

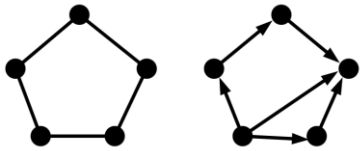
   Sehingga, dalam graf sederhana, setiap pasangan simpul hanya data dihubungkan oleh satu sisi, dan tidak ada sisi yang menghubungkan simpul dengan dirinya sendiri.
2. Graf Tak-Sederhana (*Unsimple Graph*)
   Graf tak-sederhana merupakan jenis graf yang mengandung sisi ganda atau sisi gelang, atau keduanya.
   - Graf Ganda (*Multigraph*)
     Graf ganda merupakan jenis graf yang memperbolehkan adanya lebih dari satu sisi yang menghubungkan dua simpul yang sama. Dengan kata lain, dua simpul yang identik dapat dihubungkan oleh beberapa sisi yang terpisah.
   - Graf Semu (*Pseudograph*)
     Graf semu adalah graf yang memungkinkan adanya sisi gelang.



Gambar 2.1. Jenis-Jenis Graf Berdasarkan Keberadaan Sisi Ganda dan Sisi Gelang (dari kiri: graf sederhana, graf ganda, dan graf semu)
Sumber: Koleksi Pribadi

Berdasarkan orientasi arah pada sisi atau tepi, graf dibedakan menjadi dua jenis, yaitu graf berarah dan graf tidak berarah.

1. Graf Tak Berarah (*Undirected Graph*)
   Graf tak berarah adalah jenis graf dengan hubungan antar dua simpul (*node*) tidak memiliki arah. Hal ini berarti jika ada sebuah sisi yang menghubungkan dua simpul, maka hubungan tersebut bersifat timbal balik. Contoh: jika terdapat sisi yang menghubungkan simpul A ke simpul B, maka simpul B juga terhubung dengan simpul A.
2. Graf Berarah (*Directed Graph*)
   Graf berarah merupakan jenis graf yang setiap sisinya memiliki arah tertentu. Setiap sisi menghubungkan dua simpul dengan arah yang jelas dari satu simpul ke simpul lainnya, sehingga hubungan antar simpul-simpul tidak bersifat timbal balik.
   Contoh: Jika ada sisi dari simpul A ke simpul B, mengartikan hanya A yang terhubung ke B, dan bukan sebaliknya, kecuali ada sisi yang berbeda yang mengarah dari B ke A.



Gambar 2.2. Jenis-Jenis Graf Berdasarkan Orientasi Sisi (dari kiri: graf tak berarah dan graf berarah)
Sumber: Koleksi Pribadi

c. Istilah-Istilah dalam Teori Graf

Berikut adalah beberapa istilah-istilah yang ada dalam teori graf, yaitu:

1. Ketetanggaan (*Adjacent*), dua simpul dikatakan saling bertetangga jika terdapat sebuah sisi yang menghubungkan keduanya secara langsung.
2. Bersisian (*Incidency*),sebuah simpul dikatakan bersisian dengan sebuah sisi jika sisi tersebut terhubung langsung dengan simpul tersebut, baik sisi tersebut berasal dari atau berakhir di simpul tersebut.
3. Simpul Terpencil (*Isolated Vertex*), merupakakan simpul yang tidak memiliki sisi yang menghubungkannya dengan simpul lainnya, sehingga tidak ada sisi yang bersisian dengan simpul tersebut.
4. Graf Kosong (*Null Graph* atau *Empty Graph*), merupakan jenis graf yang tidak memiliki sisi sama sekali, sehingga himpunan sisi pada graf ini adalah himpunan kosong.
5. Derajat (*Degree*), merupakan jumlah sisi yang terhubung langsung dedngan suatu simpul, menggambarkan banyaknya hubungan atau koneksi yang dimiliki simpul tersebut.
6. Lintasan (*Path*), merupakan urutan simpul yang dihubungkan oleh sisi, yang bisa berupa rangkaian terbatas maupun tak terbatas, yang menggambarkan jalur dari suatu simpul ke simpul lainnya dalam graf.
7. Siklus (*Cycle*) atau Sirkuit (*Circuit*), merupakan sebuah lintasan yang dimulai dan diakhiri pada simpul yang sama. Sirkuit memiliki panjang yang diukur berdasarkan jumlah sisi yang ada dalam lintasan tersebut. Hal ini serupa dengan pengukuran panjang lintasan.
8. Keterhubungan (*Connected*), dua simpul dikatakan terhubung jika terdapat sebuah lintasan yang menghubungkan simpul pertama dengan simpul kedua. Sebuah graf dikatakan graf terhubung (*connected graph*) jika setiap pasangan simpul dalam himpuinan simpul (v) dapat dihubungkan oleh sebuah lintasan.
9. Upagraf (*Subgraph*), sebuah graf G1 = {v_1, e_2,}

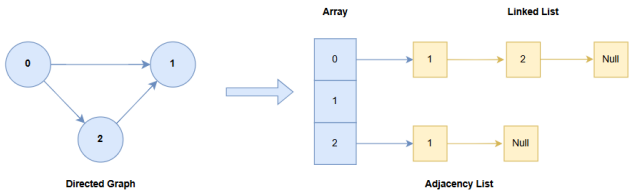
dikatakan sebagai upagraf dari graf G1 = (v, e) jika himpunan simpul v_1 merupakan bagian dari v ($v_1 \subseteq v$) and himpuna sisi e1 merupakan bagian dari e ($e_1 \subseteq e$).

10. Komplemen Upagraf (*Complement of a Subgraph*), komplemen dari upagraf G1 terhadap graf G adalah graf lain, misalnya G2= {v_2, e_2,} yang memenuhi syarat bahwa himpunan sisi $e_2 = e - e_1$, dan himpunan simpul v_2 adalah himpunan simpul yang sisi-sisi dalam e_2 yang beririsan dengannya.
11. Upagraf Merentang (*Spanning Subgraph*), upagraf merentang adalah sebuah upagraf yang mencakup semua simpul yang ada dalam graf asli.
12. Cut-Set, cut-set adalah himpunan sisi dalam sebuah graf yang jika dihapus, akan memutuskan keterhubungan antara beberapa simpul dalam graf sehingga graf menjadi tidak terhubung.

d. Representasi Graf

Graf dapat direpresentasikan dengan beberapa cara, antara lain:

1. Matriks Ketetanggaan (*Adjacency Matrix*)
   Representasi graf ini menggunakan matriks berukuran $m \times m$, di mana m adalah jumlah simpul dalam graf. Jika terdapat sisi yang menghubungkan simpul ke-i dengan simpul ke-j, maka elemen matriks pada posisi matriks[i][j] akan bernilai 1 atau sesuai bobot sisi tersebut. Sebaliknya, jika tidak ada sisi yang menghubungkan kedua simpul tersebut, nilai elemen matriks adalah 0.
2. Matriks Bersisian (*Incidency Matrix*)
   Pada representasi ini, baris pada matriks melambangkan simpul, sedangkan kolomnya merepresentasikan sisi. Nilai dalam matriks menunjukkan hubungan antara simpul dan sisi dalam graf.
3. List Ketetanggaan (*Adjancency List*)
   Graf direpresentasikan dalam bentuk struktur data berbasis daftar (list), di mana setiap simpul (*node*) memiliki daftar simpul lain yang terhubung dengannya melalui sisi. Representasi ini lebih hemat memori, terutama untuk graf yang jarang (*sparse graph*)



Gambar 2.3. Representasi Graf dengan Adjancency List
Sumber: Koleksi Pribadi

III. IMPLEMENTASI

Anagram adalah istilah dalam linguistik yang merujuk pada pembentukan sebuah kata atau frasa baru dengan menyusun ulang huruf-huruf dari kata atau frasa lain. Anagram dapat dianggap sebagai hasil permutasi huruf-huruf dari kata asal, dengan beberapa syarat berikut:

1. Jumlah huruf sama: Panjang kedua kata atau frasa harus identik.
2. Jenis huruf sama: Setiap huruf dalam kata atau frasa pertama harus sama dengan huruf-huruf dalam kata atau frasa kedua, meskipun urutannya berbeda.
3. Menggunakan semua huruf: Setiap huruf dalam kata atau frasa asal harus digunakan tepat satu kali dalam kata atau frasa hasil.

Pencarian kata-kata anagram dapat dioptimalkan dengan menggunakan teori graf. Graf yang digunakan dalam konteks ini adalah graf tak berarah (undirected graph), yang merepresentasikan hubungan antara kata-kata berdasarkan *signature*-nya.

A. Sampel Data

Dataset yang digunakan dalam implementasi program berasal dari Kaggle.com, sebuah platform yang menyediakan kompetisi *data science*, dataset publik, dan komunitas daring dengan lebih dari 16 juta anggota. Dataset yang dipilih adalah *English Word Frequency* yang disusun oleh Rachael Tatman, yang terdiri dari 333.333 kata yang sering digunakan dalam bahasa Inggris. Untuk pengaplikasian tertentu, dataset ini dapat diseleksi sehingga hanya sebagian data dalam rentang tertentu yang digunakan.

B. Struktur Graf untuk Anagram

Dalam implementasi ini, graf direpresentasikan sebagai list ketetanggaan (adjacency list), di mana setiap simpul (*node*) merepresentasikan huruf awal *signature*, *signature* itu sendiri, dan kata-kata yang berbagi *signature*.

Signature adalah representasi unik dari suatu kata, yang diperoleh dengan mengurutkan huruf-huruf dalam kata tersebut secara alfabetis. Misalnya, kata “listen”, “silent”, dan “enlist” memiliki huruf yang sama tetapi dalam urutan yang berbeda. Ketika huruf-huruf ini diurutkan, semuanya menghasilkan *signature* yang sama, yaitu “eilnst”. Dalam graf ini, setiap simpul mewakili sebuah *signature*, dan kata-kata yang berbagi *signature* terhubung melalui hubungan langsung.

Relasi antar simpul dalam graf dibangun berdasarkan aturan berikut:

1. Huruf awal ke *signature*: Simpul yang merepresentasikan huruf awal dari *signature* dihubungkan ke simpul *signature* dengan bobot sisi yang sesuai dengan panjang *signature*.
2. *Signature* ke kata anagram: Simpul *signature* dihubungkan ke simpul-simpul kata yang merupakan anagram dengan bobot sisi sebesar 1.
3. Kata ke kata: Kata-kata yang berbagi *signature* yang sama juga dihubungkan langsung satu sama lain dengan bobot sisi sebesar 1. Hal ini memastikan bahwa hubungan antara kata-kata anagram tetap terwakili secara eksplisit.

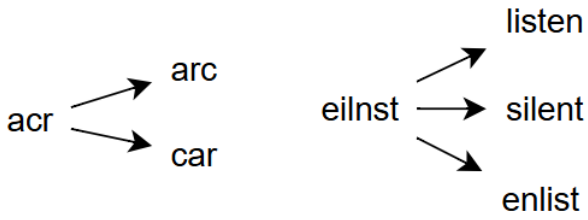
Pendekatan ini memungkinkan graf untuk secara efektif merepresentasikan hubungan antar kata dalam dataset besar dan mendukung proses identifikasi anagram dengan cepat dan efisien. Representasi *adjacency list* juga memberikan keuntungan dalam hal efisiensi memori, terutama saat bekerja dengan graf yang jarang (*sparse graph*).

C. Tahapan Implementasi

1. Identifikasi *Signature*

Tahap pertama adalah menghitung *signature* untuk setiap kata dalam dataset. *Signature* diperoleh dengan mengurutkan

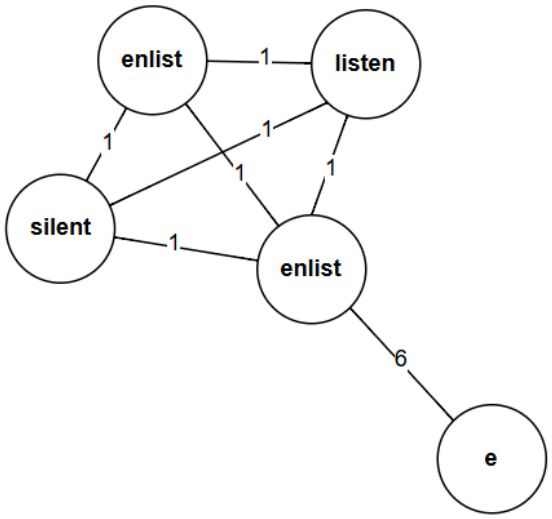
huruf-huruf pada sebuah kata secara alfabetis. Sebagai contoh, kata “arc” dan “car” memiliki *signature* yang sama, yaitu “acr”.



Gambar 3.1. Contoh Signature dan Hubungannya
Sumber: Koleksi Pribadi

2. Membangun Graf

Setelah *signature* diidentifikasi, graf dibangun berdasarkan aturan yang telah dideklarasikan sebelumnya.



Gambar 3.2. Graf Anagram disertai Aturan Sisi dan Simpul
Sumber: Koleksi Pribadi

Jika terdapat *signature* yang berbeda tetapi memiliki huruf awal yang sama, kedua *signature* tersebut akan terhubung melalui simpul *parent* yang sama. Namun, bobot pada sisi yang menghubungkan simpul-simpul tersebut berbeda, tergantung pada panjang kata masing-masing. Pendekatan ini memastikan bahwa graf dapat merepresentasikan hubungan antar kata dengan struktur yang terorganisir dan efisien.

3. Pencarian Anagram

Setelah graf selesai dibangun dari dataset, proses pencarian anagram dilakukan berdasarkan kata masukan (*input*). Langkah-langkahnya meliputi:

a. Mengidentifikasi huruf awal dari *signature* kata *input* dan mencari *signature* yang terhubung dengan bobot yang sesuai dengan panjang *signature*.
b. Membandingkan *signature* kata *input* dengan *signature* yang ditemukan untuk memastikan kecocokan.
c. Menelusuri kata-kata yang berbagi *signature* yang sama untuk menemukan semua kata yang merupakan anagram dari kata *input*.

D. Visualisasi Graf

Graf yang telah dibangun dapat divisualisasikan untuk mempermudah pemahaman tentang hubungan antara huruf awal, *signature*, dan kata-kata anagramnya. Dalam visualisasi, setiap simpul (*node*) dan sisi (*edge*) diberi label:

1. Simpul mewakili huruf awal, *signature*, atau kata anagram.
2. Sisi diberi bobot yang menunjukkan panjang *signature* atau hubungan antara kata-kata yang merupakan anagram.

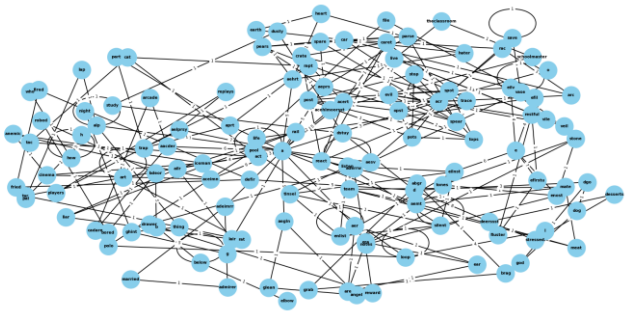
Visualisasi ini juga dapat membantu memvalidasi hasil implementasi dan menunjukkan pola hubungan antar kata secara intuitif. Program yang dibuat akan menampilkan graf berdasarkan dataset yang digunakan.

E. Penyimpanan Graf

Graf yang telah dibangun sesuai dengan aturan yang ada dapat disimpan untuk meningkatkan efisiensi proses. Penyimpanan ini memungkinkan penghematan waktu karena menghindari pembuatan ulang graf saat diperlukan. Ketika pencarian dilakukan, sistem hanya perlu mengakses graf yang sudah tersimpan tanpa perlu memulai proses dari awal. Pendekatan ini sangat bermanfaat untuk dataset besar atau ketika proses pencarian dilakukan berulang kali.

F. Implementasi Program untuk Menemukan Anagram dari Dataset Kata Bahasa Inggris (Bahasa Pemrograman Python)

Program pencarian anagram ini dirancang menggunakan graf berbobot seperti yang dijelaskan pada Gambar 1. Program berbasis *Command Line Interface* (CLI) ini dibangun dengan bahasa pemrograman Python, yang menawarkan fleksibilitas dalam manipulasi data dan implementasi graf.



Gambar 3.3. Tampilan Graf pada Program
Sumber: Koleksi Pribadi

Dataset kata yang digunakan dalam program ini berasal dari platform Kaggle, sebagaimana dijelaskan dalam bagian pendahuluan. Namun, program ini dirancang agar fleksibel sehingga dapat menggunakan dataset lain atau bahkan memungkinkan pengguna untuk melakukan input data secara manual dengan memodifikasi kode. Algoritma inti dari program dapat diakses melalui repositori GitHub(" https://github.com/mineraleee/Anagram_Solver_Graf").

Tabel 3.1. Input Dataset pada Program

```
  with open(filename, 'r') as csvfile:
          csvreader = csv.reader(csvfile)
          fields = next(csvreader)
          for i,row in enumerate(csvreader):
             if i>=500: #Jika hanya ingin mengambil
beberapa (500) kata pertama saja.
                 break
```

```
            words.append(row[0])
```

Tabel 3.2. Input Kata Manual pada Program

```
    words = [
            "listen", "enlist", "silent", "tinsel",
"who", "how", "act", "cat", "tac",
            "arc", "car", "rac", "dusty", "study",
"save", "vase", "stop", "post", "spot", "tops",
            "elbow", "below", "stressed",
"desserts", "rat", "tar", "art", "evil", "vile",
"veil", "live",
            "bored", "robed", "fired", "fried",
"god", "dog", "heart", "earth", "hater",
            "stone", "notes", "tones", "cinema",
"iceman", "anemic", "angel", "glean", "brag",
"grab",
            "arcade", "cadare", "caret", "react",
"crate", "trace", "file", "life",
            "meat", "mate", "team", "part", "trap",
"rapt", "ear", "are", "era", "lap", "pal", "alp",
            "pears", "parse", "spare", "spear",
"listen", "silent", "enlist", "fluster",
"restful",
            "admirer", "married", "players",
"replays", "schoolmaster", "theclassroom",
            "rail", "liar", "lair", "drawer",
"reward",
            "save", "vase", "thing", "night",
"loop", "pool", "polo"
        ]
```

Dalam implementasi program ini, terdapat enam fungsi utama yang mendukung proses identifikasi dan pencarian anagram:

1. `get_signature(word)`

Fungsi ini menerima sebuah kata sebagai parameter dan mengembalikan *signature* dari kata tersebut. *Signature* didefinisikan sebagai versi kata yang huruf-hurufnya telah diurutkan secara alfabetis. Sebagai contoh, untuk kata *"listen"*, fungsi ini akan menghasilkan *"eilnst"*.

2. `build_graph_with_weights(words)`

Fungsi ini bertugas membangun graf berbobot berdasarkan daftar kata yang diberikan. Graf direpresentasikan menggunakan *adjacency list*. Hubungan antar simpul (*node*), baik antara kata dengan *signature* maupun antar kata yang merupakan anagram, direpresentasikan sebagai sisi (*edge*) dengan bobot tertentu.

Tabel 3.3. Fungsi `get_signature()` dan `build_graph_with_weights()`

```
    def get_signature(word):
        return ''.join(sorted(word))

    def build_graph_with_weights(words):
        graph = defaultdict(list)
        signature_map = defaultdict(list)

        for word in words:
            signature = get_signature(word)
            signature_map[signature].append(word)

        for signature, word_list in
signature_map.items():
            initial = signature[0]
            graph[initial].append((signature,
len(signature)))

            for word in word_list:
                graph[signature].append((word, 1))
            for i, word1 in enumerate(word_list):
                for word2 in word_list[i + 1:]:
                    graph[word1].append((word2, 1))
                    graph[word2].append((word1, 1))

        return graph
```

Untuk mempermudah pemahaman hubungan antar kata, program memanfaatkan library *NetworkX* untuk menampilkan visualisasi graf. Namun, visualisasi ini akan menjadi kurang efektif jika dataset yang digunakan sangat besar karena kepadatan hubungan antar simpul dapat membingungkan.

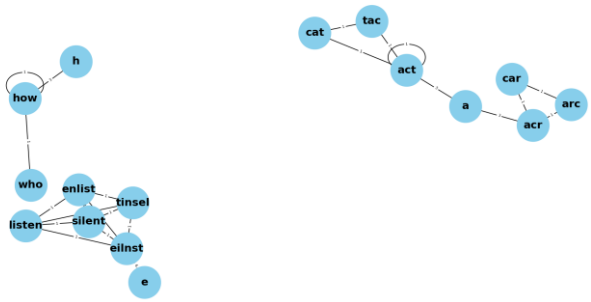
Tabel 3.4. Fungsi `visualize_graph()`

```
    def visualize_graph(graph):
        G = nx.Graph()
        for node, edges in graph.items():
            for neighbor, weight in edges:
                G.add_edge(node, neighbor,
weight=weight)

        pos = nx.spring_layout(G, k=0.7,
iterations=50)
        edge_labels = nx.get_edge_attributes(G,
'weight')

        nx.draw(G, pos, with_labels=True,
node_color='skyblue', node_size=500, font_size=5,
font_weight='bold')
        nx.draw_networkx_edge_labels(G, pos,
edge_labels=edge_labels, font_size=5)

        plt.title("Visualisasi Graf Anagram dengan
Bobot", fontsize=5)
        plt.show()
```



Gambar 3.4. Graf Anagram dengan Beberapa Kata
Sumber: Koleksi Pribadi

Setelah graf berhasil dibuat, graf yang telah dibangun dapat disimpan dalam file dengan format `.pkl` menggunakan fungsi `save_graph_to_file(graph,filename)`. Penyimpanan ini memungkinkan graf yang sama digunakan kembali tanpa perlu membangun ulang. Untuk memuat kembali graf yang telah disimpan, program menyediakan fungsi `load_graph_from_file(filename)`. Proses ini menghemat waktu dan sumber daya, terutama untuk dataset besar.

Tabel 3.5. Fungsi `save_graph_to_file()` dan `load_graph_from_file()`

```
    def save_graph_to_file(graph, file_path):
        with open(file_path, 'wb') as f:
            pickle.dump(graph, f)
        print(f"Graf disimpan ke {file_path}")
```

```
def load_graph_from_file(file_path):
    with open(file_path, 'rb') as f:
        return pickle.load(f)
```

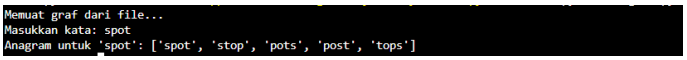
Setelah graf selesai dibangun, pengguna dapat melakukan pencarian anagram untuk kata tertentu dengan memanfaatkan fungsi pencarian. Prosesnya melibatkan:

1. Menentukan *signature* dari kata masukan (input).
2. Mencari simpul (*node*) yang memiliki huruf awal dan panjang *signature* yang sesuai.
3. Menelusuri semua kata yang berbagi *signature* yang sama dengan kata masukan untuk menampilkan daftar anagram.
4. Hasil pencarian berupa kata-kata yang memiliki hubungan dengan bobot sebesar 1, yang menandakan bahwa kata-kata tersebut adalah anagram langsung.

Tabel 3.6. Fungsi `find_anagrams()`

```
def find_anagrams(graph, word):
    signature = get_signature(word)
    initial = signature[0]
    if initial not in graph:
        return []
    anagrams = []
    for neighbor, weight in graph[initial]:
        if weight == len(signature):
            if get_signature(neighbor) ==
signature and neighbor != word:
                for sub_neighbor, sub_weight
in graph[neighbor]:
                    if sub_weight == 1:
                        anagrams.append(sub_ne
ighbor)
    return list(set(anagrams))
```

Dalam pemrosesan graf, program ini tidak hanya bertujuan untuk menemukan anagram, tetapi juga dapat digunakan untuk mengelompokkan kata-kata berdasarkan *signature*-nya. Hal ini dapat dimanfaatkan dalam analisis linguistik atau pengolahan teks lainnya. Visualisasi yang dilakukan juga dapat ditingkatkan dengan *filter* atau *zooming* untuk menampilkan hanya sebagian dataset yang relevan.



Gambar 3.5. Tampilan List Anagram sesuai Masukan Pengguna
Sumber: Koleksi Pribadi

IV. HASIL DAN PEMBAHASAN

Pencarian kata anagram pada sekumpulan data dapat dioptimalkan dengan pendekatan berbasis graf. Dalam implementasi ini, graf dibangun menggunakan *adjacency list*, di mana setiap simpul (*node*) merepresentasikan *signature* dari kata, dan setiap sisi (*edge*) menghubungkan kata-kata yang merupakan anagram satu sama lain. *Signature* dari sebuah kata dihitung dengan mengurutkan huruf-hurufnya dalam urutan alfabetis. Hal ini memastikan bahwa kata-kata dengan huruf yang sama (hanya dalam urutan berbeda) memiliki *signature* yang identik, sehingga dapat dikelompokkan secara efisien dalam graf. Dengan struktur ini, simpul yang memiliki *signature* yang sama akan menjadi pusat pengelompokan kata-kata anagram.

Proses pencarian anagram dimulai dengan menemukan simpul berdasarkan huruf pertama dari *signature* kata masukan. Setelah itu, bobot sisi yang menghubungkan simpul tersebut dengan *signature* lainnya diperiksa untuk memastikan kesesuaian panjang *signature*. Jika ditemukan kecocokan, algoritma kemudian memeriksa apakah kata-kata yang terhubung tersebut adalah anagram. Ketika satu anagram ditemukan, langkah berikutnya hanya perlu memeriksa kata-kata yang memiliki koneksi langsung ke kata tersebut (sisi berbobot 1).

Pendekatan ini efisien dalam mengelompokkan dan mencari anagram karena memanfaatkan struktur graf untuk meminimalkan redundansi dalam iterasi. Dengan adjacency list dan pemeriksaan berbasis *signature*, hubungan antar kata dapat diidentifikasi secara lebih sistematis. Efisiensi juga dapat dilakukan dengan adanya proses penyimpanan data graf sehingga cukup melakukan konstruksi graf sekali saja kemudian menggunakannya berulang kali untuk proses pencarian anagram.

Namun, kelemahan utama terletak pada kompleksitas algoritma dalam fungsi `find_anagrams()`, yang memiliki struktur loop bersarang. Loop pertama mengiterasi tetangga (neighbor) simpul berdasarkan huruf awal *signature*, sementara loop kedua mengevaluasi koneksi antar neighbor. Jika jumlah kata dalam satu kelompok *signature* cukup besar, loop bersarang ini dapat meningkatkan waktu komputasi.

Dari sisi kompleksitas, perbandingan metode graf dengan pendekatan *brute force* menunjukkan keunggulan graf dalam kondisi tertentu. Pada *brute force*, setiap kata dibandingkan dengan seluruh data yang tersedia, menghasilkan kompleksitas $O(N^2)$, di mana N adalah jumlah total kata. *Brute force* juga dapat dilakukan dengan menggunakan semua permutasi dari huruf-huruf kata yang menyebabkan kompleksitasnya adalah O(N!). Sebaliknya, graf membatasi iterasi hanya pada jumlah *signature* yang unik dan tetangga yang relevan. Namun, jika dataset memiliki banyak *signature* unik dengan sedikit kata per kelompok, metode graf dapat menjadi tidak efisien dibandingkan *brute force*.

V. KESIMPULAN

Penerapan teori graf dalam pencarian kata yang anagram menunjukkan struktur graf dapat digunakan untuk menyusun dan mencari hubungan antarkata secara efisien. Dengan menggunakan *adjancency list* dan *signature* yang dihitung berdasarkan pengurutan huruf, algoritma ini dapat secara efektif mengidentifikasi anagram dalam kumpulan kata. Pendekatan graf memberikan keunggulan untuk kasus dengan dataset besar yang sering diperbarui atau memerlukan pencarian anagram yang berulang. Namun, metode ini memiliki kelemahan dalam menangani dataset dengan banyak *signature* unik, di mana kelebihan struktur graf menjadi kurang terasa.

Meskipun algoritma ini cukup efisien dalam banyak kasus, kompleksitasnya dapat meningkat signifikan ketika jumlah kata dalam satu *signature* menjadi sangat besar. Oleh karena itu, pemilihan pendekatan harus disesuaikan dengan karakteristik data dan kebutuhan pencarian.

VII. UCAPAN TERIMA KASIH

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa, karena atas rahmat dan bimbingan-Nya, penulis dapat menyelesaikan makalah ini yang berjudul *"Optimalisasi Anagram Solver Menggunakan Pendekatan Graf"* tepat waktu. Makalah ini disusun sebagai bagian dari tugas pada mata kuliah Matematika Diskrit IF2120.

Penulis ingin menyampaikan rasa terima kasih yang mendalam kepada Bapak Dr. Ir. Rinaldi, M.T., Dr. Rila Mandala, dan Bapak Arrival Dwi Sentosa, M.T., yang dengan tulus dan penuh dedikasi telah membimbing dan membagikan ilmu selama perkuliahan berlangsung. Ilmu yang diberikan menjadi bekal berharga dalam proses penyelesaian makalah ini.

Tidak lupa, penulis juga mengucapkan terima kasih yang sebesar-besarnya kepada kedua orang tua yang selalu memberikan dukungan, baik secara moral maupun materi, selama proses pengerjaan makalah ini. Ucapan terima kasih juga penulis sampaikan kepada teman-teman dekat—Adinda Putri, Ranashahira, dan Wardatul Khoiroh—yang telah menjadi tempat berbagi ide, memberikan semangat, dan selalu mendukung selama proses ini berlangsung.

Penulis menyadari bahwa makalah ini masih jauh dari sempurna. Oleh karena itu, penulis memohon maaf apabila terdapat kesalahan atau kekurangan dalam penulisan ini. Semoga makalah ini dapat memberikan manfaat bagi para pembaca dan menjadi langkah kecil yang berkontribusi pada pengembangan pengetahuan.

REFERENSI

[1] AlgoMonster, "49. Group Anagrams," [Online]. Tersedia: https://algo.monster/liteproblems/49. [Diakses pada 21 Desember 2024].

[2] Egbunike, Crystal, "Graph Theory and Its Applications," 2002. [Diakses pada 22 Desember 2024].

[3] GeeksforGeeks, "Graph and its Representations," [Online]. Tersedia: https://www.geeksforgeeks.org/graph-and-its-representations/. [Diakses pada 25 Desember 2024].

[4] Munir, Rinaldi, "Graf (Bagian 1)," [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf. [Diakses pada 21 Desember 2024].

[5] Munir, Rinaldi, "Graf (Bagian 2)," [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf. [Diakses pada 21 Desember 2024].

[6] Sarris, Menelaos E., "Linguistic Effects on Anagram Solution: The Case of a Transparent Language," *World Journal of Education*, 2013. [Diakses pada 21 Desember 2024].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 Desember 2024

Heleni Gratia M Tampubolon
(13523107)