Perbandingan Algoritma BFS dan DFS dalam Mencari Anagram dari Suatu Kata

Michelle Eliza Gananjaya / 13516015
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Bandung, Indonesia
michelle.eliza79@gmail.com

***Abstract*—Anagram dari suatu kata adalah suatu kata lain yang memiliki huruf yang sama, tetapi dengan susunan yang berbeda. Anagram banyak ditemui dalam teka-teki dan permainan, tetapi dapat juga diaplikasikan pada hal-hal lain yang lebih serius, seperti kriptografi. Untuk mencari anagram dari suatu kata tertentu, dapat digunakan algoritma *Breadth-First Search* dan *Depth-First Search*, dan akan dibandingkan efektivitas dan efisiensi dari kedua algoritma tersebut dalam mencari anagram.**

Keywords—BFS, DFS, Algoritma, Anagram

I. PENDAHULUAN

Anagram dari suatu kata dapat ditemukan dengan mengacak huruf-huruf yang ada dalam kata tersebut dengan urutan yang berbeda, dan setiap huruf muncul sesuai dengan jumlah kemunculan pada kata pertama, tetapi anagram bukan hanya pola huruf yang acak, namun juga memiliki makna. Kata asli yang akan diacak sering disebut dengan subjek dari anagram. Selain sebuah kata, anagram juga dapat merupakan acakan dari sebuah frasa. Berikut adalah contoh-contoh anagram dari kata dan frasa:

- *secure – rescue*
- *angel – glean*
- akronim – monarki
- *garbage man – bag manager*
- *eleven plus two – twelve plus one*

Menurut beberapa ahli sejarah, anagram sudah digunakan oleh manusia sejak tahun 400 S.M, saat penyair Yunani Lycophron menggunakannya untuk merayu kaum kaya dan penguasa. Ahli yang lain berkata bahwa Pythagoras telah menggunakannya pada tahun 600 S.M. Tetapi, anagram baru mulai popular di khalayak luas di Benua Eropa pada abad pertengahan.

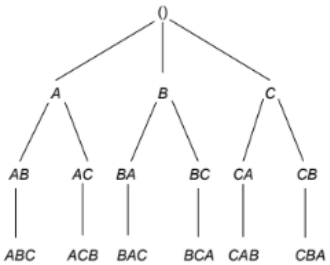
Pada abad 16-17, ilmuwan terkenal seperti Galileo, Huygens, Robert Hooke, sering menyimpan hasil penemuan mereka dengan anagram untuk menjaga hasil temuan tersebut agar tidak diklaim oleh orang lain. Hingga saat ini, selain untuk kebutuhan rekreasi seperti permainan *Scrabble*, *Bookworm*, dsb., anagram juga digunakan untuk nama samaran atau *pseudonym*, dan juga sebagai teknik untuk memecahkan beberapa jenis sandi kriptografi, yaitu teknik *multiple anagramming* yang dapat memecahkan *permutation cipher*, *transposition cipher*, dan *Jefferson disk.* [1]

II. DASAR TEORI

A. *Pohon Dinamis*

Pohon dinamis adalah struktur data yang dapat digunakan untuk mencari solusi. Pohon dinamis dibangun selama pencarian solusi berlangsung, dan dapat dibangun dengan dua metode traversal, yaitu *Breadth-First Search* dan *Depth-First Search*. Pohon dinamis menyatakan status-status persoalan saat pencarian solusi berlangsung.

- Status persoalan (*problem state*): simpul-simpul di dalam pohon yang memenuhi *constraints* dari persoalan yang dicari solusinya.
- Status solusi (*solution state*): satu atau lebih status pohon yang menyatakan solusi dari persoalan.
- Status tujuan (*goal state*): status solusi yang merupakan simpul daun dari pohon dinamis.
- Ruang solusi (*solution space*): himpunan semua status solusi.
- Ruang status (*state space*): Seluruh simpul di dalam pohon dinamis, pohonnya diebut juga pohon ruang status.[2]

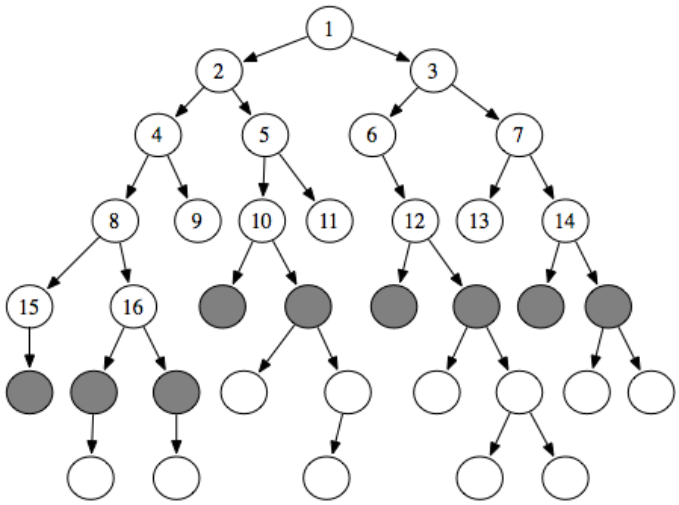


Ket: () = status kosong

Gambar 2.1
Contoh Pohon Ruang Status
Sumber: Diktat Strategi Algoritma Rinaldi Munir

B. Algoritma Breadth-First Search

Breadth-First Search atau pencarian melebar adalah salah satu algoritma traversal yang digunakan untuk menelusuri struktur data pohon atau graf. Penelusuran dimulai dari simpul akar dan akan mengunjungi semua simpul yang memiliki kedalaman selanjutnya, begitu terus berlanjut hingga menemukan solusi atau mencapai simpul daun yang terakhir.



Gambar 2.2
Ilustrasi Urutan Pencarian dengan BFS
Sumber: http://artint.info/figures/ch03/sgraph_br_new.png

Langkah dari algoritma BFS yang dimulai dari simpul *v* adalah:

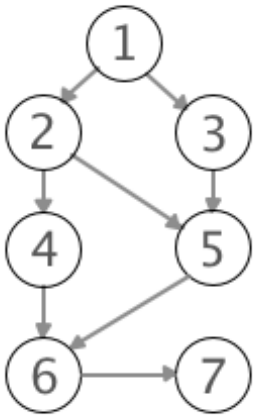
1. Kunjungi simpul *v*.
2. Kunjungi semua simpul yang bertetangga dengan simpul *v* terlebih dahulu.
3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.

Jika graf berbentuk pohon berakar, maka semua simpul pada aras *d* dikunjungi lebih dahulu sebelum simpul-simpul pada aras *d*+1.

Untuk mengimplementasikan algoritma BFS, diperlukan beberapa struktur data, salah satunya adalah matriks ketetanggaan sebagai representasi dari graf atau pohon yang akan ditelusuri. Matriks ini berukuran n x n, di mana n adalah jumlah simpul yang ada pada graf, dan nilai dari sel pada baris ke-*i* dan kolom ke-*j* akan bernilai 1 jika simpul *i* dan *j* bertetangga, dan bernilai 0 jika tidak. Selain itu, diperlukan juga struktur data antrian atau *queue* untuk menyimpan simpul-simpul yang telah dikunjungi, dan juga struktur data larik atau *array* yang berisi *boolean*, di mana elemen larik pada indeks ke-*i* akan bernilai *true* jika simpul ke-*i* telah dikunjungi.

Algoritma BFS bersifat *complete*, yang berarti solusi dari pencarian akan selalu ditemukan selama jumlah cabang maksimum yang dapat dimiliki suatu simpul terbatas. Algoritma ini juga dapat dinilai optimal jika biaya yang diperlukan untuk mencapai suatu simpul sama dengan jumlah langkah yang diperlukan untuk mencapai simpul tersebut. Kompleksitas waktu dari algoritma BFS adalah $O(b^d)$, di mana *b* adalah jumlah cabang maksimum yang dapat dimiliki suatu simpul dan *d* adalah kedalaman dari solusi optimal dari persoalan. Kompleksitas ruang dari algoritma BFS adalah $O(b^d)$, sehingga eksekusi algoritma ini membutuhkan banyak ruang dalam memori.

Contoh kasus:



Gambar 2.3
Contoh Graf Berarah
Sumber:
http://www.koderdojo.com/media/default/articles/directed-acyclic-graph-computer-science.png

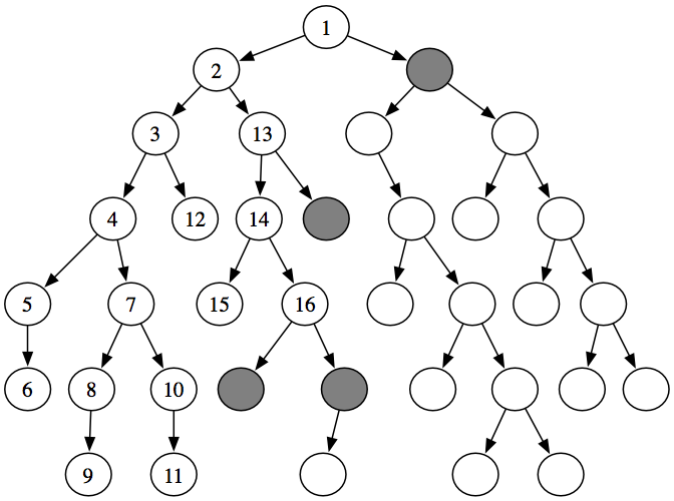
Berikut adalah ilustrasi penelusuran seluruh simpul graf pada gambar 2.3 secara traversal dengan algoritma BFS:

Iterasi	V	Q	dikunjungi						
Inisialisasi	1	{1}	1	2	3	4	5	6	7
Iterasi 1	1	{2,3}	T	F	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	F	F	F	F	F
Iterasi 3	3	{4,5}	T	T	T	F	F	F	F
Iterasi 4	4	{5,6}	T	T	T	T	F	F	F
Iterasi 5	5	{6}	T	T	T	T	T	F	F
Iterasi 6	6	{7}	T	T	T	T	T	T	F
Iterasi 7	7	{}	T	T	T	T	T	T	T

Tabel 2.1
Ilustrasi Penelusuran Graf secara Traversal dengan Algoritma BFS
Sumber: Dokumen Pribadi

C. Algoritma Depth-First Search

Depth First-Search atau pencarian mendalam juga merupakan salah satu algoritma traversal yang digunakan untuk menelusuri struktur data pohon atau graf. Penelusuran dimulai dari simpul akar, lalu dilanjutkan ke simpul yang bertetangga dengan simpul sebelumnya, hingga mencapai simpul daun. Jika simpul daun yang dicapai bukan merupakan solusi, akan dilakukan runut-balik hingga ditemukan simpul lain yang belum ditelusuri.



Gambar 2.4
Ilustrasi Urutan Pencarian dengan DFS
Sumber: http://artint.info/figures/ch03/sgraph_de_new.png

Langkah dari algoritma DFS yang dimulai dari simpul *v* adalah:

1. Kunjungi simpul *v*.
2. Kunjungi simpul *w* yang bertetangga dengan simpul *v*.
3. Ulangi DFS mulai dari simpul *w*.
4. Ketika mencapai simpul *u* sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan memiliki simpul *w* yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Khusus untuk graf berarah, mungkin terdapat beberapa simpul yang tidak dapat dicapai dari simpul awal. Penelusuran dapat dilanjutkan dengan memulai DFS dengan simpul awal berupa simpul yang belum pernah dikunjungi sebelumnya.

Untuk mengimplementasikan algoritma DFS, dibutuhkan struktur data tumpukan atau *stack* untuk menyimpan simpul-simpul yang telah dikunjungi. Simpul-simpul tersebut perlu disimpan dalam tumpukan agar memungkinkan untuk melakukan runut-balik. Saat suatu simpul dikunjungi, simpul tersebut akan di-*push* atau dimasukkan ke dalam tumpukan, dan jika dilakukan runut-balik, simpul teratas akan di-*pop* atau dikeluarkan dari tumpukan. Selain struktur data *stack*, diperlukan juga struktur data matriks ketetanggaan dan larik berisi *boolean* seperti pada algoritma BFS.

DFS adalah algoritma yang bersifat lengkap atau *complete* jika jumlah cabang maksimum yang dapat dimiliki suatu simpul terbatas, dan ada penanganan khusus untuk jalur yang tidak diperlukan (*redundant paths*) dan status yang sudah pernah muncul sebelumnya (*repeated states*). Jika kasus-kasus tersebut tidak ditangani, DFS mungkin akan terjebak pada satu cabang yang memiliki panjang tak hingga namun tidak mengarah kepada solusi, yang menyebabkan algoritma ini bukanlah algoritma pencarian solusi yang optimal. Algoritma DFS memiliki kompleksitas waktu $O(b^m)$, di mana *b* adalah jumlah cabang maksimum yang dapat dimiliki suatu simpul dan *m* adalah kedalaman maksimum dari ruang status. Sedangkan kompleksitas ruang dari algoritma DFS adalah O(bm). Dapat dilihat bahwa dalam kompleksitas ruang algoritma DFS lebih baik dari algoritma BFS, tetapi tidak dalam kompleksitas waktu.[2]

Berikut adalah ilustrasi penelusuran seluruh simpul graf pada gambar 2.3 secara traversal dengan algoritma DFS:

1. DFS(1): dikunjungi[1] = *true*; DFS(2)
2. DFS(2): dikunjungi[2] = *true*; DFS(4)
3. DFS(4): dikunjungi[4] = *true*; DFS(6)
4. DFS(6): dikunjungi[6] = *true*; DFS(7)
5. DFS(7): dikunjungi[7] = *true*; *backtrack*; DFS(5)
6. DFS(5): dikunjungi[5] = *true*; *backtrack*; DFS(3)
7. DFS(3): dikunjungi[3] = *true*

DFS merupakan algoritma pencarian yang efisien, namun tidak ada jaminan akan menemukan solusi dengan langkah dari simpul akar yang minimum. Bahkan, jika salah memilih langkah, DFS akan terjebak pada jalur yang sangat panjang atau tak hingga. Karena itu, dilakukan berbagai usaha untuk memperbaiki algoritma DFS.

Salah satu solusi dari permasalahan tersebut adalah algoritma DLS atau *Depth-Limited Search*. Algoritma ini pada dasarnya adalah DFS, namun dengan pembatasan kedalaman pencarian sampai *i*. Ketika mencapai simpul dengan kedalaman *i*, simpul tersebut dianggap tidak memiliki *successor*. Kompleksitas waktu dari algoritma DLS adalah $O(b^i)$, sedangkan kompleksitas ruangnya adalah O(bi). Algoritma ini dapat diterapkan pada permasalahan yang jumlah langkah untuk mencapai solusinya sudah diketahui. Jika tidak diketahui, algoritma ini tidak dijamin akan menemukan solusi.

Solusi lainnya adalah algoritma IDS atau *Iterative Deepening Search.* Algoritma ini juga didasarkan pada DFS, dengan batas kedalaman (*d*) yang meningkat untuk setiap iterasi, hingga ditemukan solusi. Terdapat asumsi bahwa sebagian besar simpul berada di bagian bawah, sehingga tidak akan memakan biaya besar untuk membangkitkan simpul di bagian atas berulang kali. Algoritma ini bersifat lengkap jika jumlah cabang maksimum yang dapat dimiliki suatu simpul terbatas. Kompleksitas waktu dari algoritma IDS adalah $O(b^d)$ dan kompleksitas ruangnya adalah $O(bd)$.[2]

III. Implementasi

A. Kasus Uji

Untuk menguji perbandingan kecepatan algoritma BFS dan DFS, akan disediakan kasus uji berupa kata atau frasa dan anagram padanan dari kata atau frasa tersebut. Meskipun sebenernya sebuah kata atau frasa dapat memiliki lebih dari satu anagram, dipilih kata dan frasa yang hanya memiliki satu anagram untuk memudahkan implementasi.

Terdapat lima kasus uji yang akan digunakan, yang pertama adalah sebuah kata yang terdiri dari 3 karakter, kata yang terdiri dari 5 karakter, kata yang terdiri dari 7 karakter, kata yang terdiri dari 9 karakter, dan frasa yang terdiri dari 10 karakter.

Berikut adalah kasus-kasus uji yang akan digunakan:

- *bat – tab*
- *weird – wired*
- *capitol – optical*
- *emigrants – streaming*
- *forty five – over fifty*

Dalam pengimplementasian algoritma BFS dan DFS, tanda baca dalam frasa akan diabaikan untuk memudahkan pengimplementasian dan mempercepat waktu eksekusi.

B. Implementasi Algoritma BFS

Berikut adalah *pseudocode* dari algoritma BFS[2]:

```
procedure BFS (input v: integer)
{Traversal graf dengan algoritma
pencarian BFS.
Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpul yang dikunjungi
dicetak ke layar}

Deklarasi
w: integer
q: antrian
procedure BuatAntrian(input/output q:
antrian)
{membuat antrian kosong, kepala(q) diisi
0}

procedure MasukAntrian (input/output q:
antrian, input v: integer)
{memasukkan v ke dalam antrian q pada
posisi belakang}

procedure HapusAntrian (input/output q:
antrian, output v: integer)
{menghapus v dari kepala antrian q}

function AntrianKosong (input q:
antrian) → boolean
{true jika antrian q kosong, false jika
sebaliknya}

Algoritma:
BuatAntrian(q) {buat antrian kosong}
write(v) {cetak simpul awal yang
dikunjungi}
dikunjungi[v] ← true {simpul v telah
dikunjungi, tandai dengan true}
MasukAntrian(q,v) {masukkan simpul awal
kunjungan ke dalam
antrian}

{kunjungi semua simpul graf selama
antrian belum kosong}
while not AntrianKosong(q) do
  HapusAntrian(q,v) {simpul v telah
dikunjungi, hapus dari antrian}
  for w ← 1 to n do
    if A[v, w] = 1 then
      if not dikunjungi[w] then
        write(w) {cetak simpul yang
dikunjungi}
        MasukAntrian(q,w)
        dikunjungi[w] ← true
      endif
    endif
  endfor
endwhile
{AntrianKosong(q)}
```

Berikut adalah *pseudocode* dari implementasi algoritma BFS untuk mencari anagram:

```
procedure anagramBFS(input word1:
string, input word2: string)
{Membangun pohon status dengan BFS
hingga ditemukan status yang sama dengan
word2}

Deklarasi:
q: queue of string
temp: string
i: integer
c: char
found: boolean
```

```
Algoritma:
found ← false {inisialisasi}
for i ← 1 to panjang word1 do
  c ← word1[i]
  q.enqueue(c)
  {inisialisasi queue dengan semua
karakter pada word1}
endfor
while (not q.empty() and not found) do
  state ← q.dequeue()
  for semua tetangga state do
    if belum dikunjungi then
      state ← state + tetangga
      q.enqueue(temp)
      if temp = word2 then
        found ← true
        {kata telah ditemukan, keluar
dari loop}
      endif
    endif
  endfor
endwhile
```

C. Implementasi Algoritma DFS

Karena jumlah langkah yang diperlukan untuk mencapai simpul solusi sudah diketahui, yaitu sejumlah karakter dalam kata dan frasa yang menjadi input, maka algoritma yang akan digunakan adalah salah satu variasi dari algoritma DFS yaitu *Depth-Limited Search* atau DLS.

Berikut adalah *pseudocode* dari algoritma DFS[2]:

```
procedure DFS (input v: integer)
{Mengunjungi seluruh simpul graf dengan
algoritma pencarian DFS
Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpulyang dikunjungi
ditulis ke layar}

Deklarasi
w: integer

Algoritma:
write(v)
dikunjungi[v] ← true
for w ← 1 to n do
  if A[v,w] = 1 then {simpul v dan
simpul w bertetangga}
    if not dikunjungi[w] then
      DFS(w)
    endif
  endif
endfor
```

Berikut adalah *pseudocode* dari algoritma DLS[2]:

```
function DLS (problem, limit) → rec_DLS
(make_node(init_state), problem, lim)

function rec_DLS (node, problem, limit)
if isGoal(node) then
  → solution(node)
else if depth(node) = limit then
  → cutoff
else
  for each successor in expand (node,
problem) do
    result ← rec_DLS (successor,
problem, limit)
    if result = cutoff then
      cutoff_occured ← true
    else if result ≠ failure then
      → result
  endfor
endif
if cutoff_occured then
  → cutoff
else
  → failure
endif
```

Berikut adalah *pseudocode* dari implementasi algoritma DFS-DLS untuk mencari anagram:

```
procedure anagramDFS (input word1:
string, input word2: string, input
level: integer, input limit: integer,
input/output visited: array of boolean,
input/output state: string)
{Membangun pohon status dengan DFS
hingga ditemukan status yang sama dengan
word2}

Deklarasi {tidak ada}

Algoritma:
if level = lim then
  if state = word2 then
    {kata telah ditemukan}
    → keluar dari fungsi
endif
for w ← 1 to panjang word1 do
  if not visited[i] then
    state ← state + word1[i]
    visited[i] ← true
    {pemanggilan rekursif}
    anagramDFS(word1, word2, level + 1,
limit, visited, state)
    state[i] ← NULL
    visited[i] ← false
  endif
endfor
```

IV. Hasil Uji Dan Analisis

A. *Hasil Uji*

Berikut adalah hasil uji dari implementasi algoritma BFS untuk mencari anagram dari suatu kata:

Kasus Uji	Jumlah Karakter	Waktu
1	3	0.00046227458768399575 s
2	5	0.002445650663242157 s
3	7	0.1305461725075339 s
4	9	20.46752895203078 s
5	10	69.62169774084143 s

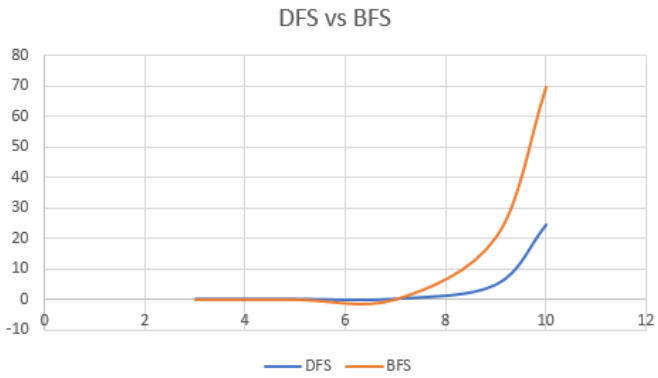
Tabel 4.1
Hasil Uji Algoritma BFS
Sumber: Dokumen Pribadi

Berikut adalah hasil uji dari implementasi algoritma DFS untuk mencari anagram dari suatu kata:

Kasus Uji	Jumlah Karakter	Waktu
1	3	0.0003155954169387501 s
2	5	0.0009579047885403798 s
3	7	0.050434542388783615 s
4	9	4.772099911179989 s
5	10	24.545838951402153 s

Tabel 4.2
Hasil Uji Algoritma DFS
Sumber: Dokumen Pribadi

B. *Analisis*



Gambar 4.1
Perbandingan Hasil Uji Algoritma BFS dan DFS
Sumber: Dokumen Pribadi

Secara waktu, berdasarkan Gambar 4.1 dan hasil uji pada Tabel 4.1 dan Tabel 4.2, terlihat bahwa performa algoritma BFS dan DFS hamper sama untuk kata atau frasa dengan jumlah karakter yang sedikit (n <= 7) dengan DFS sedikit mengungguli BFS, tetapi perbedaan sangat tampak pada hasil uji dengan kasus berupa kata atau frasa dengan jumlah karakter yang banyak (n > 7), di mana algoritma DFS lebih cepat daripada algoritma BFS. Hal ini dapat disebabkan oleh jumlah ruang status yang harus dibangkitkan untuk dapat mencapai status solusi, di mana algoritma BFS membangkitkan lebih banyak status dan menghabiskan lebih banyak waktu.

Secara kompleksitas ruang dan memori yang dipakai, saat kedua algoritma tersebut dicoba dengan kasus uji dengan jumlah karakter lebih dari 10, yaitu 11 karakter, pada eksekusi algoritma BFS terjadi *memory error*, yang berarti memori sudah tidak dapat lagi menampung pohon ruang status yang terbentuk, sedangkan eksekusi program yang menggunakan algoritma DFS masih bisa menemukan anagram dari kasus uji dengan catatan waktu 383,706262772954 detik.

V. Kesimpulan

Algoritma *Breadth-First Search* dan *Depth-First Search* merupakan algoritma pencarian solusi yang dapat menemukan anagram untuk suatu kata atau frasa. Untuk kata dengan jumlah karakter yang sedikit (di bawah 7 karakter), performa algoritma BFS dan DFS tidak jauh berbeda namun algoritma DFS lebih baik dari algoritma BFS untuk kata atau frasa dengan jumlah karakter yang lebih banyak dari 7 karakter, baik secara waktu maupun secara ruang.

Ucapan Terima Kasih

Penulis mengucapkan terimakasih kepada Tuhan YME atas bimbingannya selama pengerjaan makalah, orang tua dan teman-teman dari penulis atas segala dukungan yang telah diterima, dan kepada Dr. Ir. Rinaldi Munir, M.T., Dr. Masayu Leylia Khodra, S.T., M.T., dan Dr. Nur Ulfa Maulidevi, S.T., M.Sc selaku dosen mata kuliah IF2211 Strategi Algoritma atas segala pengajaran dan ilmu yang telah diterima penulis selama kegiatan belajar mengajar.

Referensi

[1] http://www.fun-with-words.com/anag_history.html, diakses pada 6 Mei 2018.

[2] Munir, Rinaldi, "Diktat IF2211 Strategi Algoritma", Bandung: 2018.

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, 13 Mei 2018

Michelle Eliza Gananjaya
13516015