ELINVO (Electronics, Informatics, and Vocational Education), November 2020; vol(no):141-148
ISSN 2580-6424 (printed), ISSN 2477-2399 (online,) DOI: https://doi.org/10.21831/elinvo.v5i2.28326

Traversal Struktur Data *Bipartite Graph* dalam *Graph Database* menggunakan *Depth-First Search*

Pradana Setialana[1], Muhammad Nurwidya Ardiansyah[1]
[1]Jurusan Pendidikan Teknik Elektronika dan Informatika, Universitas Negeri Yogyakarta
E-mail: pradana.setialana@uny.ac.id

ABSTRACT

Bipartite graph combined with graph database produces the right solution in storing related data, but it requires modification of the traversal algorithm to trace data on the structure. One of the algorithms that can be used to traverse the graph structure is the depth-first search (DFS). This article aims to show a modification of the DFS algorithm that can be used to traverse a bipartite graph structure in a graph database. The modification of the algorithm is then carried out by traversal testing to determine the correctness of the algorithm, complexity analysis to determine the complexity of the algorithm in asymptotic notation, time velocity test, and memory usage to determine the efficiency of the algorithm when it is run. The results of the test show that the algorithm modification can carry out traversal properly. The algorithm complexity analysis shows that the complexity of the algorithm is O(n^2). *The results of the speed and memory usage test show that the average speed in traversing is 9407.93 nanoseconds and the average memory usage is 30499929.07 bytes.*

***Keywords**: data structure, bipartite graph, graph database, graph traversal, depth-first search*

ABSTRAK

Bipartite graph yang dikombinasikan dengan *graph database* menghasilkan solusi yang tepat dalam menyimpan data berelasi tetapi diperlukan modifikasi algoritme traversal agar dapat melakukan penelusuran data pada struktur tersebut. Salah satu algoritme yang dapat digunakan untuk melakukan traversal pada struktur *graph* adalah *depth-first search* (DFS). Artikel ini bertujuan untuk menunjukan modifikasi dari algoritme DFS yang dapat digunakn untuk melakukan traversal pada struktur *bipartite graph* dalam *graph database*. Modifikasi dari algoritme tersebut kemudian dilakukan uji traversal untuk mengetahui kebenaran algoritme, analisis kompleksitas untuk mengetahui kompleksitas algoritme dalam notasi asimtotik, uji kecepatan waktu dan pengguanaan memory untuk mengetahui efisiensi algoritme ketika dijalankan. Hasil dari pengujian menunjukan bahwa modifikasi algoritme tersebut dapat melakukan traversal dengan baik dan benar. Analisis kompleksitas algoritme menunjukan kompleksitas dari algoritm tersebut adalah O(n^2). Hasil uji kecepatan dan penggunaan memory menunjukan kecepatan rata-rata dalam melakukan traversal adalah sebesar 9407,93 nanosecond dan penggunaan memory rata-rata sebesar 30499929,07 *byte*.

Kata kunci: *struktur data, bipartite graph, graph database, graph traversal, depth-first search*

PENDAHULUAN

Bipartite graph disebut juga bi*graph* yang mana dalam teori *graph* merupakan *graph* yang terdiri dari dua kelas node berbeda dimana node yang sama tidak dapat terhubung secara langsung [1] [2]. Karakteristik tersebut membuat *bipartite graph* tidak memiliki odd cycle [1]. *Bipartite graph* memiliki karakteristik yaitu terdiri dari dua jenis node yang berbeda. *Bipartite graph* dapat digunakan untuk membentuk struktur data yang berelasi seperti data hubungan keluarga atau data pohon keluarga karena memiliki karakteristik yaitu terdapat dua jenis node yang berbeda [3] [4]. Dalam data silsilah keluarga terdapat dua jenis data yaitu data keluarga dan data individu. Hal tersebut membuat *bipartite graph* cocok digunakan untuk data silsilah keluarga.

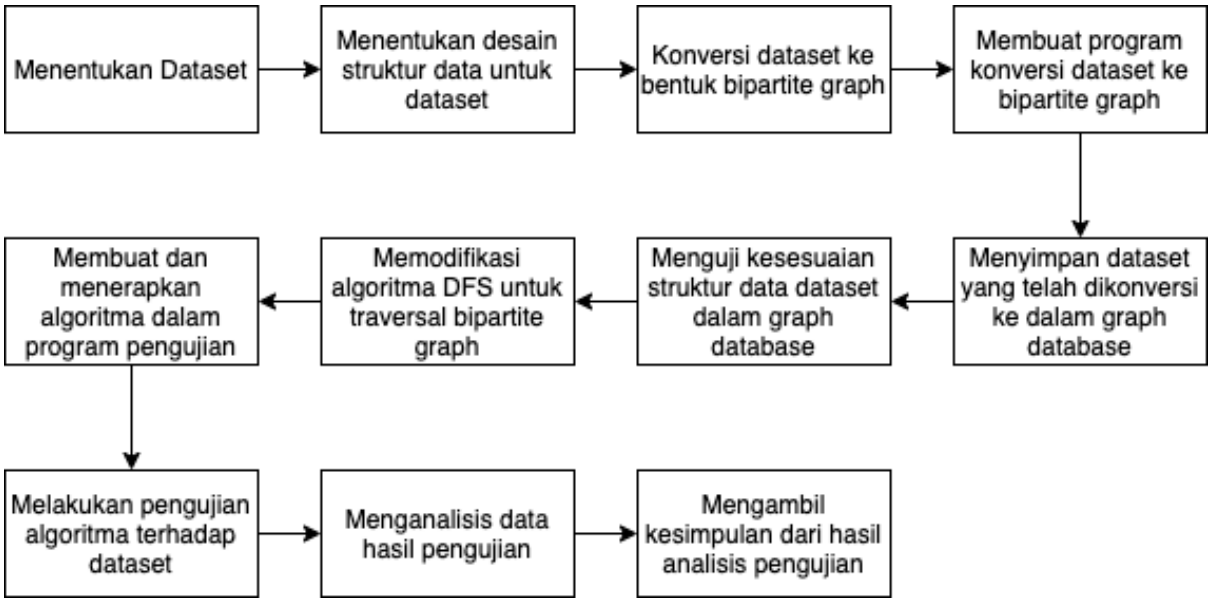
Untuk menyimpan struktur data *bipartite graph* dalam *database* dapat digunakan *graph*

database. *Graph database* adalah *database* yang menggunakan konsep model data *graph* yaitu dimana node saling terhubung dengan node yang lain [5]. *Graph database* memiliki dua komponen yaitu vertices atau node dan *relationship* atau *edges* sehingga setiap data ataupun skemanya direpresentasikan dalam bentuk *graph* [6]. *Graph database* termasuk dalam jenis NoSQL sehingga memiliki schema yang fleksibel [6]. Bentuk *graph* ini memungkinkan *graph database* dapat memodelkan data yang berelasi dalam bentuk yang alami [6]–[8]. Salah satu *graph database* management system (DBMS) yang paling populer digunakan dalam skala besar adalah Neo4J [9]. Neo4J memiliki bahasa query untuk memanipulasi data dengan nama Cyper. Selain itu, proses manipulasi data dalam Neo4J juga dapat dilakukan melalui REST API dan Java API [7].

Kombinasi dari *bipartite graph* dalam *graph database* tersebut menghasilkan solusi tepat untuk menyimpan data berelasi dengan dua jenis node yang berbeda tetapi menimbulkan permasalahan baru yaitu bagaimana pencarian atau penelusuran (traversal) terhadap data yang ada dalam struktur tersebut.

Salah satu algoritme yang dapat digunakan untuk melalukan penelurusan (traversal) terhadap data dalam struktur *graph* adalah algoritme DFS. Algoritme DFS adalah algoritme yang bekerja dengan melakukan penelurusan dimulai dari tepi simpul secara mendalam artinya bahwa DFS menelusuri simpul pertama sampai level terdalam sempai tidak ditemukan lagi simpul yang terhubung [10]. Algoritme DFS dapat digunakan untuk melakukan penelurusan pada struktur *graph* tetapi perlu dilakukan modifikasi untuk melakukan penelurusan pada stuktur *bipartite graph* dalam *graph database*.

Berdasarkan permasalahan tersebut kami memodifikasi algoritme DFS untuk digunakan dalam penelusuran (traversal) data pada struktur data *bipartite-graph* dalam *graph database*. Hasil dari modifikasi algoritme DFS tersebut kemudian dilakukan uji traversal untuk mengetahui kebenaran dari algoritme, uji kompleksitas algoritme, uji efisiensi waktu dan uji efisiensi memory untuk untuk mengetahui performa dari algoritme tersebut.



Gambar 1. Metode penelitian

METODE

Penelitian ini dilakukan dengan beberapa tahapan untuk mendapatkan simpulan akhir. Pada Gambar 1 dapat dilihat terdapat beberapa tahapan dalam penelitian ini terdiri dari menentukan dataset, mengkonversi dataset ke bentuk struktur data *bipartite graph*, menyimpan dataset dalam *bipartite graph*, memodifikasi algoritme, melakukan pengujian algoritme, dan mengambil kesimpulan dari hasil pengujian. Tahapan menentukan dataset hingga konversi dataset telah dijelaskan pada penelitian sebelumnya [4] sehingga pada penelitian ini kami menjelaskan mengenai tahapan modifikasi algoritme DFS hingga mengambil kesimpulan.

Dataset yang digunakan dalam penelitian ini adalah data silsilah keluarga Presiden Amerika Serikat. Dataset tersebut berisi lebih dari 2000 individu yang saling terhubung dengan individu yang lain dalam ikatan keluarga. Data tersebut tersimpan dalam format GEDCOM yang merupakan format standar untuk menyimpan data silsilah keluarga [4].

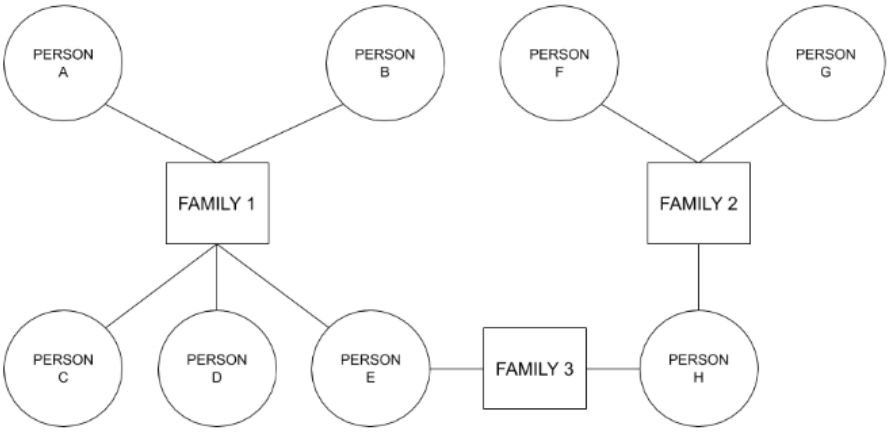
Data tersebut kemudian dibentuk dan disesuaikan sesuai dengan struktur data *bipartite graph*. Struktur data *bipartite graph* untuk data hubungan keluarga dapat dilihat pada Gambar 3. Gambar 3 menggambarkan data sililah keluarga dalam bentuk *bipartite graph* yaitu terdapat dua kelas node yaitu node individu dan node keluarga. Node individu digambarkan dalam bentuk bulat sedangkan node keluarga digambarkan dalam bentuk kotak. Dataset dalam struktur data *bipartite graph* kemudian disimpan dalam *graph database* Neo4j untuk kemudian dilakukan traversal dengan algoritme DFS yang telah dimodifikasi. Algoritme DFS adalah algoritme yang digunakan untuk penelusuran (traversal) dalam *graph*. Algoritme ini bekerja dengan cara melakukan penelusuran secara mendalam.

Untuk melakukan traversal dengan algoritme DFS pada struktur data *bipartite graph* dalam *graph database* perlu dilakukan modifikasi. Hasil dari modifikasi algoritme dalam notasi flowchart dapat dilihat pada Gambar 4. Sedangkan algoritme modifikasi DFS yang digunakan untuk melakukan traversal dalam bentuk *source code* dapat dilihat dalam pada Gambar 2.

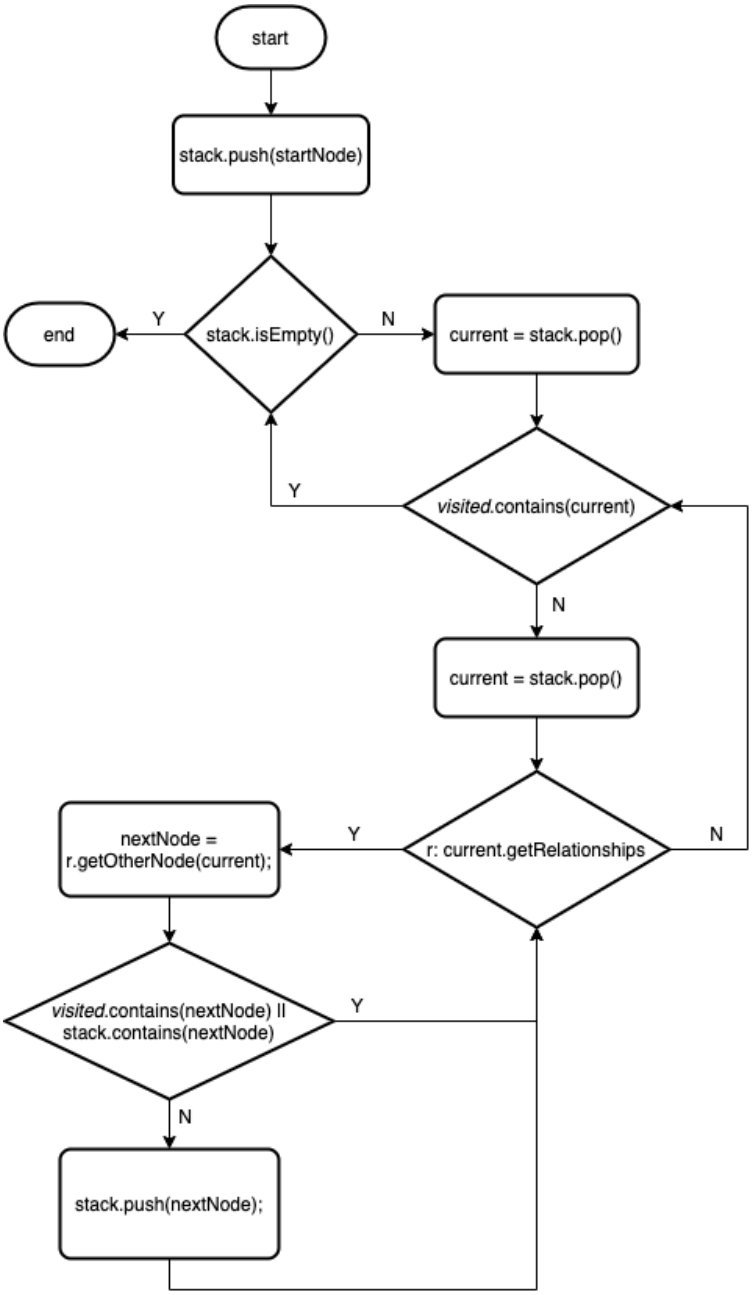
```
stack = new Stack<Node>();
visited = ArrayList<Node>();

stack.push(startNode);
while (!stack.isEmpty()) {
    current = stack.pop();
    if (!visited.contains(current)) {
        visited.add(current);
        for (Relationship r: current.getRelationships(Direction.INCOMING)) {
            Node nextNode = r.getOtherNode(current);
            if (!(visited.contains(nextNode) || stack.contains(nextNode))) {
                stack.push(nextNode);
            }
        }
        for (Relationship r: current.getRelationships(Direction.OUTGOING)) {
            Node nextNode = r.getOtherNode(current);
            if (!(visited.contains(nextNode) || stack.contains(nextNode))) {
                stack.push(nextNode);
            }
        }
    }
}
```

Gambar 2. *Source code* algoritme DFS untuk *bipartite graph* dalam *graph database*



Gambar 3. Struktur Data *Bipartite Graph*



Gambar 4. *Flowchart* algoritme DFS untuk *bipartite graph*

Pengujian traversal dan analisis notasi asimtotik tidak bergantung pada alat yang digunakan untuk pengujian. Sedangkan Efisiensi waktu dan penggunaan memori sangat bergabung pada alat pengujian. Berikut adalah spesifikasi dari alat pengujian.

Bahasa Pemrograman	: Java
IDE	: Eclipse 4.12.0
Sistem Operasi	: Mac OS 11.2.3
Processor	: 2,3 GHz Core i5
RAM	: 8 GB 2133 MHz

HASIL DAN PEMBAHASAN

Untuk menguji kebenaran dari algoritme tersebut untuk dapat menelurusi *bipartite graph* maka dilakukan uji traversal, untuk mengetahui kompleksitas *algoritme* dilakukan analisis dalam notasi asimtotik. Sedangkan untuk mengetahui efisiensi dari algoritme maka dilakukan pengujian pada kecepatan waktu dan penggunaan memory algoritme DFS yang telah dimodifisikasi dalam melakukan penelurusan.

Algoritme yang telah diimplementasikan dalam program kemudian diuji untuk melakukan traversal (penelurusan) terhadap data yang ada pada satu garis ikatan keluarga besar yaitu keluarga "George Washington /CASSIDY/". Hasil dari pengujian tersebut adalah dapat dilihat pada Gambar 5.

Hasil tersebut menunjukan bahwa algoritme yang telah dibuat dan diterapkan dapat menelusuri keluarga "George Washington /CASSIDY/" dan menampilkan setiap node yang dilalui dalam penelurusan tersebut. Dalam Gambar 5 tersebut terlihat bahwa *algoritme* selain melalui node individu juga melalui node keluarga yang ditandai dengan simbol F. Untuk mengetahui seluruh node dilalui algoritme tersebut maka dilakukan perbandingan yang dapat dilihat pada Tabel 1. Berdasarkan hasil pengujian terlihat bahwa algoritme berhasil melakukan penelusuran terhadap semua node dalam data keluarga "George Washington /CASSIDY/" yaitu sebanyak total 101 node. Dari penelusuran tersebut juga diketahui bahwa keluarga "George Washington /CASSIDY/" memiliki 35 keluarga dengan 66 individu.

Tabel 1. Detail hasil pengujian traversal

No	Parameter	Keterangan
1	Node pertama	George Washington /CASSIDY/
2	Node terakhir	Rachel
3	Jumlah node dalam studi kasus	101
4	Jumlah node yang dikunjungi	101
5	Jumlah node individu	66
6	Jumlah node keluarga	35

Selanjutnya kompleksitas algoritme *Depth-first search* (DFS) dihitung dengan menghitung jumlah langkah pada setiap baris *pseudocode*. Hasil dari perhitungan tersebut berbentuk notasi Big-O.

```
Neo4j startup succeeded
George Washington /CASSIDY/ --> F18 -->
James M. /CASSIDY/ --> F12 --> James
Eldridge /CASSIDY/ --> F9 --> Edith Vallie
(Valeria)/GRISHAM/ --> F13 --> Edna Earl
/ADAMS/ --> F21 --> Martin Franklin /ADAMS/
--> F34 --> Clarinda  /WILSON/ --> Martin
/ADAMS/ --> Sarah Jane /MAY/ --> F35 -->
Clara  /MCBRIDE/ --> Benjamin  /MAY/ -->
Lemma Newell /GRISHAM/ --> F20 --> Hattie R.
/MITCHELL/ --> F33 --> ?  /MITCHELL/ -->
Martha  // --> James B. /GRISHAM/ --> F32 --
> Moses  /GRISHAM\GRISSOM/ --> Betsey
/SLATE/ --> Virginia Dell /CASSIDY/ --> F2 -
-> William Jefferson /BLYTHE/ --> F4 -->
William Jefferson /BLYTHE/ --> F10 --> Henry
Patton Foote/BLYTHE/ --> F14 --> Esther
Elvira /BAUM/ --> F24 --> Mary  // --> Moses
/BAUM/ --> Thomas Jefferson /BLYTHE/ --> F22
--> Elizabeth Melvinia /HINES/ --> F23 -->
Andrew J. /BLYTHE/ --> Jane  // --> Frances
Ellen "Fannie"/HINES/ --> F15 --> John F.
/HINES/ --> F25 --> Mary (Or Polly)// -->
Elias  /HINES/ --> Eliza Emily /LOCKHART/ --
> F26 --> ?  /LOCKHART/ --> Martha (Or
Mattie)// --> Lou Birchie /AYERS/ --> F11 --
> Hattie  /HAYES/ --> F17 --> ?  /HAYES/ -->
?  // --> Robert E. /AYERS/ --> Simpson
Green "Dick"/AYERS/ --> F16 --> ?  // --> ?
/AYERS/ --> F27 --> ?  /AYERS/ --> William
Jefferson /CLINTON/ --> F1 --> Chelsea
/CLINTON/ --> Hillary  /RODHAM/ --> F3 -->
Dorothy  /HOWELL/ --> Hugh Ellsworth
/RODHAM/ --> F7 --> Jeff  /DWIRE/ --> F8 -->
Dick  /KELLEY/ --> F5 --> Roger  /CLINTON/ -
-> F6 --> ?  // --> Sarah Louisa /RUSSELL/ -
-> F19 --> William James /RUSSELL/ --> F30 -
-> Nancy  // --> Simon  /RUSSELL/ --> Mary
E. /SPRADLEY/ --> F31 --> Warren C.
/SPRADLEY/ --> Elsey\Alcey  /MALONE/ -->
Nancy Ann Josephine/SNELGROVE/ --> F29 -->
Elizabeth  /HOWARD/ --> Jesse  /SNELGROVE/ -
-> F28 --> Levi  /CASSIDY/ --> Rachel  //
Jumlah Node: 101
Used memory is bytes: 30463416
Waktu Eksekusi: 19078492
```

Gambar 5. Hasil traversal pada data keluarga "George Washington /CASSIDY/"

Perhitungan kompleksitas algoritme dilakukan terhadap *pseudocode* algoritme DFS untuk *bipartite graph* dalam *graph database* yang hasilnya dapat dilihat dalam Tabel 2. Berdasarkan hasil tersebut dapat diketahui kompleksitas algoritme DFS untuk *bipartite graph* dalam *graph database* adalah O(n^2).

Tabel 2. Kompleksitas Algoritme

Baris	Frekuensi	Total
5	n	n
6	1	n+1
...	...	...
9	n	(n + 1 + ...) * n
...	...	...
15	n	(n + 1 + ...) * (n + ... + n)
...	...	...
22	1	(n+1+...)*(2n+1+...) ≈ n^2

Pengujian kedua kemudian dilakukan untuk mengetahui kecepatan algoritme tersebut dalam melakukan traversal pada keluarga "George Washington /CASSIDY/". Pada pengujian kedua ini dilakukan sebanyak 30 pengujian. Hasil pengujian dapat dilihat pada **Error! Not a valid bookmark self-reference.** dan Tabel 4 yang menunjukan kecepatan algoritme pada setiap percobaan dalam melakukan penelusuran pada keluarga "George Washington /CASSIDY/". Waktu yang diambil menggunakan satuan nanosecond yaitu 10^{-9} detik. Dari 30 percobaan tersebut dapat diambil rata-rata kecepatan yang dapat dilihat pada Tabel 5.

Tabel 3. Pengujian kecepatan algoritme tahap I

Percobaan	Waktu (ns)
1	8974
2	8092
3	6781
4	7174
5	8760
6	7041
7	9112
8	8783
9	8557
10	6886
11	9859
12	6577
13	8234
14	7946
15	7571

Tabel 4. Pengujian kecepatan algoritme tahap II

Percobaan	Waktu (ns)
16	7029
17	8652
18	7421
19	13658
20	18332
21	42831
22	7858
23	5301
24	6424
25	7374
26	7302
27	9824
28	6851
29	5150
30	7884

Berdasarkan Tabel 5 dapat diketahui bahwa waktu rata-rata yang dibutuhkan algoritme tersebut untuk menelusuri data keluarga "George Washington /CASSIDY/" dalam struktur *bipartite graph* dalam *graph database* adalah 9407,93 nanosecond.

Tabel 5. Rata – rata kecepatan traversal

Keterangan	Waktu (ns)
Waktu Maksimal	42831
Waktu Minimal	5150
Rata - Rata	9407,93

Pengujian efisiensi memori digunakan untuk mengetahui besar penggunaan memori yang digunakan ketika *algoritme* tersebut berjalan. Pengujian dilakukan ketika algoritme tersebut melakukan traversal pada keluarga "George Washington /CASSIDY/" sebanyak 30 kali. Hasil dari pengujian tersebut dapat dilihat pada Tabel 6.

Tabel 6. Pengujian penggunaan memori

Percobaan	Penggunaan Memori (*byte*)
1	30531424
2	30466800
3	30466536
4	30536216
5	30464792
6	30465952
7	30535656
8	30526184
9	30531144
10	30466528
11	30536024
12	30468432
13	30527304
14	30466320
15	30534256
16	30464920
17	30469200
18	30537480
19	30463416
20	30537688
21	30534424
22	30468056
23	30533832
24	30465840
25	30535776
26	30466192
27	30531000
28	30534872
29	30465560
30	30466048

Hasil pengujian efisiensi memori pada tabel tersebut menunjukan besar memori yang dibutuhkan *algoritme* dalam melakukan traversal pada keluarga "George Washington /CASSIDY/" pada setiap percobaannya. Setelah dilakukan 30 kali percobaan, didapatkan rata-rata pengguaan memori yang dapat dilihat pada Tabel 7.

Dari Tabel 7 dapat dilihat penggunaan memori minimal adalah sebesar 30463416 *byte*, penggunaan memori maksimal sebesar 30537688 *byte* dan rata-rata penggunaan memori adalah sebesar 30499929,07 *byte*.

Tabel 7. Rata-rata penggunaan memori

Keterangan	Memory (*byte*)
Penggunaan memory minimal	30463416
Penggunaan memory maksimal	30537688
Rata-rata penggunaan memory	30499929,07

SIMPULAN

Algoritme DFS yang telah dimodifikasi dan telah diimplementasikan ke dalam program dapat menyelesaikan permasalahan penelusuran (traversal) data pada struktur data *bipartite graph* dalam *graph database*. Program dengan algoritme tersebut melakukan penelusuran (traversal) terhadap data yang ada pada satu garis ikatan keluarga besar yaitu keluarga "George Washington /CASSIDY/" dan menampilkan setiap node yang dilalui dalam penelurusan tersebut.

Hasil pengujian traversal dari algoritme tersebut diketahui bahwa jumlah node keseluruhan adalah 101, jumlah node yang dikunjungi adalah 101, jumlah node individu adalah 66 dan jumlah node keluarga adalah 35. Hasil tersebut menunjukan bahwa kebenaran algoritme tersebut dalam melakukan traversal. Sedangkan hasil uji kompleksitas dari algoritme dapat diketahui bahwa kompleksitas pada algoritme DFS yang dimodifikasi adalah O(n^2).

Hasil uji efisiensi waktu dan efisiensi memory pada algoritme DFS yang telah dimodifikasi dilakukan sebanyak 30 kali percobaan. Dari 30 kali percobaan tersebut dapat diketahui bahwa waktu maksimal untuk melakukan penelurusan dalam *graph* adalah sebesar 42831 nanosecond, waktu minimal sebesar 5150 nanosecond dan rata-rata waktu yang diperlukan adalah sebesar 9407,93 nanosecond. Sedangkan pada penggunaan memory, memory yang diperlukan minimal 30463416 *byte*, penggunaan memory maksimal sebesar 30537688 *byte* dan rata-rata penggunaan memory adalah sebesar 30499929,07 *byte*.

DAFTAR PUSTAKA

[1] R. Diestel, *Graph Theory*, vol. 3, no. 1. Springer, 2016.

[2] L. Lan, S. Ju, and H. Jin, "Anonymizing social network using *bipartite graph*," *Proc. - 2010 Int. Conf. Comput. Inf. Sci. ICCIS 2010*, pp. 993–996, 2010, doi: 10.1109/ICCIS.2010.245.

[3] A. Bezerianos, P. Dragicevic, J. D. Fekete, J. Bae, and B. Watson, "GeneaQuilts: A system for exploring large genealogies," *IEEE Trans. Vis. Comput. Graph.*, vol. 16, no. 6, pp. 1073–1081, 2010, doi: 10.1109/TVCG.2010.159.

[4] P. Setialana, T. B. Adji, and I. Ardiyanto, "Analisis *Bipartite Graph* , Ore-*Graph* dan P-*Graph* untuk Struktur Data Genealogy dalam *Graph Database*," in *The 9th National Conference on Information Technology and Electrical Engineering*, 2017, pp. 173–180.

[5] Z. J. Zhang, "*Graph Database*s for Knowledge Management," *IT Prof.*, vol. 19, no. 6, pp. 26–32, 2017, doi: 10.1109/MITP.2017.4241463.

[6] R. Angles and C. Gutierrez, "Survey of *graph database* models," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 1–39, 2008, doi: 10.1145/1322432.1322433.

[7] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*. Sebastopol: O'Reilly Media, Inc, 2015.

[8] D. Shimpi, "An overview of *Graph Database*s," *IJCA Proc. Int. Conf. Recent Trends Inf. Technol. Comput. Sci. 2012*, vol. ICRTITCS, no. 3, pp. 16–22, 2013.

[9] R. Kumar Kaliyar, "*Graph database*s: A survey," *Int. Conf. Comput. Commun. Autom. ICCCA 2015*, pp. 785–790, 2015, doi: 10.1109/CCAA.2015.7148480.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2009.