ALGORITMA PRIM DAN KRUSKAL DALAM MENCARI MINIMUM SPANNING TREE PADA BAHASA PEMROGRAMAN C

Hendarman Lubis[1], Dwi Budi Srisulistiowati[2]
[1,2] Fakultas Ilmu Komputer, Program Studi Informatika, Universitas Bhayangkara
[1]hendarman.lubis@dsn.ubharajaya.ac.id, [2]dwibudi@dsn.ubharajaya.ac.id

Abstrak

Algoritma prim dan kruskal merupakan kedua jenis algoritma yang dapat digunakan untuk mencari minimum spanning tree (MST) pada sebuah graf. Dalam pencarian MST di sebuah graf, algoritma prim berorientasi pada titik atau vertex graf, sedangkan algoritma kruskal berorientasi pada bobot (weight) sisi graf. Walaupun perbedaan orientasi namun kedua algoritma tersebut mampu memberikan solusi yang sama. Algoritma prim mempunyai kompleksitas waktu (worst case) O(E Log V), sedangkan algoritma kruskal O(E Log E) dan O(E Log V). Kompleksitas waktu tersebut sangat berpengaruh pada kecepatan waktu eksekusi atau running time dalam menjalankan algoritma proses pencarian MST, dimana algoritma prim akan mempunyai running time tercepat ketika kompleksitas graf rumit sedangkan algoritma kruskal akan lebih cepat jika kompleksitas graf sederhana. Hal tersebut juga sudah terbukti ketika diimplementasikan menggunakan bahasa pemrograman C, sehingga keefisienan waktu masing-masing algoritma dapat ditentukan berdasarkan tingkat kompleksitas graf yang diberikan.

Kata Kunci: Algoritma Kruskal, Algoritma Prim, Graf, Minimum Spanning Tree, Running Time.

Abstract

Prim and kruskal algorithms are both types of algorithms that can be used to find the minimum spanning tree (MST) on a graph. In searching MST on a graph, the prim algorithm is oriented to the point or vertex of the graph, while the Kruskal algorithm is oriented to the side weight of the graph. Despite the difference in orientation, the two algorithms are able to provide the same solution. The prim algorithm has a worst case time complexity of O (E Log V), while the Kruskal algorithm is O (E Log E) and O (E Log V). The time complexity greatly affects the speed of execution time or running time in running the MST search process algorithm, where the prim algorithm will have the fastest running time when the complexity of the graph is complicated, while the Kruskal algorithm will be faster if the complexity of the graph is simple. This has also been proven when implemented using the C programming language, so that the time efficiency of each algorithm can be determined based on the level of complexity of the given graph.

Keywords: *Kruskal's Algorithm, Prim's Algorithm, Graph, Minimum Spanning Tree, Running Time.*

1. PENDAHULUAN

Sejarah teori graph bermula saat ahli matematika Swiss Leonhard Euler memecahkan masalah jembatan Königsberg Euler berpendapat bahwa tidak ada jalan semacam itu. Buktinya hanya mengacu pada susunan fisik jembatan, namun intinya dia membuktikan teorema pertama dalam teori graph. [7]

Teori Graf merupakan salah satu metode yang digunakan untuk menyelesaikan permasalahan diskrit yang ada, misalnya merepresentasikan objek–objek diskrit dan hubungan antara objek–objek tersebut [1]. Contoh penggunaan graf yang paling umum adalah mencari suatu rute tujuan pada peta. Dengan menggunakan teori graf ini akan didapatkan hasil dengan keuntungan tertentu, misalnya rute dengan waktu tempuh yang tercepat, rute dengan jarak terpendek,

rute dengan biaya yang paling murah, rute dengan tingkat efisiensi yang paling tinggi, dan lain sebagainya.

Konsep pohon (*tree*) adalah konsep yang paling terkenal dan cukup penting dibandingkan konsep lainnya yang ada dalam teori graf, karena mampu menyelesaikan permasalahan dalam berbagai terapan masalah graf, misalnya konsep pohon merentang (*Spanning Tree*) yang digunakan oleh perusahaan kereta api dalam menentukan jalur yang akan dilalui oleh kereta, dengan mempertimbangkan bobot biaya yang minimum. Jika konsep spanning tree digunakan untuk menyelesaikan suatu permasalahan dengan unsur pencarian bobot minimum pada sebuah graf berbobot, maka akan disebut sebagai MST (*Minimum Spanning Tree*) [2] [8] [12]

Terdapat 2 jenis algoritma yang dapat digunakan untuk menyelesaikan permasalahan *Minimum Spanning Tree* (MST) yaitu algoritma prim dan kruskal [3]. Pada paper ini akan berfokus pada 2 jenis algoritma tersebut. Tujuannya adalah untuk menjelaskan secara rinci cara kerjanya kemudian membandingkan *running tim*e, CPU *consumption*, dan *memory consumption* dari kedua algoritma tersebut, serta menerapkannya pada bahasa pemrograman C.

Sesi 2 akan menjelaskan dasar teori dari algoritma prim dan kruskal. Sesi 3 akan menjelaskan eksperimen dan analisis dari algoritma prim dan kruskal. Sesi 4 akan berisikan sebuah kesimpulan akhir dari semua penjelasan dan eksperimen yang ada.

2. METODE PENELITIAN

2.1. Extreme Programming (XP)

Extreme programming (XP) merupakan salah satu cabang dari metode Agile sebagai pengembangan perangkat lunak yang digunakan untuk menyesuaikan kebutuhan pengembangan. XP digunakan dalam merancang bangun aplikasi untuk memprediksi kelulusan (Rusdiana & Marfuah, 2017).

Extreme Programming (XP) memiliki tahapan-tahapan sebagai berikut (Schach, 2011):

a. Perencanaan (*Planning*)
Perencanaan merupakan tahapan awal untuk memulai penelitian dengan mendefinisikan kebutuhan yang diperlukan, *output* yang akan dihasilkan, layanan yang akan dikembangkan pada aplikasi, dan fitur serta fungsional dari aplikasi yang akan dikembangkan.

b. Perancangan (*Design*)
Tahapan ini merupakan bagian dari perancangan aplikasi yang sesuai dengan kebutuhan dari penggunaannya.

c. Pengkodean (*Coding*)
Tahapan pengkodean merupakan tahapan dalam menyiapkan kode pada *software* yang dapat digunakan dalam pengembangan aplikasi sehingga dapat menjadi pemecahan masalah.

d. Pengujian (*Testing*)
Tahapan pengujian merupakan tahapan terakhir untuk menguji layanan atau fitur dan fungsionalitas yang terdapat pada aplikasi yang dibangun. Sehingga dapat diambil kesimpulan dari pengujian yang dilakukan.

Pengujian dilakukan sebagai bagian dari fundamental untuk menyelesaikan permasalahan yang ditangani, sehingga dapat digunakan untuk mengetahui kekuatan dan kelemahan sistem serta dapat memproyeksi kerangka kerja XP yang baru (Hameed, 2016).

2.2. Langkah-langkah XP

2.2.1. Perencanaan (Planning)

Tahapan ini dengan mencari Minimum Spanning Tree pada Algoritma Prim dan Kruskal.

2.2.2. Perancangan (*Design*)

Tahapan ini merancang tampilan menggunakan *software* yang dimplementasikan melalui Bahasa Pemrograman C.

2.2.3. Pengkodean (*Coding*)

Tahapan pengkodean menggunakan Algoritma.

2.2.3.1 Cara Kerja Algoritma

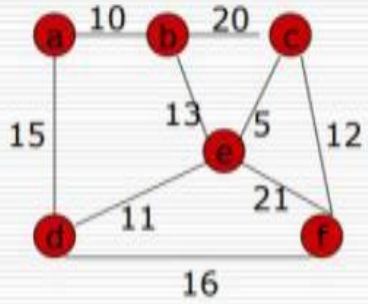
2.2.3.1.1. Algoritma Prim

Algoritma prim adalah suatu algoritma yang termasuk dalam suatu teori graf, dapat digunakan untuk mendapatkan hasil *minimum spanning tree* dari graf berbobot kemudian menghubungkannya, dengan orientasi titik graf [4].

Ada 3 tahapan cara kerja untuk mendapatkan *minimum spanning tree* dari sebuah graf berbobot dengan menggunakan algoritma prim yaitu [4] [11] :

1. Menentukan atau memilih sisi (*edge*) pada graf (G) dengan bobot yang paling minimum, kemudian masukkan kedalam himpunan T (*set of trees*).
2. Kemudian pilih sisi (*edge*) (u,v) yang mempunyai bobot paling minimum lagi tetapi bersisian dengan simpul di T, tetapi (u,v) tidak membentuk sirkuit (*cycle*) di T. Tambahkan sisi (u,v) kedalam T.
3. Ulangi langkah ke 2 sebanyak (n-2) kali atau sampai tidak ada lagi sisi (u,v) yang dapat dimasukkan lagi ke T dengan memperhatikan ketentuan pada langkah 2.

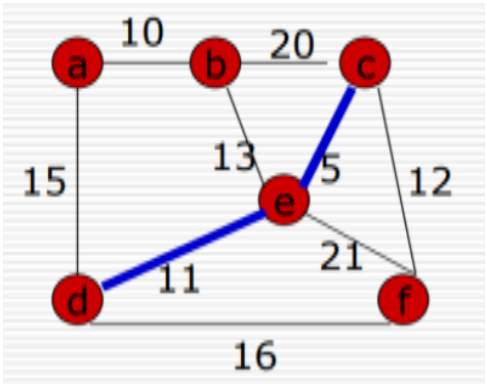
Secara sederhananya algoritma prim terlebih dahulu akan memilih suatu sisi pada sebuah graf berbobot dengan bobot yang paling kecil atau minimum, kemudian sisi pertama tersebut akan dimasukkan kedalam sebuah himpunan *tree* (T), selanjutnya akan dilihat sisi lainnya yang bertetanggaan dengan sisi awal tersebut dan memilih lagi sisi dengan bobot yang paling minimum tetapi tidak membuat sirkuit dalam T, lalu mengulangi langkah yang sama sampai semua sisi pada graf tidak ada lagi yang dapat dipilih dengan mempertimbangkan ketentuan diatas. Untuk ilustasi lebih jelasnya dapat dilihat pada gambar dibawah ini.



Langkah	Sisi	bobot
0		
1	e-c	5
2	d-e	11
3	c-f	12
4	b-e	13
5	a-b	10

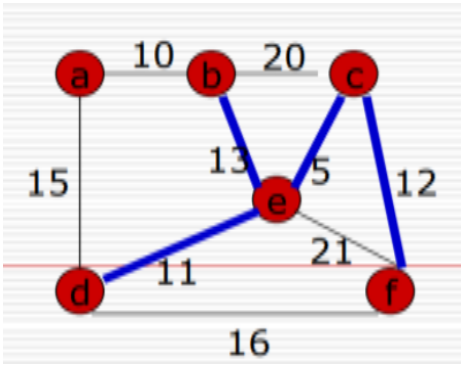
Gambar 1. Graf Berbobot

Dari gambar diatas dapat dilihat bahwa sisi pada graf dengan bobot yang paling minimum adalah sisi e-c dengan bobot 5. Sehingga sisi tersebut akan menjadi sisi awalnya.



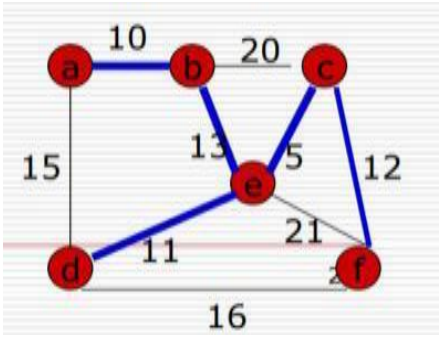
Gambar 2. Langkah 1 Algoritma Prim

Selanjutnya kita memilih sisi yang bertetanggaan dengan sisi awal (e-c) tetapi dengan bobot yang paling minimum juga. Maka akan dipilih sisi (e-d) dengan bobot 11.

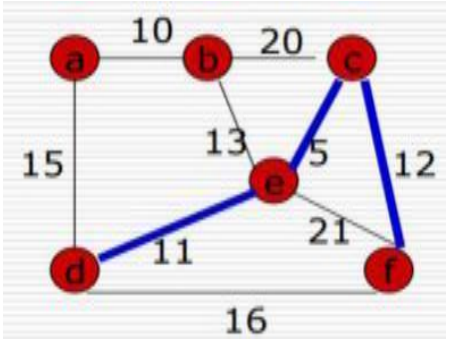


Gambar 3. Langkah 2 Algoritma Prim

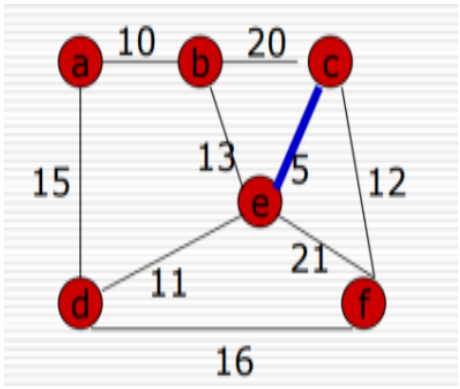
Dengan mengulangi langkah yang sama seperti sebelumnya maka akan didapatkan solusi seperti dibawah ini.



Gambar 4. Langkah 3 Algoritma Prim



Gambar 5. Langkah 4 Algoritma Prim



Gambar 6. Langkah 5 (Solusi Akhir) Algoritma Prim

Dari semua gambar diatas dapat dilihat bahwa solusi akhir nya yang ditandai dengan garis berwarna biru pada graf dan total bobot *minimum spanning tree* nya adalah 5 + 11 + 12 + 13 + 10 = 51.

Pseudocode dari Algoritma Prim adalah sebagai berikut [3] :

Algorithm 1 : Prim's Algorithm

MST-Prim (G, w, r):

```
1:                                   for each u ∈ V[G]
2:                                       do key[u]  ∞
3:        π[u]  NIL
4:    key[r]  0
5:    Q  V[G]
6:    while Q ≠ Ø
7:          do u  Extract-MIN(Q)
8:              for each v ∈ Adj [u]
9:                  do if v ∈ Q and w(u,v) < key[v]
10:                      then π[v]  u
11:                           key[v]  w(u,v)
```

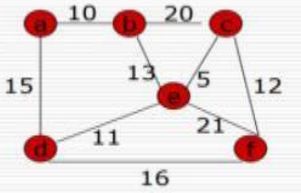
2.2.3.1.2. Algoritma Kruskal

Algoritma kruskal adalah salah satu algoritma yang termasuk dalam suatu teori graf, dapat digunakan untuk mendapatkan *minimum spanning tree* dari graf berbobot dengan orientasi bobot sisi graf [4].

Ada 3 tahapan cara kerja untuk mendapatkan *minimum spanning tree* dari sebuah graf berbobot dengan menggunakan algoritma kruskal yaitu [4] [11]:

1. Mengurutkan setiap sisi (*edge*) pada graf (G) dari bobot yang paling terkecil hingga terbesar.
2. Kemudian pilih sisi (*edge*) (u,v) yang mempunyai bobot paling minimum dan masukkan kedalam himpunan T (*set of trees*).
3. Pilih sisi (u,v) lagi dengan bobot minimum berikutnya, dengan ketentuan tidak membentuk sirkuit (*cycle*) di T, lalu tambahkan sisi (u,v) tersebut kedalam T.
4. Ulangi langkah 3 sampai tidak ada lagi sisi yang dapat dimasukkan kedalam T dengan ketentuan yang sama.

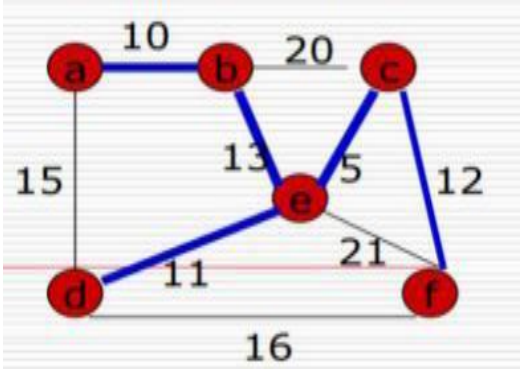
Secara sederhananya algoritma kruskal terlebih dahulu akan mengurutkan setiap bobot sisi pada graf dari yang paling minimum ke paling maksimumnya. Kemudian memilih sisi yang paling minimum dan memasukkan sisi tersebut kedalam himpunan *tree* (T). Selanjutnya memilih sisi dengan bobot minimum berikutnya dan memasukkan lagi ke T tetapi dengan ketentuan tidak membentuk sirkuit (*cycle*) di T. Mengulangi langkah yang sama sampai semua sisi pada graf tidak ada lagi yang dapat dipilih dengan mempertimbangkan ketentuan yang sama. Untuk ilustasi lebih jelasnya dapat dilihat pada gambar dibawah ini



Langkah	Sisi	bobot
0		
1	e-c	5
2	a-b	10
3	d-e	11
4	c-f	12
5	b-e	13

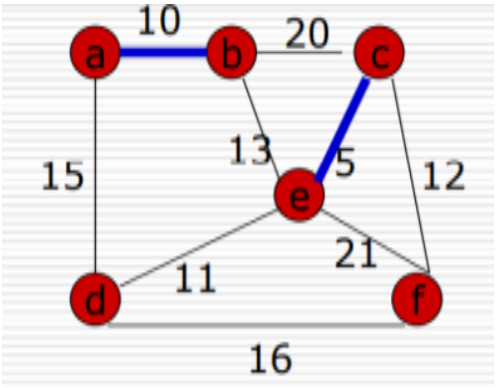
Gambar 7. Graf Berbobot dan Tabel Urutan Sisi Graf

Dari gambar diatas dapat dilihat pada graf bahwa sisi dengan bobot yang paling minimum adalah sisi e-c dengan bobot 5. Dan disebelah kanan gambar terdapat urutan bobot sisi graf yang digunakan sebagai solusi.



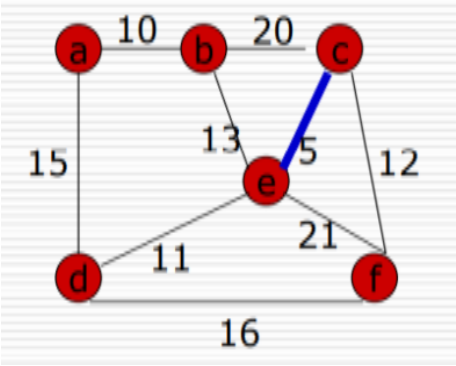
Gambar 8. Langkah 1 Algoritma Kruskal

Selanjutnya memilih bobot minimum berikutnya yang tersedia dan tidak membuat sirkuit (*cycle*).

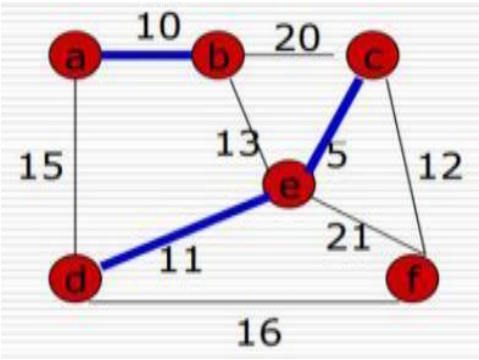


Gambar 9. Langkah 2 Algoritma Kruskal

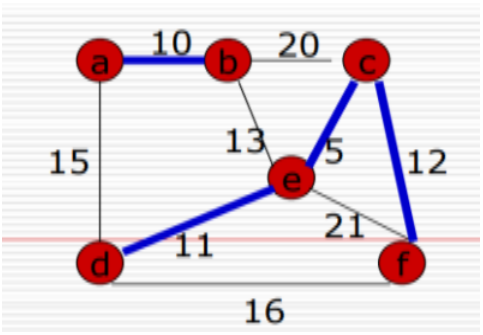
Dengan mengikuti langkah sebelumnya tetapi tetap memperhatikan ketentuan yang ada, maka solusinya sebagai berikut.



Gambar 10. Langkah 3 Algoritma Kruskal



Gambar 11. Langkah 4 Algoritma Kruskal



Gambar 12. Langkah 5 (Solusi Akhir) Algoritma Kruskal

Dari semua gambar diatas dapat dilihat bahwa solusi akhir nya yang ditandai dengan garis berwarna biru pada graf dan total bobot *minimum spanning tree* nya adalah 5 + 10 + 11 + 12 + 13 = 51.

Dengan menggunakan graf berbobot yang sama, algoritma prim dan kruskal mampu memberikan solusi *minimum spanning tree* yang sama atau benar, walaupun dalam tahapan kerjanya berbeda. Hal ini membuktikan bahwa keduanya memang benar-benar mampu dalam menyelesaikan permasalahan *minimum spanning tree* dari sebuah graf berbobot.

Pseudocode dari Algoritma Kruskal adalah sebagai berikut [3]

Algorithm 2 : Kruskal's Algorithm

MST-Kruskal (G, w):

1: A □ Ø

2: **for** each vertex v ∈ V[G] 3: **do** MAKESET(v)

4: sort the edges of E into nondecreasing order by weight w

5: **for** each edge (u,v) E, taken in nondecreasing order by weight

6: **do if** FIND-SET(u) ≠ FIND-SET(v)

7: **then** A □ A ∪ {(u,v)} 8: UNION(u,v)

9: **return** A

2.2.3.1.3. Kompleksitas Waktu

Time Complexity Algoritma Prim dan Kruskal [5]

Tabel 1. Perbandingan *Time Complexity* dari kedua Algoritma

Algoritma	*Time Complexity*
Prim	O(E log V) untuk *Binary Heap* atau O(E + V log V) untuk *Fibonacci Heap*
Kruskal	O(E Log E) atau O(E Log V)

Keterangan : E = *edge* ; V = *vertex*.

Pada eksperimen yang akan dilakukan, *worst case* yang digunakan sebagai *time complexity* algoritma prim adalah *binary heap* karena *data structure* nya mengambil bentuk *binary tree*. Sedangkan pada algoritma kruskal, *time complexity* untuk mengurutkan sisi graf pertama kali menggunakan O(E Log E), kemudian untuk melakukan verifikasi apakah sisi tersebut aman atau tidak baru menggunakan O(E Log V). Maksud kata aman yaitu jika tidak membuat sirkuit (*cycle*) pada himpunan *tree* (T).

3. HASIL DAN PEMBAHASAN

Implementasi algoritma prim dan kruskal ini dilakukan dengan menggunakan bahasa pemrograman C. Output yang dihasilkan berupa *running time* dalam menjalankan *source code* nya beserta dengan CPU dan *memory consumption* nya.

3.1. Informasi Perangkat

Eksperimen dilakukan dengan menggunakan aplikasi *compiler* Blood Dev-C++ versi 4.9.9.2 pada laptop dengan spesifikasinya sebagai berikut.

- Windows 10, 64-bit.
- Intel Core i5-8300H CPU @ 2,3 Ghz

(Cache 8M, up to 4Ghz).

- Display : Intel® UHD Graphics 630.
- Render : NVIDIA Geforce GTX 1050 4 GB.
- 8 GB DDR4 RAM.
- 1 TB HDD.

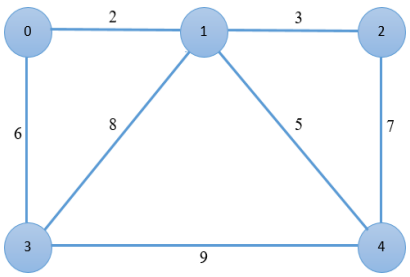
3.2. Tabel

Eksperimen ini terdiri dari 3 kali percobaan dengan bentuk graf dan jumlah *vertices* yang berbeda – beda. Output dari eksperimen ini berupa hasil *minimum spanning tree* pada graf, selain itu juga terdapat perbandingan *running time*, CPU *consumption*, dan *memory consumption* dari kedua jenis algoritma. Setiap data input yang akan dimasukkan kedalam program dapat dilihat pada tabel dibawah ini yang merupakan bentuk representasi dari graf berbobot dalam bentuk matriks *adjacency*.

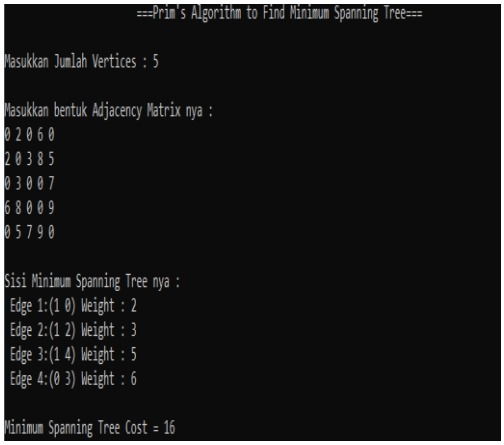
Eksperimen 1

Tabel.2 Data Input Program dalam Bentuk Matriks Adjacency untuk Eksperimen 1

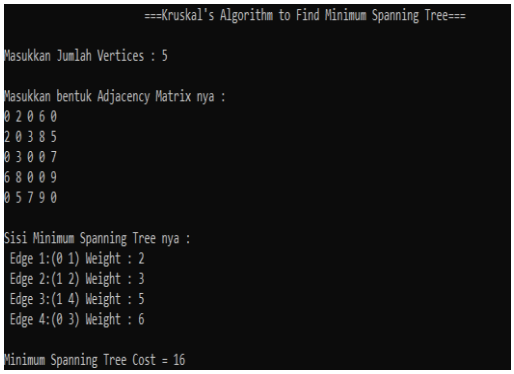
	0	1	2	3	4
0	0	2	0	6	0
1	2	0	3	8	5
2	0	3	0	0	7
3	6	8	0	0	9
4	0	5	7	9	0



Gambar 13. Graf Berbobot Eksperimen 1

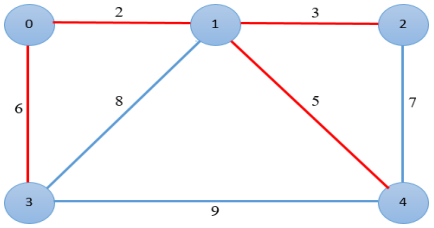


Gambar 14. Output Program (Algoritma Prim) Untuk *Minimum Spanning Tree* Eksperimen 1



Gambar 15. Output Program (Algoritma Kruskal) untuk *Minimum Spanning Tree* Eksperimen 1

Dibawah ini adalah gambar untuk hasil *minimum spanning tree* pada graf berdasarkan output dari kedua jenis program. *Minimum spanning tree* pada graf diberi tanda dengan garis berwarna merah.

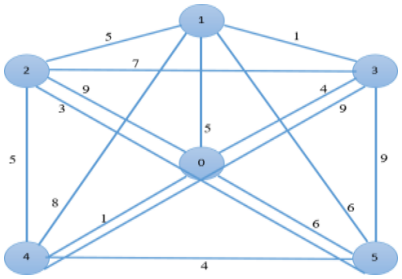


Gambar 16. *Minimum Spanning Tree* Eksperimen 1

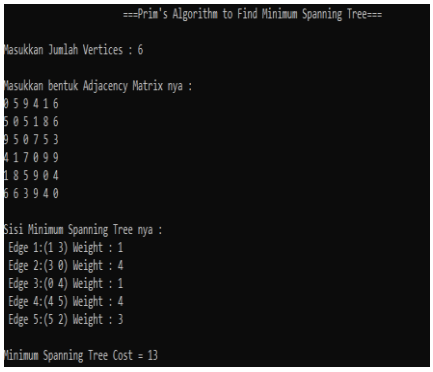
Eksperimen 2

Tabel 3. Data Input Program dlam bentuk Matriks Adjacency untuk Eksperimen 2

	0	**1**	**2**	**3**	**4**	**5**
0	0	5	9	4	1	6
1	5	0	5	1	8	6
2	9	5	0	7	5	3
3	4	1	7	0	9	9
4	1	8	5	9	0	4
5	6	6	3	9	4	0



Gambar 17. Graf Berbobot Eksperimen 2

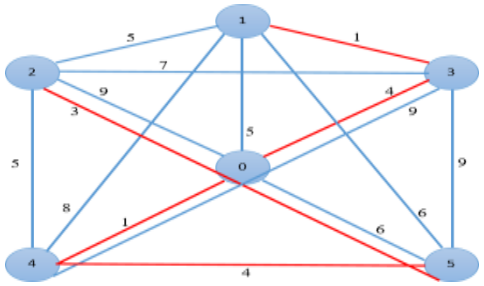


Gambar 18. Output Program (Algoritma Prim) untuk Eksperimen 2

Gambar 19. Output Program (Algoritma Kruskal) untuk Minimum Spanning Tree

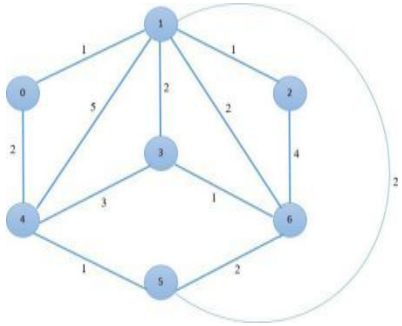
***Minimum Spanning Tree* Eksperimen 2**

Dibawah ini adalah gambar untuk hasil *minimum spanning tree* pada graf berdasarkan output dari kedua jenis program. *Minimum spanning tree* pada graf diberi tanda dengan garis berwarna merah.



Gambar 20. *Minimum Spanning Tree* Eksperimen 2

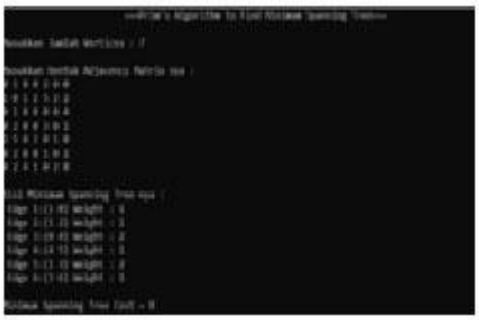
Eksperimen 3



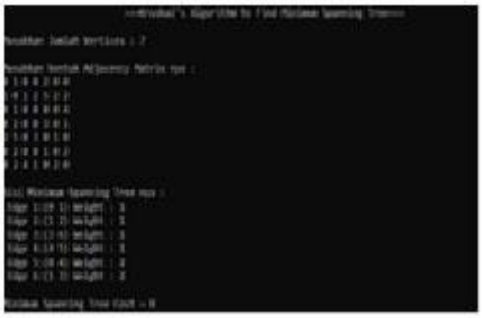
Gambar 21. Graf Berbobot Eksperimen 3

Tabel 4. Data Input Program dalam bentuk Matriks Adjacency untuk Eksperimen 3

	0	**1**	**2**	**3**	**4**	**5**	**6**
0	0	1	0	0	2	0	0
1	1	0	1	2	5	2	2
2	0	1	0	0	0	0	4
3	0	2	0	0	3	0	1
4	2	5	0	3	0	1	0
5	0	2	0	0	1	0	2
6	0	2	4	1	0	2	0

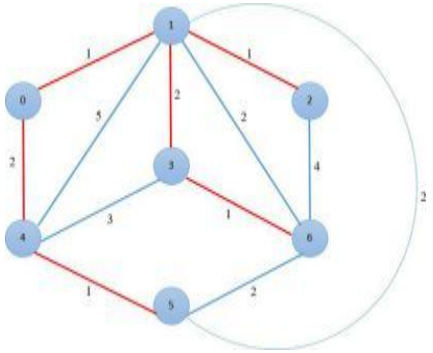


Gambar 22. Output Program (Algoritma Prim) untuk *Minimum Spanning Tree* Eksperimen 3



Gambar23. Output Program (Algoritma Kruskal) untuk Minimum Spanning Tree Eksperimen 3

Dibawah ini adalah gambar untuk hasil *minimum spanning tree* pada graf berdasarkan output dari kedua jenis program. *Minimum spanning tree* pada graf diberi tanda dengan garis berwarna merah.



Gambar 24. *Minimum Spanning Tree*

Eksperimen 3

Untuk menjalankan setiap program diatas disarankan untuk mematikan *back-ground process*, memastikan perangkat tidak terhubung ke jaringan, menon-aktifkan *antivirus*, dan sebagainya agar *running time*, CPU *consumption*, dan *memory consumption* yang dihasilkan akurat. Tabel dibawah ini akan menampilkan perbandingan dari setiap algoritma.

Tabel 5. Perbandingan Algoritma Prim dan Kruskal

Algoritma	Eksperimen	*Runnin g Time*	CPU *Consumpt ion*	*Memory Consumpt ion*
Prim	1	±1,002 ms	±0,3%	±0,2%
	2	±1,000 ms	±0,4%	±0,2%
	3	±0,789 ms	±0,6%	±0,2%
Kruskal	1	±0,974 ms	±0,2%	±0,2%
	2	±0,988 ms	±0,3%	±0,2%
	3	±0,997 ms	±0,7%	±0,2%

4. Analisis

Pada tabel 5 dapat terlihat dengan jelas bahwa masing-masing algoritma mempunyai *running time* yang lebih cepat tergantung dari kompleksitas grafnya. Ketika kompleksitasnya sederhana atau

jumlah *vertices* pada grafnya adalah 5 maka algoritma kruskal akan mempunyai *running time* yang lebih cepat dibandingkan dengan algoritma prim. Tetapi ketika jumlah *vertices* nya adalah 7 maka algoritma prim lebih cepat dibandingkan algoritma kruskal dalam mencari *minimum spanning tree* nya.

Hal tersebut sepertinya disebabkan karena pengaruhnya kompleksitas waktu *worst case*. Untuk algoritma prim menggunakan kompleksitas *binary heap* yaitu O(E log V), karena data input yang digunakan oleh program dalam bentuk matriks *adjacency* atau *data structure* nya dalam bentuk *binary tree*, sehingga algoritma prim hanya akan memilih 1 *node* atau *vertex* yang mempunyai bobot paling minimal, kemudian mengecek dan memilih setiap *vertices* yang bersisian atau bertetanggaan dengan *vertices* yang dipilih. Berikut ini perbedaan Algoritma Prim dan Kruskal [6] [9] [10]

Tabel 6. Perbedaan Algoritma Prim dan Kruskal

dipilih harus bersisian dengan *edge* yang sudah dipilih sebelumnya.	*edge* selanjutnya dari urutan yang sudah ada.
Lebih cepat menghasilkan MST jika bentuk graf kompleks.	Lebih cepat menghasilkan MST jika bentuk graf sederhana.
Basisnya merupakan algoritma greedy.	Basisnya merupakan algoritma greedy.
Kompleksitas waktunya adalah O(E log V) untuk *Binary Heap* atau O(E + V log V) untuk *Fibonacci Heap*.	Kompleksitas waktunya adalah O(E Log E) atau O(E Log V).
Sisi yang dimasukkan kedalam T harus bersisian dengan sebuah simpul di T	Sisi yang dipilih tidak perlu bersisian dengan sebuah simpul T asalkan penambahan sisi tersebut tidak membentuk sirkuit

4. KESIMPULAN

Sebelumnya dengan ketentuan mempunyai bobot paling minimal tanpa membentuk sirkuit (*cycle*). Sedangkan algoritma kruskal ada 2 kompleksitas waktunya yaitu O(E log E) untuk mengurutkan setiap *edges* pada graf, dari bobot yang paling kecil ke yang paling besar, kemudian untuk melakukan verifikasi apakah sisi tersebut aman (tidak membentuk *cycle*) baru menggunakan O(E Log V).

Oleh karena itu jika bentuk graf agak kompleks maka algoritma kruskal akan mengalami kesulitan, karena akan melakukan pengurutan semua *edges* terlebih dahulu baru kemudian melakukan verifikasi pada setiap *edges*. Berbeda dengan algoritma prim yang langsung melakukan verifikasi *edges* tanpa harus mengurutkannya terlebih dahulu. Tetapi jika grafnya sederhana memang sangat menguntungkan untuk menggunakan algoritma kruskal, karena tidak perlu mengecek setiap kemungkinan pada *edges*, cukup mengecek dari *edges* yang sudah diurutkan saja.

Memory consumption atau alokasi memory yang digunakan untuk menjalankan kedua program kurang lebih sama, sedangkan CPU *consumption* akan meningkat seiring dengan tingkat kompleksitas graf. Dibawah ini terdapat tabel yang berisi ringkasan perbedaan dari kedua jenis algoritma tersebut.

Tabel 2. Perbandingan Algoritma Prim dan Algoritma Kruskal

Algoritma Prim	Algoritma Kruskal
Untuk membentuk MST, pertama memilih *edges* yang mempunyai bobot paling minimum kemudian *edge* selanjutnya yang akan memilih	Untuk membentuk MST, pertama mengurutkan setiap *edges* dari bobot paling minimum ke maksimum, dan memilih

Dari semua eksperimen yang telah dilakukan dapat disimpulkan bahwa algoritma itu tidak ada yang salah atau buruk, hanya lebih baik dan efisien dibandingkan dengan yang lainnya, ke-efisienan algoritma diukur dari seberapa cepat waktu eksekusi atau *running time* nya dalam menyelesaikan permasalahan yang ada. Contohnya dalam eksperimen pencarian *minimum spanning tree* pada sebuah graf yang telah dilakukan, jika kompleksitas graf sederhana maka algo-ritma kruskal akan mempunyai waktu eksekusi yang lebih cepat sehingga lebih efisien, sedangkan jika kompleksitas grafnya rumit maka algoritma prim lebih cepat. Hal tersebut dipengaruhi oleh kompleksitas waktu *worst case* nya dimana algoritma prim itu O (E Log V) dan algoritma kruskal itu O (E Log E) dan O (E Log V).

REFERENCES

[1] F. Daniel and P. N. L. Taneo, Teori Graf. Yogyakarta, Indonesia: Deepublish, 2019.
[2] M. K. Siregar, Matematika Diskrit. Lampung, Indonesia: Perahu Litera, 2018.
[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction To Algorithms, 3rd ed. London, England: Eastern Economy, 2010.
[4] Marsudi, Teori Graf. Malang, Indonesia: Universitas Brawijaya Press, 2016.
[5] H. M. Pandey, Design Analysis and Algorithm. New Delhi, India: Firewall Media, 2008.
[6] S. S. Skiena, The Algorithm Design Manual, 2nd ed. New York, USA: Springer Science & Business Media, 2020.
[7] Carlson, S. C. Graph theory. Retrieved from https://www.britannica.com/topic/graph-theory, 2017.
[8] P. Simanjuntak, E. Elisa, H. Pangaribuan, Pengantar Konsep Struktur Data, Pustaka Galeri Mandiri, 2020.
[9] F. Daniel, Prida N. L. Taneo, Teori Graf, Deepublish, 2019.
[10] Majidah Khairani Siregar, Matematika Diskrit, Perahu Litera, 2018.
[11] Nugraha, D.W, Aplikasi Algoritma Prim Untuk Menentukan Minimum Spanning Tree Suatu Graf Berbobot dengan Menggunakan Pemrograman Berorientasi Objek. Jurnal Ilmiah Foristek. Volume 1, Nomor 2, September, 2011.
[12] S. Rizki, Penerapan Teori Graf untuk Menyelesaikan Masalah Minimum Spanning Tree (MST) Menggunakan Algoritma Kruskal, Jurnal Aksioma , Volume 1, Nomor 2, 2012.
[13] Rusdiana, L., & Marfuah, M. The Application of Determining Students' Graduation

Status of STMIK Palangkaraya Using K-Nearest Neighbors Method. *International Conference on Environment and Technology (IC-Tech).* Pekanbaru, Indonesia: IOP Conf. Series: Earth and Environmental Science, 2017.

[14] Schach, S. R. *Object-Oriented and Classical Software Engineering.* New York: McGraw-Hill. Widya, M. A., Agustiawan, Y., Fibrian, I. D., & Muttaqin, Z. 2011.

[15] Hameed, A, Software Development Lifecycle for Extreme Programming. *International Journal of Information Technology and Electrical Engineering*, 2016.