Penerapan Algoritma Greedy pada Least Recently Used (LRU) untuk Memoization dalam Optimasi Waktu Pemrosesan

Marcellus Michael Herman Kahari - 13520057
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
michaelkahari.mk@gmail.com

***Abstract*—Memoization adalah teknik pemrograman yang diciptakan untuk mempercepat performance dari suatu aplikasi menggunakan cache untuk mengembalikan suatu nilai yang didapatkan dari proses yang memakan waktu dan sumber daya cukup besar. Memoization dapat diaplikasikan menggunakan Least Recently Used (LRU) yang diimplementasikan menggunakan algoritma Greedy. Memoization dapat digunakan untuk berbagai hal, salah satunya adalah menyimpan data hasil pemrosesan pada suatu cache.**

Keywords— Greedy, Memoization, LRU, Optimasi, Cache

I. PENDAHULUAN

Memoization atau memoisasi adalah salah satu teknik pemrograman yang diciptakan untuk mempercepat *performance* dari suatu aplikasi menggunakan cache untuk mengembalikan suatu nilai yang didapatkan dari proses yang memakan waktu dan sumber daya cukup besar.

Salah satu aplikasi *memoization* adalah mendapatkan respons yang tepat untuk permintaan pengguna. Pengguna akan melakukan permintaan dan permintaan tersebut dikirim ke web server. Jika pada *cache* terdapat permintaan yang diinginkan oleh pengguna, web server akan mengambil hasil pemrosesan dari *cache* sesuai dengan permintaan pengguna dan mengirimkan kembali ke pengguna.

Jika pada *cache* tidak ditemukan (*cache miss*), web server akan melakukan pencarian ke *backend storage* yang tentu akan memakan sumber daya dan waktu, kemudian mengirimkan hasil pemrosesan ke web server. Web server kemudian akan menyimpan hasil pemrosesan tersebut ke *cache* sehingga jika di masa depan pengguna ingin mendapatkan kembali hasil dari permintaan tersebut, web server cukup mengambilnya dari *cache* yang tentu tidak memakan waktu yang lama.

Cache memiliki kapasitas yang terbatas. Oleh karena itu, perlu diterapkan Least Recently Used (LRU) pada *cache* sedemikian sehingga hasil pemrosesan yang jarang diakses akan dihapus dan digantikan dengan yang baru jika ada.

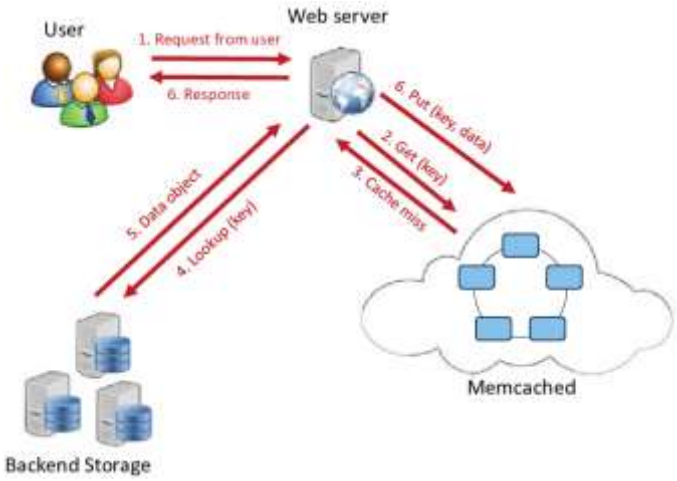

Fig. 1. Ilustrasi penggunaan cache

Sumber : https://www.researchgate.net/figure/Front-end-web-server-requests-for-a-data-objects-to-the-appropriate-cache-server-The_fig1_318474318

Pada makalah ini, penulis akan menjabarkan kegunaan *memoization* dalam memproses permintaan pengguna dengan waktu pemrosesan yang relatif singkat jika dibandingkan tanpa menggunakan *memoization.*

II. TEORI DASAR

A. *Algoritma Greedy*

Algoritma Greedy adalah algoritma yang memecahkan persoalan dengan membaginya menjadi langkah per langkah. Dalam penerapannya, setiap langkah pada algoritma Greedy selalu diusahakan merupakan langkah terbaik yang dapat diperoleh saat itu. Akan tetapi, langkah ini tidak memperhatikan konsekuensi ke depan yang mungkin akan terjadi, seperti justru semakin menjauhnya himpunan solusi dari tujuan utamanya.

Pada algoritma Greedy, setiap langkah yang diambil juga diharapkan merupakan langkah yang optimum lokal. Dari setiap langkah optimum lokal ini, diharapkan bahwa pada hasil akhir akan terjadi optimum global.

Akan tetapi, tidak selamanya algoritma Greedy menghasilkan optimum global. Bisa jadi solusi yang dihasilkan oleh algoritma Greedy merupakan solusi sub-optimum atau solusi pseudo-optimum.

Hal ini disebabkan oleh 2 hal sebagai berikut.

1. Berbeda dengan exhaustive search yang bekerja untuk seluruh kemungkinan solusi, algoritma Greedy hanya bekerja pada kemungkinan-kemungkinan solusi tertentu.
2. Algoritma Greedy memiliki fungsi-fungsi seleksi yang berbeda-beda sehingga diperlukan pengetahuan lebih untuk menentukan kira-kira solusi mana yang akan menghasilkan optimum global.

Terdapat 6 elemen algoritma Greedy.

1. Himpunan kandidat, C : berisi daftar kandidat yang mungkin akan dipilih pada setiap langkahnya.
2. Himpunan solusi, S : berisi daftar kandidat yang telah terpilih.
3. Fungsi solusi: fungsi untuk menentukan apakah himpunan kandidat yang telah terpilih sudah memberikan solusi.
4. Fungsi seleksi (*selection function*): memilih kandidat untuk dimasukkan ke daftar kandidat berdasarkan strategi greedy tertentu. Strategi greedy bersifat heuristik.
5. Fungsi kelayakan (*feasible*): menentukan apakah kandidat yang akan dipilih layak untuk dimasukkan ke dalam himpunan solusi.
6. Fungsi obyektif : memaksimumkan atau meminimalkan sesuatu.

B. Least Recently Used (LRU)

a. Cache

Dilansir dari aws.amazon.com, *cache* adalah salah satu tempat penyimpanan data dengan kecepatan tinggi yang menyimpan sebagian sekumpulan data sedemikian sehingga jika di masa depan data tersebut ingin digunakan kembali, data tersebut bisa didapatkan kembali dengan cepat. Dengan menggunakan *cache*, pengguna bisa menggunakan kembali data yang telah didapatkan atau diproses sebelumnya.

Cache pada umumnya disimpan pada *fast access hardware* seperti RAM (*Random Access Memory*). Cache dapat digunakan pada berbagai teknologi seperti sistem operasi, CDN dan DNA, *Databases*, dan Web *Application*. Fitur *cache* dapat digunakan menggunakan berbagai tools, salah satunya adalah menggunakan cloud server Amazon dengan fitur ElastiCache.

b. Definisi Least Recently Used Cache

Least Recently Used (LRU) Cache adalah perorganisasian item terurut berdasarkan penggunaannya dan algoritma tersebut dapat membuat kita mengetahui item mana yang sudah tidak digunakan dalam jangka waktu tertentu. LRU Cache memiliki kelebihan yaitu dapat diakses dengan cepat sebab diurutkan berdasarkan Most Recently Used (MRU) ke Least Recently Used (LRU) dan dapat diperbaharui dengan lebih cepat. Akan tetapi, LRU Cache memiliki kekurangan, yaitu memakan memori yang cukup besar.

Kasus	Kasus Terburuk
Ruang memori	O(n)
Akses item	O(1)
Mendapatkan item yang jarang digunakan	O(1)

Fig. 2. Tabel Notasi Big O LRU Cache

c. Cara Kerja Least Recently Used

Misalkan kapasitas adalah jumlah data yang dapat disimpan di cache. Ketika ada data yang masuk, akan dilakukan pengecekan, apakah kapasitas kosong di *cache* tidak sama dengan 0. Jika sama dengan 0, masukkan data dan hasil pemrosesan data ke dalam *cache*. Jika tidak sama dengan 0, cari di cache apakah ada *key* dengan data yang dimasukkan. Jika ada, ambil hasil proses data tersebut dan kembalikan ke pengguna, kemudian ubah waktu akses data di *cache*. Jika tidak ada, cari data dengan waktu akses paling lama, kemudian hapus data tersebut dan ganti dengan data terbaru yang baru dimasukkan.

d. Ilustrasi Cara Kerja Least Recently Used

Misalkan terdapat 5 buah data yang hendak diakses seperti berikut ini.

A	B	C	A	D

Sementara ukuran *cache* hanya dapat menampung maksimal 3 buah data, dengan kolom pertama adalah kata kunci dan kolom kedua adalah waktu akses.

-	0
-	0
-	0

Misalkan pengguna hendak mengakses data A, B, dan C dengan jarak waktu antar pemrosesan adalah 1 detik sehingga isi dari cache menjadi sebagai berikut.

A	1
B	2
C	3

Kemudian, pengguna hendak mengakses data A lagi. Dikarenakan data A sudah ada di *cache*, program tidak perlu memproses ulang data A dan cukup melakukan *get* pada *cache* untuk mendapatkan hasil proses data A. Pada *cache*, waktu akses data A berubah sehingga isi dari *cache* berubah menjadi seperti berikut.

A	4
B	2
C	3

Ketika pengguna hendak mengakses data D, dikarenakan tidak ada data D di *cache*, LRU akan mencari data dengan waktu akses paling jarang (atau paling kecil) untuk dihapus dan digantikan dengan data D sehingga sekarang isi dari tabel D adalah sebagai berikut.

A	4
D	5
C	3

.

C. *Memoization*

Memoization adalah tempat penyimpanan yang menyimpan hasil dari pemanggilan suatu fungsi yang memiliki sumber daya yang tinggi (waktu pemrosesan yang lama, *memory*) dan mengembalikan hasil ketika fungsi tersebut dipanggil dengan parameter input yang sama. Akan tetapi, sebelum melakukan *memoization*, perlu dipastikan bahwa fungsi yang digunakan pada *memoization* merupakan fungsi yang pure sehingga kita bisa mengharapkan hasil yang sesuai dengan keinginan kita. Jika fungsi tersebut merupakan fungsi yang impure, terdapat risiko yang cukup besar sehingga bisa jadi *memoization* mengembalikan hasil yang tidak sesuai..

D. *Optimisasi Program*

Optimisasi program adalah proses memodifikasi program sedemikian sehingga program dapat berjalan dan mengeluarkan hasil sesuai harapan dengan sumber daya yang rendah. Secara umum, program dapat dioptimasi sehingga dapat berjalan dengan kapasitas memori yang rendah dan waktu eksekusi yang relatif cepat..

III. PEMBAHASAN

Penulis akan membahas mengenai penerapan algoritma Greedy pada LRU dan pengujian kode program LRU serta membuktikan apakah memoization dengan LRU dapat mengoptimasi waktu pemrosesan data dibandingkan dengan tanpa LRU.

A. *Penerapan Algoritma Greedy*

Pada implementasi algoritma Greedy untuk memoization, algoritma Greedy diimplementasikan pada Least Recently Used (LRU). Berikut adalah langkah-langkah penerapan algoritma Greedy.

1. Cek data yang dimasukkan, jika data tersebut sudah ada di cache, ambil data tersebut dan kembalikan ke pengguna.
2. Jika data tersebut belum ada, cek apakah kapasitas cache masih dapat diisi. Jika masih terdapat ruang kosong di cache, masukkan pasangan data dan hasil pemrosesan data ke dalam cache.
3. Jika data tersebut belum ada dan kapasitas *cache* sudah penuh, cari data di *cache* dengan waktu pengaksesan paling lama. Hapus data tersebut dan gantikan dengan pasangan data dan hasil pemrosesan data yang baru. Pada bagian ini, diharapkan bahwa data yang lama diakses tersebut dan telah dihapus akan jarang digunakan kembali oleh pengguna.

B. *Analisis Element Algoritma Greedy*

Berdasarkan penjelasan pada poin A, penerapan tersebut dianalisis menjadi 6 *element* algoritma Greedy sebagai berikut ini.

1. Himpunan Kandidat

   Himpunan kandidat pada LRU adalah *key* yang akan dimasukkan pengguna dan akan diproses oleh fungsi *dummy* processKey

2. Himpunan Solusi

   Himpunan solusi pada LRU adalah Node yang terdiri dari *key* dan *value* yang terdapat di *cache*.

3. Fungsi Solusi

   Fungsi solusi pada LRU adalah memeriksa apakah tidak ada lagi *input* dari pengguna.

4. Fungsi Seleksi

   Fungsi seleksi pada LRU adalah pola pada masukan yang telah dimasukkan merupakan pola yang baik untuk dijadikan prediksi ke depannya.

5. Fungsi Kelayakan

   Fungsi kelayakan pada LRU adalah nilai *key* yang dimasukkan pengguna haruslah *input* yang valid.

6. Fungsi Objektif

   Fungsi objektif pada LRU adalah meminimalkan jumlah pergantian isi dari *cache*.

C. *Data Percobaan*

Penulis membuat data *dummy* dengan memasukkan sekumpulan data sebagai berikut.

1. [1, 2 ,3 ,1 ,3 ,6 ,5 ,4 ,5 ,6]
2. [1, 2, 3, 4, 5, 6, 7, 8]
3. [1, 1, 1, 1, 1, 1]

D. *Stuktur Kode Percobaan*

Penulis membuat dua buah *class* utama, yaitu class Traditional, yaitu *class* untuk merepresentasikan proses *request* data untuk setiap permintaan dari client ke server (tidak menerapkan algoritma apa pun), dan class LRUCache, yaitu class untuk merepresentasikan proses *request* data dari client ke server menggunakan Least Recently Used (LRU).

Dalam melakukan percobaan, penulis menggunakan sebuah fungsi *dummy* bernama processKey. Fungsi tersebut berguna untuk merepresentasikan proses *request* data dari client ke

server dengan menambahkan *sleep* sebagai penambah waktu. Berikut adalah *pseudocode* dari fungsi *dummy* processKey pada class LRUCache.

function processKey(key: integer) → string value

{ mengembalikan nilai hasil dari proses yang telah dilakukan }

Deklarasi:

Number: integer

Result: string

Algoritma:

if cache dengan kunci key tidak undefined **then**

→ get(key)

else

Number ←math.random()

Result ← "ABCDEFGHIJKL"

Sleep(100)

put(key, result[number])

→ result[number]

LRUCache dibuat menggunakan *double linked list.* Hal ini menyebabkan susunan dari LRUCache dapat diubah baik maju ataupun mundur. Berikut adalah struktur data dari LRUCache. Pada program ini, diasumsikan bahwa ukuran maksimal dari *cache* hanya mampu menampung paling banyak 3 buah Node sehingga jika data di *cache* sudah 3 buah dan ingin dimasukkan data terbaru, perlu dilakukan penghapusan Node di *cache* untuk data yang paling lama atau jarang diakses.

class LRUCache

Deklarasi:

Size: integer

Capacity: integer

Head: Node

Tail: Node

Cache: dictionary

Selain itu, dibuat sebuah fungsi main untuk menguji apakah LRU dapat meningkatkan *performance* program atau tidak. Di dalam main, terdapat sekumpulan data yang akan diuji. Terdapat 3 buah kasus pengujian, yaitu untuk kasus *average case* dari LRU, kasus *worst case* dari LRU, dan kasus *best case* dari LRU. Dari ketiga kasus tersebut, kemudian akan ditarik kesimpulan apakah LRU cukup efektif dalam meningkatkan performansi program.

E. Percobaan

Percobaan pertama akan menggunakan data nomor 1, yaitu untuk kasus *average case.*

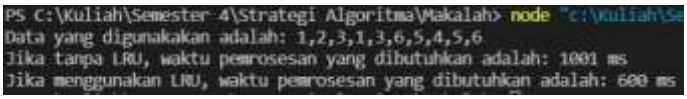

Fig. 3. Gambar Percobaan Average Case

Dari percobaan untuk *average case*, dapat dilihat bahwa waktu pemrosesan yang dibutuhkan untuk memproses data menggunakan LRU lebih cepat 401 ms dibandingkan dengan waktu pemrosesan tanpa menggunakan LRU.

Percobaan kedua akan menggunakan data nomor 2, yaitu untuk kasus worst case.

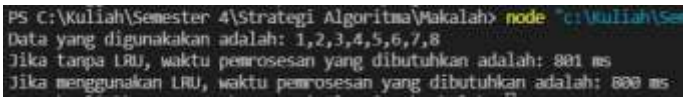

Fig. 4. Gambar Percobaan Worst Case

Dari percobaan untuk *worst case*, dapat dilihat bahwa waktu pemrosesan yang dibutuhkan untuk memproses data menggunakan LRU hanya berselisih sangat sedikit dengan waktu pemrosesan yang diperlukan tanpa LRU. Hal ini menunjukkan bahwa pada kasus *worst case*, waktu pemrosesan dengan LRU dan tanpa LRU adalah sama.

Percobaan ketiga akan menggunakan data nomor 3, yaitu untuk kasus *best case.*

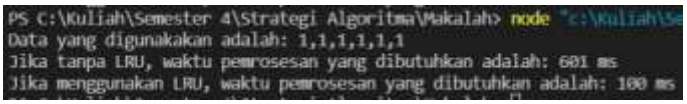

Fig. 5. Gambar Percobaan Best Case

Dari percobaan untuk kasus *best case*, dapat dilihat bahwa waktu pemrosesan yang diperlukan untuk memproses data menggunakan LRU berselisih sangat banyak, yaitu nyaris (n-1)/n dengan n adalah jumlah data yang ada..

IV. ANALISIS

Akan dilakukan analisis cara kerja LRU pada *kasus average case, worst case,* dan *best case*. Dimisalkan terdapat 3 buah kolom dan 3 buah baris dengan masing-masing baris merepresentasikan data yang disimpan di *cache* dan kolom pertama, kedua, dan ketiga secara berturut-turut merepresentasikan *key*, Node, dan waktu pengaksesan.

-	-	0
-	-	0
-	-	0

Penulis akan mengilustrasikan proses cara kerja LRU untuk data pada kasus *average case*. Data yang digunakan adalah sebagai berikut.

[1, 2 ,3 ,1 ,3 ,6 ,5 ,4 ,5 ,6]

1. Step 1

1	Node_1	1
-	-	0
-	-	0

2. Step 2

1	Node_1	1
2	Node_2	2
-	-	0

3. Step 3

1	Node_1	1
2	Node_2	2
3	Node_3	3

4. Step 4

1	Node_1	4
2	Node_2	2
3	Node_3	3

5. Step 5

1	Node_1	4
2	Node_2	2
3	Node_3	5

6. Step 6

1	Node_1	4
6	Node_6	6
3	Node_3	5

7. Step 7

5	Node_5	7
6	Node_6	6
3	Node_3	5

8. Step 8

5	Node_5	7
6	Node_6	6
4	Node_4	8

9. Step 9

5	Node_5	9
6	Node_6	6
4	Node_4	8

10. Step 10

5	Node_5	9
6	Node_6	10
4	Node_4	8

Pada kasus *average case* yang diujikan, dapat disimpulkan bahwa terdapat 6 kali pemrosesan data yang menyebabkan terjadinya *delay* waktu. Keenam pemrosesan tersebut terjadi pada step 1, 2, 3, 6, 7, dan 8. Hal ini jauh lebih singkat dibandingkan pemrosesan biasa tanpa LRU, sebab jika tanpa LRU, data akan melalui pemrosesan sebanyak 10 kali.

Penulis akan mengilustrasikan proses kerja LRU untuk data pada kasus *worst case*. Data yang digunakan adalah sebagai berikut.

[1, 2, 3, 4, 5, 6, 7, 8]

1. Step 1

1	Node_1	1
-	-	0
-	-	0

2. Step 2

1	Node_1	1
2	Node_2	2
-	-	0

3. Step 3

1	Node_1	1
2	Node_2	2
3	Node_3	3

4. Step 4

4	Node_4	4
2	Node_2	2
3	Node_3	3

5. Step 5

4	Node_4	4
5	Node_5	5
3	Node_3	3

6. Step 6

4	Node_4	4
5	Node_5	5
6	Node_6	6

7. Step 7

7	Node_7	7
5	Node_5	5
6	Node_6	6

8. Step 8

7	Node_7	7
8	Node_8	8
6	Node_6	6

.

Pada kasus *worst case* yang diujikan, dapat disimpulkan bahwa terdapat 8 kali pemrosesan data yang menyebabkan terjadinya delay waktu. Waktu yang ditunjukkan dengan algoritma menggunakan LRU sama dengan tanpa LRU, sebab pada kasus ini, tidak ada permintaan yang bisa diatasi menggunakan *cache*.

Penulis akan mengilustrasikan proses kerja LRU untuk data pada kasus *best case*. Data yang digunakan adalah sebagai berikut

[1, 1, 1, 1, 1, 1]

1. Step 1

1	Node_1	1
-	-	0
-	-	0

2.Step 2

1	Node_1	2
-	-	0
-	-	0

3.Step 3

1	Node_1	3
-	-	0
-	-	0

4.Step 4

1	Node_1	4
-	-	0
-	-	0

5.Step 5

1	Node_1	5
-	-	0
-	-	0

6.Step 6

1	Node_1	6
-	-	0
-	-	0

Pada kasus *best case* yang diujikan, dapat disimpulkan bahwa hanya terdapat 1 kali pemrosesan data yang menyebabkan terjadinya *delay* waktu. Hal ini disebabkan pada sisa data yang diminta oleh pengguna dapat diatasi semua oleh *cache* sehingga dapat mengurangi waktu pemrosesan.

V. KESIMPULAN

Berdasarkan percobaan yang telah dilakukan oleh penulis, penulis menyimpulkan bahwa Least Recently Used (LRU) dapat digunakan sebagai *memoization* dalam meningkatkan *performace* program. Terdapat 3 kondisi yang mungkin terjadi pada LRU.

Kondisi yang pertama adalah kondisi *average case*. Pada bagian ini, LRU dapat dikatakan memiliki waktu pemrosesan yang relatif lebih singkat dibandingkan tanpa menggunakan LRU.

Kondisi yang kedua adalah kondisi *worst case*. Pada bagian ini, LRU memiliki waktu pemrosesan yang sama dengan waktu pemrosesan tanpa menggunakan LRU. Hal ini disebabkan data yang diminta oleh pengguna tidak memiliki kesamaan sehingga tidak ada data yang dapat diambil dari *cache*.

Kondisi yang ketiga adalah kondisi *best case*. Pada bagian ini, LRU memiliki waktu pemrosesan yang sangat singkat dibandingkan dengan waktu pemrosesan tanpa LRU. Hal ini disebabkan cache yang ada digunakan secara maksimal.

Walaupun *memoization* sangat berguna untuk mengurangi waktu pemrosesan data, penggunaan *memoization* perlu berhati-hati. Hal ini disebabkan bisa jadi data yang hendak disimpan di *memoization* bukan data yang pure, melainkan impure sehingga bisa jadi justru mengembalikan data yang kurang tepat dan tidak sesuai dengan keinginan pengguna.

VIDEO LINK AT YOUTUBE

https://youtu.be/mhTSmP7uMj4

UCAPAN TERIMA KASIH

Pertama-tama, penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa sebab berkat bantuan-Nya, penulis dapat menyelesaikan penulisan makalah IF2211 Strategi Algoritma. Penulis juga hendak mengucapkan terima kasih kepada dosen IF2211 Strategi Algoritma Kelas 03, Pak Rinaldi Munir, yang telah memberikan ilmu yang bermanfaat untuk penulisan makalah ini. Penulis juga berterima kasih kepada keluarga dan rekan-rekan mahasiswa di prodi Teknik Informatika atas dukungan dan bantuannya selama ini sehingga penulis dapat menyelesaikan penulisan makalah ini.

REFERENCES

[1] Chabbra, Sahil. 2022. *Program for Least Recently Used (LRU) Page Replacement algorithm*. Diakses pada 19 Mei 2022 dari GeeksForGeeks.

[2] Munir, Rinaldi. 2021. *Algoritma Greedy (bagian 1)*. Diakses pada 20 Mei 2022 dari Institut Teknologi Bandung.

[3] Silberschatz, Galvin and Gagne. 2002. *Chapter 10: Virtual Memory*. Diakses pada 19 Mei 2022 dari http://www.wiley.com/college/silberschatz6e/0471417432/slides/pdf2/mod10.2.pdf.

[4] Walker, James. 2021. *What Is Memoization and Why Does It Matter?*. Diakses pada 20 Mei 2022 dari https://www.howtogeek.com/devops/what-is-memoization-and-why-does-it-matter/.

[5] Wu, David. 2019. *CS 4102: Algorithms*. Diakses pada 20 Mei 2022 dari Universitas Virginia.

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, 21 Mei 2022

Marcellus Michael Herman Kahari
13520057