Implementasi *Multiple Constraint Knapsack Problem* dengan Algoritma *Dynamic Programming* dalam Optimasi Nilai Transfer Pemain Olahraga

Naufal Dean Anugrah (13518123)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: naufal.dean@gmail.com

Abstrak*—Transfer pemain dalam kompetisi olahraga merupakan suatu hal yang sangat penting. Kegiatan ini akan menentukan kualitas suatu tim sehingga kemungkinan untuk meraih kemenangan menjadi semakin besar. Akan tetapi, proses transfer pemain dibatasi oleh keadaan keuangan suatu tim. Terdapat beberapa aspek keuangan yang harus diperhatikan, dua hal yang sangat penting adalah dana transfer dan dana gaji pemain. Permasalahan tersebut dapat dimodelkan menjadi suatu permasalahan optimasi dengan suatu batas, dalam hal ini menggunakan permasalahan knapsack. Lebih tepatnya, permasalahan 0/1 knapsack dengan multi batasan (*multiple constraint 0/1 knapsack problem*). Dalam makalah ini, akan dibahas mengenai optimasi permasalahan transfer pemain yang dimodelkan menggunakan permasalahan 0/1 knapsack multi batasan. Batasan yang digunakan yaitu batasan dana transfer dan gaji pemain. Permasalahan knapsack tersebut akan diimplementasikan menggunakan algoritma *dynamic programming.

***Kata kunci—Broadcast,* Kriptografi, RSA, Teorema Coppersmith.**

I. PENDAHULUAN

Olahraga merupakan suatu aspek yang tak terpisahkan dari kehidupan manusia. Menurut ThoughtCo. [1], olahraga telah muncul setidaknya 3000 tahun silam. Olahraga awal yang dilakukan manusia, secara umum ditujukan untuk pelatihan perang maupun persiapan berburu, misalkan dengan adanya lembar tombak, batu, dan sebagainya. Hal ini membuat suatu kelompok manusia bisa bersaing dengan kelompok lain dan dapat bertahan hidup di alam bebas kala itu.

Saat ini, olahraga sudah berkembang dengan pesat dan memiliki banyak cabang di dalamnya. Tujuan olahraga pun juga mengalami perubahan, salah satunya adalah menjadi sarana hiburan bagi umat manusia. Menurut Britannica [2], olahraga sudah menjadi pertunjukan di berbagai peradaban kuno manusia. Pertama, di Mesir kuno terdapat pertunjukan panahan. Kedua, di Yunani kuno terdapat tradisi olimpiade pertama di dunia yang masih dilanjutkan hingga saat ini. Ketiga, di Roma terdapat Colosseum yang menjadi sarana pertunjukan untuk pertarungan gladiator. Selain itu, di abad pertengahan terdapat turnamen jousting misal.

Dengan adanya berbagai turnamen dan pertunjukan tersebut, muncullah berbagai klub olahraga yang bertujuan untuk memenangkan suatu turnamen. Misalnya saja, di cabang sepakbola terdapat klub Real Madrid, Manchester United, Barcelona, dll. Di cabang basket, terdapat klub Golden State Warriors, Boston Celtics, Houston Rockets, dll. Kemunculan berbagai klub olahraga tersebut secara tidak langsung membuat persaingan dalam suatu turnamen olahraga menjadi semakin meningkat.

Dari sisi ekonomi, terdapat berbagai pihak yang menjadi sponsor untuk klub tersebut. Menurut statista [3], nilai sponsor baju sepakbola 2019 terbesar dipegang oleh bidang penerbangan dengan nilai 207 juta euro, disusul oleh bidang otomotif dengan nilai 159 juta euro. Secara umum, tawaran sponsor suatu klub sebanding dengan kemungkinan klub tersebut meraih kemenangan. Oleh karena itu, proses transfer pemain menjadi krusial. Hal ini disebabkan karena, dengan pemain yang bagus kemungkinan kemenangan tim akan meningkat dan pendanaan dari pihak sponsor, penjualan tiket, dsb. akan meningkat. Keadaan tersebut tentunya menjadi incaran untuk setiap klub sehingga keberadaan klub tersebut akan terus terjaga dan tidak mengalami kerugian secara finansial.

Untuk mendapatkan total nilai pemain yang maksimum dengan jumlah pendanaan yang terbatas, dapat dilakukan pemodelan permasalahan transfer pemain menggunakan permasalahan komputasi yang sudah dikenal. Dalam kasus ini menggunakan permasalahan knapsack. Penjelasan detail dan implementasi permasalahan ini akan dibahas lebih lanjut pada bagian berikut dalam makalah ini.

II. DASAR TEORI

A. *Dynamic Programming*

Dynamic programming adalah suatu algoritma dalam suatu proses komputasi. Algoritma ini biasanya dipakai untuk menyelesaikan suatu permasalahan optimasi di berbagai aspek kehidupan. Seperti pada algoritma *divide-and-conquer,* algoritma *dynamic programming* menyelesaikan permasalahan

dengan mengombinasikan solusi pada subpermasalahan yang ada.

Menurut Rinaldi Munir [4], terdapat 7 karakteristik persoalan dengan program dinamis, karakteristik tersebut adalah sebagai berikut:

1. Persoalan dibagi menjadi beberapa tahap dan akan diambil satu keputusan pada setiap tahap tersebut.
2. Setiap tahap tersusun atas beberapa status. Status tersebut secara umum berhubungan dengan suatu tahap dan merupakan kemungkinan masukan yang dapat diberikan dalam tahap tersebut.
3. Keputusan pada setiap tahap digunakan untuk menghitung tahap berikutnya.
4. *Cost* akan meningkat seiring dengan bertambahnya tahap.
5. *Cost* suatu tahap bergantung pada *cost* tahap sebelumnya serta *cost* untuk berpindah ke tahap selanjutnya.
6. Keputusan terbaik pada suatu tahap akan memberikan keputusan terbaik pula pada tahap berikutnya karena terdapat hubungan rekursif.
7. Berlaku prinsip optimalitas dalam permasalahan tersebut.

Dalam proses pembangunan tabel pada algoritma *dynamic programming*, terdapat dua pendekatan yang bisa dilakukan, yaitu:

1. Tabulation (*bottom-up*).

   Pendekatan ini dilakukan dengan membuat formulasi permasalahan secara rekursif. Nilai untuk subpermasalahan akan diselesaikan terlebih dahulu. Kemudian, kalkulasi nilai permasalahan yang lebih besar dilakukan dengan menggunakan nilai subpermaslaahan yang telah dihitung.

2. Memoization (*top-down*).

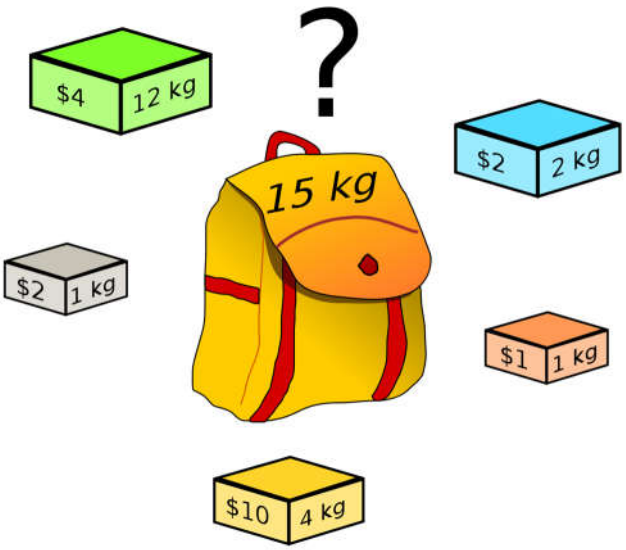
   Pendekatan ini dilakukan dengan menghitung nilai dari permasalahan yang besar hingga menuju permasalahan basis. Setiap nilai permasalahan yang sudah dihitung akan disimpan dalam suatu tabel. Oleh karena itu, pendekatan ini disebut memoization.

Algoritma *dynamic programming* sudah banyak digunakan untuk mengembangkan berbagai algoritma lain. Dalam permasalahan graf misalnya, terdapat algoritma lintasan terpendek Dijkstra, Floyd-Warshall, dan Bellman-Ford. Dalam permasalahan grammar, terdapat algoritma Cocke-Younger-Kasami (CYK) untuk validasi suatu string apakah merupakan hasil dari suatu Context-Free-Grammar.

Dalam makalah ini, *dynamic programming* akan digunakan untuk membuat implementasi solusi permasalahan 0/1 knapsack dengan multi batasan (*multiple constraint 0/1 knapsack problem*).

B. *Permasalahan Knapsack*

Permasalahan knapsack adalah suatu permasalahan optimasi. Definisi permasalahannya adalah diberikan suatu himpunan barang, setiap barang dalam himpunan tersebut memiliki suatu nilai dan berat, tentukan kombinasi barang untuk dimasukkan ke dalam ransel (*knapsack*) sehingga total nilainya maksimum dan beratnya tidak melebihi kapasitas dari ransel tersebut.



Gambar 1 Ilustrasi Permasalahan Knapsack

Secara umum, terdapat beberapa varian dari permasalahan Knapsack. Pertama, permasalahan 0/1 knapsack. Kedua, permasalahan knapsack terbatas (*bounded knapsack problem*). Ketiga, permasalahan knapsack tanpa batas (*unbounded knapsack problem*).

Terdapat kesamaan dalam ketiga varian permasalahan knapsack tersebut di atas, yaitu pada nilai yang dimaksimasi dan batasan pada berat yang dapat diambil. Secara matematis dapat dituliskan sebagai berikut:

$$\textbf{\textit{maksimasi}}: \sum_{i=1}^{n} v_i x_i$$

$$\textbf{\textit{batasan}}: \sum_{i=1}^{n} w_i x_i \leq W$$

dengan v_i adalah nilai barang ke-i, w_i adalah berat barang ke-i, x_i adalah jumlah barang ke-i yang diambil, dan W adalah kapasitas ransel.

Perbedaan antara ketiga varian tersebut adalah pada batasan jumlah barang yang diambil untuk tiap jenis barang dalam himpunan. Pada permasalahan 0/1 knapsack, maksimum hanya mengambil satu untuk setiap barang yang ada. Pada permasalahan knapsack terbatas, maksimum dapat mengambil sejumlah c untuk tiap barang. Pada permasalahan knapsack tanpa batas, tidak terdapat jumlah maksimum yang boleh diambil untuk tiap jenis barang.

Secara matematis, perbedaan batasan pada ketiga varian permasalahan knapsack tersebut adalah sebagai berikut:

$$\textbf{0/1 knapsack}: 0 \leq x_i \leq 1$$

$$\textbf{knapsack terbatas}: 0 \leq x_i \leq c$$

$$\textbf{knapsack tak terbatas}: 0 \leq x_i$$

C. Multiple Constraint 0/1 Knapsack Problem

Multiple constraint 0/1 knapsack problem (permasalahan 0/1 knapsack dengan multi batasan) adalah salah satu pengembangan dari permasalahan 0/1 knapsack biasa. Perbedaannya terletak pada batasan yang diberikan pada permasalahan yang ada. Dalam *Multiple constraint 0/1 knapsack problem*, terdapat lebih dari satu batasan yang diberikan. Misalnya saja, batasan untuk volume dan berat barang yang bisa dimasukkan ke dalam ransel.

Misal terdapat kumpulan item yang dapat dilihat pada tabel berikut:

Tabel 1 Contoh Data Item

No.	Nilai	Volume	Berat
1	4	3	2
2	4	2	2
3	5	3	3
4	1	2	1
5	3	2	1
6	3	1	3

dengan volume maksimum adalah 6 dan berat maksimum adalah 4.

Dari data, pada Tabel 1, bisa didapatkan bahwa total nilai maksimum item yang dapat dimasukan ke dalam ransel adalah 8. Dengan solusi sebagai berikut:

Tabel 2 Solusi Contoh Permasalahan Knapsack

Solusi	Item Dipilih	Nilai	Volume	Berat
1	1, 2	8	5	4
2	3, 5	8	5	4

Sebagai catatan, untuk selanjutnya kita akan fokus pada kasus 0/1 knapsack. Hal ini karena dalam permasalahan transfer pemain tidak mungkin melakukan transfer pada satu pemain sebanyak dua kali secara bersamaan. Kemudian, penyebutan *multiple constraint knapsack problem* merujuk pada *multiple constraint 0/1 knapsack problem.*

III. Implementasi

Pada bagian ini akan dijelaskan mengenai langkah implementasi *multiple constraint 0/1 knapsack problem* dengan menggunakan algoritma *dynamic programming*. Pertama, akan dibahas langkah-langkah algoritmanya. Kemudian, akan diberikan potongan kode program dalam bahasa python.

Dalam implementasi ini, *constraint* untuk permasalahannya ada dua, yaitu dana transfer dan dana pemain. Akan dicari satu buah kombinasi pemain yang memiliki nilai total maksimum dan total harga transfer (`tf_price`) tidak melebihi dana transfer (`max_tf_price`) serta total gaji (`salary`) tidak melebihi dana gaji (`max_salary`).

A. Langkah Algoritma

Berikut adalah definisi yang digunakan untuk implementasi kasus:

1. Tahap (i) berupa proses penentuan, apakah pemain ke-i akan dibeli atau tidak.
2. Status (j) berupa dana transfer yang tersisa setelah membeli pemain pada tahap sebelumnya.
3. Status (k) berupa dana gaji yang tersisa setelah membeli pemain pada tahap sebelumnya.

Berikut langkah algoritma untuk implementasi *multiple constraint 0/1 knapsack problem* dengan menggunakan algoritma *dynamic programming*. Langkah di bawah dibuat dengan memisalkan nama matriks yang digunakan adalah *dp.* Kemudian, $dp[i][j][k]$ adalah nilai untuk tahap ke-*i* pada status (*j,k*) atau dalam notasi matematisnya $f_i(j,k)$.

1. Buat matriks 3 dimensi. Dimensi pertama untuk penentuan pembelian pemain tiap tahap. Dimensi kedua untuk menyatakan status j. Dimensi ketiga untuk menyatakan status k.
2. Inisiasi nilai $dp[i][0][0]$, $dp[0][j][0]$, dan $dp[0][0][k]$ dengan nilai 0 untuk $0 \leq i \leq jumlah\_pemain$, $0 \leq j \leq max\_tf\_price$ dan $0 \leq k \leq max\_salary$. Nilai ini akan menjadi basis nilai dari tabel *dynamic programming*.
3. Lakukan *loop i* dari *1* hingga *jumlah_pemain* yang merepresentasikan tiap tahap penentuan pembelian pemain ke-*i.*
4. Lakukan *loop j* dari *0* hingga *max_tf_price* yang merepresentasikan penggunaan dana transfer hingga batas kapasitas maksimumnya.
5. Lakukan *loop k* dari *0* hingga *max_salary* yang merepresentasikan penggunaan dana gaji hingga batas kapasitas maksimumnya.

6. Di dalam ketiga *loop* di atas, bagi menjadi dua kondisi.
   a. Jika *tf_price* pemain ke-i melebihi nilai j atau *salary* pemain ke-i melebihi nilai k, pemain ke-i tidak dibeli. Hal ini karena, dana transfer atau dana gaji tidak mencukupi.

      Set:

      $dp[i][j][k] = dp[i-1][j][k]$.
   b. Selain pada kasus poin a (dana transfer dan dana gaji mencukupi), beli pemain ke-i.

      Set:

      $dp[i][j][k] = \max($
      $dp[i-1][j][k],$
      $dp[i-1][j - tf\_price\_pemain\_i]$
      $[k - salary\_pemain\_i] + nilai\_pemain\_i$
).
7. Hasil optimasinya bisa kita dapatkan dengan melakukan cek sederhana. Jika nilai $dp[i][j][k] \neq dp[i-1][j][k]$, pemain ke-$i$ dibeli, dan sebaliknya. Hal ini berlaku, karena hanya satu solusi yang akan diambil jika terdapat beberapa solusi.

B. Implementasi Kode Program

Berikut adalah kelas yang digunakan untuk menyimpan data pemain yang akan ditransfer:

```
class Player(object):
    """Player class implementation."""

    def __init__(self, value, tf_price, salary):
        self.value = value
        self.tf_price = tf_price
        self.salary = salary

    def get_value(self):
        return self.value

    def get_tf_price(self):
        return self.tf_price

    def get_salary(self):
        return self.salary
```

Kemudian, berikut implementasi fungsi knapsack untuk menyelesaikan *multiple constraint knapsack problem* dalam kasus ini. Algoritma yang digunakan mengacu pada penjelasan di bagian A bab ini.

```
def knapsack(max_tf_price, max_salary,
                                    player_list):
    # Init dp matrix
    max_i = len(player_list) + 1
    max_j = max_tf_price + 1
    max_k = max_salary + 1
    dp = [[[0 for k in range(max_k)]
              for j in range(max_j)]
              for i in range(max_i)]

    # Build dp matrix
    for i in range(1, max_i):
        for j in range(0, max_j):
            for k in range(0, max_k):
                # Get i-th player stats
                # NB: saved in idx i - 1
                #     in the array
                v = player_list[i - 1]
                                .get_value()
                t = player_list[i - 1]
                                .get_tf_price()
                s = player_list[i - 1]
                                .get_salary()
                # Check condition
                if t > j or s > k:
                    dp[i][j][k] = dp[i-1][j][k]
                else:
                    dp[i][j][k] = max(
                        dp[i-1][j][k],
                        dp[i-1][j-t][k-s] + v
                    )
    return dp
```

Untuk mendapatkan salah satu solusi kombinasi yang optimum, dapat menggunakan kode berikut:

```
max_tf_price = …
max_salary = …
player_list = […]

dp = knapsack(max_tf_price, max_salary,
                                    player_list)

j = max_tf_price
k = max_salary
opt_val = opt_tfp = opt_sal = 0
for i in range(len(player_list), 0, -1):
    if dp[i][j][k] != dp[i - 1][j][k]:
        v = player_list[i - 1].get_value()
        t = player_list[i - 1].get_tf_price()
        s = player_list[i - 1].get_salary()
```

```
        print(f"Player-{i} taken:
                                ({v},{t},{s})")
        opt_val += v
        opt_tfp += t
        opt_sal += s
        j -= t
        k -= s
print(f"Optimum value: {opt_val}")
print(f"Optimum transfer price: {opt_tfp}")
print(f"Optimum salary: {opt_sal}")
```

IV. Studi Kasus

Pada bagian ini, akan dilakukan pengujian terhadap aplikasi yang telah dibuat. Data pemain sepakbola yang digunakan, merupakan hasil kompilasi dari berbagai sumber. Nilai pemain diambil dari basis data permainan sepakbola PES 2020 [8]. Harga transfer pemain diambil dari laman web transfermarkt [6]. Kemudian, gaji pemain diambil dari gaji bulanan pemain [7] dikalikan 12. Berikut data yang digunakan dalam pengujian kali ini:

Tabel 3 Data Pemain Sepakbola

No	Nama	Nilai	Harga Transfer (juta euro)	Gaji Tahunan (juta euro)
1	C. Ronaldo	94	60	54
2	L. Messi	94	112	102
3	Neymar	92	128	48
4	S. Aguero	91	52	16
5	L. Suarez	91	28	36
6	E. Hazard	91	80	19
7	V. van Dijk	91	80	12
8	S. Ramos	90	14	22
9	Pique	90	20	22
10	David de Gea	90	40	21
11	Kylian Mbappe	90	180	21
12	R. Lukaku	86	68	9
13	G. Buffon	84	1	7
14	S. Cazorla	83	4	7
15	M. Rashford	83	64	13

Untuk uji coba pertama, digunakan nilai $max\_tf\_price = 250$ dan max $\_salary = 125$. Setelah menjalankan program, didapat hasilnya adalah sebagai berikut:

Tabel 4 Hasil Uji Coba 1

No	Nama	Nilai	Harga Transfer (juta euro)	Gaji Tahunan (juta euro)
4	S. Aguero	91	52	16
5	L. Suarez	91	28	36
7	V. van Dijk	91	80	12
8	S. Ramos	90	14	22
9	Pique	90	20	22
13	G. Buffon	84	1	7
14	S. Cazorla	83	4	7
	Total	620	199	122

Kemudian, untuk uji coba 2, digunakan nilai $max\_tf\_price = 500$ dan max $\_salary = 300$. Setelah menjalankan program, didapat hasilnya adalah sebagai berikut:

Tabel 5 Hasil Uji Coba 2

No	Nama	Nilai	Harga Transfer (juta euro)	Gaji Tahunan (juta euro)
1	C. Ronaldo	94	60	54
3	Neymar	92	128	48
4	S. Aguero	91	52	16
5	L. Suarez	91	28	36
6	E. Hazard	91	80	19
8	S. Ramos	90	14	22
9	Pique	90	20	22
10	David de Gea	90	40	21
12	R. Lukaku	86	68	9
13	G. Buffon	84	1	7
14	S. Cazorla	83	4	7
	Total	982	495	261

Dapat dilihat dari Tabel 5, C. Ronaldo sedangkan L. Messi tidak. Hal ini disebabkan karena C. Ronaldo memiliki harga transfer dan gaji yang jauh lebih kecil daripada L. Messi padahal nilai keduanya sama. Hal yang serupa juga dapat dilihat pada kasus Kylian Mbappe yang tidak dipilih karena harga transfernya sangat besar.

Kemudian untuk uji coba ketiga, akan menguji *edge case* yaitu dengan contoh kasus saat $max\_tf\_price = 1000$ dan max $\_salary = 500$. Kedua batasan nilai tersebut mencukupi untuk membeli semua pemain. Saat program dijalankan, hasilnya pun sesuai dengan perkiraan. Seluruh pemain dibeli dan

menghasilkan total nilai 1340, total harga transfer 931 juta euro, dan total gaji tahunan 409 juta euro.

Kemudian untuk uji coba keempat, juga akan menguji *edge case* yaitu dengan contoh kasus saat $max\_tf\_price = 0$ dan max $\_salary = 0$. Saat program dijalankan, hasilnya juga sesuai perkiraan. Tidak ada pemain yang dibeli sehingga total nilai, harga transfer, dan gaji yang optimum adalah 0.

V. KESIMPULAN

Permasalahan optimasi transfer pemain dapat dimodelkan menjadi *multiple constraint 0/1 knapsack problem* yang diimplementasikan menggunakan algoritma *dynamic programming*. Dengan menggunakan teknik ini, proses transfer pemain akan menjadi lebih efektif dan efisien.

LINK VIDEO YOUTUBE

Berikut ini adalah link video Youtube yang berisi tambahan penjelasan mengenai permasalahan yang telah dibahas dalam makalah ini:

https://youtu.be/8YYYWAwnkZ0

ACKNOWLEDGMENT

Penulis ingin mengucapkan terima kasih kepada berbagai pihak yang telah membantu proses penyelesaian makalah ini. Pertama, kepada Tuhan Yang Maha Esa yang senantiasa memberikan berkah dan rahmat-Nya sehingga penulis dapat menyelesaikan pembuatan makalah ini dengan baik. Kedua, kepada Bapak Dr. Ir. Rinaldi Munir, M.T. selaku dosen mata kuliah IF2211 Strategi Algoritma K-3 yang selalu membimbing kami dalam memahami materi yang dipakai dalam makalah ini. Ketiga, kepada penulis sumber referensi makalah ini yang namanya dicantumkan dalam bagian referensi. Keempat, orang tua penulis yang senantiasa memberi motivasi dan semangat untuk terus belajar. Kelima, teman-teman jurusan Teknik Informatika ITB, khususnya angkatan 2018, yang memberikan semangat untuk bersaing secara sehat selama kuliah di ITB.

REFERENCES

[1] Bellis, Mary. "A Brief History of Sports". https://www.thoughtco.com/history-of-sports-1992447. 2019.

[2] William N. Thompson et al. "Sports". https://www.britannica.com/sports/sports.

[3] Lange, David. "Europe: shirt sponsorship value of leading 50 football clubs 2019, by industry". https://www.statista.com/statistics/1024836/shirt-sponsorship-value-top-football-clubs-by-industry/. 2020.

[4] Munir, Rinaldi. "Program Dinamis (*Dynamic Programming*) Bagian 1". Institut Teknologi Bandung: Bandung, page 7-9, 2020.

[5] Cormen, Thomas H. et al. *Introduction to Algorithms, Second Edition.* The MIT Press: Massachusetts. 2001.

[6] Sportekz. "https://www.sportekz.com/list/highest-paid-footballers-2020/". 2020.

[7] Transfermarkt. "https://www.transfermarkt.com/marktwertetop/wertvollstespieler". 2020.

[8] *PesDB. "http://pesdb.net/pes2020/"*. 2020.

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, 1 Mei 2020

Naufal Dean Anugrah
13518123