Analisis Kompleksitas Algoritma dalam implementasi *Schulze Methods* berbasis teori graf pada *Preferential Single-Winner Election system*

Raffael Boymian Siahaan - 13522046[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522046@std.stei.itb.ac.id

***Abstract*— Makalah ini mengeksplorasi implementasi dan kompleksitas algoritma pada *Schulze Methods* dalam konteks sistem Pemilihan Pemenang Tunggal (Single-Winner Election). Dengan fokus pada algoritma Floyd-Warshall dan Pencarian Breadth-First (BFS) dan memahami bagaimana algoritma komputasi dapat dimanfaatkan untuk meningkatkan efisiensi dan keadilan dalam proses pemilihan.**

***Keywords*—Kompleksitas, Algoritma, Schulze Method, Breadth-First Search, BFS, Floyd-Warshall**

I. PENDAHULUAN

Dalam konteks pemilihan umum, sistem pemilihan atau *electoral system* merupakan aturan penting yang mengatur proses pemilihan dan penentuan pemenang. Salah satu jenis pemilihan yang umum digunakan adalah *Single-Winner Election*, di mana pemilih hanya memilih satu kandidat atau pilihan dari beberapa kandidat, dan kandidat dengan suara terbanyak akan menjadi pemenang tunggal. Pemilihan ini penting karena sering digunakan dalam pemilihan presiden, walikota, atau anggota legislatif, di mana fokusnya adalah menentukan satu pemenang tunggal dari sejumlah kandidat yang bersaing.

Makalah ini bertujuan untuk mengeksplorasi implementasi dan kompleksitas algoritma dalam konteks pemilihan *Single-Winner Election* dengan *Schulze Methods*, dengan fokus khusus pada algoritma *Floyd-Warshall* dan *Breadth-First Search (BFS)*. Penelitian ini penting karena memberikan wawasan tentang bagaimana teknologi dan algoritma komputasi dapat dimanfaatkan untuk meningkatkan efisiensi dan keadilan dalam proses pemilihan. Dengan memahami penerapan algoritma ini, kita dapat lebih memahami potensi dan tantangan dalam penggunaan teknologi dalam pemilihan.

II. LANDASAN TEORI

A. Kompleksitas Algoritma

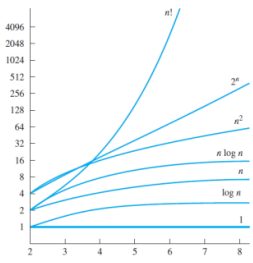
Algoritma adalah suatu urutan logis mengenai langkah-langkah penyelesaian masalah secara sistematis. Sebuah algoritma tidak saja harus benar, tetapi juga harus mangkus (efisien). Sebuah algoritma yang dirancang juga harus mempertimbangkan waktu yang diperlukan untuk menjalankan serta ruang memori yang diperlukan untuk berbagai jenis masukan, mulai dari masukan yang sedikit hingga sangat besar sehingga algoritma dapat dikatakan sangkil apabila algoritma tersebut dapat meminimumkan kebutuhan waktu dan ruang.

Kompleksitas algoritma diukur berdasarkan berapa jumlah waktu dan ruang (space) memori yang dibutuhkan untuk menjalankan suatu algoritma. Kompleksitas algoritma memiliki kegunaan dalam membandingkan beberapa jenis algoritma. Seperti contoh, algoritma pengurutan (sorting) memiliki berbagai jenis algoritma penyelesaian, seperti pengurutan seleksi (selection sort), pengurutan sisipan (insertion sort), pengurutan apung (bubble sort), pengurutan gabung (merge sort), pengurutan cepat (quick sort), dsb. Namun, bagaimana cara memilih algoritma yang terbaik untuk diimplementasikan? Untuk menjawab pertanyaan tersebut, kompleksitas algoritma diperlukan untuk menyelesaikan persoalan tersebut.

Berbicara mengenai kompleksitas dan kebutuhan waktu dan ruang, Terminologi yang biasa diperlukan dalam membahas kompleksitas waktu/ruang dari suatu algoritma adalah :

1. Ukuran besar masukan data (n)
2. Kompleksitas waktu T(n)
3. Kompleksitas ruang S(n)

Untuk mengukur kompleksitas waktu, kita perlu menghitung banyaknya operasi yang dilakukan oleh suatu algoritma. Operasi yang dapat dipertimbangkan adalah operasi penjumlahan, pengurangan, perbandingan, pembagian, pembacaan, pemanggilan prosedur, dsb. Namun, kita tidak perlu menghitung secara keseluruhan. Kita cukup menghitung jumlah operasi dasar dari algoritma tersebut, sebagai contoh, pada algoritma pengurutan (sorting), operasi dasarnya adalah operasi perbandingan elemen dan operasi pertukaran elemen-elemen yang ada pada array tersebut.



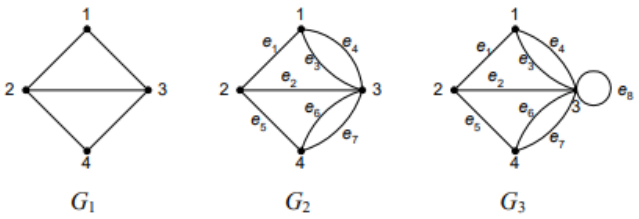
***Gambar 2.1** Perbandingan pertumbuhan notasi Big-Oh*
Diambil dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/25-

Kompleksitas-Algoritma-Bagian2-2023.pdf

B. Graf

Graf merupakan sebuah representasi objek-objek diskrit yang digunakan untuk menentukan hubungan antara objek-objek yang ada. Graf digambarkan sebagai kumpulan simpul dan sisi.

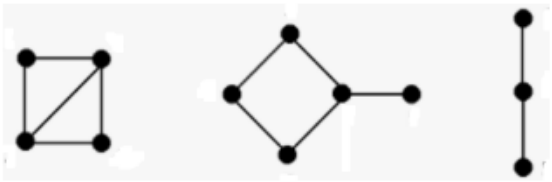
Graf didefinisikan sebagai G = (V, E) dengan V merupakan himpunan tidak kosong dari simpul-simpul $\{ v1, v2. v3, \dots, vn \}$ dan E merupakan himpunan tidak kosong dari sisi-sisi $\{ e1, e2. e3, \dots, en \}$



Gambar 2.2 *Contoh graf*
Diambil dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf

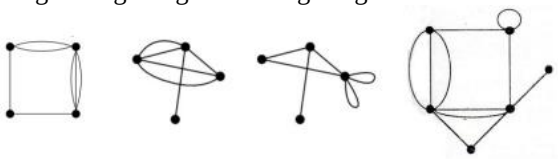
Jenis-Jenis graf berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf digolongkan menjadi dua jenis, yaitu:

1. Graf sederhana (simple graph), yaitu graf yang tidak mengandung gelang ataupun sisi ganda.



Gambar 2.3 *Graf sederhana*
Diambil dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf

2. Graf tak sederhana (unsimple-graph), yaitu graf yang mengandung sisi ganda atau gelang.



Gambar 2.4 *Graf berarah*
Diambil dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf

C. Schulze Method

Metode Schulze adalah sistem pemilihan yang dikembangkan oleh Markus Schulze pada tahun 1997. Metode ini juga dikenal sebagai Schwartz Sequential dropping (SSD), cloneproof Schwartz sequential dropping (CSSD), the beatpath method, beatpath winner, path voting, dan path winner. Cara kerja dari Schulze Method secara garis besar adalah jika ada satu calon yang memenangkan suara mayoritas saat dibandingkan dengan calon lainnya, calon tersebut akan menjadi pemenang. Hasil dari metode ini berupa urutan kandidat terurut mengecil. Metode ini diadopsi oleh beberapa organisasi besar dunia, seperti Debian, Ubuntu, Gentoo, Wikimedia, dan masih banyak lagi.

Metode schulze merupakan modifikasi dari metode Simpson-Kramer yang telah populer sebelumnya, tetapi mendapat beberapa kritikan, seperti tidak memilih betul-betul berdasarkan opsi teratas, rentan terhadap klon (sengaja menominasikan beberapa kandidat yang mirip untuk memanipulasi hasil pemilihan), dan melanggar simetri pembalikan (situasi aneh di mana opsi yang sama dipilih bahkan ketika semua suara dibalik. Artinya, opsi yang sama diidentifikasi sebagai yang terbaik dan sekaligus sebagai yang terburuk).

Implementasi dari Schulze Method secara garis besar adalah setiap pemilih mendapatkan daftar pilihan kandidat (yang disebut “alternatif”) dan mereka harus mengurutkan pilihan tersebut berdasarkan kandidat yang paling mereka sukai. Namun, hal yang perlu diperhatikan adalah :

1. Pemilih bisa memberikan peringkat yang sama untuk beberapa pilihan. Artinya, mereka merasa bahwa beberapa pilihan tersebut sepadan.
2. Pemilih tidak harus memberikan peringkat untuk semua pilihan. Jika ada pilihan yang tidak diberi peringkat, pemilih merasa pilihan tersebut kurang baik dibandingkan yang diberi peringkat, dan mereka tidak membedakan antara pilihan yang tidak diberi peringkat.
3. Pemilih bisa melewatkan beberapa peringkat, tapi ini bukan masalah besar karena hal terpenting dari metode ini adalah urutan pilihan, bukan angka peringkatnya.

Berikut adalah contoh studi kasus ketika terdapat 23 pemilih yang memberi peringkat pada 4 kandidat

Jumlah Pemilih	Urutan Preferensi
8	ACDB
2	BCDA
3	BDAC
5	CBDA
1	CDBA
3	DABC
1	DBAC

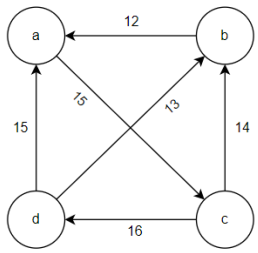
Tabel 2.1 *Tabel studi kasus mengenai jumlah pemilih dan urutan preferensi*

Pertama, kita harus menghitung preferensi pasangan. Misalkan kita ingin membandingkan A dan B. Terdapat 8 + 3 = 11 pemilih yang lebih memilih A daripada B (ACDB dan DABC) dan 2 + 3 + 5 + 1 + 1 = 12 pemilih yang memilih B daripada A (BCDA, CBDA, CDBA, dan DBAC) sehingga N[A, B] = 11 dan N[B, A] = 12, dimana N[X, Y] adalah jumlah pemilih yang memilih kandidat X dan Y. Dengan melanjutkan perbandingan, didapat matriks preferensi berpasangan, yang ditunjukkan sebagai berikut :

	N[*,a]	N[*,b]	N[*,c]	N[*,d]
N[*,a]	---	11	15	8
N[*,b]	12	---	9	10
N[*,c]	8	14	---	16

N[*,d]	15	13	7	---

Tabel 2.2 *Tabel matriks prefrensi berpasangan*

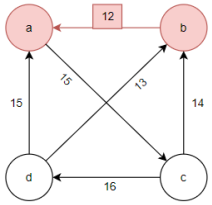


Gambar 2.5 *Graf yang menyatakan hubungan antara preferensi a, b, c, dan d*

Sel pada tabel diwarnai hijau jika N[X, Y] > N[Y, X]. Selanjutnya, kita perlu mengindentifikasi serta memvisualisasikan jalur terkuat menggunakan algoritma Floyd-Warshall.

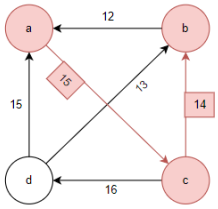
Saat ingin melakukan perhitungan jalur terkuat, terdapat 2 kemungkinan kasus, yaitu :

1. Jalur langsung, yaitu semisal kita ingin mencari jalur terkuat dari simpul b ke a (N[b, a]). Nilai jalur terkuat dapat diperoleh dengan langsung melakukan penelusuran dari b ke a dan didapat bobot 12.



Gambar 2.6 *Graf ilustrasi poin (1)*

2. Jalur tak langsung, yaitu semisal kita ingin mencari jalur terkuat dari simpul a ke b (N[a, b]). ketika jalur yang dilewati tidak langsung ke b, melainkan harus melewati C terlebih dahulu dengan sisi berbobot 15, dilanjutkan ke B dengan sisi berbobot 14. Jika berada pada kondisi seperti ini, maka kita akan mengambil sisi dengan bobot yang minimum ($min(15, 14) = 14$).



Gambar 2.7 *Graf ilustrasi poin (2)*

Dengan melanjutkan algoritma tersebut untuk mencari jalur terkuat dari simpul ke simpul, didapat sekumpulan data bobot jalur terkuat dari masing-masing hubungan antarsimpul sebagai berikut :

	$P_D[*,a]$	$P_D[*,b]$	$P_D[*,c]$	$P_D[*,d]$
$P_D[*,a]$	---	14	15	15
$P_D[*,b]$	12	---	12	12
$P_D[*,c]$	15	14	---	16
$P_D[*,d]$	15	14	15	---

Tabel 2.3 *Tabel matriks data bobot jalur terkuat dari masing-masing simpul*

Selanjutnya, cara menentukan pemenang yang sesuai adalah dengan membandingkan setiap simpul yang ada. Misalkan simpul a dan b, terdapat bobot $P_D[a,b] = 14$ dan $P_D[b,a] = 12$. Maka dapat disimpulkan bahwa $P_D[a,b] > P_D[b,a]$, yang berarti kandidat a mendapatkan bobot lebih (dapat dikatakan lebih baik) dari kandidat b, dan seterusnya sehingga didapatkan pewarnaan tabel sebagai berikut: (warna hijau menyatakan bobot jalur terkuat terbesar dari simpul X ke Y)

	$P_D[*,a]$	$P_D[*,b]$	$P_D[*,c]$	$P_D[*,d]$
$P_D[*,a]$	---	14	15	15
$P_D[*,b]$	12	---	12	12
$P_D[*,c]$	15	14	---	16
$P_D[*,d]$	15	14	15	---

Tabel 2.4 *Tabel penentuan pemenang yang sesuai*

Pemenang m harus memenuhi kondisi dimana $P_D[m,X] \geq P_D[X,m]$ untuk setiap kandidat X lainnya. Berdasarkan tabel di atas, didapat a dan c memiliki suara yang sama dalam pewarnaan hijau, sehingga pemenang berdasarkan metode ini adalah a dan c (a = c > d > b).

III. PEMBAHASAN

A. Implementasi dan Analisis Kompleksitas Algoritma pada Algoritma Floyd-Warshall

Implementasi dari Algoritma Floyd-Warshall diawali dengan pendefinisian fungsi dengan parameter N, yaitu sebuah matriks array yang berisi preferensi pemilih antara kandidat yang ada, kemudian output dari fungsi tersebut adalah :

1. PD, yaitu bobot jalur terkuat dari alternatif i ke j.
2. Pred, yaitu predecessor (pendahulu), dapat diartikan sebagai alternatif yang langsung mendahului j dalam jalur terkuat dari i ke j.
3. Winner, yaitu array boolean yang menyatakan apakah alternatif i adalah bagian dari sebuah himpunan (set) pemenang potensial.

Tahap pertama yang dilakukan adalah inisialisasi, yaitu melakukan inisialisasi pada matriks bobot jalur (PD) dan matriks predecessor (pred) berukuran C x C, dimana C meripakan jumlah alternatif. Selanjutnya, Matriks P diisi dengan bobot jalur antara setiap pasangan alternatif, yaitu ketika i ≠ j, $P_D[i,j]$ akan diatur sebagai sebuah tupel yang berisi jumlah pemilih yang lebih memilih anternatif i daripada j dan sebaliknya (N[i][j], N[j][i]).

Tahap kedua adalah implementasi dari Algoritma Floyd-Warshall. Algoritma ini digunakan untuk mencari jalur terpendek antara semua pasangan simpul dalam graf. Dalam konteks schulze method, algoritma ini digunakan untuk menghitung bobot jalur terkuat ($P_D[i,j]$) dari alternatif i ke

alternatif j.

Cara kerja dari tahap ini adalah :

1. Algoritma akan memerika setiap pilihan (alternatif) satu per satu.
2. Kemudian, untuk setiap pasangan pilihan yang berbeda, algoritma ini akan mencari jalur terkuat di antara keduanya dengan melakukan pengecekan terhadap i, j dan k (`i != j, i != k, j != k`) untuk memastikan bahwa i, j, dan k adalah tiga alternatif yang berbeda.
   Jalur terkuat adalah jalur dengan bobot terbesar, yang dalam konteks ini diukur oleh jumlah pemilih yang lebih memilih satu pilihan dibandingkan dengan yang lain.
3. Jika ternyata ada jalur yang lebih kuat melalui pilihan lain (`PD[j][k]>min(PD[j][i], PD[i][k]`), algoritma ini akan memilih jalur tersebut sebagai jalur terkuat (`PD[j][k] = min(PD[j][i], PD[i][k]`).
4. Selain itu, algoritma ini juga mencatat pilihan mana yang langsung mendahului pilihan lain dalam jalur terkuat, yang maksudnya adalah jika ada jalur yang lebih kuat melalui pilihan tertentu, pilihan tersebut akan dicatat sebagai pilihan yang langsung mendahului pilihan lain dalam jalur terkuat (`pred[j][k] = pred[i][k]`).

Tahap ketiga adalah melakukan penentuan pemenang potensial. Jika bobot jalur terkuat dari alternatif j ke alternatif i lebih besar atau sama dengan boboy jalur terkuat dari i ke j, i bukanlah pemenang dan ditandai sebagai False dalam array Winner.

Implementasi fungsi ini dalam bahasa python adalah sebagai berikut :

```python
# Perhitungan Jalur Terkuat Menggunakan Algoritma Floyd-Warshall
def floyd_warshall_schulze(N):
    # Tahap inisialisasi
    C = len(N)
    PD = [[None]*C for _ in range(C)]
    pred = [[None]*C for _ in range(C)]

    for i in range(C):
        for j in range(C):
            if i != j:
                PD[i][j] = (N[i][j], N[j][i])
                pred[i][j] = i

    # Algoritma Floyd-Warshall
    for i in range(C):
        for j in range(C):
            if i != j:
                for k in range(C):
                    if i != k and j != k:
                        if PD[j][k] > min(PD[j][i], PD[i][k]):
                            PD[j][k] = min(PD[j][i], PD[i][k])
                            pred[j][k] = pred[i][k]

    # Metode Schulze
    winner = [True]*C
    for i in range(C):
        for j in range(C):
            if i != j:
                if PD[j][i] >= PD[i][j]:
                    winner[i] = False

    return PD, pred, winner
```

Gambar 3.1 Fungsi Untuk Perhitungan Jalur Terkuat
Menggunakan Algoritma Floyd-Warshall
Diambil dari dokumen pribadi

Analisis kompleksitas algoritma akan berfokus pada operasi penting yang dilakukan. Pada tahap inisialisasi, matriks PD dan pred diinisialisasi seukuran C x C. Selanjtunya, dilakukan pengisian matriks PD dengan tuple (N[i][j], N[j][i]).

Pada tahap inisialisasi matriks, jumlah operasi yang dilakukan untuk melakukan inisialisasi None adalah $T(C) = 2 \times C \times C = 2C^2$. Namun, pada beberapa bahasa pemrograman, inisialisasi matriks tidak terlalu berpengaruh signifikan sehingga dapat diabaikan. Kemudian, pengisian matriks PD dengan tuple memiliki jumlah operasi sejumlah $T(C) = C \times C = C^2$.

Selanjutnya, pada Algoritma Floyd-Warshall, terdapat tiga loop bersarang yang masing-masing berjalan C kali. Pada kasus terburuk, jumlah assignment yang dilakukan adalah sejumlah $T(C) = C \times C \times C = C^3$.

Dalam penentuan pemenang potensial, terdapat dua loop bersarang yang masing-masing berjalan C kali. oleh karena itu, jumlah operasi yang dilakukan adalah sejumlah $T(C) = C \times C = C^2$. Namun, kasus terburuk dapat terjadi apabila i != j, yang sebenarnya operasi dilakukan sebanyak $T(n) = C^2 - C$ kali.

Maka, jumlah operasi secara keseluruhan adalah $T(C) = C^2 + C^3 + C^2 = c^3 + 2C^2$.

Berdasarkan perkembangan asimptotik paling besar pada T(C), didapat kompleksitas waktu dari fungsi adalah $O(C^3)$.

B. Implementasi dan Analisis Kompleksitas Algoritma dalam mengidentifikasi pemenang potensial

Implementasi ini merupakan algoritma lebih lanjut apabila kita hanya ingin mengidentifikasi apakah suatu kandidat merupakan pemenang potensial atau bukan.

Fungsi memiliki parameter N (matriks yang menyimpan jumlah pemilih yang lebih memilih alternatif i daripada j) dan m (indeks alternatif (kandidat) yang ingin dilakukan validasi). Output dari fungsi berupa boolean Winner yang menunjukkan apakah alternatif m adalah pemenang potensial dan matriks PD yang berisi jalur terkuat antara setiap pasangan alternatif.

Tahap pertama yang dilakukan adalah inisialisasi Matriks PD dan array marked. PD akan menyimpan bobot jalur terkuat antara setiap pasangan alternatif, dan marked digunakan untuk melacak alternatif mana yang telah diproses.

Tahap selanjutnya adalah menghitung bobot jalur terkuat dari alternatif m ke alternatif lainnya. $PD[m][i]$ diisi dengan tuple $(N[m][i], N[i][m])$ untuk setiap alternatif i yang berbeda dari m, yang merupakan jumlah pemilih yang lebih memilih m daripada i dan sebaliknya. Kemudian, kita mencari jalur terkuat dari m ke setiap alternatif lainnya dan memperbarui $PD[m][k]$ jika kita menemukan jalur yang lebih kuat melalui alternatif lain.

Selanjutnya, tahap ini mirip dengan Tahap sebelumnya, tetapi sekarang kita mencari jalur terkuat dari setiap alternatif lainnya ke m.

Tahap terakhir adalah menentukan pemenang potensial, algoritma akan menentukan apakah m adalah pemenang

potensial. Jika ada alternatif i yang jalur terkuatnya ke m lebih kuat atau sama kuat dengan jalur terkuat dari m ke i, maka m bukanlah pemenang potensial.

Implementasi fungsi ini dalam bahasa python adalah sebagai berikut :

```python
def Potential_winner(N, m):
    # Tahap Inisialisasi
    C = len(N)
    PD = [[None]*C for _ in range(C)]
    marked = [False]*C
    n = 2 if m == 1 else 1

    # Menghitung Kekuatan Jalur Terkuat dari Alternatif m ke Alternatif Lainnya
    for i in range(C):
        if i != m:
            PD[m][i] = (N[m][i], N[i][m])
            marked[i] = False
    marked[m] = True
    for _ in range(C - 1):
        x1, x2 = PD[m][n]
        j = n
        for k in range(C):
            if not marked[k] and ((x1, x2) <= PD[m][k] or marked[j]):
                x1, x2 = PD[m][k]
                j = k
        marked[j] = True
        for k in range(C):
            if not marked[k] and PD[m][k] < min(PD[m][j], (N[j][k], N[k][j])):
                PD[m][k] = min(PD[m][j], (N[j][k], N[k][j]))

    # Menghitung Kekuatan Jalur Terkuat dari Alternatif Lainnya ke Alternatif m
    for i in range(C):
        if i != m:
            PD[i][m] = (N[i][m], N[m][i])
            marked[i] = False
    marked[m] = True
    for _ in range(C - 1):
        x1, x2 = PD[n][m]
        j = n
        for k in range(C):
            if not marked[k] and ((x1, x2) <= PD[k][m] or marked[j]):
                x1, x2 = PD[k][m]
                j = k
        marked[j] = True
        for k in range(C):
            if not marked[k] and PD[k][m] < min(PD[j][m], (N[k][j], N[j][k])):
                PD[k][m] = min(PD[j][m], (N[k][j], N[j][k]))

    # Menentukan Pemenang Potensial
    winner = True
    for i in range(C):
        if i != m and PD[i][m] >= PD[m][i]:
            winner = False

    return PD, winner
```

Gambar 3.2 Fungsi Untuk mengidentifikasi pemenang potensial
Diambil dari dokumen pribadi

Pada tahap inisialisasi matriks, jumlah operasi yang dilakukan untuk melakukan inisialisasi matriks PD dan array marked adalah $T(C) = C \times C + C = C^2 + C$. Namun, pada beberapa bahasa pemrograman, inisialisasi matriks dan array tidak terlalu berpengaruh, sehingga dapat diabaikan.

Pada saat menghitung bobot jalur terkuat dari alternatif m ke alternatif lainnya, terdapat dua proses yang berbeda. Pertama adalah looping inisialisasi matriks PD dan array marked sebelum memulai perhitungan, dan yang kedua adalah proses peritungan bobot jalur terkuat. Pada proses yang pertama, loop akan berjalan sejumlah C. Pada proses kedua, loop bagian luar berjalan C – 1 kali, dan didalam loop tersebut terdapat $2C$ operasi pada dua looping. Oleh karena itu, jumlah assignment yang dilakukan adalah sejumlah $T(C) = C + 2C(C-1) = C + 2C^2 - 2C = 2C^2 - C$.

Pada tahap selanjutnya, yaitu menghitung bobot jalur terkuat dari alternatif lain ke alternatif m, proses yang dilakukan mirip dengan tahap sebelumnya, sehingga jumlah operasi yang dilakukan adalah sejumlah $T(C) = C + 2C(C-1) = C + 2C^2 - 2C = 2C^2 - C$.

Pada tahap penentuan pemenang potensial, proses validasi pemenang dilakukan sebanyak C kali dalam loop, sehingga jumlah assignment yang dilakukan adalah sejumlah $T(C) = C$.

C. Implementasi dan Analisis Kompleksitas Algoritma dalam melakukan verifikasi hubungan khusus antar alternatif

Implementasi ini menggunakan algoritma Breadth-First Search (BFS) untuk memvalidasi hubungan tertentu antara dua pilihan spesifik, misalnya untuk membuktikan bahwa tidak ada jalur dari alternatif B ke A yang memiliki bobot yang memenuhi kriteria tertentu. Fokus utama implementasi ini adalah untuk mengkonfirmasi atau menyangkal hipotesis spesifik tentang hubungan antara alternatif, bukan untuk menganalisis seluruh set alternatif secara menyeluruh.

Fungsi ini memiliki parameter N, yang merupakan matriks yang mencatat jumlah pemilih yang lebih memilih alternatif i daripada j. Parameter lainnya adalah a dan b, yang merupakan dua alternatif yang hubungannya ingin diverifikasi. Selanjutnya, x1 dan x2 adalah bobot jalur yang telah kita temuan dari alternatif a ke b.

Tahap pertama adalah Inisialisasi. Di sini, kita menginisialisasi empat array: B1, B2, array1, dan array2. B1 dan B2 akan menjadi dua set yang kita cari. B1 akan berisi alternatif yang dapat dicapai dari alternatif b dengan bobot jalur setidaknya (x1, x2), sedangkan B2 akan berisi sisanya. array1 dan array2 digunakan untuk membantu kita mencari set tersebut. Kita menandai alternatif b sebagai bagian dari set B1 dan menambahkannya ke array1.

Selanjutnya, kita menggunakan algoritma Breadth-First Search (BFS) untuk mencari semua alternatif yang dapat dicapai dari alternatif b dengan bobot jalur setidaknya (x1, x2). Untuk setiap alternatif dalam array1, kita memeriksa semua alternatif lainnya. Jika alternatif lain tersebut belum termasuk dalam B1 dan bobot jalur dari alternatif saat ini ke alternatif lain tersebut dan kembali setidaknya (x1, x2), kita menambahkan alternatif tersebut ke B1 dan array1.

Setelah kita mendapatkan set B1, kita menghitung set B2 sebagai selisih antara set semua alternatif dan B1. Dengan kata lain, B2 berisi semua alternatif yang tidak termasuk dalam B1.

Implementasi kode di dalam bahasa python adalah sebagai berikut :

```python
# Melakukan verifikasi Hubungan Khusus antar Alternatif
def calculate_sets(N, a, b, x1, x2):
    # Mendapatkan jumlah alternatif
    C = len(N)

    # Inisialisasi array B1, B2, array1, dan array2
    B1 = [False]*C
    B2 = [True]*C
    array1 = [0]*C
    array2 = [0]*C
    B1[b] = True
    array1[0] = b
    m = 1

    while m > 0:
        n = m
        for k in range(m):
            array2[k] = array1[k]
        m = 0
        for i in range(n):
            j = array2[i]
            for k in range(C):
                if not B1[k] and (N[j][k], N[k][j]) >= (x1, x2):
                    B1[k] = True
                    array1[m] = k
                    m += 1

    # Hitung set B2 sebagai selisih antara set semua alternatif dan B1
    for i in range(C):
        B2[i] = not B1[i]

    return B1, B2
```

Gambar 3.3 Fungsi Untuk melakukan verifikasi hubungan khusus antar alternatif
Diambil dari dokumen pribadi

Inisialisasi array dapat diabaikan karena tidak berpengaruh signifikan terhadap perkembangan waktu asimptotik algoritma. Selanjutnya, loop while akan berjalan hingga tidak ada lagi elemen yang ditambahkan ke B1. Dalam kasus terburuk, loop while akan berjalan sebanyak C kali karena harus memeriksa setiap alternatif yang ada. Di dalam loop while, terdapat loop pertama sejumlah m kali, selanjutnya terdapat loop kedua sejumlah n kali yang di dalamnya terdapat loop for sebanyak C kali pada potongan kode `for k in range(C)`, sehingga jumlah operasi yang dilakukan untuk while loop adalah $T(C) = C * (m + n * C) = C(m + n * C)$.

Selanjutnya, pada saat menghitung set B2 sebagai selisih antara set semua alternatif dan B1, terdapat looping sebanyak C kali, sehingga jumlah operasi yang dilakukan adalah $T(C) = C$.

Maka, jumlah operasi secara keseluruhan adalah $T(C) = C(m + n * C) + C$.

Berdasarkan perkembangan asimptotik paling besar pada T(C), didapat kompleksitas waktu dari fungsi adalah $O(C^2)$.

V. KESIMPULAN

Makalah ini membahas secara tuntas mengenai kompleksitas algoritma yang berkaitan dengan *Schulze Method* memiliki kompleksitas yang bervariasi, yang disesuaikan dengan kebutuhan yang diinginkan. Kompleksitas Algoritma pada Algoritma Floyd-Warshall memiliki kompleksitas waktu $O(C^3)$, Kompleksitas Algoritma dalam mengidentifikasi pemenang potensial memiliki kompleksitas waktu $O(C^2)$, dan Kompleksitas Algoritma dalam melakukan verifikasi hubungan khusus antar alternatif memiliki kompleksitas waktu $O(C^2)$.

VI. UCAPAN TERIMA KASIH

Dalam pemenuhan tugas makalah untuk mata kuliah Matematika Diskrit (IF2120), penulis ingin mengawali dengan ungkapan rasa syukur kepada Tuhan yang Maha Esa atas berkah, bimbingan, serta dukungan-Nya yang memungkinkan penulis menyelesaikan makalah berjudul "Analisis Kompleksitas Algoritma dalam implementasi *Schulze Methods* berbasis teori graf pada *Preferential Single-Winner Election system*".

Penulis juga ingin mengucapkan terima kasih kepada Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc., sebagai dosen mata kuliah Matematika Diskrit IF2120 Kelas K1 yang telah dengan penuh dedikasi memberikan pengajaran dan bimbingan kepada kami, para mahasiswa, selama satu semester ini. Penulis juga ingin menyampaikan terima kasih kepada bapak Dr. Ir. Rinaldi, M.T. atas sumber pembelajaran matematika diskrit yang diberikan melalui website beliau, dan tentunya kepada semua pihak yang telah memberikan kontribusi, baik secara langsung maupun tidak langsung, dalam penulisan makalah ini. Selain itu, penulis ingin menyampaikan permohonan maaf apabila terdapat kesalahan dalam penulisan makalah ini yang sekiranya dapat menyinggung pihak tertentu.

REFERENSI

[1] https://arxiv.org/ftp/arxiv/papers/1804/1804.02973.pdf (paper "The Schulze Method of Voting" by Markus Schulze)
[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf
[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/24-Kompleksitas-Algoritma-Bagian1-2023.pdf
[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/25-Kompleksitas-Algoritma-Bagian2-2023.pdf
[5] https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16
[6] https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

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, 11 Desember 2023

Raffael Boymian Siahaan
13522046