Analisis Kompleksitas Algoritma di Fase Pindai dan Konvolusi dari Algoritma Teiresias

Aditya Prawira Nugroho - 13520049
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13520049@mahasiswa.itb.ac.id

***Abstract*—Algoritma Teiresias adalah sebuah algoritma kombinatorial yang mampu mengembalikan semua pola yang memiliki jumlah salinan yang sudah ditentukan pengguna dari masukan yang diberikan. Algoritma ini memiliki dua fase, yaitu fase pindai dan fase konvolusi. Kedua fase tersebut memiliki implementasi yang berbeda. Karena perbedaan implementasi tersebut, dapat dianalisis kompleksitas dari kedua fase dalam algoritma Teiresias sehingga dapat ditentukan fase manakah yang sangat memengaruhi kompleksitas algoritma Teiresias. Selain itu, dapat diketahui seberapa efektif algoritma ini dibandingkan dengan algoritma lainnya yang memecahkan masalah yang sama.**

***Keywords*—algoritma Teiresias, kompleksitas algoritma, fase pindai, fase konvolusi**

I. PENDAHULUAN

Bioinformatika merupakan bidang ilmu yang mempelajari penerapan teknik komputasional dalam menganalisis informasi biologis yang dimiliki makhluk hidup. Bidang keilmuan ini menerapkan teknik komputasional untuk memecahkan masalah biologis, terutama menggunakan sekuens DNA dan asam amino sehingga didapatkan informasi yang berkaitan dengannya. Salah satu produk hasil keilmuan bioinformatika adalah algoritma Teiresias.

Algoritma Teiresias diciptakan oleh Isidore Rigoustos dan Aris Floratos untuk menawarkan satu solusi terhadap salah satu permasalahan dalam analisis sekuens biologis, yaitu mencari kesamaan sekuens dalam struktur primer protein dan gen yang berkorelasi. Algoritma ini terbagi menjadi dua fase, yaitu pindai dan konvolusi. Kedua fase tersebut memiliki tujuan yang berbeda, sehingga implementasinya juga berbeda. Karena perbedaan itu, dapat dianalisis kompleksitas algoritma dari kedua fase sehingga dapat ditentukan fase manakah yang lebih kompleks waktunya dan seberapa efektif algoritma ini dibandingkan dengan algoritma lainnya yang menawarkan solusi terhadap permasalahan yang sama. Pada tulisan ini, saya akan berfokus pada kompleksitas waktunya dan hanya menyinggung sedikit kompleksitas ruang.

II. LANDASAN TEORI

A. Algoritma Teiresias

Algoritma Teiresias adalah algoritma yang berbasis penemuan pola kombinatorial yang dapat digunakan untuk mencari pola dengan panjang yang bervariasi. Algoritma ini mampu mencari semua pola yang ada dalam sebuah himpunan sekuens masukan tanpa mengenumerasi keseluruhan solusi. Pola yang diidentifikasi pada algoritma ini didefinisikan sebagai *regular expression* dalam bentuk

$$\Sigma(\Sigma + \{'.'\})^*\Sigma$$

dengan Σ = {A, C, G, T} (simbol DNA). Karakter '.' melambangkan dan merepresentasikan 'any' atau 'do not care' yang artinya karakter tersebut merepresentasikan semua *string* dari Σ. Dengan begitu, misal kita memiliki sebuah pola P, haruslah P dalam bentuk *regular expression* di atas, contoh P yang valid, yaitu ACGT, A..T, A.T.A.

Kemudian, untuk mendefinisikan lebih lanjut, sebuah pola P disebut sebagai pola <L, W>, dengan $L \leq W$, jika setiap subpola dari P dengan panjang W atau lebih memiliki setidaknya karakter didefinisikan pada ∑ dengan panjang L.

Selain itu, parameter ketiga dari pola P adalah K. Parameter K bisa merujuk kepada dua bentuk, yaitu jumlah minimal dari sekuens unik yang diinginkan dalam sebuah masukan yang diberikan atau jumlah minimal dari kemunculan total yang harus dimiliki sebuah pola.

Beberapa terminologi yang digunakan dalam algoritma Teiresias adalah *elementary pattern* dan *maximal pattern*. *Elementary pattern* adalah sebuah pola yang dihasilkan dari fase pindai. *Elementary pattern* harus berupa string dengan karakter yang didefinisikan pada ∑ dengan panjang tepat sebanyak L.

Maximal pattern adalah pola yang dihasilkan dari fase konvolusi yang diulang. Pada dasarnya, *maximal pattern* adalah kombinasi dari *elementary pattern* secara rekursif untuk membentuk pola yang lebih panjang.

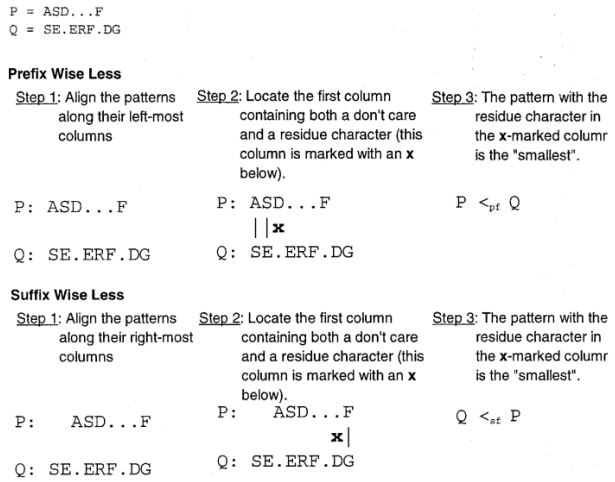
Algoritma Teiresias terdiri dari dua fase primer, yaitu pindai dan konvolusi. Fase pindai ini adalah fase pertama dari algoritma Teiresias. Secara garis besar, fase ini akan memindai sekuens masukan untuk *elementary pattern*, yaitu pola yang memenuhi parameter L, W, K, dan kemudian akan diekstraksi.

Fase kedua adalah fase konvolusi. Fase ini menerima masukan berupa *elementary pattern* yang sudah ditemukan pada fase pindai dan kemudian dikombinasikan atau dikonvolusikan untuk membentuk pola yang lebih panjang dan kompleks. Contoh, konvolusi dari *elementary pattern* A.CT dengan CT.G akan menghasilkan A.CT.G. Terlihat bahwa dua karakter terakhir dari pola pertama dan dua karakter pertama pola kedua beririsan. Irisan seperti ini terdeteksi pada fase ini dan

digunakan untuk membuat pola baru. Dengan begitu, operasi konvolusi dapat didefinisikan sebagai

$$R = P \oplus Q = \begin{cases} PQ' \; jika \; sufiks(P) == prefiks(P) \\ \emptyset \end{cases} \quad (1)$$

dengan $\oplus$ berarti konvolusi dan Q' adalah sisa dari string Q setelah prefiks(Q) dibuang. Simbol ∅ melambangkan konvolusi yang gagal yang berupa string kosong. Urutan dari konvolusi sangat penting. Pengurutan konvolusi dilakukan berdasarkan beberapa aturan dan dengan langkah-langkah pada gambar berikut



Gambar 1 Langkah-langkah untuk menentukan urutan konvolusi (Sumber: Combinatorial pattern discovery in biological sequences: the TEIRESIAS algorithm *dalam Jurnal Bioinformatik Vol. 14 no. 1 1998, hal: 58)*

Dengan mengombinasikan kedua fase ini, akan didapatkan *maximal pattern* yang bersesuaian dengan parameter L, W, K yang diberikan. Beban dari eksekusi program bisa jatuh ke fase pindai atau konvolusi bergantung pada parameter yang diberikan. Secara umum, semakin besar nilai L dan W, semakin kompleks fase pindainya dan semakin kecil nilai K, semakin kompleks fase konvolusinya.

Contoh penerapan dan permasalahan yang diselesaikan oleh algoritma Teiresias adalah seperti berikut. Misal diberikan sebuah masukan string ACGTCCACTGCATGGACGATGAT dengan parameter L = 3, W = 5, K = 2, maka akan dihasilkann pola pada tabel berikut.

Jumlah Kemunculan	Jumlah Sekuens	Pola
2	1	ACG
2	1	ATG
2	1	GA.GAT
2	1	TG.A.G
2	1	T.CA..G
2	1	AC..C
2	1	G.C.A
2	1	C..TG
2	1	T..AC

Tabel 1 Himpunan *maximal pattern* yang didapatkan setelah menjalankan algoritma Teiresias

Misal kita ambil pola T.CA..G sebagai contoh, maka dapat kita lihat bahwa fase pindai akan menghasilkan *elementary pattern*, yaitu T.CA dan CA..G. Karena parameter L = 3, maka jumlah karakter harus ada 3 dan panjang pola haruslah ≤ 5 karena W = 5. Terlihat bahwa T.CA dan CA..G beririsan, sehingga pada fase konvolusi dihasilkan pola T.CA..G.

B. Kompleksitas Algoritma

Dalam studi informatika, kompleksitas algoritma adalah seberapa besar sumber daya yang digunakan untuk mengeksekusi algoritma tersebut, terutama sumber daya waktu dan ruang. Jika algoritma semakin kompleks, maka waktu dan ruang yang dipakai juga akan semakin besar. Kompleksitas yang ditinjau dari seberapa besar ruang yang digunakan disebut sebagai kompleksitas ruang dan dilambangkan sebagai S(n).

Sebaliknya, kompleksitas algoritma yang ditinjau dari seberapa besar waktu yang digunakan, disebut kompleksitas waktu. Kompleksitas waktu biasa dilambangkan sebagai T(n) dan diukur berdasarkan berapa banyak operasi dilakukan, seperti operasi aritmetika, baca/tulis, dll. Akan tetapi, jika kita menghitung banyaknya operasi yang dilakukan, akan sangat rumit, sehingga lebih baik kita mengukur laju perubahan kebutuhan waktu sebuah algoritma jika masukan yang diberikan meningkat. Maka dari itu, kompleksitas waktu lebih sering ditulis dalam notasi O-besar (Big-O Notation).

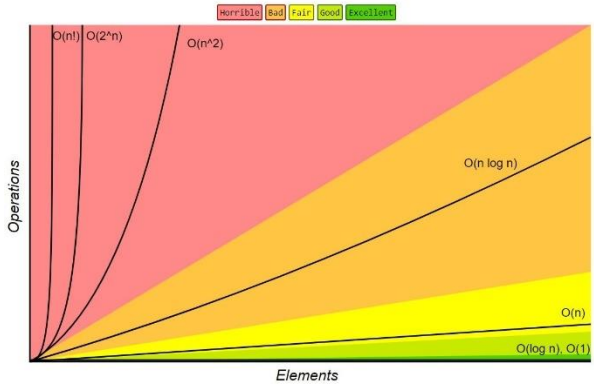
Big-O Notation dituliskan dalam bentuk O(n). Definisi dari notasi O-besar adalah T(n) = O(f(n)), yang dibaca T(n) adalah O(f(n)), jika terdapat sebuah konstanta C dan n_0 sedemikian sehingga T(n) ≤ C f(n) untuk n ≥ n_0. Maka dari itu, f(n) dapat dianggap sebagai batas atas dari T(n) untuk nilai n yang besar.

Ada beberapa pengelompokan algoritma berdasarakn notasi O-besar yang dapat disajikan dalam tabel seperti berikut

Notasi O-Besar	Nama
O(1)	Konstan
O(log n)	Logaritmik
O(n)	Linier
O(n log n)	Linier logaritmik
$O(n^2)$	Kuadratik
$O(n^3)$	Kubik
$O(2^n)$	Eksponensial
O(n!)	Faktorial

Tabel 2 Pengelompokan algoritma berdasarkan *Big O-Notation*

Dari tabel tersebut, urutan notasi O-besar dari yang laju perubahan kebutuhan waktu terkecil hingga terbesar adalah 1 < log n < n < n log n < n^2 < n^3 < … < 2^n < n!.



Gambar 2 Grafik pertumbuhan laju kebutuhan waktu notasi O-besar (Sumber: https://devopedia.org/images/article/17/4996.1513922020.jpg)

1. O(1)

O(1) memiliki arti bahwa waktu yang dibutuhkan algoritma akan selalu konstan, tidak berubah meskipun data yang dimasukan sangat besar. Contoh operasi yang memiliki notasi ini adalah pengaksesan elemen array, notasi algoritmanya sebagai berikut

```
arr: array[0…100] of integer
output(arr[44])
```

2. O(log n)

Kompleksitas O(log n) berati kebutuhan waktu algoritma bertumbuh secara logaritmik terhadap data yang dimasukan. Contoh dari algoritma yang memiliki kompleksitas O(log n) adalah *binary search*. Algoritma *binary search* dalam bahasa C adalah sebagai berikut

```
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= 1) {
      int mid = l + (r-1)/2;
      if (arr[mid] == x) {
        return mid;
      if (arr[mid] > x) {
        return binarySearch(arr, l, mid – 1, x);
      return binarySearch(arr, l, mid + 1, x);
    }
    return -1;
}
```

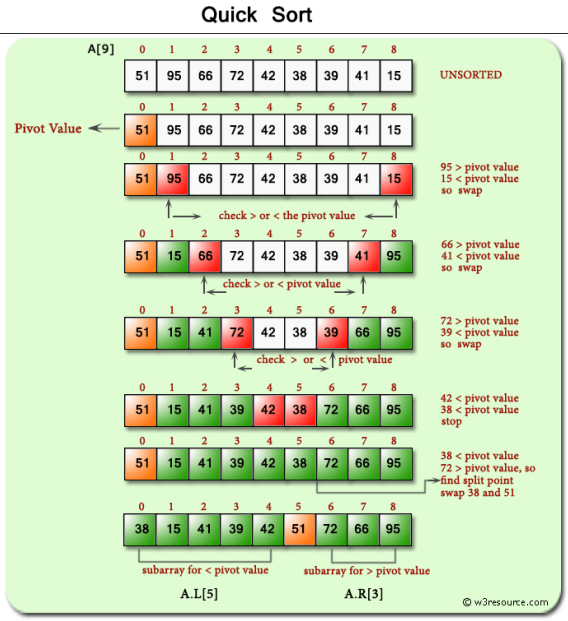
3. O(n)

Kompleksitas O(n) berarti laju pertumbuhan waktu yang dibutuhkan oleh algoritma sebanding dengan jumlah masukan. Contoh algoritma yang memiliki kompleksitas ini adalah algoritma pencarian elemen dalam array. Implementasi algoritma pencarian elemen dalam bahasa C adalah sebagai berikut

```
int elmtSearch(int arr[], int arrLenght, int X)
{
    int i = 0;
    while (i != arrLength and arr[i] != X) {
      i += 1;
    }
    if (arr[i] == X) {
      return i;
    } else {
      return -1;
    }
}
```

4. O(n log n)

Ciri khas dari algoritma dengan kompleksitas waktu ini adalah memiliki strategi *divide and conquer*, yaitu strategi algoritma yang memecahkan masalah ke masalah lebih kecil, kemudian menggabungkan semua solusinya. Contoh dari algoritma denga kompleksitas waktu O(n log n) adalah quick sort.



Gambar 3 Ilustrasi algoritma quick sort (Sumber: https://www.w3resource.com/w3r_images/quick-sort-part-1.png)

5. O(n^2)

Algoritma yang memiliki kompleksitas waktu O(n^2) biasanya memiliki 2 *nested loop* dalam kodenya. Algoritma seperti ini hanya cocok untuk ukuran masukan yang relatif kecil. Contoh algoritma dalam bahasa Python dengan kompleksitas ini adalah

```
for i in range(10):
  for j in range(10):
    A[i][j] = A[i][j] * 2
```

6. O(n^3)

Kompleksitas O(n^3) berarti laju kebutuhan waktu bertambah secara kubik terhadap ukuran masukan. Algoritma ini biasanya memiliki 3 *nested loop*. Contoh dari algoritma yang memiliki 3 *nested loop* dalam bahasa C adalah

```
for (int i = 0; i < 10; i++) {
  for (int j = 0; j < 10; j++) {
    sum = 0;
    for (int k = 0; k < 10; k++) {
      sum += Arr[j][k];
    }
  }
}
```

7. O(2^n)

Algoritma dengan kompleksitas waktu ini memerlukan banyak waktu karena setelah melakukan satu operasi, operasi setelahnya adalah pemanggilan 2 operasi lain. Contohnya adalah penentuan suku ke-n dari barisan Fibonacci dengan algoritma rekursif, yang dalam bahasa python akan terlihat seperti berikut

```
def fibonacci(n):
  if n <= 1:
```

```
    return n
  else
    return fibonacci(n - 2) + fibonacci (n – 1)
```

8. O(n!)

Sebuah algoritma yang memiliki kompleksitas waktu ini biasanya mencoba semua kemungkinan yang ada dan pada umumnya merupakan algoritma yang berbasis kombinatorial. Contoh permasalahan yang memiliki kompleksitas ini adalah *traveling salesman problem* dengan solusi *brute-force*.

III. Kompleksitas Algoritma Fase Pindai

Kita akan membahas fase pertama dari algoritma Teiresias terlebih dahulu, yaitu fase pindai. Seperti yang sudah dijelaskan di awal, fase pindai bertujuan untuk mengekstraksi *elementary pattern* dari sekues masukan yang memenuhi parameter L, W, K yang diinginkan pengguna. Secara garis besar, algoritma pada fase pindai diimplementasikan dengan *regular expression* dan *nested loop* untuk mencoba semua kemungkinan. Berikut *pseudocode* untuk algoritma pada fase pindai yang diambil dari laporan riset *On the Time Complexity of the TEIRESIAS Algorithm*

```
EP: linked list of string containing set of
elementary patterns
Si[j]: the j-th char in the i-th sequence
EP = null
for all σ in ∑                          (5)
  if support(σ) >= K
    Extend(σ)
  end if
end for

Extend(pattern P)
offset list counts(|∑|)

A = Number of regular characters in P
if A = L
  Add P to EP                           (1)
  return
end if

for i = 0 to (W * |P| * L+A)            (2)
  for all σ in ∑                        (3)
    counts(σ) = empty
  end for
  P' = P concatenated with i dots
  for (x, Y) in LS(P)                   (4)
    if (y + |P| + i) < |Sx|
      σ = Sx[y + |P| + i]
      Add (x, y+|P|+i) to counts(σ)
    end if
   end for
  for all σ in ∑                        (6)
    if support(P'σ) >= K
      Extend(P'σ)
    end if
  end for
end for
```

Terlihat dari *pseudocode* bahwa masukan akan diperiksa berulang-ulang untuk membentuk *elementary pattern* yang kemudian dimasukkan ke dalam *linked list* EP. Pada baris (1), dilakukan pengecekan jika panjang string P sama dengan L, jika sama, maka masukkan P ke EP, sehingga baris (1) membutuhkan waktu yang konstan. Selain itu, pada baris (5) dan (6) dilakukan iterasi sebanyak |∑| sehingga T(n) untuk saat ini adalah 2|∑| + 1.

Inisiasi counts(σ) pada baris (3) dilakukan sebanyak W * |P| * L+A dikali dengan |∑|, Kemudian, dapat dilihat pada loop baris (2) dan (4) bahwa masing-masing karakter dari masukan akan diperiksa sebanyak W * |P| * L+A dikali dengan |LS(P)|. Terlebih lagi, kita ketahui bahwa semua nilai kombinasi pasangan L dan W yang mungkin akan diperiksa. Untuk mempermudah menghitung semua kombinasi L dan W, kita definisikan A(L,W) sebagai jumlah dari *template* (L,W), yang artinya *elementary pattern* dengan jumlah karakter dalam ∑ sebanyak L dan panjang string maksimal 5, maka dapat dihitung bahwa tiap karakter dari masukan akan diperiksa dengan kasus terburuk sebanyak

$$A(1, W-L+1) + A(2, W-L+2) \ldots + A(L, W) = \sum A(i, W-L+i)$$

Misalkan banyak karakter masukan adalah m, maka total waktu yang akan dihabiskan adalah

$$T(n) = m\sum A(i, W-L+i) + 2|\sum| + 1 \leq mLA(L, W) \leq mLW^{L-1} \leq mW^{L}$$

Kita dapatkan f(n) = mW^L, C = 1, dan $n_0 \geq 1$ sehingga notasi O-besarnya adalah O(mW^L).

IV. Kompleksitas Algoritma Fase Konvolusi

Setelah memindai, fase berikutnya adalah konvolusi. Sudah ditentukan urutan melakukan konvolusi dan aturan-aturan konvolusi di awal. Untuk implementasi fase ini digunakan sebuah stack untuk menyimpan *elementary pattern* dan struktur data pohon. *Pseudocode* dari fase ini adalah sebagai berikut

```
Order EP according to prefix-wise less and let
all entries in EP be unmarked.
Initialize DirS and DirP
DirP(w): subset of P containing all the
elementary patterns starting with w
DirS(w): subset of P containing all the
elementary patterns ending in w
Maximal: set of maximal patterns
isMaximal(R): returns 0 if R is subsumed by a
pattern in Maximal, and 1 otherwise

Maximal = empty
S: Stack
S = empty
while EP not empty
  P = prefix-wise smallest unmarked element in
EP
  mark P
  push(P)
  while stack not empty
```

```
    start:
      T = top of stack
      # Process suffix-wise less
      w = prefix(T)
      U = {Q in DirS(w) | Q, T have not been
convolved yet}
      while U not empty
        Q = minimum element of U                (1)
        R = Q ⊕ T                               (2)
        if |Ls(R)| = |Ls(T)|
          pop stack
        end if
        if support(R) >= K && isMaximal(R)      (3)
          push(R)
          goto start
        end if
      end while

      # Process preifx-wise less
      w = suffix(T)
      U = {Q in DirP(w) | Q, T have not been
convolved yet}
      while U not empty
        Q = minimum element of U
        R = T ⊕ Q
        if |Ls(R)| = |Ls(T)|
          pop stack
        end if
        if support(R) >= K && isMaximal(R)
          push(R)
          goto start
        end if
      end while

      T = pop stack
      Add T in Maximal                          (4)
      return T

  end while
end while
```

Sebelumnya, kita harus menghitung terlebih dahulu jumlah EP (*elementary pattern)* yang dihasilkan pada fase pindai. Karena ukuran masukan atau jumlah karakter adalah m, jumlah EP yang dihasilkan setelah satu kali iterasi fase pindai maksimal adalah A(L, W), dan minimal jumlah kemunculan EP adalah K, maka |EP| selalu kurang dari atau sama dengan mA(L,W)/K.

Untuk mempercepat algoritma dan membuatnya lebih efisien, digunakan struktur data pohon biner seimbang untuk DirS dan DirP. Kemudian, dilakukan sorting terhadap EP berdasarkan urutan *prefix-wise less* atau *suffix-wise less*. Dengan menggunakan algoritma *sorting* yang standar, didapatkan kompleksitas waktu O(|EP| log|EP|) (dari bentuk O(n log n)) dan dalam *sorting* akan dilakukan perbandingan untuk mengurutkan EP sebanyak jumlah karakter pada string, yaitu W, sehingga kompleksitas waktu total adalah O(W(|EP| log|EP)). Setalah itu, dilakukan pencarian dan penyisipan ke dalam pohon dengan waktu O(log |EP|) karena DirS dan DirP adalah pohon biner yang seimbang. Karena (log |EP|) < W(|EP| log|EP), untuk pembentukan DirS, DirP, dan baris (1) dibutuhkan waktu sebesar

$$O\big(\mathrm{W}(|\mathrm{EP}|\log|\mathrm{EP})\big) = O(W\left(\frac{\mathrm{mA(L,W)}}{\mathrm{K}}\right)\log\frac{\mathrm{mA(L,W)}}{\mathrm{K}}$$
$$= O(W^L m\log m)$$

Setelah itu, barulah proses konvolusi dimulai. Pada baris (2), konvolusi akan dilakukan sebanyak karakter masukan m dikali dengan panjang dari T (*top of stack*) saat itu, yaitu W. Sehingga, konvolusi T dengan string Q adalah O(mW).

Pada baris (3), terdapat pemanggilan fungsi isMaximal. Fungsi tersebut akan mengecek apakah pola R merupakan pola yang maksimal dengan membandingkannya dengan stack. Oleh karena itu, banyak jumlah pengecekan tiap iterasi adalah sejumlah panjang stack saat itu hingga akhir iterasi, sehingga dapat kita tulis sebagai $\sum_{i=0}|S_i|$. Oleh karena itu, total waktu untuk melakukan konvolusi adalah

$$O(mW\sum_{\mathrm{i=0}}|\mathrm{S_i}|)$$

Dengan begitu, kita dapatkan total kompleksitas waktu pada fase konvolusi adalah

$$O(W^L m\log m + mW\sum_{\mathrm{i=0}}|\mathrm{S_i}|)$$

V. Perbandingan Algoritma Teiresias dengan Algoritma Lain

Dari kedua kompleksitas waktu pada fase pindai dan konvolusi, algoritma Teiresias tergolong cukup cepat karena masalah pencarian pola pada bioinformatika tergolong masalah *NP-hard* atau masalah yang hanya bisa diselesaikan dengan waktu polinomial. Beberapa algoritma pendahulunya adalah Needleman and Wunsch, Delcoigne and Hansen, Martinez, dan lain-lain. Algoritma pendahulunya cenderung memiliki kompleksitas waktu yang lebih tinggi daripada Teiresias atau tidak mampu menyelesaikan pencarian pola secara lengkap seperti Teiresias.

Seperti contoh, algoritma Needleman and Wunsch memiliki kompleksitas

$$F_{ij} = \max_{h<i,k<j}\{F_{h,j-1} + S(A_i, B_j), F_{i-1,k} + S(A_i, B_j)\}$$

Gambar 4 Kompleksitas algoritma Needleman and Wunsch (Sumber: https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm#Complexity)

Algoritma tersebut menggunakan rekursif dan *dynamic programming* sehingga waktu yang diperlukan sebesar kubik terhadap masukan.

Terlihat bahwa algoritma Teiresias lebih cepat dibandingkan algoritma Needleman and Wunsch karena waktu terbesar pada algoritma Teiresias berupa log dan perkalian dibandingkan dengan waktu kubik yang dimiliki Needleman and Wunsch. Selain itu, hasil dari algoritma Teiresias lebih lengkap dibandingkan dengan Needleman and Wunsch.

VI. KESIMPULAN

Setelah dianalisis, terlihat bahwa fase yang memakan waktu paling banyak adalah fase konvolusi, dengan kompleksitas waktu

$$O(W^L m \log m + mW \sum_{i=0} |S_i|)$$

yang lebih besar dibandingkan dengan kompleksitas waktu pada fase pindai, yaitu $O(mW^L)$. Oleh karena itu, keseluruhan algoritma Teiresias memiliki kompleksitas waktu yang sama dengan kompleksitas waktu fase konvolusi.

Mempertimbangkan masalah pencarian pola yang mirip adalah masalah kombinatorial yang pada umumnya memiliki kompleksitas waktu O(n!), algoritma Teiresias tergolong efisien dan cepat dibandingkan dengan O(n!). Selain itu, algoritma ini juga lebih cepat dibandingkan algoritma pencarian pola bioinformatika lainnya.

VII. UCAPAN TERIMA KASIH

Penulis bersyukur kepada Tuhan Yang Maha Esa karena telah melimpahkan rahmat-Nya sehingga penulis mampu menyelesaikan makalah berjudul "Analisis Kompleksitas Algoritma di Fase Pindai dan Konvolusi dari Algoritma Teiresias". Penulis mengucapkan terima kasih kepada orang tua, bapak/ibu dosen, teman-teman, dan sahabat penulis yang selalu memberikan dukungan, motivasi, dan masukan sehingga penulis mampu menyelesaikan makalah dengan baik.

REFERENSI

[1] Floratos, Aris., Rigoutsos, Isidore. 1998. "On the Time Complexity of the TEIRESIAS Algorithm". New York: IBM.

[2] Floratos, Aris dan Rigoutsos, Isidore. 1998. "Combinatorial pattern discovery in biological sequences: the TEIRESIAS algorithm". *Bioinformatics, Vol. 14 No. 1.* New York: IBM.

[3] Nau, Lee J. 2011. "A Scalable, Memory Efficient Multicore TEIRESIAS Implementation". Ohio: Ohio University.

[4] Munir, Rinaldi. 2021. "Kompleksitas Algoritma (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kompleksitas-Algoritma-2020-Bagian1.pdf (Diakses pada tanggal 6 Desember 2021 Pukul 19.20 WIB)

[5] Munir, Rinaldi. 2021. "Kompleksitas Algoritma (Bagian 2)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kompleksitas-Algoritma-2020-Bagian2.pdf (Diakses pada tanggal 6 Desember 2021 Pukul 19.20 WIB)

[6] Needleman, Saul B. dan Wunsch, Christian D. 1970. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". *Journal of Molecular Biology*. **48** (3): 443–53.

[7] Susilawati dan Bachtiar, N. 2018. *Biologi Dasar Terintegrasi*: 4. Pekanbaru: Kreasi Edukasi.

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, 5 Desember 2021

Aditya Prawira Nugroho 13520049