Penerapan Pencocokan *String* dalam Tes DNA *Ancestry*

Nicholas Wijaya 13516121[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1] *nic.nicholas1907@gmail.com*

***Abstract*—DNA test are often performed to obtain some information. One of them is to find relation between two DNA, including ancestry test. This paper aims to show how string matching algorithm can be used to obtain relation between two DNA by matching its nitrogen bases.**

***Keywords*—DNA test, string matching, ancestry**

I. PENDAHULUAN

DNA (*deoxyribonucleic acid*), merupakan material yang terdapat pada tubuh makhluk hidup yang diturunkan secara turun-temurun, termasuk pada manusia. Hampir setiap sel yang terdapat pada tubuh manusia mengandung DNA yang sama. [1] [2] mengatakan bahwa DNA yang dimiliki oleh seorang anak merupakan campuran dari bagian-bagian DNA kedua orang tuanya (ayah dan ibunya).

Gambar 1.1 DNA
sumber: kor.pngtree.com

DNA memiliki banyak informasi mengenai individu, seperti gender, warna rambut, warna mata, ADHD, serta sifat-sifat termasuk penyakit yang diderita oleh manusia. [3] DNA juga dapat menunjukkan orang tua kandung dari seseorang. Orang dapat melakukan tes untuk mengetahui apakah seseorang merupakan anak/orang tua kandungnya. Tes tersebut disebut *Ancestry DNA Test*.

II. TEORI DASAR

A. Struktur DNA

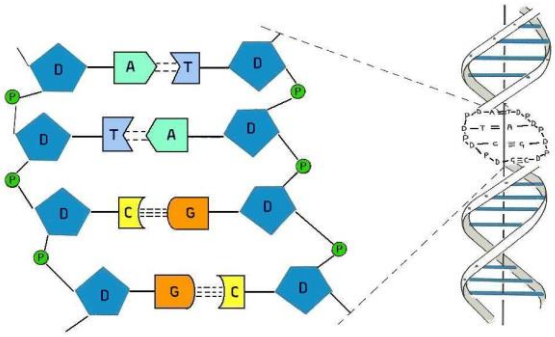
DNA terdiri atas dua utas benang *polinukleotida* yang saling berpilin membentuk heliks ganda (*double helix*). Benang DNA tersebut tersusun atas rangkaian nukleotida. Setiap nukleotida tersusun atas: [4]

1. Gugusan gula deoksiribosa
2. Gugusan asam fosfat
3. Gugusan basa nitrogen.

Basa nitrogen penyusun DNA terdiri dari dua jenis, yaitu basa purin, yaitu adenin (A) dan guanin (G), serta basa pirimidin yaitu sitosin (C) dan timin (T). Ikatan antara basa nitrogen dan gula pentosa disebut nukleosida. [4]

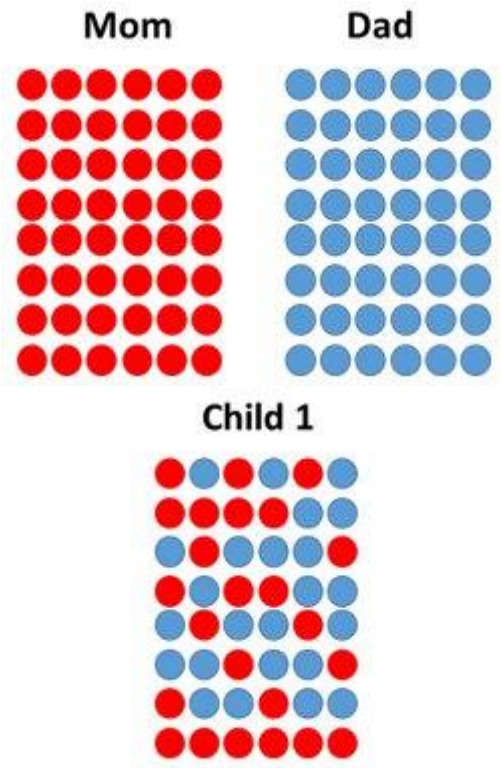
Basa-basa nitrogen pada utas benang yang satu berpasangan dengan basa-basa nitrogen pada utas benang yang lain. Setiap jenis basa nitrogen pada satu benang memiliki jenis pasangan yang tetap pada benang yang lain. Pasangan yang terjadi yaitu: [4]

1. Adenin dengan Timin (A dengan T)
2. Guanin dengan Sitosin (G dengan C)



Gambar 2.1 Basa Nitrogen DNA
sumber: https://bikifi.com/biki/kalitim-ve-genetik-dnanin-yapisi

Susunan basa nitrogen dari DNA pada tubuh manusia merupakan gabungan dari susunan basa nitrogen dari DNA ayah dan ibunya. Anak menerima bagian-bagian dari DNA ayah dan ibunya secara acak. Sebagai contoh, lihat Gambar 2.2. [2]



Gambar 2.2 DNA Anak Merupakan Campuran dari DNA Ayah dan Ibunya

sumber: http://genetics.thetech.org/ask-a-geneticist/same-parents-different-ancestry

Dengan mengetahui hal ini, manusia dapat mengetahui garis keturunan mereka dengan melakukan pencocokan pola susunan basa nitrogen pada DNAnya dengan DNA orang tuanya/anaknya.

Dalam pencocokan DNA, terdapat suatu istilah yaitu *centimorgan* (cM). *Centimorgan* merupakan istilah satuan pada genetika yang merupakan ukuran frekuensi rekombinasi. Satu centimorgan sama dengan 1% kemungkinan bahwa penanda pada satu lokus genetik akan dipisahkan dari penanda pada lokus kedua karena menyeberang dalam satu generasi. *Centimorgan* pada setiap spesies makhluk hidup berbeda-beda. Pada manusia, 1 *centimorgan* rata-rata yaitu setara dengan sekitar 1 juta pasangan basa. [11] Walaupun begitu, jumlah *centimorgan* pada pria dan wanita berbeda. Genom wanita memiliki panjang 4782 *centimorgan*, sedangkan genom pria hanya 2809 *centimorgan*. [12]

Jumlah dari *centimorgan* dapat memprediksi hubungan antar manusia. Tabelnya adalah sebagai berikut. [13]

Aproksimasi Jumlah Kecocokan DNA (*centimorgan* [cM])	**Hubungan yang Mungkin**
3.475 cM	orang tua, anak, atau kembar identik
2.400-2.800 cM	saudara kandung
1.450-2.050 cM	keluarga dekat: kakek, nenek, cucu, paman, bibi, keponakan
680-1.150 cM	sepupu dengan kakek/nenek yang sama
200-620 cM	sepupu dengan kakek/nenek buyut yang sama
90-180 cM	sepupu dengan ayah/ibu dari kakek/nenek buyut yang sama
20-85 cM	sepupu dengan kakek/nenek dari kakek/nenek buyut yang sama
6-20 cM	sepupu jauh

Tabel 2.1 Tabel Hubungan Centimorgan dengan Hubungan yang Mungkin

B. Pencocokan String (*String Matching*)

Pencocokan *String* atau yang biasa disebut *String Matching* merupakan algoritma yang digunakan untuk mencari seluruh kemunculan sebuah *string* (*pattern*) tertentu pada sebuah *string* yang lebih besar atau teks. Contohnya yaitu saat kita menekan [ctrl+f] untuk melakukan pencarian kata pada peramban atau aplikasi pengedit teks. [5]

Beberapa contoh *string matching*:

1. *Pattern*: ikan
   Teks: Saya suka makan **ikan**. Ibu suka membakar **ikan** untuk saya.
2. *Pattern*: apa
   Teks: Si**apa**kah yang menemukan lampu?

Banyak algoritma *string matching* yang telah ditemukan dan dikembangkan oleh orang-orang. Jenis-jenis dari algoritma *string matching* ini dapat diklasifikasikan berdasarkan berbagai klasifikasi, salah satunya yaitu berdasarkan strategi yang digunakan. Berdasarkan strategi yang digunakan, *string matching* dapat diklasifikasikan menjadi: [6]

1. *From left to right*
   Strategi ini melakukan pencocokan *string* dimulai dari huruf paling kiri pada *pattern* yang kemudian dilanjutkan hingga ke huruf paling kanan.
   Algoritma yang termasuk ke dalam klasifikasi *from left to right* yaitu algoritma
   a. Brute Force
   b. Karp and Rabin
   c. Shift Or
   d. Morris and Pratt
   e. Knuth, Morris and Pratt
   f. Deterministic Finite Automaton
   g. Forward Dawg Matching
   h. Apostolico-Crochemore
   i. Not So Naïve
2. *From right to left*
   Strategi ini melakukan pencocokan *string* dimulai dari huruf paling kanan pada *pattern* yang kemudian dilanjutkan hingga ke huruf paling kiri.
   Algoritma yang termasuk ke dalam klasifikasi *from left to right* yaitu algoritma
   a. Boyer-Moore
   b. Apostolico and Giancarlo
   c. Crochemore et alii (Turbo BM)
   d. Colussi (Reverse Colussi)
   e. Quick Search
   f. Reverse Factor

g. Turbo Reverse Factor
h. Zhu Takaoka
i. Berry-Ravindran

3. *In a specific order*
   Strategi ini melakukan pencarian *string* dengan melakukan partisi dari pada teks yang akan di-iterasi.
   Algoritma yang termasuk ke dalam klasifikasi *in a specific order* yaitu algoritma
   a. Two Way
   b. Colussi
   c. Galil-Giancarlo
   d. Optimal Mismatch
   e. Maximal Shift
   f. Skip Search
   g. KMP Skip Search
   h. Alpha Skip Search
4. *In any order*
   Algoritma yang termasuk ke dalam klasifikasi *in any order* yaitu algoritma
   a. Horspool
   b. Tuned Boyer-Moore
   c. Smith
   d. Raita

Pada makalah ini, hanya cara kerja dari beberapa algoritma yang cukup terkenal yang akan dibahas.

Algoritma yang paling mudah yaitu algoritma Brute Force. Brute Force melakukan pencarian secara naïve dari kiri ke kanan dan di-iterasi per huruf. Cara kerjanya sebagai berikut.

Pattern: o wo
Teks: hello world!

1. hello world!
   ≠
   o wo
2. hello world!
    ≠
    o wo
3. hello world!
     ≠
     o wo
4. hello world!
      ≠
      o wo
5. hello world!
       =
       o wo
6. hello world!
        =
        o wo
7. hello world!
         =
         o wo
8. hello world!
          =
          o wo

Implementasi dari brute force ini:

```
function sub_string($pattern, $subject)
{
        $n = strlen($subject);
        $m = strlen($pattern);
        for ($i = 0; i < $n-$m; $i++) {
                $j = 0;
                while ($j < $m &&
$subject[$i+$j] == $pattern[$j]) {
                        $j++;
                }
                if ($j == $m) return $i;
        }
        return -1;
} [7]
```

Kompleksitas waktu dari algoritma Brute Force ini:
Kasus terburuk: O(mn)
Kasus terbaik: O(n)
Kasus rata-rata: O(m+n)
dengan n = panjang *string* teks dan m = panjang *string pattern*.

Algoritma *string matching* pertama yang memiliki waktu linear yaitu algoritma Morris and Pratt, yang kemudian disempurnakan oleh Knuth, Morris and Pratt yang sering disebut algoritma Knuth-Morris-Pratt (KMP). Cara kerja dari algoritma KMP ini sebagai berikut. [8]

Pattern: atcacatcatca
Teks: gatcgatcacatcatcacgaaaaa

1. *Preprocess*: mencari fungsi pinggiran KMP dari *pattern*.

	a	t	c	a	c	a	t	c	a	t	c	a
j	1	2	3	4	5	6	7	8	9	10	11	12
f	0	0	0	1	0	1	2	3	4	2	3	4

   Arti dari f(9) = 4 yaitu
   - cari ukuran *prefix* terbesar dari P[1..9] (“atcacatca”) yang merupakan *suffix*-nya
   - cari ukuran dari “atca” = 4
2. Memulai pencarian *pattern* dalam teks. KMP melakukan pengecekan *string* dari kiri ke kanan.
   gatcgatcacatcatcacgaaaaa
   atcacatcatca
    atcacatcatca
       atcacatcatca
        atcacatcatca (matched)

Implementasi program pada bahasa Python dari algoritma KMP ini:

```
def part(pattern):
    """
    find suffix and prefix match table for pattern
    :param pattern:
    :return: table
    """
    table = [0]

    for i in range(1, len(pattern)):
        j = table[i-1]
        while j > 0 and pattern[j] != pattern[i]:
            j = table[j-1]
        table.append(j+1 if pattern[j] ==
pattern[i] else j)

    return table


def kmp(text, pattern):
    """
```

```
    kmp pattern matching algorithm
    :param text:
    :param pattern:
    :return: index of first occurrence of pattern
in text
    """
    if len(pattern) > len(text):
        return -1
    else:
        pattern = pattern.lower()
        text = text.lower()
        table = part(pattern)
        j = 0
        sum = 0

        for i in range(len(text)):
            if text[i] == pattern[j]:
                if j == len(pattern)-1:
                    return i - len(pattern) + 1
                else:
                    j = j+1
            else:   # text[i] != pattern[j]
                j = table[j-1]
                i = i + len(pattern) - j
    return -1
```

Kompleksitas waktu dari algoritma KMP ini: O(m+n) dengan n = panjang *string* teks dan m = panjang *string pattern*.

Algoritma Boyer-Moore merupakan algoritma yang banyak digunakan untuk pengembangan algoritma-algoritma *string matching* lainnya. Variasi dari algoritma Boyer-Moore sangat banyak, contohnya Turbo Boyer-Moore, Quick Search, Zhu and Takaoka, Berry-Ravindran, Apostolico and Giancarlo, Colussi, Tuned Boyer-Moore, dan Horspool. Algoritma Boyer-Moore merupakan algoritma *from right to left*. Terdapat dua jenis pergeseran pada algoritma Boyer-Moore: [9]

1. *Good-suffix shift* (*matching shift*)
   Good-suffix shift memosisikan bagian *string* dari teks yang telah cocok (misalnya m) dengan bagian *string* dari *pattern* yang paling kanan (setelah bagian teks yang cocok). Contohnya:

```
          v
...abcdabceabcfabc...
   ...xabcfabcfabc
       ...xabcfabcfabc
```

   Apabila tidak ada bagian yang cocok seluruhnya, maka *pattern* akan digeser hingga bagian *prefix* dari *pattern* yang cocok dengan *suffix*-nya. Contohnya:

```
       v
...robocab....
    abacab
        abacab
```

2. *Bad-character shift* (*occurrence shift*)
   Bad-character shift menggeser *pattern* agar karakter dari teks yang tidak cocok hingga kemunculan terakhir dari karakter tersebut pada *pattern* (sebelum bagian dari *pattern* yang telah dicocokkan). Contohnya:

```
          v
...abcdabceabcfabc...
   ...eabcfabcfabc
       ...eabcfabcfabc
```

   Bila tidak ada, akan digeser hingga satu posisi setelah *bad-character* tersebut. Contohnya:

```
       v
...robocab....
    abacab
       abacab
```

Saat terjadi ketidakcocokan, akan dibandingkan *bad-character shift* dengan *good-character shift* lalu dipilih yang paling baik.

Berikut contoh dari *string matching* dengan algoritma Boyer-Moore:

Pattern: EXAMPLE
Teks: SSIMPLE EXAMPLE
Langkah-langkah penyelesaian:

1. `SSIMPLE EXAMPLE`
   `EXAMPLE`
2. `SSIMPLE EXAMPLE`
   `EXAMPLE`
3. `SSIMPLE EXAMPLE`
   `EXAMPLE`
4. `SSIMPLE EXAMPLE`
   `EXAMPLE`

Implementasi program pada bahasa Python dari algoritma Boyer-Moore ini:

```
def heuristic(string, size):
    """
    creating list for heuristic that will be used
in Boyer-Moore string matching
    :param string: pattern
    :param size: size of pattern
    :return: heuristic table
    """
    # Initialize all occurence as -1 in ASCII char
array
    bad_char = [-1] * 256

    # Fill the actual value of last occurence
    for i in range(size):
        bad_char[ord(string[i])] = i

    # return initialized list
    return bad_char


def boyer_moore(big_string, pattern):
    """
    string matching with boyer-moore algorithm
    :param big_string:
    :param pattern:
    :return: index of first occurrence of pattern
in big_string
    """
```

```
    big_string = big_string.lower()
    pattern = pattern.lower()
    len_s = len(big_string)
    len_p = len(pattern)

    bad_char = heuristic(pattern, len_p)

    if len_s < len_p:
        return -1
    else:
        shift = 0
        while shift <= len_s - len_p:
            j = len_p - 1

            while j >= 0 and pattern[j] ==
big_string[shift + j]:
                j -= 1

            if j < 0:
                return shift
            else:
                shift += max(1, j -
bad_char[ord(big_string[shift + j])])

    return -1
```

Kompleksitas waktu dari algoritma Boyer-Moore ini:

Preprocessing: O(m+σ)

Pencarian *string*:

- Kasus terbaik: O(n/m)
- Kasus terburuk: O(mn)

dengan n = panjang *string* teks dan m = panjang *string pattern*. [10]

III. Tes DNA *Ancestry* dengan Pencocokan *String*

Langkah pertama dari pencocokan DNA ini dilakukan dengan mengambil sampel DNA dari ayah/ibu dan anaknya. Setelah itu barulah dilakukan pencocokan DNA antara anak dengan ayah/ibu serta dilakukan penghitungan jumlah basa nitrogen yang cocok. Pada pembahasan ini, akan digunakan contoh DNA ayah dan anak sebagai berikut.

Ayah:

CACTCTCAGCTAGATGATCGTGACTACTACTAGCTAGCTACGATCA
GCGACTGATCGTGAGCGCGTAGATTACGGCGCCCATGCTACTAGTA
GTCACTACTGACTGACTACGTAGTACGATCACGACGAGCTAGATCG

Anak:

CTAGCTAGCTACGATCAATCGATCATCAGCTCACTCTCAGCTAGAT
TAGCATCAGTCAGTACGACGAGCTAGATCGTAGATCGACTACGAAG
TAGTCACTACTGACGATCATCAGTAGTATGCAGCGCGTAGATTACG

dengan basa nitrogen yang berwarna merah adalah milik ayah dan yang berwarna biru adalah milik ibu.

A. Menghitung Jumlah Kecocokan DNA dengan Pencocokan String

Awalnya dilakukan pencocokan sejumlah blok DNA anak dengan DNA ayah. Yang menjadi *pattern* pada pencocokan *string* ini adalah bagian dari DNA anak, dan yang menjadi teksnya adalah DNA ayah.

Langkah yang dilakukan:

1. Mencari bagian dari DNA ayah yang cocok dengan DNA anak
2. Mencocokan sejumlah bagian dari DNA anak yang cocok dengan DNA ayah sekaligus menghitung jumlah basa nitrogen yang cocok
   i. Menghilangkan bagian dari DNA yang telah dicocokkan pada DNA ayah serta DNA anak
   ii. Bila ada bagian dari DNA anak yang tidak cocok dengan bagian manapun dari DNA ayah, maka dihilangkan.
   Bagian dari DNA dihitung tidak cocok bila jumlah basa nitrogen dari bagian tersebut kurang dari jumlah tertentu. Misalnya pada pembahasan kali ini digunakan jumlah 8.
3. Mengulang langkah 1-2 hingga DNA anak telah habis dicocokkan.

Pada contoh dari DNA ayah dan anak di atas, maka langkahnya sebagai berikut.

1. Awalnya jumlah cocok diinisasi dengan 0.
   Ayah:
   CACTCTCAGCTAGATGATCGTGACTACTACTAGCTAGCTACGA
   TCAGCGACTGATCGTGAGCGCGTAGATTACGGCGCCCATGCTA
   CTAGTAGTCACTACTGACTGACTACGTAGTACGATCACGACGA
   GCTAGATCG
   Anak:
   CTAGCTAGCTACGATCAATCGATCATCAGCTCACTCTCAGCTA
   GATTAGCATCAGTCAGTACGACGAGCTAGATCGTAGATCGACT
   ACGAAGTAGTCACTACTGACGATCATCAGTAGTATGCAGCGCG
   TAGATTACG
   Bagian CTAGCTAGCTACGATCA ditemukan pada DNA ayah. Hilangkan bagian tersebut. Kemudian jumlah cocok ditambahkan dengan panjang bagian tersebut yaitu 17.
2. Jumlah cocok = 17
   Ayah:
   CACTCTCAGCTAGATGATCGTGACTACTAGCGACTGATCGTGA
   GCGCGTAGATTACGGCGCCCATGCTACTAGTAGTCACTACTGA
   CTGACTACGTAGTACGATCACGACGAGCTAGATCG
   Anak:
   ATCGATCATCAGCTCACTCTCAGCTAGATTAGCATCAGTCAGT
   ACGACGAGCTAGATCGTAGATCGACTACGAAGTAGTCACTACT
   GACGATCATCAGTAGTATGCAGCGCGTAGATTACG
   Bagian ATCGAT hanya memiliki panjang 6 dan tidak memiliki kecocokan, sehingga A dihapus.
3. Jumlah cocok = 17
   Ayah:
   CACTCTCAGCTAGATGATCGTGACTACTAGCGACTGATCGTGA
   GCGCGTAGATTACGGCGCCCATGCTACTAGTAGTCACTACTGA
   CTGACTACGTAGTACGATCACGACGAGCTAGATCG
   Anak:
   TCGATCATCAGCTCACTCTCAGCTAGATTAGCATCAGTCAGTA
   CGACGAGCTAGATCGTAGATCGACTACGAAGTAGTCACTACTG
   ACGATCATCAGTAGTATGCAGCGCGTAGATTACG
   Bagian TCGAT hanya memiliki panjang 5 dan tidak memiliki kecocokan, sehingga T dihapus. Hal ini terus dilakukan hingga bagian biru hilang sebab tidak ada bagian yang

cocok.

4. Jumlah cocok = 17
   Ayah:
   CACTCTCAGCTAGATGATCGTGACTACTAGCGACTGATCGTGA
   GCGCGTAGATTACGGCGCCCATGCTACTAGTAGTCACTACTGA
   CTGACTACGTAGTACGATCACGACGAGCTAGATCG
   Anak:
   CACTCTCAGCTAGATTAGCATCAGTCAGTACGACGAGCTAGAT
   CGTAGATCGACTACGAAGTAGTCACTACTGACGATCATCAGTA
   GTATGCAGCGCGTAGATTACG
   Bagian CACTCTCAGCTAGAT ditemukan pada DNA ayah. Hilangkan bagian tersebut. Kemudian jumlah cocok ditambahkan dengan panjang bagian tersebut yaitu 15. Sehingga jumlah cocok menjadi 17+15 = 32.
5. Jumlah cocok = 32
   Ayah:
   GATCGTGACTACTAGCGACTGATCGTGAGCGCGTAGATTACGG
   CGCCCATGCTACTAGTAGTCACTACTGACTGACTACGTAGTAC
   GATCACGACGAGCTAGATCG
   Anak:
   TAGCATCAGTCAGTACGACGAGCTAGATCGTAGATCGACTACG
   AAGTAGTCACTACTGACGATCATCAGTAGTATGCAGCGCGTAG
   ATTACG
   Bagian TAGCA hanya memiliki panjang 5 dan tidak memiliki kecocokan, sehingga T dihapus. Ini juga terus dilakukan hingga bagian biru hilang.
6. Jumlah cocok = 32
   Ayah:
   GATCGTGACTACTAGCGACTGATCGTGAGCGCGTAGATTACGG
   CGCCCATGCTACTAGTAGTCACTACTGACTGACTACGTAGTAC
   GATCACGACGAGCTAGATCG
   Anak:
   ACGACGAGCTAGATCGTAGATCGACTACGAAGTAGTCACTACT
   GACGATCATCAGTAGTATGCAGCGCGTAGATTACG
   Bagian ACGACGAGCTAGATCG ditemukan pada DNA ayah. Hilangkan bagian tersebut. Kemudian jumlah cocok ditambahkan dengan panjang bagian tersebut yaitu 16. Sehingga jumlah cocok menjadi 32+16 = 48.

   Hal ini terus dilakukan hingga DNA anak telah habis dicocokkan dengan DNA ayah. Sehingga pada akhirnya akan didapatkan bahwa jumlah cocok = 77.

Setelah didapatkan jumlah cocok dari basa nitrogen ayah dan anak, jumlah *centimorgan* dari kecocokan dicari untuk menentukan kemungkinan hubungan antara kedua DNA yang telah dicocokkan.

B. Menghitung Jumlah Centimorgan

Pada bagian IIA, telah disebutkan bahwa pada manusia, rata-rata dari satu *centimorgan* adalah satu juta pasangan basa. Hasil dari perhitungan jumlah kecocokan basa nitrogen dari bagian sebelumnya dibagi dengan satu juta.

$$centimorgan_{manusia} = \frac{jumlah\ kecocokan}{1.000.000}$$

Jumlah dari kecocokan pada bagian A pada contoh hanya memiliki jumlah cocok 77, namun pada kenyataannya jumlahnya mencapai milyaran. Setelah dihitung, kita menggunakan Tabel 2.1 pada bagian II untuk menentukan hubungan dari DNA yang telah dihitung pada bagian A. Apabila hasil dari penghitungan *centimorgan* mencapai sekitar 3.475 cM, maka kedua DNA tersebut benar merupakan ayah/ibu dan anak.

IV. KESIMPULAN

DNA dari tubuh makhluk hidup mengandung banyak informasi yang dapat diambil, salah satunya yaitu hubungan antar dua DNA. Pencocokan basa nitrogen pada DNA dapat digunakan untuk menentukan hubungan antar dua orang, termasuk apakah kedua orang tersebut memiliki hubungan ayah/ibu dan anak. Tes DNA tersebut disebut tes DNA Ancestry. *String matching* merupakan suatu teknik yang dapat digunakan untuk mencocokkan basa nitrogen dari dua DNA.

V. UCAPAN TERIMA KASIH

Saya mengucapkan terima kasih kepada Tuhan Yang Maha Esa atas berkat dan rahmatnya sehingga saya dapat menyelesaikan makalh ini dengan baik dan tepat waktu. Saya juga mengucapkan terimakasih kepada Ibu Masayu selaku dosen IF2211 Strategi Algoritma kelas K01. Terakhir saya juga mengucapkan terima kasih kepada orang tua serta teman-teman saya yang telah memberikan saya dukungan dalam pembuatan makalah ini baik dalam wujud perbuatan nyata maupun doa.

REFERENSI

[1] https://ghr.nlm.nih.gov/primer/basics/dna
[2] http://genetics.thetech.org/ask-a-geneticist/same-parents-different-ancestry
[3] https://www.theguardian.com/science/2008/apr/27/genetics.cancer
[4] https://desybio.wordpress.com/2010/03/29/struktur-dna/
[5] https://catatananalgo.wordpress.com/2016/10/02/algoritma-string-matching-pencocokan-string/
[6] http://www-igm.univ-mlv.fr/~lecroq/string/node2.html
[7] http://www.stoimen.com/blog/2012/03/27/computer-algorithms-brute-force-string-matching/
[8] https://www.slideshare.net/thinkphp/string-kmp
[9] https://stackoverflow.com/questions/13175739/what-are-the-shift-rules-for-boyer-moore-string-search-algorithm
[10] http://www-igm.univ-mlv.fr/~lecroq/string/node14.html
[11] https://web.archive.org/web/20120717121400/http://rarediseases.info.nih.gov/Glossary.aspx?acronym=False#C
[12] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC52322/
[13] https://blogs.ancestry.com/ancestry/2016/05/03/the-science-behind-a-more-precise-dna-matching-algorithm/

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, 14 Mei 2018

Nicholas Wijaya
13516121