Pemanfaatan Pencocokan String dalam Prediksi Teks

Yahya 13518029
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518029@std.stei.itb.ac.id

***Abstract*—Pada saat ini banyak orang tidak luput dari gawai. Pada saat mengetik tentunya tidak jarang sering terjadi kesalahan atau bahkan bingung untuk menentukan kosa kata yang sesuai. Maka dari itu prediksi teks yang terdapat pada saat mengetik menjadi alat bantu yang dapat berguna untuk menyelesaikan permasalahan tersebut. Prediksi teks dapat dilakukan dengan pencocokan string terhadap setiap kosa kata pada suatu bahasa.**

Kata kunci—Prediksi teks; KMP; Boyer-Moore; Regex; Trie

I. PENDAHULUAN

Pada zaman sekarang ini, kita tidak jarang melihat orang yang sedang memainkan gawainya. Baik itu di rumah, di lingkungan sekolah, atau bahkan di tempat-tempat umum kita tidak jarang melihat orang yang berjalan sambal menatap layar gawainya. Hal itu bisa jadi dikarenakan banyak sekali yang dapat kita lakukan dengan gawai. Banyak sekali informasi yang bisa kita dapatkan dari genggaman tangan. Dan juga kita bisa tetap terhubung dengan orang lain.

Pada saat memainkan gawai, kita biasa menggunakan satu tangan ataupun dua tangan. Hal itu tergantung kebiasaan tiap orang. Tentunya menggunakan dua tangan dapat memudahkan kita dalam mengetik, tetapi ada suatu kondisi ketika kita harus mengetik dengan satu tangan. Untuk mengetik dengan satu tangan itu bukan hal yang mudah, berbagai gawai menyediakan mode satu tangan untuk memudahkan penggunanya mengetik ketika menggunakan satu tangan.

Selain itu, adanya fitur prediksi teks pada saat pengetikan juga sangat membantu. Fitur prediksi teks dapat meningkatkan kecepatan kita dalam mengetik apabila mengalami kesulitan dalam mengetik. Tentunya akan sangat membantu apabila kita dapat langsung melengkapi kata yang kita ingingkan tanpa perlu mengetiknya secara penuh dan langsung disediakan oleh sistem. Fitur prediksi teks tersebut menggunakan pencocokkan string terhadap kosa kata yang ada pada suatu bahasa.

II. DASAR TEORI

A. *String*

String adalah sebuah tipe data yang digunakan dalam pemrograman sama seperti integer ataupun float yang menyatakan tipe data suatu variabel. String merepresentasikan suatu teks, bukan bilangan. String merupakan kumpulan dari karakter yang diletakkan secara sekuensial.

Jika terdapat sebuah String S dengan panjang n, dengan string tersebut didefinisikan sebagai berikut :

$$S = x_0x_1..x_{n-1}$$

Kemudian prefix dari string tersebut adalah *substring* dari string yaitu S[0..k] dengan k mulai dari 0 sampai n-1. Apabila terdapat string S = “Yahya”. Maka kemungkinan prefix yang ada yaitu “Y”, “Ya”, “Yah”, “Yahy”, dan “Yahya”.

Suffix dari sebuah string adalah *substring* dari string S yaitu S[k..n-1] dengan k mulai dari 0 sampai n-1. Apabila terdapat string S = “Yahya”. Maka kemungkinan suffix yang ada yaitu “Yahya”, “ahya”, “hya”, “ya”, dan “a”.

B. *Pencocokan String*

Pencocokan string adalah apabila terdapat dua buah string yaitu string teks dan string *pattern*. Maka akan dilakukan pencocokan string *pattern* terhadap string teks, mencari apakah string *pattern* terdapat pada string teks. Untuk melakukan pencocokan string, terdapat berbagai macam algoritma. Pencocokan string dapat dilakukan dengan algoritma *Brute Force*, Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), *Regular Expression* (Regex) dan juga dengan menggunakan struktur data Trie.

1. *Brute Force*

Jika terdapat dua buah string yaitu string teks dan juga string *pattern* dengan panjang string teks adalah n, sedangkan panjang string *pattern* adalah m. Panjang string teks jauh lebih besar dibanding panjang string *pattern* atau dengan kata lain n >> m. Pencocokan string dengan menggunakan algoritma *brute force* ini akan melakukan iterasi dimulai dari karakter pertama string teks lalu akan membandingkan string-string berikutnya dengan string *pattern*. Apabila terjadi *mismatch* antara string *teks* dengan string *pattern*, maka algoritma ini akan memulai lagi pencocokan string dari tepat satu karakter setelah awal pencocokan pada string teks tadi.

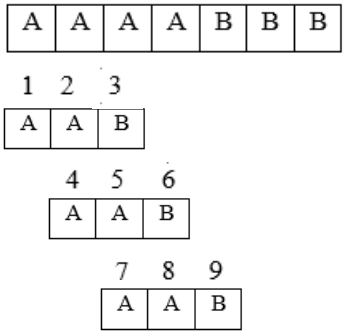
String Teks :

A	A	A	A	B	B	B

String *Pattern* :

A	A	B

Kemudian pencocokan string akan berjalan seperti berikut.



Kompleksitas algoritma *brute force* pada kasus terburuk yaitu O(m*n). Kasus terburuk ini terjadi apabila string sering terjadi mismatch pada akhir dari string *pattern*. Sedangkan, kompleksitas algoritma pada kasus terbaik yaitu O(n). Kasus terbaik terjadi ketika *mismatch* sering kali ditemukan pada awal karakter dari string *pattern*. Kompleksitas algoritma untuk rata-rata kasus yaitu O(m + n), yang mana kompleksitas waktu tersebut juga sudah cukup cepat. Algoritma *brute force* sangat tidak efektif apabila string yang dicocokkan tidak bervariasi. Akan tetapi, algoritma *brute force* ini cukup cepat jika string yang dicocokkan cukup beragam.

2. Knuth-Morris-Pratt

Knuth-Morris-Pratt (KMP) adalah algoritma pencocokan string yang akan melakukan pencocokan terhadap suatu string *pattern* pada string *text* yang dimulai dari kiri ke kanan. KMP mirip seperti pendekatan *Brute Force*, namun bedanya yaitu iterasi yang dilakukan pada string *text* pada algoritma KMP tidak akan pernah "mundur" lagi, dan akan terus maju hingga index terakhir. Algoritma KMP menggunakan penghitungan *border function* dan menggunakan nilai tersebut ketika terjadi *mismatch*. *Border Function* adalah suatu fungsi yang menyatakan panjang prefix dari string *pattern* yang juga merupakan suffix dari string tersebut. *Border Function* didefinisikan dengan b(k) menyatakan *border function* ketika terjadi *mismatch* pada indeks sebelum *mismatch* tersebut. Dengan kata lain, jika *mismatch* terjadi pada indeks j, maka k merupakan j – 1. Nilai *border function* b(k) adalah prefix terbesar dari P[0..k] yang juga merupakan suffix dari P[1..k].

Jika terdapat string *pattern* P sebagai berikut :

P = "ABAABA"

Maka *border function* yang terbentuk adalah sebagai berikut :

j	0	1	2	3	4	5
P[j]	A	B	A	A	B	A
k	-	0	1	2	3	4
b(k)	-	0	0	1	1	2

Algoritma KMP adalah algoritma yang tidak memperhatikan karakter apa yang menyebabkan terjadinya *mismatch*. Pembentukan *border function* juga hanya bergantung pada string *pattern* saja tanpa memerhatikan string teks. Proses penghitungan *border function* pada algoritma KMP ini diperlukan kompleksitas waktu sebesar O(m). Berikut contoh pencocokan string dengan algoritma KMP.

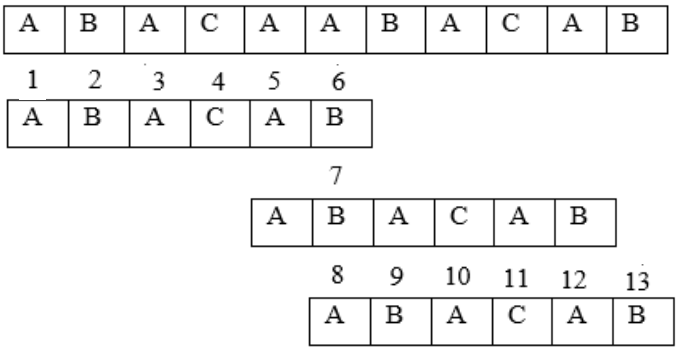
String teks T = "ABACAABACAB"

String *pattern* P = "ABACAB"

Perhitungan *border function* untuk string pattern adalah sebagai berikut.

j	0	1	2	3	4	5
P[j]	A	B	A	C	A	B
k	-	0	1	2	3	4
b(k)	-	0	0	1	0	1

Ketika *border function* sudah dibuat, maka algoritma pencocokan string dapat langsung dilakukan.



Kompleksitas waktu untuk pencarian string dengan algoritma KMP adalah O(m + n). Pencocokan string dengan algoritma KMP jauh lebih cepat dibandingkan dengan algoritma *brute force*. Algoritma ini sangat cocok untuk melakukan pemrosesan pada string yang sangat panjang dan juga jarang terjadinya *mismatch* yang berarti string tersebut tidak cukup beragam. Namun, algoritma ini akan kurang begitu efektif apabila string yang dicocokkan kemungkinan terjadi *mismatch* nya cukup tinggi, atau dengan kata lain, string nya memiliki huruf yang cukup bervariasi.

3. Boyer-Moore

Algoritma Boyer-Moore (BM) adalah algoritma pencocokan string yang menggunakan teknik *looking-glass* dan teknik *character jump*. Algoritma ini menggunakan *fungsi last occurrence* untuk mencari kapan suatu karakter

pada teks terakhir kali muncul pada string *pattern*. Penghitungan *last occurrence* dilakukan sebelum pencocokan string dimulai. Karena itulah algoritma BM ini memperhitungkan karakter apa pada string teks yang menyebabkan terjadinya *mismatch* dengan string *pattern*. Algoritma BM ini menggunakan 2 teknik untuk melakukan pencocokan string. Teknik-teknik tersebut yaitu *looking-glass technique* dan *character jump technique.*

a. Teknik *looking-glass*

Teknik ini yang membedakan algoritma pencocokan string ini dengan algoritma yang telah dibahas sebelumnya. Dengan menggunakan teknik ini, pencocokan string dimulai dengan mencocokan karakter dari karakter terakhir pada string *pattern* lalu kemudian akan terus "mundur" hingga mencapai karakter pertama pada *pattern*.

b. Teknik *character jump*

Teknik ini akan dilakukan ketika terjadi *mismatch* pada saat mencocokan string teks dengan string *pattern*. Pada tahap inilah kita akan memperhatikan string apa pada teks yang menjadi penyebab *mismatch* dengan *pattern*. Jika x adalah string pada teks yang menyebabkan *mismatch*, kemudian kita akan membandingkan *last occurrence* dari string tersebut dengan indeks terjadinya *mismatch* pada *pattern.* Terdapat 3 kasus yang mungkin terjadi, yaitu :

1. Kasus Pertama

Kasus pertama ini terjadi ketika *last occurrence* dari string x berada di sebelah kiri indeks terjadinya *mismatch* pada *pattern*. Pada kasus ini, string *pattern* akan seolah-olah digeser ke kanan hingga karakter x pada *pattern* sejajar dengan string x penyebab *mismatch* pada teks.

2. Kasus Kedua

Kasus kedua ini terjadi ketika *last occurrence* dari string x berada di sebelah kanan indeks terjadinya *mismatch* pada *pattern.* Pada kasus ini, string pattern akan seolah-olah digeser ke kanan satu karakter.

3. Kasus Ketiga

Kasus ketiga ini terjadi ketika string x tidak pernah muncul pada *pattern*. Pada kasus ini, string *pattern* akan seolah-olah digeser hingga indeks pertama *pattern* sejajar dengan tepat satu indeks setelah terjadinya *mismatch* pada teks.

Kompleksitas algoritma untuk algoritma Boyer-Moore pada kasus terburuk yaitu O(nm + A). Algoritma Boyer-Moore akan sangat efektif apabila alfabet A ini cukup panjang, dengan A adalah variasi alfabet yang terdapat pada string teks. Algoritma ini akan berjalan dengan lambat apabila melakukan pencocokan string untuk *binary*. Tetapi, algoritma ini akan berjalan dengan efektif untuk melakukan pencocokkan pada teks yang berbahasa Inggris.

C. Regular Expression

Regulrar Expression (Regex) adalah pencocokan string yang menggunakan suatu pola tertentu yang diberikan sesuai dengan notasi yang berlaku. Regular Expression akan mencari semua posisi yang pada string teks yang cocok dengan *pattern* regex yang diberikan. Berikut beberapa notasi yang terdapat pada regex yang bersumber dari Modul Praktikum NLP : Regex.

Notasi	Deskripsi
[abc]	a, b, atau c
[^abc]	Semua karakter selain a,b,c
[a-zA-z]	a sampai z atau A sampai Z, inclusive
.	Semua karakter
\d	Digit [0-9]
\D	Non digit [^0-9]
\s	Whitespace character
\S	Non whitespace character
\w	Word character
\W	Non wod character
X?	X muncul satu kali atau tidak sama sekali
X*	X muncul 0 atau banyak
X+	X muncul satu atau lebih
X{n}	X muncul tepat n kali
X{n,}	X muncul setidaknya n kali
X{n,m}	X muncul antara n sampai m kali
^	Awal baris
$	Akhir baris
\b	Batas kata

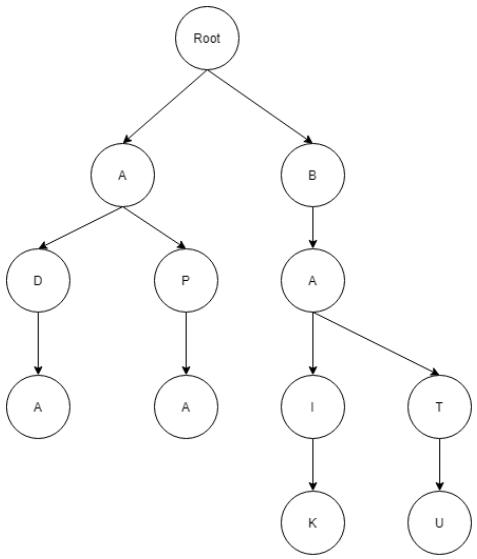
D. Struktur Data

Struktur Data adalah pengorganisasian data, manajemen data dan format penyimpanan suatu data yang memungkinkan pengaksesan dan modifikasi data dengan efektif. Struktur data digunakan untuk menyelesaikan berbagai macam persoalan pemrograman. Dengan menggunakan struktur data, suatu persoalan dapat diselesaikan dengan efektif dan mangkus.

E. Trie

Trie adalah struktur data yang mirip seperti struktur data pohon yang digunakan untuk menyelesaikan permasalahan dalam merepresentasikan kumpulan string. Trie memiliki *node* yang menyimpan huruf-huruf pada setiap *node* nya dan memiliki *reference* ke *node* lain. Trie diawali dengan *root node* yang tidak menyimpan huruf apapun dan baru mulai menghubungkan ke setiap node lain yang berisi tiap alfabet dari A sampai Z. Setiap percabangan yang ada pada trie merepresentasikan kata-kata yang disimpan pada trie tersebut. Kompleksitas waktu terburuk pada pembentukan trie apabila

string terpanjang memiliki panjang m dan terdapat n kata yang disimpan di dalam trie, maka kompleksitas waktunya adalah O(m*n). Sedangkan kompleksitas waktu untuk mencari suatu kata yang ada pada trie, apabila kata yang ingin dicari memiliki panjang m maka kompleksitas waktunya adalah O(m). Berikut adalah contoh struktur data trie.



Gambar 1. Struktur Data Trie

III. Pencocokan String pada Prediksi Teks

Dalam melakukan prediksi teks ketika melakukan pengetikan, ada hal penting yang perlu diperhatikan, yaitu bahasa. Bahasa yang digunakan sangat penting untuk menentukan prediksi teks yang selanjutnya akan dimunculkan untuk membantu pengguna dalam mengetik. Pemilihan bahasa yang digunakan akan menjadi acuan sebagai daftar kosa kata yang terdapat pada bahasa tersebut untuk dilakukan pencocokan string terhadap kata yang saat ini diketik oleh pengguna. Kosa kata yang perlu diperhatikan, tidak hanya kosa kata yang terdapat di kamus, karena yang terdapat di kamus bisa jadi hanya kata – kata dasar saja. Sedangkan ketika mengetik, kita bisa menggunakan kata turunan dari kata tersebut, seperti memberikan imbuhan, atau pun kata – kata lainnya yang biasa digunakan dalam komunikasi sehari - hari.

Untuk melakukan pendataan kosa kata pada bahasa Indonesia, kita dapat mengambil dari KBBI. Untuk melakukan pencocokkan string, kita bisa menyimpan setiap kosa kata yang ada pada sebuah struktur data array satu dimensi. Pencocokan string kemudian akan menelusuri tiap elemen pada array untuk dilakukan pencocokan string dan apabila terdapat elemen yang cocok mulai dari huruf pertama, maka kata tersebut akan ditampilkan sebagai kandidat pada teks prediksi. Pencocokan string ini tidak memperhatikan huruf besar ataupun huruf kecil.

Untuk persoalan ini, pencocokan string dapat dioptimasi. Karena hanya melihat kecocokan pada elemen pertama, maka ketika terdapat *mismatch*, tidak perlu lagi meneruskan algoritma pencocokan string. Berikut merupakan implementasi algoritma – algoritma pencocokan string berdasarkan teori yang sudah dipaparkan.

1. Algoritma Brute Force

Berikut implementasi algoritma *brute force* dalam bahasa python.

```python
def bruteforce(teks, pattern) :
    n = len(teks)
    m = len(pattern)
    for i in range(n - m + 1) :
        j = 0
        while j < m and teks[i+j].lower() == pattern[j].lower() :
            j += 1
        if j == m :
            return i

    return -1
```

Gambar 2 Implementasi Algoritma Brute Force

2. Algoritma Knuth-Morris-Pratt (KMP)

Pada algoritma KMP ini, mula-mula perlu dilakukan pemrosesan awal pada string *pattern* dengan menghitung *border function* terlebih dahulu. Berikut implementasinya dalam bahasa python.

```python
def borderFunction(pattern) :
    fail = [0 for i in range(len(pattern))]
    fail[0] = 0

    m = len(pattern)
    j = 0
    i = 1
    while i < m :
        if pattern[j].lower() == pattern[i].lower() :
            fail[i] = j + 1
            i += 1
            j += 1
        elif j > 0 :
            j = fail[j - 1]
        else :
            fail[i] = 0
            i += 1

    return fail
```

Gambar 3 Implementasi *Border Function*

Kemudian, setelah menghitung *border function*, dapat langsung diteruskan pada pencocokan stringnya. Berikut implementasi algoritma KMP dalam bahasa python.

```python
def kmp(teks, pattern):
    border = borderFunction(pattern)
    n = len(teks)
    m = len(pattern)

    i = 0
    j = 0

    while i < n :
        if pattern[j].lower() == teks[i].lower() :
            if j == m - 1 :
                return i - m + 1
            i += 1
            j += 1
        elif j > 0 :
            j = border[j - 1]
        else :
            i += 1

    return -1
```

Gambar 4 Implementasi Algoritma KMP

3. Algoritma Boyer-Moore

Untuk menggunakan algoritma Boyer-Moore perlu pemrosesan awal untuk menghitung *last occurrence* dari variasi karakter yang ada pada string teks. Berikut implementasinya dalam bahasa python.

```python
def buildLastOccurence(teks, pattern) :
    # membentuk last occurence
    lo = {}

    # inisialisasi -1
    for char in teks.lower() :
        lo[char] = -1

    # membentuk last occurence dari pattern
    for i, char in enumerate(pattern) :
        lo[char.lower()] = i

    return lo
```

Gambar 5 Implementasi *Last Occurrence*

Setelah mendapatkan *last occurrence*, kita dapat melanjutkan proses pencocokan string dengan algoritma Boyer-Moore. Berikut implementasi algoritma Boyer-Moore dalam bahasa python.

```python
def boyerMoore(teks, pattern) :
    lo = buildLastOccurence(teks, pattern)

    n = len(teks)
    m = len(pattern)
    i = m - 1
    if(i > n - 1) :
        return -1 #artinya tidak ada

    j = m - 1

    while True :
        if teks[i].lower() == pattern[j].lower() : # kalau character match
            if j == 0 : # udah ketemu
                return i
            else :
                i -= 1
                j -= 1
        else : # terjadi mismatch
            lastOccur = lo[teks[i].lower()]
            i = i + m - min(j, 1 + lastOccur)
            j = m - 1

        if i > n - 1 :
            break

    return -1
```

Gambar 6 Implementasi Algoritma Boyer-Moore

Selanjutnya, untuk melakukan prediksi teks pada saat melakukan pengetikan dapat dilakukan beberapa langkah sebagai berikut :

1. Mendata kosa kata yang ada di kamus

   Pendataan ini berguna untuk selanjutnya akan dicocokkan pada kata yang sedang diketik. Pendataan ini akan menjadi sumber kata-kata yang kita miliki. Untuk mendapatkan data tersebut, kita dapat mengambilnya dari KBBI.

2. Menyimpan kosa kata pada sebuah struktur data

   Setelah mendapatkan kata-kata apa saja yang akan dilakukan pengecekan, selanjutnya kita akan menentukan struktur data untuk menyimpan kata tersebut. Pada kasus ini, apabila algoritma pencocokan string yang digunakan menggunakan algoritma *brute force*, KMP, ataupun Boyer-Moore, maka struktur data yang akan dipakai adalah List of String. Apabila pencocokan string menggunakan regex, maka cukup simpan seluruh kosa kata tersebut pada satu string yang sangat panjang, karena regex akan langsung dapat mencari semua kata yang cocok dengan pattern. Kemudian apabila pencocokkan string menggunakan Trie, maka tentunya akan menggunakan struktur data Trie.

3. Melakukan pencocokkan string

   Setelah membentuk struktur data yang akan digunakan, kemudian kita akan melakukan pencocokkan string untuk mencari pada kata apa saja, tulisan yang sudah kita ketik mengalami kecocokan.

4. *Generate* daftar kata yang cocok

   Setelah menemukan kata mana saja yang cocok, selanjutnya kita akan mendata kata tersebut dan menyimpannya ke dalam sebuah *container* berupa List of String.

5. Menampilkan daftar kata yang cocok

Setelah daftar kata tersebut dibentuk, tahap terakhir ini tentunya adalah menampilkan kata tersebut ke layar. Apabila List of String dari kata yang cocok tersebut kosong, maka layar hanya akan menampilkan kembali kata yang sedang kita ketik.

Karena kata yang akan dilakukan pencocokkan tidak memperdulikan huruf kapital atau bukan. Maka pada kasus ini akan dianggap semua huruf bukan huruf kapital. Begitu juga pada teks yang sedang diketik akan diperlakukan sebagai huruf yang bukan kapital pada saat melakukan pencocokan teks. Selanjutnya akan dilakukan pemrosesan sesuai dengan langkah-langkah yang telah didefinisikan.

1. Mendata kosa kata yang ada di kamus

   Misal, kosa kata yang dimiliki yaitu “aku”, “ada”, “akhirnya”, “akan”, “buat”, “bikin”, “cari”, “cuci”, “cucu”.

2. Menyimpan kosa kata pada sebuah struktur data

   Untuk pemrosesan dengan algoritma *brute force*, KMP, dan Boyer-Moore. Kosa kata tersebut akan disimpan pada sebuah List of String seperti berikut.

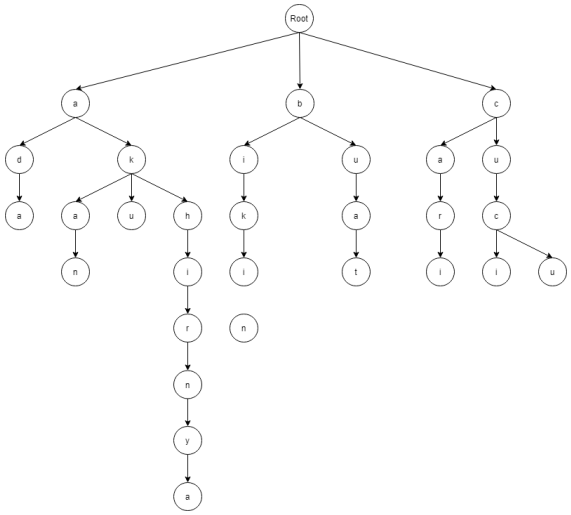
aku	ada	akhirnya	akan	buat

bikin	cari	cuci	cucu

   Untuk pemrosesan dengan menggunakan regex, daftar kosa kata akan disimpan pada sebuah string yang memanjang seperti berikut.

   String S = “aku ada akhirnya akan buat bikin cari cuci cucu”

   Untuk pemrosesan menggunakan struktur data Trie, berikut daftar kosa kata yang telah dibentuk menjadi struktur data Trie.



Gambar 7 Struktur Data Trie dari Kosa Kata

3. Melakukan pencocokkan string

   Pada tahap ini, baru akan dimulai algoritma pencocokan string untuk mendapatkan prediksi teks yang akan ditampilkan ketika sedang mengetik. Misal, kata yang sedang diketikkan adalah “Ak”. Kemudian kata tersebut akan dilakukan pencocokkan tanpa mempertimbangkan huruf besar atau kecilnya. Jadi apabila terdapat “A” dan “a” akan dianggap sama.

   Pencocokkan string dengan menggunakan algoritma *brute force*, KMP, dan Boyer-Moore akan menggunakan string teks T = “Ak”. Dan kemudian akan melakukan iterasi terhadap List of String dengan setiap elemen akan menjadi string *pattern* yang kemudian akan melakukan pencocokkan. Pencocokkan yang dilakukan ini hanya mempertimbangkan apabila kecocokan tersebut benar benar mulai dari awal string yang akan menjadi teks yang diprediksi tersebut untuk kata yang diketikkan. Maka, akan seperti sebagai berikut.

   String Teks T = “Ak”

   String *pattern* :

   1) P = “aku”

a	k

a	k	u

   String tersebut cocok.

   2) P = “ada”

a	k

a	d	a

String tersebut tidak cocok.

3) *P = "akhirnya"*

a	k

a	k	h	i	r	n	y	a

String tersebut cocok.

4) *P = "akan"*

a	k

a	k	a	n

String tersebut cocok.

5) *P = "buat"*

a	k

b	u	a	t

String tersebut tidak cocok.

6) *P = "bikin"*

a	k

b	i	k	i	n

String tersebut tidak cocok.

7) *P = "cari"*

a	k

c	a	r	i

String tersebut tidak cocok.

8) *P = "cuci"*

a	k

c	u	c	i

String tersebut tidak cocok.

9) *P = "cucu"*

a	k

c	u	c	u

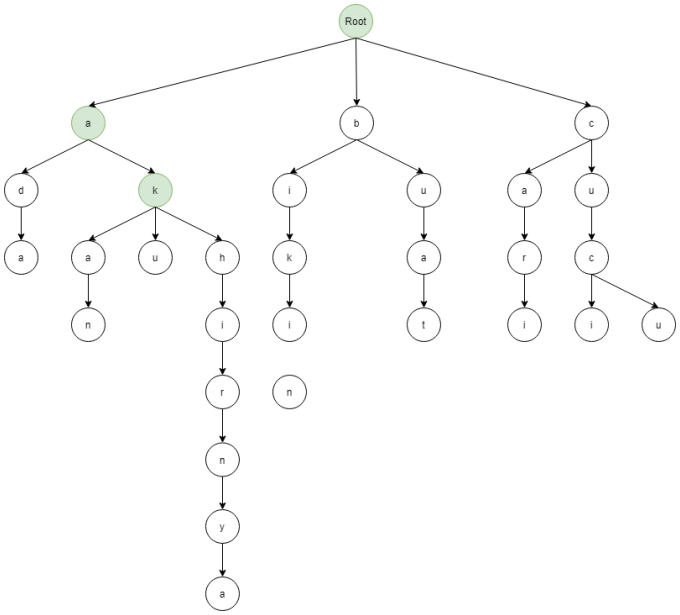
String tersebut tidak cocok.

String yang dihasilkan dari pencocokkan tersebut adalah "aku", "akhirnya", dan "akan".

Kemudian, pencocokkan string dengan menggunakan regex akan menggunakan suatu pola berdasarkan notasi yang telah ditentukan. Jika x merupakan string yang diketik, maka pola tersebut akan menjadi seperti berikut "\bx[\S]*\b". Dengan ketentuan, x merupakan string yang telah diubah menjadi huruf kecil semua. Pola tersebut akan mencocokkan sebuah string x yang berupa satu kata yang dibatasi dengan \b untuk menandai batas kata, dan juga akan melengkapi kata tersebut hingga akhir batas kata tersebut dengan pola [\S]*. Apabila x = "ak". Maka dari string *pattern* P = "aku ada akhirnya akan buat bikin cari cuci cucu", akan didapatkan hasil sebagai berikut.

String hasil regex : S = ["aku", "akhirnya", "akan]

Pencocokkan string dengan menggunakan struktur data Trie akan melakukan *traversal* pada node sesuai dengan teks yang diketikkan. Jika x = "ak". Maka dari root, akan menelusuri node "a". Apabila tidak terdapat node "a", maka hentikan pencarian, karena pasti tidak ada yang cocok. Kemudian apabila terdapat node "a", lanjutkan menuju node "k". Lakukan hal serupa sampai karakter terakhir dari string tersebut. Berikut hasil penelusuran string yang diketikkan terhadap Trie.



Gambar 8 Pencocokan String dengan Trie

4. *Generate* daftar kata yang cocok

Setelah melakukan pencocokan pada string sesuai metode masing-masing. Setiap metode memiliki berbagai cara untuk mendata kata yang cocok.

Algoritma *brute force*, KMP, dan Boyer-Moore apabila setiap pencocokkan menemukan kecocokan, kata tersebut dapat langsung dimasukkan ke dalam sebuah *list*. Sedangkan ketika menggunakan regex, akan otomatis menghasilkan *list* dari daftar kata yang cocok dengan pola. Apabila menggunakan struktur data Trie, ketika sudah mencapai node dari karakter terakhir pada string yang diketik, kemudian akan melakukan DFS. Ketika proses DFS tersebut sudah mencapai *leaf* node, *path* dari *root* hingga ke *leaf* tersebut, merupakan kata yang cocok.

5. Menampilkan daftar kata yang cocok

Setelah semua daftar kata yang cocok telah terkumpul, maka langkah selanjutnya dan merupakan langkah terakhir adalah menampilkannya ke layar sehingga pengguna bisa memilih kata tersebut dan memakainya.

Berikut merupakan contoh dari hasil pencocokan yang telah dilakukan pada langkah-langkah sebelumnya.

Gambar 9 Hasil Prediksi Kata

IV. Kesimpulan Dan Saran

1. Kesimpulan

Pada prediksi teks ini, terdapat banyak cara pencocokan string yang bisa dilakukan. Namun yang paling efektif dari semua algoritma yang telah disebutkan yaitu menggunakan struktur data Trie. Karena kata yang memiliki *prefix* yang sama tidak akan ditulis ulang dan itu akan meminimalisir penggunaan *memory*. Dan juga, dengan menggunakan struktur data Trie, pencarian tidak akan menuju ke kata yang tidak terdapat pada string yang telah ditentukan.

2. Saran

Untuk memperbaiki prediksi teks ini agar lebih efektif dan tepat guna, daftar kata yang akan ditampilkan akan diurutkan sesuai prioritas. Prioritas tersebut yang dimaksud dapat berupa kata yang sering digunakan oleh pengguna, atau pun berdasarkan struktur kalimat yang sering dipilih oleh pengguna. Hal itu bisa memungkinkan kata yang akan tampil di layar adalah kata yang paling mungkin untuk dipilih pengguna.

Link Video di Youtube

Video untuk pemaparan makalah ini dapat diakses pada link youtube sebagai berikut.

https://youtu.be/nhDMpEbR5hg

Video tersebut berjudul "Pemanfaatan Pencocokan String dalam Prediksi Teks" yang sesuai dengan judul makalah ini.

Ucapan Terima Kasih

Penulis ingin mengucapkan puji syukur kepada Tuhan Yang Maha Esa karena dengan rahmat-Nya penulis dapat menyelesaikan makalah yang berjudul "Pemanfaatan Pencocokan String dalam Prediksi Teks" ini. Penulis juga berterima kasih kepada seluruh dosen pengampu mata kuliah strategi algoritma yang telah memberikan ilmu dan pelajaran yang sangat berharga untuk penulis.

Penulis juga berterima kasih kepada keluarga penulis, terutama kepada orang tua penulis, yang selalu mensupport penulis dalam menjalani perkuliahan ini dalam kondisi apapun. Karena support yang diberikan sangat berarti bagi penulis.

Referensi

[1] https://www.geeksforgeeks.org/advanced-data-structures/#Trie. Diakses 25 April 2020.

[2] Modul Praktikum NLP : Regex. Diakses 26 April 2020.

[3] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf. Diakses 26 April 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, 2 Mei 2020

Yahya
13518029