Penyaringan Kata Kasar Menggunakan Hamming Distance pada Algoritma Brute Force Pattern Matching

Marchotridyo - 13520119
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): acoxstpd@gmail.com

***Abstract*—Di era perkembangan internet ini, orang-orang dapat saling berkomunikasi dengan sangat mudah melalui berbagai media, salah satunya ada media visual melalui ketikan. Orang-orang dapat mengetikkan apa saja yang ia ingin katakan. Akibatnya, tak jarang teknologi ini digunakan untuk berkata-kata yang tidak sepantasnya. Makalah ini mempelajari bagaimana cara menggunakan prinsip *pattern matching* untuk mendeteksi sekaligus menyaring kata-kata kasar dari suatu masukan. Algoritma yang digunakan berdasar kepada algoritma *brute force pattern matching* yang dimodifikasi agar pencarian dapat mengakomodasi kata-kata kasar yang tidak eksak dengan kumpulan kata kasar pada sistem dengan menggunakan *hamming distance*.**

Keywords—komunikasi visual, pattern matching, brute force, hamming distance, penyaringan kata-kata kasar

I. PENDAHULUAN

Di era perkembangan internet ini, orang-orang dapat saling berkomunikasi dengan sangat mudah melalui berbagai media, salah satunya ada media visual melalui ketikan. Bentuk komunikasi ini sangat sering dipakai dalam kehidupan sehari-hari, misal melalui aplikasi *instant messaging* seperti LINE, aplikasi media sosial seperti Twitter, melalui postingan blog, bahkan melalui permainan *multiplayer* yang bersifat *online*.

Kemudahan mengekspresikan informasi ini terkadang disalahgunakan karena ada beberapa orang yang tidak memikirkan dahulu apa yang ia ketikkan. Ketikan-ketikan yang dihasilkan, walaupun diunggah di ruang publik, tidak jarang mengandung kata-kata yang kurang pantas untuk dilihat oleh banyak orang, apalagi pada media-media yang tidak hanya dapat dilihat oleh orang dewasa tapi juga oleh anak-anak. Hal ini dapat memberikan pengaruh negatif dengan bentuk menormalisasikan berkata yang kurang baik di ruang publik.

Gambar 1.1 Contoh penyimpangan yang terjadi di ruang media yang bisa dikonsumsi oleh publik

Makalah ini mempelajari bagaimana suatu prinsip yang telah dipelajari di mata kuliah IF2211 Strategi Algoritma, yaitu prinsip penyocokan *string*, dapat digunakan untuk melakukan penyaringan terhadap suatu masukan untuk melakukan *filter* terhadap kata-kata kasar. Dalam aplikasinya, algoritma yang digunakan adalah modifikasi dari algoritma *brute force pattern matching* dengan cara menambahkan perhitungan *hamming distance* pada setiap pengecekannya. Ini bertujuan agar sistem penyaringan dapat bekerja tidak hanya kata yang eksak sama dengan kumpulan kata kasar, misalnya aplikasi akan tetap menyaring kata ‘anjink’ walaupun yang kata kasar yang tersimpan pada sistem adalah kata ‘anjing’.

II. TEORI DASAR

A. *Brute force*

Algoritma *brute force* adalah suatu algoritma yang pendekatannya *straightforward* atau naif. Algoritma yang disusun didasarkan pada pernyataan pada persoalan dan/atau definisi atau konsep yang dilibatkan. Ciri dari algoritmanya adalah cara kerjanya sangat sederhana dan jelas caranya.

Algoritma *brute force* umumnya tidak cerdas karena membutuhkan volume komputasi yang besar dan waktu yang lama dalam penyelesaiannya, sehingga lebih cocok digunakan untuk persoalan yang masukannya kecil. Algoritmanya juga terkesan tidak sekonstruktif/sekreatif strategi pemecahan masalah lainnya.

Disamping kekurangannya, algoritma *brute force* juga memiliki beberapa kegunaan. Biasanya, algoritma *brute force* ini digunakan sebagai pembanding untuk algoritma-algoritma yang lebih mangkus. Algoritma yang disusun sederhana dan

mudah dimengerti untuk orang-orang yang baru belajar tentang algoritma. Selain itu, algoritma *brute force* dapat diterapkan untuk memecahkan hampir sebagian besar masalah.

B. *Pattern matching*

Pattern matching, atau pencocokan *string*, secara sederhana, adalah masalah mengenai pencarian sebuah pola pada suatu teks yang diberikan. Inti dari permasalahan ini adalah mencari lokasi dari pola tersebut pada suatu teks, misal ditentukan dari indeks pertama kemunculan pattern tersebut. Sebagai contoh, apabila diberikan teks "nasi dan ayam", apabila diberikan *pattern* "si" maka harapannya algoritma yang dibuat dapat mengembalikan hasil '2' yang menandakan *pattern* ditemukan pada indeks ke-2 pada teks (indeks teks diasumsikan mulai dari 0).

Pencocokan *string* ini sering sekali digunakan dalam kehidupan sehari-hari kita, misalnya dalam pencarian pada *editor text*, pada *search engine*, pada analisis citra, juga pada *bioinformatics* khususnya pada pencocokan rantai DNA/RNA.

Metode *string matching* sendiri beragam, contohnya ada algoritma *brute force pattern matching*, algoritma Knuth-Morris-Pratt (KMP), serta algoritma Boyer-Moore (BM). Pada makalah ini, yang digunakan adalah algoritma *brute force pattern matching*.

C. *Brute force pattern matching*

Seperti namanya, algoritma *brute force pattern matching* adalah algoritma pencocokan *string* berdasarkan algoritma *brute force*. Pencarian dilakukan secara naif, dengan langkah-langkah secara umum sebagai berikut:

1. Pencarian, yang berupa sebuah iterasi, dimulai dari indeks pertama pada teks.
2. Di setiap iterasi, akan dicoba dilakukan pengecekan apakah pola P dimulai pada indeks tersebut di teks T.
3. Jika pola P ditemukan, kembalikan indeks tersebut. Iterasi bisa dihentikan.
4. Jika pola P tidak ditemukan, maka indeks pencarian pada teks T dimajukan satu. Iterasi dihentikan jika indeks pencarian tidak dapat mengakomodasi lagi panjang pola P.

Dalam notasi algoritmik, prosedur tersebut dapat dituliskan sebagai berikut (indeks pada string dimulai dari 0):

```
function bruteMatch(T, P: string) -> integer
KAMUS
   i, j, m, n: integer
   found: boolean
ALGORITMA
   m <- panjang dari T
   n <- panjang dari P
   found <- false
   while (i <= n - m) and (not found) do
        j <- 0
        while (j < m) and (T_{i+j} = P_j) do
                j <- j + 1
        if j = m then
                found <- true
        else
                i <- i + 1
   if found then
        -> i
   else
        -> -1
```

Prosedur yang sudah dijelaskan di atas diilustrasikan oleh pencarian pada gambar berikut ini:

Contoh 1:

Teks: NOBODY NOTICED HIM

Pattern: NOT

```
  NOBODY NOTICED HIM
1 NOT
2  NOT
3   NOT
4    NOT
5     NOT
6      NOT
7       NOT
8        NOT
```

Gambar 2.1 Ilustrasi pencarian menggunakan algoritma *brute force pattern matching*

(Sumber: *Slide* perkulahan *brute force* bagian 1)

D. *Hamming distance*

Hamming distance adalah suatu kuantitas yang dimiliki saat membandingkan dua *string* dengan panjang yang sama. Kuantitas tersebut menyatakan berapa simbol yang berbeda pada kedua string. Contoh, *hamming distance* dari *string* 'karolin' dan 'kathrin' adalah 3 karena ada tiga simbol yang berbeda (ditandai dengan warna merah).

Pencariannya dapat dilakukan secara naif (*brute force*) dengan langkah-langkah secara umum sebagai berikut:

1. Awalnya, *hamming distance* dari dua *string* dengan panjang yang sama adalah nol.
2. Mulai iterasi dari indeks pertama *string*.
3. Di setiap iterasi, cek apakah simbol yang terdapat pada indeks tersebut pada kedua *string* sama atau tidak.
4. Jika kedua simbol sama, cukup lanjutkan iterasi saja.

5. Jika kedua simbol berbeda, tambahkan *hamming distance* sebanyak satu, lalu lanjutkan iterasi.
6. Iterasi dihentikan ketika seluruh karakter pada *string* sudah diperiksa. Hasil yang dicari adalah hasil *hamming distance* yang sudah dihitung.

Dalam notasi algoritmik, prosedur tersebut dapat dituliskan sebagai berikut (indeks pada string dimulai dari 0):

```
function hammingDistance(S1, S2: string) -> integer
KAMUS
   i, dist: integer
ALGORITMA
   { Prekondisi: S1 dan S2 memiliki panjang yang sama
}
   dist <- 0
   i traversal [0 .. (panjang dari S1) - 1]
        if (S1i != S2i) then
                dist <- dist + 1
   -> dist
```

E. *Modifikasi algoritma brute force pattern matching dengan perhitungan hamming distance*

Perhatikan bahwa algoritma *brute force pattern matching* sebelumnya hanya mengembalikan indeks pertama ditemukannya pola. Hal ini kurang cocok untuk apa yang diinginkan dalam pendeteksian kata kasar, misalnya apabila diberikan masukan 'anjing kucing anjing' dengan anggapan 'anjing' adalah suatu kata kasar, kata yang akan disaring hanyalah kata anjing yang pertama dan kata anjing yang kedua tidak disaring.

Oleh karena itu, algoritma *brute force pattern matching* diberikan modifikasi sedemikian sehingga hasil pengembaliannya adalah larik *integer* yang berisikan indeks-indeks awal ditemukannya pola pada teks. Dalam contoh sebelumnya, 'anjing kucing anjing' apabila dicocokkan dengan 'anjing' akan dibuat mengembalikan larik [0, 14]. Apabila pola tidak ditemukan, akan dikembalikan larik kosong [].

Dalam notasi algoritmik, modifikasi tersebut dapat dilakukan sebagai berikut: (res adalah suatu array dinamis, pemanggilan res.push(x) artinya menambahkan x ke dalam res).

```
function bruteMatchLarik(T, P: string) -> array of
integer
KAMUS
   i, j, m, n: integer
   res: array of integer
   different: boolean
ALGORITMA
   m <- panjang dari T
   n <- panjang dari P
   res <- []
   i traversal [0 .. m-n]
        different <- false
        j <- 0
        while (j < m) and (not different) do
                if (Ti+j != Pj) then
                        different <- true
                else
                        j <- j + 1
        if (j = m) then
                res.push(i) {Tambahkan indeks i ke res }
   -> res
```

Selain itu, peneliti menginginkan agar kata-kata yang memiliki kemiripan setidaknya 80% ikut terdeteksi sebagai kata kasar, misalnya 'anjink' dan 'anjing' memiliki perbedaan 1 karakter dari 6 karakter, artinya memiliki kesamaan 5 karakter dari 6 karakter (83,33%) sehingga kata 'anjink' juga akan disaring.

Untuk mengimplementasikan hal tersebut, digunakan *hamming distance* antara dua *string* dengan panjang yang sama untuk menghitung persentase kesamaan dari cuplikan teks dengan pola. Apabila persentase kesamaan setidaknya 80%, indeks tersebut akan dimasukkan ke larik.

Dalam notasi algoritmik, penambahan aplikasi *hamming distance* untuk menghitung persentase kesamaan antara dua string pada fungsi *bruteMatchLarik* dilakukan sebagai berikut:

```
function bruteMatchHamming(T, P: string) -> array of
integer
KAMUS
   i, j, m, n, dist: integer
   res: array of integer
   kesamaan: real
ALGORITMA
   m <- panjang dari T
   n <- panjang dari P
   res <- []
   i traversal [0 .. m-n]
        dist <- 0
        j traversal [0.. n]
                if (Ti+j != Pj) then
                        dist <- dist + 1
        { Hamming distance cuplikan teks dengan pola
sudah terhitung }
        { Cari persentase kesamaan }
        kesamaan <- (n - dist) / n
        if (kesamaan >= 0.8) then
                res.push(i)
   -> res
```

Dengan fungsi yang sudah diberikan di atas, pencocokan teks 'anjink kucing anjing' terhadap pola 'anjing' akan mengembalikan larik [0, 14].

F. *Algoritma penyaringan terhadap suatu masukan*

Peneliti ingin penyaringan dilakukan dengan cara mengubah karakter-karakter pada bagian yang terdeteksi sebagai kata kasar menjadi '*' kecuali kata pertama dan kata terakhirnya. Sebagai contoh, 'anjink kucing anjing' akan disaring menjadi 'a****k kucing a****g'.

Selain itu, karena kata-kata kasar jumlahnya banyak, peneliti menginginkan penyaringan terhadap semua kata kasar dapat dilakukan dengan memanggil satu fungsi saja, misalnya kata 'anjing kucing babi ayam' akan disaring menjadi 'a****g kucing b**i ayam', menganggap larik data kata kasar mencakup kata 'anjing' dan 'babi'.

Garis besar proses penyaringan ini akan dituangkan dalam fungsi *filterText*, yang secara garis besar mengikuti langkah-langkah berikut:

1. Iterasikan setiap kata kasar yang terdapat dalam larik kumpulan kata kasar.
2. Untuk setiap kata yang sedang diproses *brute force pattern matching* terhadap kata tersebut untuk mendapat indeks-indeks awalnya pada teks.

3. Iterasikan larik indeks (misal i) tersebut dengan cara mengubah karakter-karakter dari indeks i + 1 sampai dengan karakter sebelum karakter terakhir pola menjadi '*'.
4. Proses selesai apabila semua kata kasar pada larik kumpulan kata kasar sudah diproses.

Dalam notasi algoritmik, prosedur tersebut dapat dilakukan sebagai berikut:

```
function filterText(T: string, kataKasar: array of
string) -> string
KAMUS
   i, j, k, p: integer
   res: string
   indexes: array of integer
ALGORITMA
   res <- copy dari T
   i traversal [0 .. panjang dari kataKasar]
        indexes <- bruteMatchHamming(T, kataKasar_i)
        j traversal [0 .. panjang dari indexes]
                p <- panjang dari kataKasar_i
                k traversal [indexes_j + 1 .. indexes_j + p
- 2]
                        res_k = '*'
   -> res
```

G. *Analisis kompleksitas algoritma yang dibuat*

Perhatikan bahwa pada fungsi *bruteMatchHamming*, terjadi iterasi sebanyak $(m - n)(n) = mn - n^2$ dengan m adalah panjang teks dan n panjang pola. Operasi-operasi yang dilakukannya adalah operasi O(1) dengan mengganggap operasi .push() pada larik dinamis dapat dilakukan dengan kompleksitas O(1). Dengan demikian, fungsi *bruteMatchHamming* memiliki kompleksitas $O(mn - n^2)$.

Sedangkan, untuk kompleksitas algoritma *filterText*, kompleksitasnya bergantung kepada banyak kata pada larik kataKasar serta banyaknya indeks awal pola yang ditemukan pada teks. Apabila dimisalkan k adalah banyak kataKasar dan i adalah banyak indeks terbanyak dari hasil pemanggilan *bruteMatchHamming*, m adalah panjang teks, dan n adalah panjang pola (kata kasar) terpanjang, kompleksitasnya adalah $O(k((mn - n^2) + in))$.

III. Hasil Eksperimen

Eksperimen dilakukan dengan cara membuat suatu aplikasi *desktop* menggunakan *framework* Electron dalam bahasa JavaScript. Aplikasi ini bernama SwearFilter dan dapat diunduh melalui tautan acomarcho/SwearFilter: A desktop application for swear word detection using brute force pattern matching algorithm + hamming distance. (github.com). Aplikasi yang dibuat cukup sederhana, pengguna cukup memberikan *input* teks lalu menekan tombol 'Filter!'.

Daftar kata-kata kasar menggunakan data yang sudah ada dari modul bernama 'badwords' karya MauriceButler pada GitHub yang dapat dilihat melalui tautan MauriceButler/badwords: A highly consumable list of bad (profanity) english words (github.com). Daftar ini berisikan kata-kata pada bahasa Inggris yang dianggap kasar (*profane*).

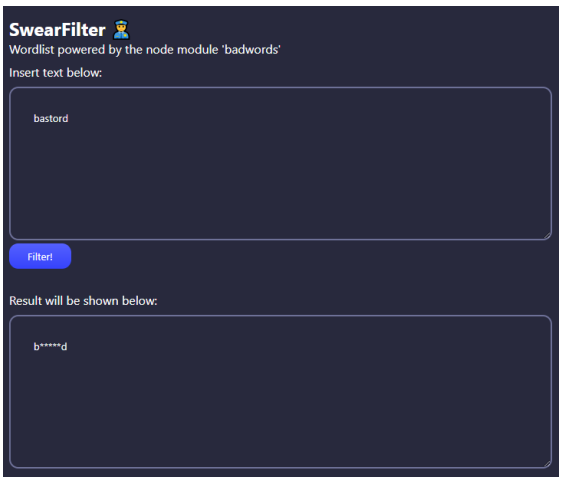
Pengujian dilakukan secara bertahap:

1. Menguji satu kata kasar yang secara eksak terletak pada kumpulan kata.
2. Menguji satu kata kasar yang memiliki kemiripan setidaknya 80% terhadap suatu kata pada kumpulan kata.
3. Menguji berbagai kata kasar yang sama untuk menguji apakah *pattern matching* berhenti pada penemuan pertama atau tidak.
4. Menguji sebuah cuplikan lirik lagu berbahasa Inggris yang memiliki beberapa kata kasar untuk menguji fungsionalitas aplikasi secara penuh.
5. Menguji kasus di mana ada kata yang tidak kasar tetapi memiliki setidaknya 80% kemiripan dengan kata kasar.

Pertama-tama, dilakukan pengujian terhadap kata kasar 'bastard' yang secara eksak terletak pada kumpulan kata.

Gambar 3.1 Pengujian untuk kata 'bastard'

Kedua, dilakukan pengujian terhadap kata 'bastord' yang memiliki kesamaan >= 80% terhadap kata 'bastard' yang terdaftar pada kumpulan kata.



Gambar 3.2 Pengujian untuk kata 'bastord'

Ketiga, dilakukan pengujian terhadap kalimat ‘bastard bastard bastard’ untuk menguji apakah *pattern matching* berhenti di penemuan pertama saja, atau tidak.

Gambar 3.3 Pengujian untuk kalimat ‘bastard bastard bastard’

Keempat, dilakukan pengujian terhadap cuplikan dari lagu Payphone karya dari Maroon 5.

Gambar 3.4 Pengujian pada kasus dunia nyata terhadap beberapa lirik lagu yang mengandung kata kasar

Kelima, dilakukan pengujian terhadap kasus di mana lirik lagu mengandung suatu kata yang dideteksi 80% mirip dengan suatu kata kasar, walaupun setidaknya tidak berupa kata kasar.

Gambar 3.5 Pengujian pada kasus dunia nyata terhadap suatu lirik lagu yang memiliki kata yang seharusnya tidak kasar, tetapi ditandai seebagai kata kasar.

Pada contoh di atas, kata *bells* disensor karena memiliki kemiripan setidaknya 80% dengan kata *balls* yang ada pada data kata kasar serta kada *rinks* disensor karena memiliki kemiripan setidaknya 80% dengan kata *dinks* yang ada pada data kata kasar.

Hasil eksperimen membuktikan bahwa algoritma yang diberikan cukup efektif untuk mendeteksi kata-kata yang memang kasar. Namun, ada kekurangan yang cukup jelas yaitu algoritma dapat menandai kata-kata yang tidak kasar sebagai kata yang kasar seperti contoh pada Gambar 3.5. Hal ini dapat dipertimbangkan untuk peneliti-peneliti selanjutnya, misal dengan mengecek kepada suatu kamus asli yang bisa menandai apakah kata ini sebenarnya bukan kata kasar atau tidak.

IV. KESIMPULAN.

Menggunakan prinsip-prinsip yang telah diajarkan pada mata kuliah IF2211 Strategi Algoritma, berbagai macam persoalan di dunia nyata dapat diselesaikan, bahkan menggunakan tipe algoritma yang sederhana yaitu algoritma *brute force*. Dalam kasus pada makalah ini, persoalan penyaringan kata kasar dapat dilakukan dengan memodifikasi algoritma *brute force pattern matching* dengan menambahkan perhitungan *hamming distance* pada setiap segmen teks yang dibandingkan. Ini menunjukkan bahwa algoritma-algoritma yang sudah ada sebelumnya (dalam artian sudah diajarkan) dapat dimodifikasi untuk mengakomodasi kebutuhan penggunanya. Algoritma yang disusun juga bisa langsung diaplikasikan ke dalam bentuk aplikasi asli menggunakan bahasa pemrograman tertentu, dalam kasus ini menggunakan bahasa JavaScript dengan bantuan *framework* Electron.

V. Ucapan Terima Kasih

Penulis mengucapkan terima kasih kepada seluruh pihak yang telah membantu pengerjaan makalah ini sehingga pengerjaan makalah ini dapat dikerjakan tepat pada waktunya, khususnya kepada Tuhan yang Maha Esa karena tanpa rahmat dan karunia-Nya makalah ini tentu saja tidak akan ada. Selain itu, penulis secara khusus mengucapkan terima kasih kepada keluarga dan Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc. selaku dosen pengampu mata kuliah IF2211 Strategi Algoritma kelas 02. Penulis juga berterima kasih kepada pembuat referensi yang penulis gunakan untuk makalah ini.

VI. Link YouTube

https://youtu.be/DiroS6y9_gE

Referensi

[1]	Munir, Rinaldi. 2022. *Algoritma Brute Force (Bagian 1)*. Diunduh dari situs Pak Rinaldi Munir pada tanggal 20 Mei 2022.
[2]	Tim Dosen IF2211 Strategi Algoritma. 2022. *Pencocokan String (String/Pattern Matching)*. Diunduh dari situs Pak Rinaldi Munir pada tanggal 20 Mei 2022.
[3]	Studený, Jan dan Uznański, Przemyslaw. 2019. *Approximating Approximate Pattern Matching*. arXiv:1810.01676v3 [cs.DS] 23 Jul 2019.

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, 20 Mei 2022

Marchotridyo 13520119