Penerapan Algoritma FFT dalam Wildcard String Matching

Kristo Anugrah - 13522024[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522024@std.stei.itb.ac.id

Abstrak*—Persoalan wildcard string matching adalah persoalan pencocokan dua buah string yang berisi huruf Latin kecil dan karakter wildcard. Karakter wildcard dalam string dianggap sama dengan seluruh alfabet. Persoalan ini dapat diselesaikan secara naif dalam O(nm), dengan n dan m sebagai ukuran kedua buah string. Namun, untuk n dan m yang besar pendekatan ini tidak efisien. Makalah ini membahas pendekatan wildcard string matching dalam O(nlogn) dengan menggunakan algoritma Fast Fourier Transform.

***Kata kunci*— discrete fourier transform, fast fourier transform, kompleksitas algoritma, string matching**

I. Pendahuluan

Persoalan *wildcard string matching* menerima dua buah *string* S dan P yang berisi huruf Latin kecil dan karakter *wildcard*. Kemudian perlu dihasilkan seluruh posisi *substring* S sehingga *substring* tersebut sama dengan P. Dua buah *string* dikatakan sama jika dan hanya jika seluruh karakternya bersesuaian. Dua buah karakter dikatakan bersesuaian jika dan hanya jika dua buah karakter tersebut sama atau salah satu merupakan karakter *wildcard*.

Persoalan ini dapat diselesaikan secara naif dalam waktu $O(nm)$ dengan mencoba seluruh indeks i dan kemudian mencocokkan *substring* dengan string P. Namun, algoritma naif tersebut tidak efisien karena kompleksitas waktu yang bertumbuh secara kuadratik. Dalam makalah ini akan ditunjukkan algoritma yang dapat menyelesaikan persoalan *wildcard string matching* dalam $O(nlogn)$ dengan menggunakan algoritma Fast Fourier Transform.

II. Rumusan Persoalan

Diberikan sebuah *string* S dan P (*0-indexed*) dengan panjang masing-masing n dan m $(m \leq n)$. *String* S terdiri dari huruf Latin kecil (a, b, c,…, z), sedangkan *string* P terdiri dari huruf Latin kecil serta karakter '?' (ASCII 63). Sebuah indeks $i \in [0, n-m]$ dikatakan 'cocok' jika dan hanya jika untuk seluruh $j \in [0, m)$, $S_{i+j} = P_j$ atau $P_j = "?"$. Keluarkan seluruh indeks $i \in [0, n-m]$ sehingga indeks i merupakan indeks yang 'cocok'.

Sebagai contoh, jika $S = "informatika"$ dan $P = "i?f"$, program akan mengeluarkan indeks 0, karena *substring* pada indeks tersebut diawali dengan huruf 'i' yang diikuti huruf 'f' pada indeks 2. Sedangkan jika $S = "aabbaa"$ dan $P = "a?b"$, program akan mengeluarkan indeks 0 dan 1.

III. Landasan Teori

A. Bilangan Kompleks dan Akar Persatuan

Bilangan kompleks adalah bilangan dengan bentuk $a + bi$, dengan $a, b \in \mathbb{R}$. Bilangan kompleks juga dapat divisualisasikan sebagai sebuah titik dalam $\mathbb{R}^2$, dengan sumbu x menandakan bagian real dan sumbu y menandakan bagian imajiner. Sebuah bilangan kompleks juga dapat direpresentasikan dengan bentuk $re^{i\theta}$, dengan r sebagai jarak bilangan dari origin dan theta sudut antara titik dengan sumbu x positif. Bentuk tersebut juga dapat ditulis sebagai kombinasi fungsi cos dan sin

$$e^{i\theta} = \cos\theta + i\sin\theta \tag{1}$$

Akar persatuan ke-n (*n-th root of unity*) adalah seluruh bilangan kompleks z yang memenuhi $z^n = 1$ untuk suatu bilangan bulat positif n. Perhatikan bahwa akar persatuan ke-n pasti memiliki $r = 1$. Perhatikan juga karena fungsi cos dan sin adalah fungsi siklis dengan periode 2π, berlaku

$$1 = \cos 2\pi m + i\sin 2\pi m, m \in \mathbb{N} \tag{2}$$

Sehingga z dapat diekspresikan dengan

$$z^n = \cos 2\pi m + i\sin 2\pi m, m \in \mathbb{N} \tag{3}$$

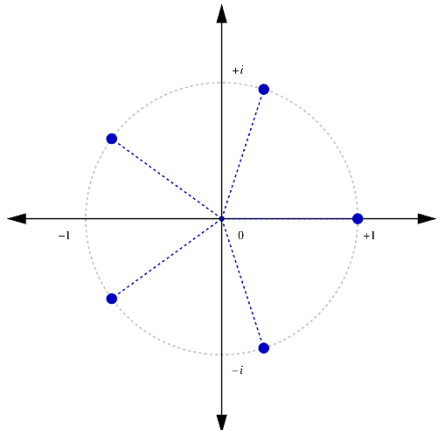
Dengan mengambil akar ke-n dari kedua ruas, didapat

$$z = \sqrt[n]{\cos 2\pi m + i\sin 2\pi m}, m \in \mathbb{N} \tag{4}$$

Dengan menggunakan teorema de Moivre, persamaan (4) dapat diubah menjadi

$$z = \cos\frac{2\pi m}{n} + i\sin\frac{2\pi m}{n} = \exp\left(\frac{i2\pi m}{n}\right), m \in \mathbb{N} \tag{5}$$

Akar persatuan ke-n dapat divisualisasikan sebagai n titik pada *unit circle* yang berjarak sama. Hal ini dapat dilihat pada persamaan (5). Menambahkan m sama halnya dengan memutar titik *counterclockwise* sejauh $\frac{2\pi}{n}$ radian.



Gambar 1 Akar persatuan ke-5 [1]

B. Polinomial dalam Satu Variabel

Dalam makalah ini, *single variable polynomial* merujuk pada polinomial dengan bentuk $\sum_{i=0}^{n-1} a_i x^i$, $a_i \in \mathbb{R}, n \in \mathbb{N}$.

Untuk suatu sembarang polinomial $f(x)$, ada tiga operasi yang dapat dilakukan, yaitu:

1) Evaluasi; diberikan $x_0 \in \mathbb{R}$, hitung $f(x_0)$
2) Penjumlahan; diberikan dua polinomial $A(x)$ dan $B(x)$, hitung $C(x) = A(x) + B(x)$
3) Perkalian; diberikan dua polinomial $A(x)$ dan $B(x)$, hitung $C(x) = A(x)B(x)$

Sebuah polinomial $f(x)$ dengan derajat $n-1$ dapat direpresentasikan dengan n koefisien ataupun n buah titik (x, y) yang memenuhi $f(x) = y$. Misalnya, sebuah polinomial $f(x) = 1 + x^2$ dapat direpresentasikan dengan *list* koefisien (1, 0, 1) atau dengan *list* titik ((0, 1), (1, 2), (2, 5)). Perlu diingat bahwa n titik berbeda memiliki polinomial unik dengan derajat $n-1$ yang melewati semua titik tersebut.

Kedua representasi tersebut dapat digunakan untuk mendeskripsikan suatu polinomial $f(x)$. Perlu diperhatikan bahwa dalam mendeskripsikan suatu polinomial dengan *list* koefisien, penulisan *list* harus urut dari koefisien x^0 hingga koefisien x^{n-1}.

Dua representasi tersebut memiliki keuntungan dan kekurangannya masing-masing. Kompleksitas waktu untuk tiap representasi pada operasi di atas ditunjukkan pada tabel berikut

TABEL 1
KOMPLEKSITAS WAKTU OPERASI DARI SETIAP REPRESENTASI POLINOMIAL

label	koefisien	titik
evaluasi	$O(n)$	$O(n^2)$
penjumlahan	$O(n)$	$O(n)$
perkalian	$O(n^2)$	$O(n)$

Algoritma yang berkaitan dengan tiap operasi dan representasinya tidak dibahas dalam makalah ini. Dari tabel di atas, dapat dilihat bahwa tiap representasi kompleksitas waktu $O(n^2)$ pada suatu operasi.

C. Fast Fourier Transform

Algoritma Fast Fourier Transform adalah sebuah algoritma yang menggunakan teknik *divide and conquer*. Dalam makalah ini, algoritma FFT akan digunakan dalam konteks perkalian dua buah polinomial dalam representasi koefisien.

Perhatikan pada tabel 1 di atas, representasi titik dapat melakukan operasi perkalian dalam $O(n)$. Fakta ini membuat kita untuk merancang sebuah algoritma yang dapat mengubah sebuah representasi koefisien dari sebuah polinomial menjadi representasi titik. Secara formal, kita ingin sebuah set X dan mengevaluasi $f(x)$ untuk setiap $x \in X$. Diperlukan $|X| = n$ untuk sebuah polinomial f dengan derajat $n-1$. Namun, proses evaluasi secara naif akan berujung pada kompleksitas $O(n^2)$.

Dalam proses ini kita dapat menggunakan teknik *divide and conquer*. Untuk suatu polinomial F dengan derajat $n-1$, definisikan F_{even} dan F_{odd} masing-masing sebagai polinomial yang mengandung koefisien genap dan ganjil pada F. Atau secara formal:

$$F_{even} = \sum_{i=0}^{floor\left(\frac{n-1}{2}\right)} F_{2i} x^i \tag{6}$$

$$F_{odd} = \sum_{i=0}^{floor\left(\frac{n-2}{2}\right)} F_{2i+1} x^i \tag{7}$$

Fungsi $floor(X)$ mengeluarkan bilangan bulat terbesar Y sedemikian sehingga $Y \leq X$.

Misal, jika $F(x) = 1 + x^2 + 2x^3$, $F_{even}(x) = 1 + x$ dan $F_{odd}(x) = 2x$.

Dengan definisi tersebut, mudah dilihat bahwa berlaku

$$F(x) = F_{even}(x^2) + x \cdot F_{odd}(x^2) \tag{8}$$

Jadi, jika kita dapat menghitung polinomial F_{even} dan F_{odd}, kita dapat menghitung polinomial F. Definisikan $degree(F)$ sebagai derajat dari suatu polinomial F.

Skema algoritma sejauh ini adalah:

1. Buat suatu fungsi $FFT(F, X)$ yang mengambil sebuah polinomial F dan himpunan X sebagai argumen dan mengembalikan himpunan titik L sehingga untuk setiap $x \in X$, $(x, F(x)) \in L$.
2. Basis: jika polinomial F berderajat 0, lakukan proses evaluasi untuk seluruh $x \in X$. Proses ini membutuhkan waktu $O(|X|)$.
3. Divide: jika tidak, panggil $FFT(F_{even}, X^2)$ dan $FFT(F_{odd}, X^2)$, dengan X^2 himpunan nilai kuadrat untuk seluruh $x \in X$. Proses ini membutuhkan waktu $O(degree(F) + |X|)$.
4. Conquer: untuk setiap $x \in X$, hitung nilai $F(x)$ dengan menggunakan persamaan (8). Proses ini membutuhkan waktu $O(|X|)$

Sayangnya, jika skema tersebut dianalisis, total kompleksitasnya masih $O(n^2)$. Hal ini dikarenakan kardinalitas himpunan X selama berjalannya algoritma tetap.

Supaya algoritma makin cepat, dibutuhkan himpunan X yang

kardinalitasnya berkurang jika kita mengkuadratkan seluruh elemennya. Dengan kata lain, $|X^2| < |X|$.

Sebuah fakta menarik: setiap bilangan kompleks memiliki 2 akar. Contoh, akar dari 1 adalah -1 dan 1, akar dari -1 adalah i serta $-i$.

Berbekal fakta tersebut, mudah dilihat bahwa jika kita memilih $X = \{1, -1\}$, kita akan mendapat $X^2 = \{1\}$. Sama halnya dengan $X = \{1, -1, i, -i\}$, kita akan mendapat $X^2 = \{1, -1\}$. Salah satu himpunan X yang memiliki properti ini adalah himpunan akar persatuan ke-n.

Dengan menggunakan X sebagai himpunan akar persatuan ke-n, didapat skema algoritma yang baru:

1. Buat suatu fungsi $FFT(F, X)$ yang mengambil sebuah polinomial F dan himpunan X sebagai argumen dan mengembalikan himpunan titik L sehingga untuk setiap $x \epsilon X$, $\big(x, F(x)\big) \epsilon L$.
2. Basis: jika polinomial F berderajat 0, lakukan proses evaluasi untuk seluruh $x \epsilon X$. Perhatikan bahwa pada tahap ini, $X = \{1\}$. Proses ini membutuhkan waktu $O(1)$.
3. Divide: jika tidak, panggil $FFT(F_{even}, X^2)$ dan $FFT(F_{odd}, X^2)$, dengan X^2 himpunan nilai kuadrat untuk seluruh $x \epsilon X$. Dengan menggunakan X sebagai himpunan akar persatuan, maka $|X^2| = \frac{|X|}{2}$. Proses ini membutuhkan waktu $O(degree(F) + |X|)$.
4. Conquer: untuk setiap $x \epsilon X$, hitung nilai $F(x)$ dengan menggunakan persamaan (8). Proses ini membutuhkan waktu $O(|X|)$

Jadi, untuk sebuah polinomial F dengan derajat $n - 1$ kita dapat memilih X sebagai himpunan akar persatuan ke-n. Perhatikan juga bahwa seluruh akar persatuan ke-N memiliki bentuk $\exp\left(\frac{i2\pi m}{n}\right), m \epsilon \{x | 0 \leq x < n, x \epsilon \mathbb{N}\}$. Definisikan $\omega_N = \exp\left(\frac{i2\pi}{n}\right)$, maka seluruh akar persatuan ke-n dapat diekspresikan sebagai $\omega_n^m, 0 \leq m < n$, dan m bilangan bulat. Untuk menghitung ω_N^m dapat digunakan pula bentuk deret yang didefinisikan sebagai berikut:

$$\omega_n^m = \exp\left(\frac{i2\pi}{n}\right) * \omega_n^{m-1}, \omega_n^0 = 1 \tag{9}$$

Dalam implementasinya, digunakan beberapa lemma mengenai akar persatuan untuk memudahkan perhitungan[2].

Lemma 1: untuk bilangan bulat $n \geq 0, k \geq 0, d \geq 0$ *berlaku* $\omega_{dn}^{dk} = \omega_n^k$

Bukti: $\omega_{dn}^{dk} = \exp\left(\frac{2\pi i}{dn}\right)^{dk} = \exp\left(\frac{2\pi i}{n}\right)^k = \omega_n^k$

Lemma 2: untuk bilangan bulat genap $n > 0$, *berlaku* $\omega_n^{n/2} = \omega_2 = -1$

Bukti: $\omega_n^{n/2} = \omega_{2*n/2}^{n/2} = \omega_2^1 = -1$

Lemma 3: untuk bilangan bulat genap $n > 0$, *berlaku* $(\omega_n^k)^2 = \left(\omega_n^{k+n/2}\right)^2 = \omega_{n/2}^k$

Bukti: $(\omega_n^k)^2 = \omega_{2*n/2}^{2*k} = \omega_{n/2}^k$

$\left(\omega_n^{k+n/2}\right)^2 = \omega_n^{2k+n} = \omega_n^{2k} * \omega_n^n = \omega_n^{2k} = \omega_{2*n/2}^{2*k} = \omega_{n/2}^k$

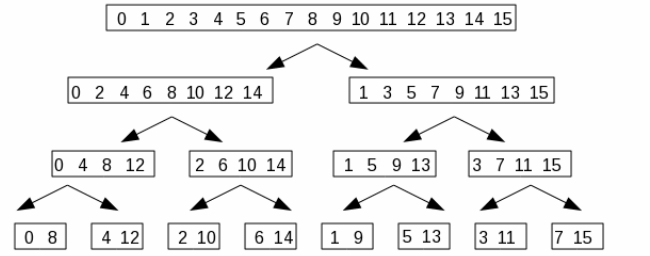
Lemma 4: untuk bilangan bulat genap $n \geq 0, k \geq 0$, *berlaku identitas* $\omega_n^{k+n/2} = -\omega_n^k$

Bukti: $\omega_n^{k+n/2} = \omega_n^k * \omega_n^{n/2} = \omega_n^k * -1 = -\omega_n^k$

Dengan lemma di atas, kita dapat menulis ulang persamaan (8) menjadi

$$F(\omega_n^k) = F_{even}((\omega_n^k)^2) + \omega_n^k * F_{odd}((\omega_n^k)^2),$$
$$F(\omega_n^k) = F_{even}\left(\omega_{n/2}^k\right) + \omega_n^k * F_{odd}\left(\omega_{n/2}^k\right), \text{untuk } k < n/2$$
$$F(\omega_n^k) = F_{even}\left(\omega_{n/2}^k\right) - \omega_n^k * F_{odd}\left(\omega_{n/2}^k\right), \text{untuk } k \geq n/2 \tag{10}$$

Skema algoritma di atas digambarkan dalam diagram di bawah:



Gambar 2 Diagram algoritma FFT[3]

Definisikan $T(n)$ sebagai waktu yang diperlukan untuk menjalankan algoritma FFT untuk suatu polinomial F berderajat n. Maka, didapat *formula*:

$$T(n) = 2 * T\left(\frac{n}{2}\right) + O(n) \tag{11}$$

Dengan menggunakan teorema Master, didapat $T(N) = nlog_2 n$. Maka dengan menjalankan algoritma FFT, sebuah representasi koefisien dapat diubah menjadi representasi titik dalam waktu $O(nlog_2 n)$.

D. Inverse Fast Fourier Transform

Setelah representasi koefisien diubah menjadi koefisien titik, proses perkalian dapat dilakukan dalam $O(n)$, mengingat perkalian polinomial dengan representasi titik dapat dilakukan dengan rumus:

$$C(x) = A(x) * B(x) \tag{12}$$

Ketika nilai $C(x)$ sudah dihitung untuk seluruh $x \epsilon X$, maka proses selanjutnya adalah mengubah representasi titik menjadi representasi koefisien.

Proses konversi dari representsi koefisien menjadi representasi titik dapat dituliskan dengan sebuah perkalian matriks:

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{0+1} & \cdots & \omega_n^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \cdots & \omega_n^{(n-1)^2} \end{pmatrix} \begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} \tag{13}$$

Pada persamaan (13), matriks di sebelah kiri disebut dengan matriks Vandermonde (V_n), dimana entri ke-(k, j) sama dengan ω_n^{kj}.

Mengembalikan representasi koefisien dari representasi titik, sama halnya dengan melakukan invers, atau seperti pada persamaan (13), perkalian dengan invers matriks.

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{0+1} & \cdots & \omega_n^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \cdots & \omega_n^{(n-1)^2} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_{n-1} \end{pmatrix} \tag{14}$$

Ternyata, entri ke-(k, j) pada invers dari matriks Vandermonde (V_n^{-1}) sama dengan ω_n^{-kj}/n. Bukti secara lengkap berada di luar cakupan makalah ini, sehingga akan dilakukan pembuktian terbalik yang menunjukkan bahwa hal tersebut benar adanya.

Untuk sembarang matriks persegi M, berlaku $MM^{-1} = I$, dengan I adalah matriks identitas. Sehingga, untuk matriks Vandermonde (V_n), berlaku

$$V_n V_n^{-1} = I \tag{15}$$

Lemma 5: untuk $n \geq 1$ dan sebuah bilangan bulat k yang bukan kelipatan n, berlaku $\sum_{j=0}^{n-1}(\omega_n^k)^j = 0$.

Bukti: $\sum_{j=0}^{n-1}(\omega_n^k)^j = \frac{(\omega_n^k)^n - 1}{(\omega_n^k - 1)}$, jumlah barisan geometri

$$\frac{(\omega_n^k)^n - 1}{(\omega_n^k - 1)} = \frac{(\omega_n^n)^k - 1}{(\omega_n^k - 1)} = \frac{1 - 1}{\omega_n^k - 1} = 0$$

Lemma 6: entri ke-(k, j) pada invers matriks Vandermonde (V_n^{-1}) sama dengan ω_n^{-kj}/n

Bukti: Definisikan $M(p, q)$ sebagai entri ke-(p, q) pada matriks $V_n^{-1}V_n$, maka berlaku

$$M(p,q) = \sum_{r=0}^{n-1}(\omega_n^{-rp}/n) * (\omega_n^{rq}) = \sum_{r=0}^{n-1}(\omega_n^{r(q-p)}/n)$$

Ketika $q \neq p$, berdasarkan lemma 5, hasil penjumlahannya 0. Namun ketika $q = p$, didapat $\omega_n^0 = 1$, sehingga hasil penjumlahannya adalah 1. Didapat $M(p, q) = 1$ ketika $p = q$, dan $M(p, q) = 0$ ketika $p \neq q$. Perhatikan bahwa matriks M adalah definisi dari matriks identitas, sehingga telah dibuktikan bahwa entri ke-(k, j) pada invers matriks Vandermonde (V_n^{-1}) sama dengan ω_n^{-kj}/n.

Didapat bahwa matriks Vandermonde untuk proses FFT dan IFFT sangatlah mirip. Kita hanya perlu menggunakan akar konjugat dari akar persatuan ke-n, dan kemudian membagi hasilnya dengan n. Karena inilah implementasi invers FFT hampir identikal dengan impelementasi FFT, hanya saja akar yang digunakan berbeda serta perlu dilakukan pembagian dengan n pada akhir perhitungan.

E. String

Dalam bidang *computer science*, *string* adalah sebuah sekuens karakter. *String* biasanya diimplementasikan seagai sebuah *byte array* yang menyimpan karakter dengan sebuah *character-encoding* (seperti ASCII, UTF-8).

Beberapa bahasa pemograman seperti C++, dan Ruby memungkinkan program untuk mengubah elemen *string* setelah deklarasi. Namun, dalam bahasa pemograman seperti Java dan Python, elemen sebuah *string* tidak dapat diubah (*immutable*).

IV. SOLUSI PERSOALAN

Definisikan suatu fungsi $CHAR2INT(c)$ yang bernilai -1 jika $c = "?"$ dan bernilai u dengan alfabet ke-u sama dengan c. Contoh, $CHAR2INT(a) = 0$, $CHAR2INT(b) = 1$. Definisikan barisan U dan V dengan panjang masing-masing n dan m sebagai

$$U_i = CHAR2INT(S_i) \tag{16}$$

$$V_i = CHAR2INT(P_i) \tag{17}$$

Tinjau dua buah polinomial A dan B dengan masing-masing derajat n dan m. Polinomial A dan B didefinisikan dengan[4]:

$$A_i = \cos(\alpha_i) + i * \sin(\alpha_i), \alpha_i = \frac{2\pi U_i}{26} \tag{18}$$

$$B_i = \begin{cases} \cos(\beta_i) - i * \sin(\beta_i), \beta_i = \frac{2\pi V_{m-i}}{26}, \mathrm{V_{m-i}} \neq -1 \\ 0, V_{m-i} = -1 \end{cases} \tag{19}$$

Tinjau sebuah polinomail $C(x) = A(x) * B(x)$.

$$C_{m+i} = \sum_{j=0}^{m} A_{i+j}B_{m-j}$$

$$C_{m+i} = \sum_{j=0}^{m}(\cos(\alpha_u) + i * \sin(\alpha_u)) * (\cos(\beta_v) - i * \sin(\beta_v))$$

$$u = i + j, v = m - j \tag{20}$$

Perhatikan bahwa jika $S_{i+j} = P_j$, pastilah $\alpha_{i+j} = \beta_j$. Kondisi ini membuat persamaan (20) menjadi:

$$C_{m+i} = \sum_{j=0}^{m}(\cos^2(\alpha_u) + \sin^2(\alpha_u)) = m \tag{21}$$

Persaman (21) menggunakan teorema Pythagoras yang menyatakan

$$\cos^2(x) + \sin^2(x) = 1 \tag{22}$$

Namun, jika $P_j = "?"$, maka $\beta_j = 0$, yang menghasilkan nilai 0. Jadi, jika $P_j = "?"$, dengan algoritma ini karakter tersebut seolah-olah akan "dilompati", karena tidak berpengaruh ke polinomial C.

Dengan observasi tersebut, misalkan CNT adalah jumlah karakter "?" pada string P. Maka suatu indeks i dikatakan cocok jika dan hanya jika $C_{i+m} = m - CNT$.

Penghitungan polinomial C dapat dilakukan dalam $O(nlogn)$ dengan menggunakan FFT. Sedangkan kalkulasi deret U dan V dapat dilakukan dalam $O(n)$. Total kompleksitas algoritma ini adalah $O(nlogn)$.

Berikut ini adalah implementasi algoritma tersebut dalam bahasa C++.

```
#include <iostream>
#include <vector>
#include <math.h>
#include <complex>
```

```
using namespace std;
using cNum = complex<double>;
const double PI = acos(-1);

void fft(vector<cNum> &F, bool invert){
    int n = F.size();
    if (n == 1){
        return;
    }
    vector<cNum> Feven(n / 2), Fodd(n / 2);
    for(int i = 0; 2 * i < n; ++i){
        Feven[i] = F[2 * i];
        Fodd[i] = F[2 * i + 1];
    }
    fft(Feven, invert);
    fft(Fodd, invert);

    double ang = 2 * PI / n;
    if(invert){
        ang *= -1;
    }
    cNum w = {1, 0}, wn = {cos(ang), sin(ang)};
    for(int i = 0; 2 * i < n; ++i){
        F[i] = Feven[i] + w * Fodd[i];
        F[i + n / 2] = Feven[i] - w * Fodd[i];
        if(invert) {
            F[i] /= 2;
            F[i + n / 2] /= 2;
        }
        w *= wn;
    }
}

vector<int> multiply(vector<cNum> const &A, vector<cNum> const &B){
    vector<cNum> copyA(A.begin(), A.end());
    vector<cNum> copyB(B.begin(), B.end());
    int n = 1;
    while(n < (int)copyA.size() + (int)copyB.size()){
        n <<= 1;
    }
    copyA.resize(n);
    copyB.resize(n);

    fft(copyA, false);
    fft(copyB, false);

    for(int i = 0; i < n; ++i){
        copyA[i] *= copyB[i];
    }
    fft(copyA, true);

    vector<int> result(n);
    for (int i = 0; i < n; ++i){
        result[i] = round(copyA[i].real());
    }
    return result;
}

int CHAR2INT(char c){
    if(c == '?'){
        return -1;
    }
    return c - 'a';
}

void stringMatchingWithFFT(string S, string P){
    int n = S.size();
    int m = P.size();
    int CNT = 0;
    vector<int> U(n);
    vector<int> V(m);
    for(int i = 0; i < n; ++i){
        U[i] = CHAR2INT(S[i]);
    }
    for(int i = 0; i < m; ++i){
        V[i] = CHAR2INT(P[i]);
        if(V[i] == -1){
            CNT++;
        }
    }
    vector<cNum> A(n);
    vector<cNum> B(m);
    int degB = m - 1;
    for(int i = 0; i < n; ++i){
        double alpha = (2.0 * PI * U[i]) / 26;
        cNum temp(cos(alpha), sin(alpha));
        A[i] = temp;
    }
    for(int i = 0; i < m; ++i){
        if(V[degB - i] != -1){
            double beta = (2.0 * PI * V[degB - i]) / 26;
            cNum temp(cos(beta), -sin(beta));
            B[i] = temp;
        }else{
            cNum temp(0, 0);
            B[i] = temp;
        }
    }
    vector<int> C = multiply(A, B);
    cout << "Indeks yang cocok adalah (0-indexed): " << endl;
    for(int i = 0; i <= n - m; ++i){
        if(C[degB + i] == m - CNT){
            cout << i << " ";
        }
    }
    cout << endl;
}

int main()
{
    string s1, s2;
```

```
    cout << "Masukkan string S: ";
    cin >> s1;
    cout << "Masukkan string P: ";
    cin >> s2;
    stringMatchingWithFFT(s1, s2);
    return 0;
}
```

V. KESIMPULAN

Persoalan *string matching* dapat diselesaikan dalam waktu $O(nlogn)$ dengan bantuan algoritma FFT serta perumusan secara matematik. Algoritma menggunakan teorema Pythagoras serta ilmu bilangan kompleks dalam perumusannya. Perumusan ini mengambil fakta bahwa perkalian sebuah bilangan kompleks dengan konjugatnya sama dengan 1. Untuk karakter *wildcard*, konstruksi algoritma menjadikan supaya hasil perkalian bernilai 0. Hal ini menjadikan karakter *wildcard* seolah-olah "dilompati" dalam proses pencocokan.

VI. APENDIKS

Algoritma FFT mengasumsikan polinomial F memiliki derajat $n-1$ dengan $n = 2^u, u \epsilon \mathbb{N}$. Jika kondisi ini tidak terpenuhi, sebelum melakukan FFT, perlu dilakukan *padding* terhadap koefisien F dengan nilai 0 hingga kondisi tersebut terpenuhi.

Dalam algoritma FFT, transformasi yang digunakan adalah transformasi DFT (*Discrete Fourier Transform*).

VII. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada:

1. Tuhan Yang Maha Esa,
2. orang tua penulis,
3. Bapak dan Ibu dosen pengampu mata kuliah Matematika Diskrit IF2120,
4. teman-teman penulis, dan
5. pihak-pihak lain yang telah mendukung penulis selama pembelajaran dan proses pengerjaan makalah ini yang tidak dapat penulis sebutkan satu per satu.

REFERENSI

[1] https://commons.wikimedia.org, David R. Tribble. https://commons.wikimedia.org/wiki/File:One5Root.svg. [Diakses 5 Desember 2023 15.53 WIB]

[2] https://codeforces.com, Tähvend Uustalu, "[Tutorial] FFT". https://codeforces.com/blog/entry/111371. [Diakses 5 Desember 2023 19.05 WIB]

[3] https://secretmango.com, Jim Blakey, "Discrete and Fast Fourier Transforms". http://www.secretmango.com/jimb/Whitepapers/fft/fft_v2.html. [Diakses 10 Desember 2023 18.54 WIB]

[4] https://cp-algorithms.com, Jakob Kogler, "Fast Fourier Transform". https://cp-algorithms.com/algebra/fft.html. [Diakses 7 Desember 2023 7.09 WIB]

[5] R. Munir, 'Homepage Rinaldi Munir'. https://informatika.stei.itb.ac.id/~rinaldi.munir. [Diakses 11 Desember 2023 15.09 WIB]

[6] Thomas, H., Charles, E., Ronald, R., Clifford, S., (2022). Introduction to Algorithms. *MIT Press.*

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 11 Desember 2023

Kristo Anugrah 13522024