Aplikasi Fungsi Pembangkit dalam Penyelesaian Persoalan *Stars and Bars* yang Dibatasi

Firizky Ardiansyah - 13520095
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13520095@std.stei.itb.ac.id

***Abstrak*—Fungsi pembangkit merupakan sebuah teknik menghitung dalam kombinatorial dengan mengeksploitasi sifat bijeksi sebuah persoalan. Teknik ini memungkinkan persoalan kombinatorial untuk dimodelkan ke dalam bentuk aljabar sehingga lebih mudah untuk dimanipulasi. Salah satu persoalan dalam kombinatorial yang cukup terkenal adalah persoalan *stars and bars*. Persoalan ini dapat dimodelkan menjadi fungsi pembangkit berbentuk suku banyak yang kemudian bisa dimanipulasi sehingga persoalan yang lebih umum bisa diselesaikan. Makalah ini akan membahas aplikasi fungsi pembangkit dalam penyelesaian persoalan *stars and bars* dengan peubah yang dibatasi serta implementasinya dalam algoritma.**

***Kata Kunci*—algoritma, fungsi pembangkit, kombinatorial, *stars and bars*.**

I. PENDAHULUAN

Persoalan *stars and bars* merupakan persoalan yang solusinya memiliki cakupan keterkaitan cukup luas di bidang kombinatorial terutama dalam perhitungan distribusi objek atau partisi objek. Persoalan ini digunakan dalam teknik perhitungan dasar kombinatorial yang membutuhkan pencarian banyaknya solusi sebuah persamaan. Solusi persoalan ini kemudian dikenal sebagai teknik *stars and bars* atau teorema *stars and bars*.

Misalkan dimiliki n buah kotak dan r buah objek identik. Persoalan *stars and bars* pada dasarnya meminta kita untuk menghitung banyaknya cara mendistribusi semua objek tersebut ke dalam kotak yang tersedia sedemikian sehingga setiap kotak berisi sejumlah objek atau tidak sama sekali. Bentuk yang lebih umum dikenali dari persoalan *stars and bars* adalah bentuk pencarian banyaknya solusi sebuah persamaan dengan beberapa peubah bilangan bulat non-negatif tak-diketahui. Jika didefinisikan $\mathbb{Z}^+$ adalah himpunan bilangan bulat non-negatif dan $\mathbb{Z}$ adalah himpunan seluruh bilangan bulat, secara matematis, diberikan $x_1, x_2, x_3, \ldots, x_n, r \in \mathbb{Z}^+$, persoalan *stars and bars* adalah pencarian banyaknya solusi persamaan $x_1 + x_2 + \cdots + x_n = r$.

Sebuah ekstensi dari persoalan *stars and bars* adalah dengan membatasi peubah yang tak diketahui dalam interval berhingga tertentu. Sehingga secara matematis, persoalan *stars and bars* menjadi pencarian banyaknya solusi $x_1 + x_2 + \cdots + x_n = r$, dengan $s_i \leq x_i \leq t_i$ untuk sembarang $s_i, t_i \in \mathbb{Z}, s_i \leq t_i$ untuk setiap $i \in [1, n]$ dan $r \in \mathbb{Z}^+$. Perhatikan bahwa s_i dan t_i tidak perlu non-negatif. Salah satu algoritma paling sederhana untuk penyelesaian masalah ini adalah dengan mengenumerasi setiap peubah dengan interval yang diberikan, sehingga memiliki kompleksitas relatif terhadap operasi dasar sejumlah $T(n) = \prod_{1 \leq i \leq n}(t_i - s_i)$.

Aplikasi fungsi pembangkit di bidang kombinatorial telah banyak digunakan untuk memudahkan manipulasi persoalan yang cukup kompleks. Sifat bijeksi dari sebuah persoalan yang kompleks dapat dieksploitasi untuk dapat dimodelkan menjadi persoalan yang lebih mudah dengan teknik yang disebut *double counting*. Fungsi pembangkit memodelkan persoalan yang kompleks dalam bentuk fungsi umum sehingga memungkinkan perhitungan yang lebih sistematik dengan manipulasi aljabar.

Pada implementasinya, kita akan melihat bahwa penggunaan fungsi pembangkit dalam pemodelan persoalan ini memerlukan algoritma efisien untuk mengalikan suku banyak. Salah satu algoritma yang cukup terkenal dalam pencarian perkalian dua buah suku banyak adalah algoritma *fast fourier transform* (FFT). Makalah ini akan menelisik lebih dalam penggunaan fungsi pembangkit untuk kasus yang diberikan sekaligus implementasinya dalam algoritma dan kompleksitasnya.

II. LANDASAN TEORI

A. Kombinatorial

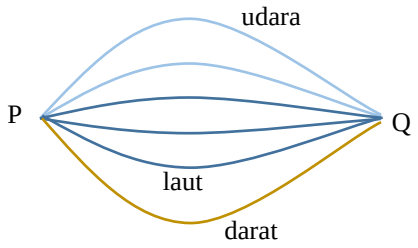
Kombinatorial adalah cabang matematika untuk menghitung (*counting*) jumlah penyusunan objek-objek tanpa harus mengenumerasi semua kemungkinan. [9] Terdapat dua buah prinsip dalam kombinatorial yaitu *the addition principle* (AP) dan *the multiplication principle* (MP). [1] prinsip ini merupakan operasi paling dasar dalam menyelesaikan persoalan-persoalan perhitungan enumerasi.

Jika dimiliki k kejadian dengan:

1. Banyak cara untuk mencapai kejadian ke-i adalah n_i untuk setiap $i \in [1, k]$,
2. Setiap kejadian saling lepas, artinya kejadian ke-i tidak ada kaitannya dengan kejadian ke-j, untuk sembarang i dan j,

menurut kaidah penjumlahan atau AP, banyaknya cara untuk setidaknya satu kejadian terjadi adalah $n_1 + n_2 + \cdots + n_k$. [1] Contoh aplikasi kaidah penjumlahan adalah dalam persoalan pencarian banyaknya cara seseorang pergi dari kota P ke kota Q jika ada 2 cara untuk melalui jalur udara, 3 cara untuk melalui jalur laut, dan 1 cara untuk melalui jalur darat. Menurut AP,

diperoleh banyaknya cara dari kota P ke kota Q adalah $2 + 3 + 1 = 6$ cara.



Gambar 1 Ilustrasi Persoalan Menggunakan AP

Adapun kaidah perkalian atau MP menyatakan bahwa jika sebuah kejadian yang dapat dipartisi menjadi k kejadian terurut dengan banyak cara untuk mencapai kejadian ke-i adalah n_i untuk setiap $i \in [1, k]$, total banyaknya cara untuk kejadian tersebut dapat terjadi adalah $n_1 n_2 n_3 \dots n_k$. [1] Contoh aplikasinya adalah ketika seseorang ingin pergi dari kota A ke kota D dan harus melewati kota B dan kota C seperti berikut:

Gambar 2 Ilustrasi Persoalan Menggunakan MP[1].

Total banyaknya cara untuk pergi dari kota A ke kota D adalah perkalian banyaknya cara dari kota A ke kota B, dari kota B ke kota C, dan dari kota C ke kota D. Sehingga diperoleh total banyaknya cara adalah $2x5x3 = 30$ cara.

Prinsip dasar kombinatorial ini akan dijadikan alat untuk memperoleh teknik-teknik kombinatorial lain yang lebih penting seperti permutasi dan kombinasi.

B. Permutasi

Misalkan terdapat n buah objek tidak identik dan r buah kotak dengan $r \leq n$. Banyaknya cara berbeda untuk mengisi setiap kotak sedemikian sehingga setiap kotak tepat memiliki sebuah objek adalah sebagai berikut:

- Banyaknya cara untuk mengisi kotak pertama adalah n.
- Banyaknya cara untuk mengisi kotak kedua adalah $(n-1)$.

  ...
- Banyaknya cara untuk mengisi kotak ke-r adalah $(n-r-1)$.

Berdasarkan MP, diperoleh bahwa banyaknya cara mengisi r buah kotak dengan setiap kotak berisi tepat sebuah objek dari n buah objek dan $r \leq n$ adalah $n(n-1) \dots (n-r-1)$. Bentuk kasus ini disebut sebagai permutasi. Secara umum, permutasi r dari n elemen dinotasikan dengan $P_r^n = \frac{n!}{(n-r)!}$. Simbol tanda seru adalah faktorial, didefinisikan sebagai $x! = x.(x-1) \dots 2.1$ dan $0! = 1$. [9]

Perhatikan bahwa permutasi n dari n elemen ekuivalen dengan banyaknya pengacakan n buah objek, dinotasikan sebagai $P_n = n!$.

C. Kombinasi

Kombinasi adalah bentuk khusus dari permutasi, bedanya dengan permutasi, kombinasi tidak memperhitungkan urutan kemunculan. [9] Misalkan A adalah sebuah himpunan dengan setiap elemennya berbeda satu sama lain. Himpunan-kombinasi dari A didefinisikan sebagai subhimpunan dari A. Lebih khusus, himpunan-kombinasi-r dari A adalah subhimpunan A yang memiliki r elemen. [1]

Kombinasi r dari n adalah banyaknya himpunan-kombinasi-r dari sebuah himpunan dengan n elemen. Untuk mengaitkan kombinasi dengan permutasi, agar memperoleh nilai kombinasi r dari n elemen sesuai definisi, tinjau bahwa banyaknya cara memasukkan n objek ke dalam r kotak adalah P_r^n. Karena urutan objek di setiap kotak dalam definisi kombinasi tidak diperhatikan, diperoleh bahwa kombinasi r dari n elemen adalah P_r^n / P_r^r, secara matematis,

$$C_r^n = \binom{n}{r} = \frac{P_r^n}{P_r^r} = \frac{n!}{(n-r)!\, r!}.$$

Simbol kolom untuk merepresentasikan kombinasi akan lebih banyak digunakan terutama dalam aplikasi kombinasi dalam perhitungan ekspansi binomial.

Perhatikan bahwa kombinasi r dari n elemen hanya terdefinisi saat $r \leq n$. Adapun properti penting dalam kombinasi, yaitu,

$$\binom{n}{n-r} = \frac{n!}{(n-(n-r))!\,(n-r)!} = \binom{n}{r}.$$

D. Teorema Binomial

Teorema binomial adalah metode kombinatorial dalam mencari koefisien sebuah suku dalam ekspansi binomial berbentuk $(x+y)^n$, teorema ini ditemukan oleh Isaac Newton pada 1676. [1] Secara matematis, untuk setiap bilangan bulat $n \geq 0$, berlaku

$$(x+y)^n = \binom{n}{0} x^0 y^n + \binom{n}{1} x^1 y^{n-1} + \dots + \binom{n}{n-1} x^{n-1} y^1 + \binom{n}{n} x^n y^0 = \sum_{r=0}^{n} \binom{n}{r} x^r y^{n-r}.$$

Menariknya, koefisien-koefisien ini bisa disusun dalam segitiga yang sangat terkenal disebut segitiga paskal. Setiap baris dari segitiga paskal merepresentasikan koefisien hasil binomial seperti pada contoh berikut.

$(x+y)^0 = 1$: 1
$(x+y)^1 = x + y$: 1 1
$(x+y)^2 = x^2 + 2xy + y^2$: 1 2 1
$(x+y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$: 1 3 3 1
$(x+y)^4 = x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$: 1 4 6 4 1
$(x+y)^5 = x^5 + 5x^4y + 10x^3y^2 + 10x^2y^3 + 5xy^4 + y^5$: 1 5 10 10 5 1

Gambar 3 Segitiga Pascal.[10]

Lebih lanjut, teorema ini memungkinkan manipulasi bentuk aljabar binomial, sehingga komputasi kombinatorial akan lebih variatif dan mudah dicari. Contohnya adalah pencarian jumlahan kombinasi n sebagai berikut

$$S = \sum_{r=0}^{n} \binom{n}{r}, n \geq 0.$$

Penjumlahan tersebut dapat dicari dengan mengeksploitasi ruas kiri teorema binomial, sulihkan $x = 1$ dan $y = 1$, diperoleh

$$(1+1)^n = \sum_{r=0}^{n} \binom{n}{r} (1)^r (1)^{n-r} = S \Rightarrow S = 2^n.$$

E. Prinsip Bijeksi

Sebuah korespondensi $f: A \to B$ disebut bijektif jika dan hanya jika f injektif dan surjektif. f injektif jika $f(a_1) \neq f(a_2)$ untuk setiap $a_1 \neq a_2$ di A, dan surjektif jika $\forall b \in B \to \exists a. (a \in A, f(a) = b)$.

Prinsip bijeksi menyebutkan bahwa sebuah solusi persoalan yang dapat dibangun menjadi sebuah himpunan berhingga memiliki sifat bijeksi jika terdapat sebuah pemetaan bersifat bijeksi f yang memetakan persoalan tersebut ke persoalan lain. Secara analitis, misalkan solusi suatu persoalan dapat dibangun menjadi himpunan berhingga A, definisikan persoalan lain yang memiliki solusi berhingga pada himpunan B. Jika terdapat bijeksi dari A ke B, haruslah $|A| = |B|$. Dengan kata lain, banyaknya solusi dari A sama dengan banyaknya solusi dari B. Prinsip ini juga dikenal sebagai *double counting*. [1]

F. Teorema Akar Persatuan dan Formula Euler

Definisikan i adalah bilangan kompleks, yaitu $i^2 = -1$. Formula euler menyatakan bahwa,

$$e^{ix} = cis(x) = \cos(x) + isin(x).$$

Adapun akar persatuan adalah akar-akar kompleks dari persamaan $x^n = 1$ untuk sebuah $n \in \mathbb{N}$. Teorema akar persatuan menyatakan bahwa akar kompleks persamaan $x^n = 1$ akan berbentuk $e^{\frac{2k\pi i}{n}}$ untuk $k \in [1, n]$. Bukti teorema ini bisa diperoleh menggunakan formula euler. Perhatikan bahwa

$$e^{2\pi i} = 1 \Rightarrow e^{2k\pi i} = 1, \forall k \in \mathbb{N}$$

Akibatnya, tinjau bahwa

$$x^n = 1 = e^{2\pi i} = e^{4\pi i} = e^{6\pi i} = \cdots$$

Pangkatkan $\frac{1}{n}$ kedua ruas, diperoleh sebuah himpunan akar-akar persatuan derajat n yaitu

$$U_n = \left\{ e^{\frac{2k\pi i}{n}} \mid k \in \{1,2,3, \ldots, n\} \right\}.$$

Akar persatuan memiliki banyak kegunaan di berbagai bidang. Dalam kombinatorial, akar persatuan banyak digunakan dalam fungsi pembangkit dan kita akan lihat juga bahwa akar persatuan adalah alat paling penting dalam algoritma FFT. [4]

G. Teorema Stars and Bars

Pada bagian ini, akan dibahas lebih dalam teorema *stars and bars.* Seperti yang dijelaskan pada bagian pendahuluan, persoalan *stars and bars* menyatakan bahwa jika dimiliki sebuah bilangan non-negatif r dan n buah peubah non-negatif $x_i, i \in [1, n]$, dengan $x_1 + x_2 + \cdots + x_n = r$, kita perlu mencari banyaknya solusi dari persamaan tersebut.

Ide penyelesaian persoalan ini adalah dengan membangun bijeksi dengan persoalan yang lebih mudah. Pertimbangkan r buah bintang identik, $n-1$ buah batang identik, dan $n+r-1$ kotak, banyaknya cara menempatkan r bintang ke dalam $n+r-1$ kotak kemudian sisanya diisi oleh $n-1$ batang adalah $\binom{n+r-1}{r}$.

Bisa dibangun sebuah korespondensi satu-satu dari persoalan bintang dan batang ini dengan persoalan *stars and bars*. Tinjau sebuah kasus ketika batang dan bintang telah dimasukkan ke dalam kotak. x_i berkorespondensi dengan banyaknya bintang di sebelah kiri batang ke-i dan di sebelah kanan batang ke-$(i-1)$ untuk $2 \leq i \leq n-1$, x_1 berkorespondensi dengan banyaknya bintang di sebelah kiri batang pertama, sedangkan x_n berkorespondensi dengan banyaknya bintang di sebelah kanan batang ke-$(n-1)$. Mudah ditunjukkan bahwa korespondensi ini merupakan korespondensi satu-satu antara persoalan *stars and bars* dengan bintang dan batang. Berdasarkan teorema bijeksi, karena solusi persoalan bintang dan batang adalah $\binom{n+r-1}{r}$, diperoleh solusi persoalan $stars\ and\ bars$ juga $\binom{n+r-1}{r}$.

Untuk memvisualisasikan bijeksi lebih baik, tinjau $a+b+c=13$, dengan $a, b, c \in \mathbb{Z}^+$. Pertimbangkan sebuah kombinasi bintang dan batang berikut {****|***|******}, sesuai definisi korespondensi, representasi tersebut menunjukkan salah satu solusi persamaan $a+b+c=13$, yaitu $a=4, b=3, c=6$. Perhatikan bahwa di antara dua batang tidak perlu ada bintang, hal ini merepresentasikan nilai 0 pada korespondensi di persoalan *stars and bars*. [3]

Lebih lanjut, *stars and bars* bisa digeneralisasi dengan membatasi x_i pada interval bawah tertentu. Jika dimiliki sebuah bilangan non-negatif r dan n buah peubah non-negatif $x_i, i \in [1, n]$, dengan $x_1 + x_2 + \cdots + x_n = r$, definisikan $s_i \in \mathbb{Z}, \forall i \in [1, n]$, sedemikian sehingga $x_i \geq s_i\ \forall i \in \mathbb{Z}$, lagi-lagi diminta untuk menemukan banyaknya solusi persamaan tersebut. Ide penyelesaian persoalan ini adalah dengan membuat peubah baru, misalnya $a_i = x_i - s_i, \forall i \in [1, n]$, perhatikan bahwa $a_i \geq 0, \forall i \in [1, n]$, akibatnya, x_i pada persamaan awal dapat kita sulihkan dengan $a_i + s_i$, sehingga $(a_1 + s_1) + (a_2 + s_2) + \cdots + (a_n + s_n) = r$, atau $a_1 + a_2 + \cdots + a_n = r - \Sigma s_i$. Persamaan terakhir merupakan bentuk umum persoalan *stars and bars*. Jika $r - \Sigma s_i \geq 0$ dan s_i diketahui untuk setiap $i \in [1, n]$, solusi persoalan ini adalah $\binom{n+r-\Sigma s_i-1}{r-\Sigma s_i}$.

H. Kompleksitas Waktu Algoritma

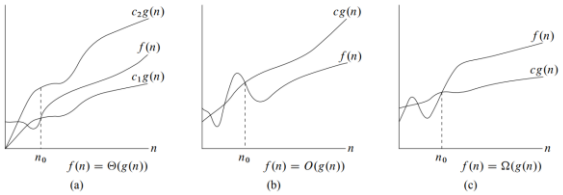
Pekerjaan utama di dalam kompleksitas waktu adalah menghitung jumlah tahapan komputasi di dalam algoritma. Tahapan komputasi diukur, biasanya, berdasarkan banyaknya operasi dasar yang dilakukan dalam algoritma. Ukuran kompleksitas waktu sebuah algoritma ditetapkan berdasarkan besarnya ukuran masukan. Ada tiga macam kompleksitas waktu:

1. $T_{max}(n)$: Kompleksitas waktu untuk kasus terbutuk (*worst case)*, kebutuhan waktu maksimum,
2. $T_{min}(n)$: Kompleksitas waktu untuk kasus terbaik (*best case*), kebutuhan waktu minimum,
3. $T_{avg}(n)$: Kompleksitas waktu untuk kasus rata-rata (*average case*), kebutuhan waktu secara rata-rata. [11]

Jenis kompleksitas waktu yang dipertimbangkan dalam makalah ini adalah kompleksitas waktu maksimum.

Representasi kompleksitas waktu biasanya dituliskan dalam bentuk sifat asimtotiknya, terdapat 3 buah fungsi asimtotik, yaitu *big-O*, *big-Omega*, dan *big-Theta*. Masing-masing definisinya adalah sebagai berikut:

1. *big-O*, atau $O(f(n))$ adalah fungsi *asimtotik atas* dari $T(n)$ jika terdapat konstanta C dan n_0 sedemikian sehingga $T(n) \leq Cf(n)\ \forall n \geq n_0$.
2. *big-Omega*, atau $\Omega(f(n))$ adalah fungsi *asimtotik bawah* dari $T(n)$ jika terdapat konstanta C dan n_0 sedemikian sehingga $T(n) \geq Cf(n)\ \forall n \geq n_0$.
3. *big-Theta*, atau $\Theta(f(n))$ adalah fungsi *asimtotik* dari $T(n)$ jika $T(n)$ memiliki asimtotik bawah $\Omega(f(n))$ dan asimtotik atas $O\big(f(n)\big)$.



Gambar 4 (a) Fungsi Asimtotik (b) Fungsi Asimtotik Atas (c) Fungsi Asimtotik Bawah[2]

Pada umumnya, representasi fungsi asimtotik kompleksitas waktu yang digunakan adalah fungsi asimtotik atas atau $O(f(n))$. Properti yang paling umum dari fungsi asimtotik ini adalah sebagai berikut,

1. $O\big(f(n) + g(n)\big) = O(\max\big(f(n), g(n)\big)$.
2. $O(f(n)g(n)) = O\big(f(n)\big)O\big(g(n)\big)$.
3. $O\big(cf(n)\big) = O\big(f(n)\big)$.
4. $f(n) = O\big(f(n)\big)$.
5. $O\big(P(n)\big) = O(x^n)$, dengan $P(n) = a_0x^n + a_1x^{n-1} + \cdots + a_n$. [12]

I. Fungsi Pembangkit

Salah satu alat yang sangat penting untuk proses komputasi dalam kombinatorial adalah konsep fungsi pembangkit. Misalkan $(a_r) = (a_0, a_1, \dots, a_r, \dots)$ sebuah barisan bilangan, bentuk umum fungsi pembangkit dari barisan tersebut adalah barisan geometri

$$A(x) = \sum_{r=0}^{\infty} a_r x^r = a_0 + a_1x + a_2x^2 + \cdots.$$

Dalam mempertimbangkan fungsi pembangkit, x selalu diasumsikan sedemikian sehingga fungsi pembangkit konvergen karena fokus utama dalam fungsi pembangkit di ranah kombinatorial adalah koefisien dari fungsi pembangkit tersebut. Operasi paling dasar yang bisa diterapkan pada fungsi pembangkit adalah operasi penjumlahan dan perkalian.

Contoh pencarian fungsi pembangkit, misalnya dimiliki sebuah barisan $\left(\binom{n}{0}, \binom{n}{1}, \dots, \binom{n}{n}, 0,0, \dots\right)$, dari definisi, fungsi pembangkit barisan tersebut akan membentuk barisan koefisien yang diperoleh dari teorema binomial yaitu $A(x) = (1 + x)^n$.

Contoh lain, tinjau sebuah deret geometri tak hingga S_∞ sebagai

$$S_\infty = 1 + x + x^2 + x^3 + \cdots$$

Jumlahan deret geometri pada kasus di atas memiliki nilai konvergensi $S_\infty = \frac{1}{1-x}$ untuk $|x| < 1$. Akibatnya,

$$\frac{1}{1-x} = 1 + x + x^2 + \cdots.$$

Hal ini menunjukkan bahwa fungsi pembangkit untuk barisan $(1,1,1,1, \dots)$ adalah $A(x) = \frac{1}{x-1}$. Perhatikan bahwa perolehan di atas dapat digeneralisasi dengan sedikit manipulasi. Misalnya, dengan menurunkan $A(x)$ sebanyak n kali, akan diperoleh fungsi pembangkit baru dari barisan baru, yaitu

$$A^{(n)}(x) = \frac{n!}{(1-x)^{n+1}} = \sum_{r=0}^{\infty} \frac{(r+n)!}{r!} x^r$$

Persamaan di atas mudah untuk dibuktikan menggunakan induksi matematika. Jika n disulihkan dengan $n-1$, akan diperoleh

$$\frac{1}{(1-x)^n} = \sum_{r=0}^{\infty} \frac{(r+n-1)!}{r!\,(n-1)!} x^r = \sum_{r=0}^{\infty} \binom{r+n-1}{r} x^r.$$

Secara umum, fungsi pembangkit untuk barisan $\left(\binom{n-1}{0}, \binom{n}{1}, \binom{n+1}{2}, \dots, \binom{r+n-1}{r}, \dots\right)$ adalah $A(x) = \frac{1}{(1-x)^n}$. [1]

J. Algoritma Fast Fourier Transform

Sebuah suku banyak berderajat $n-1$, $P(x) = a_0x^0 + a_1x^1 + a_2x^2 + \cdots + a_{n-1}x^{n-1}$ dapat direpresentasikan dalam bentuk vektor koefisien $(a_0, a_1, a_2, \dots, a_{n-1})$.

Definisikan *Discrete Fourier Transform* (DFT) dari sebuah suku banyak $P(x)$ berderajat $n-1$ sebagai sebuah suku banyak baru yang koefisiennya adalah $P(\omega_{n,k})$ untuk setiap $0 \leq k \leq n-1$ dengan $\omega_{n,k}$ adalah akar persatuan ke-k dari $x^n = 1$. Secara matematis, DFT bisa ditulis dalam representasi vektor koefisiennya sebagai berikut

$$DFT(a_0, a_1, a_2, \dots, a_{n-1}) = \Big(P\big(\omega_{n,0}\big), P\big(\omega_{n,1}\big), \dots, P\big(\omega_{n,n-1}\big)\Big)$$

Perhatikan bahwa $\omega_{n,k} = e^{\frac{2\pi k}{n}} = \left(e^{\frac{2\pi}{n}}\right)^k = \big(\omega_{n,1}\big)^k$, misalkan $\omega_{n,1} = \omega$, diperoleh, $DFT(a_0, a_1, a_2, \dots, a_{n-1}) = \big(P(\omega^0), P(\omega^1), \dots, P(\omega^{n-1})\big)$. Selain itu, definisikan juga DFT^{-1} sebagai invers DFT dari P yaitu $DFT^{-1}(DFT(P)) = P$.

Perhatikan saat dua buah suku banyak dikalikan, misal A dan B, nilai dari $P = (A \cdot B)$ pada sembarang titik akan sama dengan perkalian hasil A di titik tersebut dan B di titik tersebut. Secara matematis,

$$(A \cdot B)(x) = A(x) \cdot B(x).$$

Dampak dari properti ini, bisa diperoleh hasil perkalian dua buah suku banyak menggunakan persamaan

$$A \cdot B = DFT^{-1}(DFT(A) \cdot DFT(B))$$

Algoritma FFT, memungkinkan pencarian DFT sebuah suku banyak dengan kompleksitas waktu $O(n \log n)$. Ide algoritma

ini adalah membagi suku banyak P menjadi

$$P(x) = P_1(x^2) + xP_2(x^2).$$

Asumsikan P berderajat $n-1$, dengan n adalah perpangkatan 2. Definisikan $P_1(x) = (a_0, a_2, \dots, a_{n-2})$ dan $P_2(x) = (a_1, a_3, \dots, a_{n-1})$. Mudah dilihat bahwa P_1 dan P_2 masing-masing berderajat $\frac{n}{2} - 1$. Tinjau suku ke-k dari $DFT(P)$,

$$DFT(P)_k = DFT(P_1)_k + \omega^k DFT(P_2)_k$$

untuk setiap $k \in [0, \frac{n}{2} - 1]$, persamaan tersebut merupakan perolehan langsung dari hubungan P dengan P_1 dan P_2. Untuk nilai $k \in [\frac{n}{2}, n-1]$ bisa diperoleh dengan persamaan berikut

$$DFT(P)_k = P(\omega^k) = P_1(\omega^{2k}) + \omega^k P_2(\omega^{2k}).$$

Misalkan $k = \frac{n}{2} + m$, dengan $m \in [0, \frac{n}{2} - 1]$ diperoleh

$$DFT(P)_{\frac{n}{2}+m} = P_1(\omega^{n+2m}) + \omega^m \omega^{\frac{n}{2}} P_2(\omega^{n+2m}).$$

Berdasarkan formula euler, $\omega^n = \cos(2\pi) + i\sin(2\pi) = 1$, dan $\omega^{\frac{n}{2}} = \cos(\pi) + i\sin(\pi) = -1$, diperoleh

$$\begin{aligned} DFT(P)_{\frac{n}{2}+m} &= P_1(\omega^{2m}) - \omega^m P_2(\omega^{2m}) \\ &= DFT(P_1)_m - \omega^m DFT(P_2)_m. \end{aligned}$$

Hasil tersebut memungkinkan pencarian DFT secara rekursif maupun iteratif.

Misalkan $T_{DFT}(n)$ adalah kompleksitas waktu pencarian DFT. Karena setiap operasi polinomial dibagi menjadi dua buah polinomial baru berderjat setengahnya, diperoleh

$$T_{DFT}(n) = T_{DFT}\left(\frac{n}{2}\right) + T(op).$$

Pencarian DFT pada satu operasi memerlukan operasi $assignment$ sebanyak n kali, sehingga memiliki kompleksitas linear, diperoleh

$$T_{DFT}(n) = T_{DFT}\left(\frac{n}{2}\right) + T(n).$$

Jika $n = 2^k$, akan diperoleh $T_{DFT}(n) = k \cdot n = O(n \log n)$.

Pencarian invers DFT, seperti terdapat pada [6] membutuhkan pengetahuan mengenai matriks vandermonde dan *advanced calculus*, pada makalah ini, penulis hanya akan memberikan formula langsung dari invers DFT, yaitu

$$DFT^{-1}(Q)_k = \frac{1}{n}\sum_{m=0}^{n-1}(DFT(P)_m \omega^{-km}).$$

Algoritma efisien untuk memperoleh penjumlahan di atas adalah dengan algoritma yang sama dengan pencarian DFT, perbedaannya yaitu alih-alih ω^k, yang digunakan adalah ω^{-k}. Selain itu, pada akhir komputasi, koefisien yang diperoleh perlu dibagi dengan n sesuai dengan formula di atas. Kompleksitas waktu pencarian invers DFT sama dengan pencarian DFT yaitu $O(n \log n)$.

Perhatikan bahwa dalam algoritma FFT, tipe data yang digunakan adalah bilangan kompleks, sehingga dimungkinkan adanya ketidaktelitian dalam komputasi. Total kompleksitas waktu untuk algoritma FFT adalah $O(n \log n)$.

III. Pencarian Banyaknya Solusi Persamaan yang Dibatasi

Sekarang kita siap untuk memecahkan persoalan *stars and bars*. Meninjau kembali, *problem statement* persoalan *stars and bars* yang dibatasi ini adalah sebagai berikut.

Diberikan dua buah bilangan bulat non negatif n dan r, serta barisan bilangan bulat $(s_1, s_2, s_3, \dots, s_n)$, dan $(t_1, t_2, t_3, \dots, t_n)$, dengan $s_i \le t_i \; \forall i \in [1, n]$. Partisi r menjadi n buah elemen, $x_i, \forall i \in [1, n]$ sedemikian sehingga $x_1 + x_2 + \cdots + x_n = r$, dengan $s_i \le x_i \le t_i$ untuk setiap $i \in [1, n]$. Tentukan banyaknya cara mempartisi r dengan batasan yang diberikan.

Kita akan menggunakan fungsi pembangkit untuk menyelesaikan persoalan ini. Namun, sebelum itu, bentuk *stars and bars* di atas bisa digeser batas bawahnya menjadi 0, misalkan $x_i = a_i + s_i, \forall i \in [1, n]$, diperoleh batas $0 \le a_i \le t_i - s_i, \forall i \in [1, n]$ dan $a_i + a_2 + \cdots + a_n = r - \sum_{i=1}^{n} s_i$. Misalkan $p_i = t_i - s_i$, dan $r' = r - \sum_{i=1}^{n} s_i$, persoalan di atas disederhanakan menjadi pencarian banyaknya solusi non negatif $(a_1, a_2, \dots, a_n)$, sedemikian sehingga

$$a_1 + a_2 + a_3 + \cdots + a_n = r'$$

dengan $a_i \le p_i$, jika diberikan n, r' dan barisan $(p_1, p_2, \dots, p_n)$.

Akan dibangun sebuah fungsi pembangkit untuk persoalan ini. Misalkan sebuah barisan $(y_1^{(n)}, y_2^{(n)}, \dots, y_r^{(n)}, \dots)$ adalah barisan solusi untuk persoalan di atas, dengan $y_i^{(n)}$ adalah banyaknya solusi dari $a_1 + a_2 + a_3 + \cdots + a_n = i$. Fungsi pembangkit untuk barisan ini adalah

$$A^{(n)}(x) = \sum_{j=0}^{\infty} y_j^{(n)} x^j = y_0^{(n)} + y_1^{(n)} x + y_2^{(n)} x^2 + \cdots$$

Perhatikan bahwa $A^{(k)}(x)$ menunjukkan penjumlahan yang sedang dikonsiderasi, yaitu penjumlahan a_1 hingga a_k, atau $a_1 + a_2 + \cdots + a_k$.

Dari definisi fungsi pembangkit di atas pandang subpersoalan berikut:

- *Solusi partisi menjadi sebuah bilangan a_1*. perhatikan bahwa solusi $a_1 = \alpha$ memiliki satu cara untuk $0 \le \alpha \le p_1$ dan 0 cara di luar interval tersebut. Akibatnya, fungsi pembangkit pada subpersoalan ini adalah fungsi pembangkit dari barisan $(1,1,1,1, \dots, 1,0,0, \dots)$, dengan banyaknya angka 1 pada barisan tersebut adalah $p_1 + 1$, atau $A^{(1)}(x) = A_1(x) = \sum_{j=0}^{p_1} x^j$.

- *Solusi partisi menjadi dua buah bilangan a_1 dan a_2*. tinjau bahwa subpersoalan ini bisa dipecah menjadi subpersoalan sebelumnya, yaitu mencari banyaknya solusi untuk $a_1 = \alpha$ dan banyaknya solusi untuk $a_2 = \beta$. Jika fungsi pembangkit untuk $a_1 = \alpha$ diperoleh $A_1(x)$ dan untuk $a_2 = \beta$ fungsi pembangkitnya adalah $A_2(x)$, mudah dibuktikan bahwa koefisien x^i dari $A_1(x)A_2(x)$ adalah banyaknya solusi persamaan $a_1 + a_2 = i$. Dengan demikian, fungsi pembangkit dari subpersoalan ini adalah $A^{(2)}(x) = A_1(x)A_2(x)$.

Akan dibuktikan secara induksi, bahwa fungsi pembangkit solusi persoaalan *stars and bars* terbatas dapat dituliskan sebagai

$$A^{(n)}(x) = A_1(x)A_2(x) \dots A_n(x)$$

dengan $A_i(x)$ adalah fungsi pembangkit dari solusi persamaan $a_i = \alpha$ untuk sembarang α. Bentuk eksplisit fungsi $A_i(x)$ sama seperti pada subpersoalan pertama, yaitu $A_i(x) =$

$\sum_{j=0}^{p_i} x^j$ untuk setiap $i \in [1, n]$.

Basis induksi telah dibuktikan pada subpersoalan pertama dan kedua. Asumsikan $A^{(n-1)}(x) = A_1(x)A_2(x) \dots A_{n-1}(x)$ adalah solusi persoalan *stars and bars* dengan $n-1$ peubah. Tinjau

$$A^{(n-1)}(x) = \sum_{j=0}^{\infty} y_j^{(n-1)} x^j.$$

Untuk sembarang u, banyaknya solusi dari persamaan $a_1 + a_2 + \cdots + a_{n-1} = u$, sesuai hipotesis, adalah $y_u^{(n-1)}$. Pilih sembarang v, sedemikian sehingga $a_n = v$. Banyaknya solusi dari $a_n = v$ adalah koefisien x^v dari $A_n(x)$. Koefisien x^{u+v} dari $A^{(n-1)}(x)A_n(x)$, akibatnya adalah banyaknya solusi dari $a_1 + a_2 + \cdots + a_n = u + v$. Karena pemilihan u dan v tidak mengurangi keumuman (dipilih secara sembarang), untuk setiap r, banyaknya solusi dari $a_1 + a_2 + \cdots + a_n = r$ adalah koefisien x^r dari $A^{(n-1)}(x)A_n(x)$. Dengan demikian, terbukti bahwa $A^{(n-1)}(x)A_n(x)$ adalah fungsi pembangkit untuk $a_1 + a_2 + \cdots + a_n = r$ dan dapat dituliskan sebagai

$$A^{(n)}(x) = A^{(n-1)}(x)A_n(x) = A_1(x)A_2(x) \dots A_n(x).$$

Secara aljabar, dengan menyulihkan bentuk eksplisit dari $A_i(x)$, diperoleh bahwa solusi dari $a_1 + a_2 + \cdots + a_n = r'$ dengan $0 \le a_i \le p_i$ adalah sebagai berikut

$$[x^{r'}](A^{(n)}(x)) = [x^{r'}]\left(\prod_{i=1}^{i=n}\left(\sum_{j=0}^{j=p_i} x^j\right)\right)$$

Notasi $[x^k](f(x))$ adalah koefisien x^k dari $f(x)$. Tinjau bahwa deret pada ruas kanan adalah deret geometri, jumlahannya dapat dicari dengan formula deret geometri, yaitu

$$\sum_{j=0}^{p_i} x^j = 1 + x + x^2 + \cdots + x^{p_i} = \frac{(1 - x^{p_i+1})}{(1-x)}.$$

Dengan demikian, bentuk di atas dapat disederhanakan menjadi

$$[x^{r'}](A^{(n)}(x)) = [x^{r'}]\left(\prod_{i=1}^{i=n} \frac{(1 - x^{p_i+1})}{(1-x)}\right).$$

Misalkan $P(x) = \prod_{i=1}^{n}(1 - x^{p_i+1})$, diperoleh bahwa

$$[x^{r'}](A^{(n)}(x)) = [x^{r'}]\left(\frac{P(x)}{(1-x)^n}\right).$$

Kita telah mengetahui bahwa $\frac{1}{(1-x)^n}$ merupakan fungsi pembangkit dari sebuah suku banyak bersuku tak hingga, sulihkan bentuknya dalam persamaan di atas, akan diperoleh

$$[x^{r'}](A^{(n)}(x)) = [x^{r'}]\left(P(x) \cdot \sum_{k\ge 0}\binom{k+n-1}{k} x^k\right)$$

$$[x^{r'}](A^{(n)}(x)) = \sum_{k\ge 0}\left(\binom{k+n-1}{k} \cdot [x^{r'-k}](P(x))\right)$$

Koefisien $P(x)$ memerlukan komputasi perkalian n buah suku banyak, dalam hal ini, algoritma FFT dapat diterapkan.

Misalkan $T_{mult}(n)$ adalah kompleksitas waktu untuk menghitung $P(x)$, perhatikan bahwa untuk menghitung $P^{(n-1)}(x) \cdot (1 - x^{p_n+1})$ menggunakan FFT memerlukan $\Omega\left(\left(\left|P^{(n-1)}\right| + p_n + 1\right)\log\left(\left|P^{(n-1)}\right| + p_n + 1\right)\right)$, dengan $P^{(n-1)}(x) = \prod_{i=1}^{n-1}(1 - x^{p_i+1})$ dan $\left|P^{(n-1)}\right|$ adalah banyaknya suku pada $P^{(n-1)}$. Jika diasumsikan hasil perkalian setelah FFT sudah dikondensasi (perkalian suku banyak berderajat x dengan suku banyak berderajat y menghasilkan suku banyak berderajat $(x+y)$), diperoleh bahwa kompleksitas waktu perkalian $P^{(n-1)}(x) \cdot (1 - x^{p_n+1})$ adalah $\Omega((S+n)\log(S+n))$ dengan $S = \Sigma p_i$. Perhatikan bahwa asimtotik atas untuk algoritma ini bisa sangat besar jika vektor koefisien tidak dikondensasi. Kompleksitas waktu total dapat dicari dari persamaan rekursi

$$T_{mult}(n) = T_{mult}(n-1) + T_{FFT}(n)$$
$$T_{mult}(n) = T_{mult}(n-1) + \Omega((S+n)\log(S+n))$$
$$= \Omega(nS\log S).$$

Perhatikan bahwa, jika vektor koefisien selalu dikondensasi, misalkan $2^{q-1} < S \le 2^q$, kompleksitas asimtotik atas untuk algoritma ini adalah $T_{mult}(n) = O(n(2^q + n)\log(2^q + n))$.

Dengan demikian, kompleksitas total pencarian persoalan *stars and bars* yang dibatasi adalah $T(n) = T(C) + T(Op) + T_{mult}(n)$, dengan C adalah operasi pencarian kombinasi dan Op adalah operasi perkalian kombinasi dengan $P(x)$. Mudah dilihat bahwa Op dilakukan sebanyak n kali, sehingga hanya membutuhkan waktu linear, sedangkan C dapat diperoleh dengan algoritma efisien yaitu dengan *precompute* semua kemungkinan faktorial yang akan dihitung. Mudah dilihat bahwa $O(C) + O(n(2^q + n)\log(2^q + n)) = O(n(2^q + n)\log(2^q + n))$. Dengan demikian $T(n) = O(n(2^q + n)\log(2^q + n))$.

Hal menarik dari solusi pencarian ini adalah, untuk r' yang berubah-ubah, karena $P(x)$ tidak berubah, pencarian solusi bisa dilakukan hanya dalam $O(\max(C, n))$. Kita juga dapat melakukan optimasi jika p_i memiliki keseragaman, misalnya dalam kasus ekstrem ketika $p_1 = p_2 = \cdots = p_n = p$, bentuk fungsi pembangkit dari solusi akan menjadi

$$[x^{r'}](A^{(n)}(x)) = [x^{r'}]\left((1 - x^{p+1})^n \cdot \sum_{k\ge 0}\binom{k+n-1}{k} x^k\right)$$

$$[x^{r'}](A^{(n)}(x))$$
$$= [x^{r'}]\left(\left(\sum_{m=0}^{n}\binom{n}{m}(-x^{p+1})^m\right)\left(\sum_{k\ge 0}\binom{k+n-1}{k} x^k\right)\right)$$

$$[x^{r'}](A^{(n)}(x)) = \sum_{k=0}^{\min\left(\frac{r'}{p+1}, n\right)} (-1)^k \binom{n}{k}\binom{r' - kp - k + n - 1}{r' - kp - k}$$

dengan kompleksitas waktu $O\left(\max\left(C, min\left(\frac{r}{p+1}, n\right)\right)\right)$.

Lebih lanjut, untuk nilai-nilai p_i yang tidak menyebar, pencarian perkalian suku banyak berulang (perpangkatan suku banyak) bisa dilakukan dengan menggunakan algoritma *fast exponentiation*.

Selain itu, formula bentuk umum *stars and bars* bisa juga diperoleh dari hasil fungsi pembangkit ini dengan menyulihkan $p_i \to \infty$ untuk setiap $i \in [1, n]$. Akan diperoleh

$$[x^{r'}]\left(A^{(n)}(x)\right) = \left[x^{r'}\right]\left(\frac{1}{(1-x)^n}\right)$$
$$= [x^{r'}]\left(\sum_{k\ge 0}\binom{k+n-1}{k} x^k\right).$$

Dengan demikian, banyaknya solusi adalah $\binom{r'+n-1}{r'}$, konsisten dengan yang diperoleh menggunakan formula *stars and bars.*

IV. IMPLEMENTASI ALGORITMA

Hasil implementasi algoritma pencarian DFT dengan bahasa C++ adalah sebagai berikut.

```cpp
const double pi = acos(-1);

void dft(vector<complex<long double>> &a){
    // I.S a adalah vektor koefisien suku banyak
    // F.S a adalah DFT dari vektor koefisien masukan
    int n = a.size();
    if(n==1){
        return;
    }
    vector<complex<long double>> a1(n/2), a2(n/2); // membentuk vektor koefisien a1 dan a2
    for(int i=0; i<n/2; i++){
        a1[i]=a[2*i];
        a2[i]=a[2*i+1];
    }
    dft(a1); dft(a2);
    complex<long double> w = 1; // pengali saat ini
    complex<long double> w_n(cos(2*pi/n), sin(2*pi/n)); // akar persatuan ke n
    for(int i=0; i<n/2; i++){
        a[i] = a1[i]+w*a2[i];
        a[i+n/2] = a1[i]-w*a2[i];
        w = w*w_n;
    }
}
```

Gambar 5 Implementasi Pencarian DFT

Adapun untuk pencarian invers, hasil implementasinya adalah sebagai berikut.

```cpp
void invdft(vector<complex<long double>> &a){
    // I.S a adalah vektor koefisien suku banyak
    // F.S a adalah DFT^-1 dari vektor koefisien masukan
    int n = a.size();
    if(n==1){
        return;
    }
    vector<complex<long double>> a1(n/2), a2(n/2); // membentuk vektor koefisien a1 dan a2
    for(int i=0; i<n/2; i++){
        a1[i]=a[2*i];
        a2[i]=a[2*i+1];
    }
    invdft(a1); invdft(a2);
    complex<long double> w = 1; // pengali saat ini
    complex<long double> w_n(cos(-2*pi/n), sin(-2*pi/n)); // akar persatuan ke n
    for(int i=0; i<n/2; i++){
        a[i] = (a1[i]+w*a2[i])/2.0L;
        a[i+n/2] = (a1[i]-w*a2[i])/2.0L;
        w = w*w_n;
    }
}
```

Gambar 6 Implementasi Pencarian Invers DFT

Mengaplikasikan kedua prosedur di atas, perkalian dua buah suku banyak dapat langsung diimplementasi dengan formula $DFT^{-1}(DFT(A) \cdot DFT(B))$, implementasinya dalam bahasa C++ diperoleh sebagai berikut.

```cpp
vector<ll> multiply(vector<ll> a, vector<ll> b) {
    vector<complex<long double>> dfta(a.begin(), a.end()), dftb(b.begin(), b.end());
    int n = 1;
    while (n<a.size()+b.size()){
        n <<= 1;
    }
    dfta.resize(n); dftb.resize(n);
    dft(dfta); dft(dftb);
    for (int i = 0; i < n; i++){
        dfta[i] *= dftb[i];
    }
    invdft(dfta);
    vector<ll> result(n);
    for (int i = 0; i < n; i++){
        result[i] = llround(dfta[i].real());
    }
    int i=n-1;
    while(result[i]==0){
        i--;
    }
    result.resize(i+1);
    return result;
}
```

Gambar 7 Algoritma Perkalian Dua Buah Suku Banyak

Perhatikan bahwa vektor *result* pada perolehannya dilakukan operasi *rounding* menjadi *long long int*, sehingga sangat memungkinkan untuk terjadi ketidaktelitian.

Berikut ini adalah program utama pencarian solusi *stars and bars* terbatas interval dengan algoritma FFT.

```cpp
vector<ll> createNum(int r){
    vector<ll> res(r+2);
    res[0] = 1LL;
    res[r+1] = -1LL;
    return res;
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    // IO
    int n, r;
    printf("Masukkan nilai n: ");
    scanf("%d", &n);
    printf("Masukkan nilai r: ");
    scanf("%d", &r);
    int s[n], t[n], p[n];
    vector<ll> P[n], res;
    for(int i=0; i<n; i++){
        printf("Masukkan nilai s_%d dan t_%d (dipisahkan spasi): ", i+1, i+1);
        scanf("%d %d", &s[i], &t[i]);
        r-=s[i];
        p[i]=t[i]-s[i];
        if(p[i]<0){
            printf("tidak memiliki solusi\n");
            exit(-1);
        }
        P[i] = createNum(p[i]);
    }
    if(n<0){
        printf("tidak memiliki solusi\n");
        exit(-1);
    }
    chrono::steady_clock::time_point begin = chrono::steady_clock::now();
    // precompute faktorial
    long long fac[20];
    fac[0]=1;
    for(ll i=1; i<20; i++){
        fac[i] = (fac[i-1]*i);
    }
    // Proses
    res = P[0];
    for(int i=1; i<n; i++){
        res = multiply(res, P[i]);
    }
    long long ans = 0;
    for(int i=0; i<min((int)res.size(), r+1); i++){
        ans = ans + res[i]*1LL*fac[n-1+(r-i)]/(1LL*fac[n-1]*1LL*fac[r-i]);
    }
    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Time difference = " << chrono::duration_cast<chrono::microseconds>(end - begin).count() << "[µs]" << endl;
    cout << "Time difference = " << chrono::duration_cast<chrono::nanoseconds> (end - begin).count() << "[ns]" << endl;
    cout<< "Solusinya adalah: " << ans << "\n";
    return 0;
}
```

Gambar 8 Program Utama Algoritma Pencarian Solusi Persoalan *Stars And Bars* yang dibatasi interval

Faktorial pada bahasa C++ hanya mampu menampung hingga $19! = 1.21x10^{17}$, mengingat batasan *long long int* pada bahasa C++ adalah $-9x10^{18}$ hingga $9x10^{18}$. Akibatnya nilai $r + n - 1$ perlu lebih kecil dari 19. Untuk kasus-kasus besar, implementasi bisa dilakukan pada bahasa lain yang bisa menampung bilangan bulat besar seperti java atau python.

Algoritma *time difference* digunakan sebagai pengukur waktu eksekusi. [13]

Untuk membandingkan kompleksitas waktu, algoritma yang mengimplementasikan *brute force* dari penyelesaian persoalan ini juga dibuat. Berikut hasil implementasinya.

```cpp
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int n;
    printf("Masukkan nilai n: ");
    scanf("%d", &n);
    chrono::steady_clock::time_point begin = chrono::steady_clock::now();
    int ans = 0;
    for(int i=0; i<=20; i++){
        for(int j=0; j<=20; j++){
            for(int k=0; k<=20; k++){
                for(int l=0; l<=20; l++){
                    for(int p=0; p<=20; p++){
                        for(int q=0; q<=20; q++){
                            if(i+k+j+l+p+q==n){
                                ans++;
                            }
                        }
                    }
                }
            }
        }
    }
    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Time difference = " << chrono::duration_cast<chrono::microseconds>(end - begin).count() << "[µs]" << endl;
    cout << "Time difference = " << chrono::duration_cast<chrono::nanoseconds> (end - begin).count() << "[ns]" << endl;
    cout << "Banyaknya solusi menggunakan metode naif: "<< ans << "\n";

    return 0;
}
```

Gambar 9 Algoritma *Brute Force*

Batas-batas dan jumlah iterasi dimodifikasi secara manual disesuaikan dengan input.

Misalnya akan diselesaikan persamaan $x_1 + x_2 + \cdots + x_7 = 12$, dengan batasan berikut $0 \leq x_1 \leq 12$, $1 \leq x_2 \leq 12$, $1 \leq x_3 \leq 11$, $2 \leq x_4 \leq 11$, $0 \leq x_5 \leq 10$, $0 \leq x_6 \leq 9$, $1 \leq x_7 \leq 9$.

Menggunakan algoritma FFT, diperoleh hasil sebagai berikut.

```
Masukkan nilai n: 7
Masukkan nilai r: 12
Masukkan nilai s_1 dan t_1 (dipisahkan spasi): 0 12
Masukkan nilai s_2 dan t_2 (dipisahkan spasi): 1 12
Masukkan nilai s_3 dan t_3 (dipisahkan spasi): 1 11
Masukkan nilai s_4 dan t_4 (dipisahkan spasi): 2 11
Masukkan nilai s_5 dan t_5 (dipisahkan spasi): 0 10
Masukkan nilai s_6 dan t_6 (dipisahkan spasi): 0 9
Masukkan nilai s_7 dan t_7 (dipisahkan spasi): 1 9
Time difference = 4200[┬┤s]
Time difference = 4200300[ns]
Solusinya adalah: 1716
```

Gambar 10 Hasil Eksekusi Studi Kasus Interval Berbeda Menggunakan Algoritma FFT

Adapun hasil eksekusi menggunakan algoritma *brute force*, sebagai berikut.

```
Masukkan nilai n: 12
Time difference = 35418[┬┤s]
Time difference = 35418900[ns]
Banyaknya solusi menggunakan metode naif: 1716
```

Gambar 11 Hasil Eksekusi Studi Kasus Interval Berbeda Menggunakan Algoritma *Brute Force*

Perhatikan bahwa algoritma FFT delapan kali lebih cepat dibandingkan *brute force*. Hasil yang diperoleh pun memiliki akurasi yang cukup baik.

Selain itu, untuk kasus khusus ketika $p_i = p_j \; \forall i, j \in [1, n]$, penulis membuat optimasi sesuai dengan formula hasil penyesuaian fungsi pembangkit, berikut hasil implementasinya.

```
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int n, r;
    printf("Masukkan nilai n: ");
    scanf("%d", &n);
    printf("Masukkan nilai r: ");
    scanf("%d", &r);
    int p;
    printf("Masukkan nilai p: ");
    scanf("%d", &p);
    chrono::steady_clock::time_point begin = chrono::steady_clock::now();
    // precompute faktorial
    long long fac[20];
    fac[0]=1;
    for(ll i=1; i<20; i++){
        fac[i] = (fac[i-1]*i);
    }
    long long ans = 0;
    for(int k=0; k<=min(r/(p+1), n); k++){
        int c = k%2?-1:1;
        ll m = r-k*p-k;
        ans += c*(fac[n]/(fac[n-k]*fac[k]))*(fac[m+n-1]/(fac[n-1]*fac[m]));
    }
    chrono::steady_clock::time_point end = chrono::steady_clock::now();
    cout << "Time difference = " << chrono::duration_cast<chrono::microseconds>(end - begin).count() << "[µs]" << endl;
    cout << "Time difference = " << chrono::duration_cast<chrono::nanoseconds> (end - begin).count() << "[ns]" << endl;
    cout<< "Solusinya adalah: " << ans << "\n";
    return 0;
}
```

Gambar 12 Implementasi Optimasi Pada Kasus Interval Sama

Misalnya akan diuji kasus untuk $x_1 + x_2 + \cdots + x_{13} = 7$ dengan $0 \leq x_i \leq 7$ untuk setiap $i \in [1,13]$. Hasil eksekusi dengan metode *brute force* diperoleh sebagai berikut.

```
Masukkan nilai n: 7
Time difference = 1603116808[┬┤s]
Time difference = 1603116808400[ns]
Banyaknya solusi menggunakan metode naif: 50388
```

Gambar 13 Hasil Eksekusi Menggunakan Algoritma *Brute Force*

Kasus uji juga dijalankan pada algoritma yang menggunakan FFT, hasil yang diperoleh adalah sebagai berikut.

```
Masukkan nilai n: 13
Masukkan nilai r: 7
Masukkan nilai s_1 dan t_1 (dipisahkan spasi): 0 7
Masukkan nilai s_2 dan t_2 (dipisahkan spasi): 0 7
Masukkan nilai s_3 dan t_3 (dipisahkan spasi): 0 7
Masukkan nilai s_4 dan t_4 (dipisahkan spasi): 0 7
Masukkan nilai s_5 dan t_5 (dipisahkan spasi): 0 7
Masukkan nilai s_6 dan t_6 (dipisahkan spasi): 0 7
Masukkan nilai s_7 dan t_7 (dipisahkan spasi): 0 7
Masukkan nilai s_8 dan t_8 (dipisahkan spasi): 0 7
Masukkan nilai s_9 dan t_9 (dipisahkan spasi): 0 7
Masukkan nilai s_10 dan t_10 (dipisahkan spasi): 0 7
Masukkan nilai s_11 dan t_11 (dipisahkan spasi): 0 7
Masukkan nilai s_12 dan t_12 (dipisahkan spasi): 0 7
Masukkan nilai s_13 dan t_13 (dipisahkan spasi): 0 7
Time difference = 8462[┬┤s]
Time difference = 8462300[ns]
Solusinya adalah: 50388
```

Gambar 14 Hasil Eksekusi Kasus Interval Sama Menggunakan Algoritma FFT

Adapun hasil penggunaan algoritma optimasi, diperoleh sebagai berikut.

```
Masukkan nilai n: 13
Masukkan nilai r: 7
Masukkan nilai p: 7
Time difference = 0[┬┤s]
Time difference = 300[ns]
Solusinya adalah: 50388
```

Gambar 15 Hasil Eksekusi Kasus Interval Sama Menggunakan Algoritma Optimasi

Terlihat bahwa algoritma *brute force* membutuhkan waktu sebanyak 26 menit untuk mendapatkan solusi. Algoritma FFT pada kasus ini masih cukup efisien, waktu eksekusi masih < 1 detik. Menariknya, dengan algoritma optimasi, waktu yang dibutuhkan hanya berorde ratusan nanodetik, ribuan kali lebih cepat dibanding dengan algoritma FFT.

Kasus-kasus uji di atas terbatas pada *range limit* yang disediakan oleh tipe data C++, penggunaan bahasa lain dimungkinkan untuk mencari solusi dengan n dan r yang cukup besar. Masalah batasan tipe data ini sebenarnya bisa diakali dengan penggunaan modulo, tetapi hal ini membutuhkan banyak modifikasi pada algoritma FFT.

V. KESIMPULAN

Fungsi pembangkit memiliki aplikasi yang sangat luas di bidang komputasi. Salah satu contoh aplikasi fungsi pembangkit adalah pada penyelesaian persoalan *stars and bars* yang digeneralisasi. Namun, di luar itu, fungsi pembangkit juga bisa diterapkan pada lebih banyak kasus-kasus lain yang lebih kompleks, misalnya pada kasus *stars and bars* dengan masing-masing peubah memiliki koefisien tertentu atau ketika yang dicari adalah banyaknya solusi pertidaksamaan. Lebih lanjut, penggunaan fungsi pembangkit memungkinkan untuk mendapatkan solusi yang lebih umum, sehingga proses komputasi bisa dilakukan lebih dinamis dan adaptif.

VI. UCAPAN TERIMA KASIH

Penulis mengucapkan rasa syukur kepada Tuhan Yang Maha Esa karena berkat rahmat-Nya, penulis bisa menyelesaikan makalah ini dengan baik. Tidak lupa, penulis mengucapkan terima kasih kepada orang tua, kerabat, dan teman-teman atas dukungan dan doanya. Selain itu, penulis mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T., Ibu Harlili, M.Sc., dan Ibu Dr. Nur Ulfa Maulidevi S.T., M.Sc., selaku dosen pengampu mata kuliah matematika diskrit tahun 2021/2022 atas bimbingan, ilmu, dan motivasinya selama satu semester ini. Akhir kata, penulis menyadari bahwa masih terdapat banyak kekurangan dalam pembuatan makalah ini. Karena itu, kritik dan saran yang membangun sangat terbuka dan sangat diapresiasi.

REFERENCES

[1] Chuan-Chong, C., Khee-Meng, K., "Principles and Techniques in Combinatorics", Singapore: World Scientific, 1992, ch. 1-4.

[2] Cormen, T. H., Leiserson, C. E., & Rivest, R. L., Stein, C, "Introduction to algorithms", 3rd ed. London: McGraw-Hill, 2009, ch. 1.

[3] https://brilliant.org/wiki/integer-equations-star-and-bars/ diakses pada 11 Desember 2021.

[4] https://brilliant.org/wiki/roots-of-unity/ diakses pada 9 Desember 2021.

[5] https://aofa.cs.princeton.edu/30gf/ diakses pada 11 Desember 2021.

[6] https://cp-algorithms.com/algebra/fft.html diakses pada 10 Desember 2021.

[7] https://cp-algorithms.com/combinatorics/inclusion-exclusion.html diakses pada 8 Desember 2021.

[8] https://ichi.pro/id/identitas-euler-dan-akar-persatuan-126277470404742 diakses pada 11 Desember 2021.

[9] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kombinatorial-2020-Bagian1.pdf diakses pada 8 Desember 2021.

[10] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kombinatorial-2020-Bagian2.pdf diakses pada 9 Desember 2021.

[11] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kompleksitas-Algoritma-2020-Bagian1.pdf diakses pada 10 Desember 2021.

[12] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kompleksitas-Algoritma-2020-Bagian2.pdf diakses pada 10 Desember 2021.

[13] https://www.pluralsight.com/blog/software-development/how-to-measure-execution-time-intervals-in-c-- diakses pada 12 Desember 2021.

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, 12 Desember 2021

Firizky Ardiansyah 13520095