TUGAS AKHIR - KI141502

DESAIN DAN ANALISIS ALGORITMA KOMPUTASI FORMULA $\sum_{i=1}^{n} a^i i^r$, STUDI KASUS : PERSOALAN SPOJ MOON SAFARI

ANTON KRISTANTO
NRP 5112100078

Dosen Pembimbing 1
Arya Yudhi Wijaya, S.Kom.,M.Kom.

Dosen Pembimbing 2
Rully Soelaiman, S.Kom.,M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2016

Halaman ini sengaja dikosongkan

TUGAS AKHIR – KI141502

DESAIN DAN ANALISIS ALGORITMA KOMPUTASI FORMULA $\sum_{i=1}^{n} a^i i^r$, STUDI KASUS : PERSOALAN SPOJ MOON SAFARI

ANTON KRISTANTO
NRP 5112100078

Dosen Pembimbing 1
Arya Yudhi Wijaya, S.Kom.,M.Kom.

Dosen Pembimbing 2
Rully Soelaiman, S.Kom.,M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2016

Halaman ini sengaja dikosongkan

UNDERGRADUATED THESIS – KI141502

DESIGN AND ANALYSIS OF ALGORITHM FOR COMPUTING THE FORMULA $\sum_{i=1}^{n} a^i i^r$, CASE STUDY SPOJ MOON SAFARI PROBLEM

ANTON KRISTANTO
NRP 5112100078

Dosen Pembimbing 1
Arya Yudhi Wijaya, S.Kom.,M.Kom.

Dosen Pembimbing 2
Rully Soelaiman, S.Kom.,M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2016

Halaman ini sengaja dikosongkan

DESAIN DAN ANALISIS ALGORITMA KOMPUTASI FORMULA $\sum_{i=1}^{n} a^i i^i$, STUDI KASUS : PERSOALAN SPOJ MOON SAFARI

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Dasar dan Terapan Komputasi
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:
Anton Kristanto
NRP: 5112100078

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Arya Yudhi Wijaya, S.Kom.,M.Kom. .....
NIP: 198409042010121002 (Pembimbing 1)

Rully Soelaiman, S.Kom.,M.Kom. .....
NIP: 197002131994021001 (Pembimbing 2)

SURABAYA
MEI 2016

Halaman ini sengaja dikosongkan

DESAIN DAN ANALISIS ALGORITMA KOMPUTASI FORMULA $\sum_{i=1}^{n} a^i i^r$, STUDI KASUS : PERSOALAN SPOJ MOON SAFARI

Nama	: Anton Kristanto
NRP	: 5112100078
Jurusan	: Teknik Informatika FTIF - ITS
Dosen Pembimbing I	: Arya Yudhi Wijaya, S.Kom.,M.Kom.
Dosen Pembimbing II	: Rully Soelaiman, S.Kom.,M.Kom.

Abstrak

Komputasi perhitungan rumus berikut ini jika diketahui nilai bilangan bulat n, a dan r :

$$f(n, a, r) = \sum_{i=1}^{n} a^i i^r$$

Perhitungan rumus diatas dapat dilakukan dengan loop sebanyak n kali. Jika n bernilai sangat besar maka menghitung rumus tersebut dengan loop tentunya tidak efisien.

Pada Tugas Akhir ini akan dirancang penyelesaian permasalahan di atas dengan melakukan transformasi rumus diatas menjadi rumus baru. Masalah diatas akan diselesaikan dengan berbagai macam algoritma antara lain : Eulerian Polynomials, Fast Multiplication Polynomials dan Interpolation Polynomials.

Hasil dari Tugas Akhir ini telah berhasil menyelesaikan permasalahan di atas dengan cukup efisien dengan kompleksitas waktu O(r log r) dimana r adalah nilai dari r pada rumus tersebut.

Kata Kunci: Rumus, Polynomial, Pangkat, Geometri, Interpolation, Multiplication, Eulerian, Fourier

Halaman ini sengaja dikosongkan

DESIGN AND ANALYSIS OF ALGORITHM FOR COMPUTING THE FORMULA $\sum_{i=1}^{n} a^i i^r$, CASE STUDY SPOJ MOON SAFARI PROBLEM

Name	: Anton Kristanto
NRP	: 5112100078
Major	: Informatics Department Faculty of IT - ITS
Supervisor I	: Arya Yudhi Wijaya, S.Kom.,M.Kom.
Supervisor II	: Rully Soelaiman, S.Kom.,M.Kom.

Abstract

Computing the following calculation formula if known integer n, a and r:

$$f(n, a, r) = \sum_{i=1}^{n} a^i i^r$$

Calculation formula above can be done with the loop n times. If n is very large, the formula calculates the loop is certainly not efficient.

In this final project will be designed completion of the above problems with the above formula to transform into a new formula. The above problems will be solved with a variety of algorithms, among others: Eulerian polynomials, Fast Multiplication Polynomials and Interpolation polynomials.

The results of this final project has successfully completed the above problems quite efficiently with time complexity O (r log r) where r is the value of r in the formula.

Kata Kunci: Rumus, Polynomial, Pangkat, Geometri, Interpolation, Multiplication, Eulerian, Fourier

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas pimpinan, penyertaan, dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul :

DESAIN DAN ANALISIS ALGORITMA KOMPUTASI FORMULA $\sum_{i=1}^{n} a^i i^r$, STUDI KASUS : PERSOALAN SPOJ MOON SAFARI

Penelitian Tugas Akhir ini dilakukan untuk memenuhi salah satu syarat meraih gelar Sarjana di Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya.

Dengan selesainya Tugas Akhir ini diharapkan apa yang telah dikerjakan penulis dapat memberikan manfaat bagi perkembangan ilmu pengetahuan terutama di bidang informasi serta bagi diri penulis sendiri selaku peneliti.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama penulis mengerjakan Tugas Akhir maupun selama menempuh masa studi antara lain:

- Ayah, Ibu dan saudara penulis yang selalu memberikan perhatian, dorongan dan kasih sayang yang menjadi semangat utama bagi diri penulis sendiri selama menempuh masa kuliah maupun pengerjaan Tugas Akhir ini.
- Bapak Rully Soelaiman, S.Kom, M.Kom. selaku Dosen Pembimbing yang telah memberikan ilmu, nasehat, motivasi, arahan, pandangan, dan bimbingan kepada penulis baik selama masa kuliah maupun selama pengerjaan Tugas Akhir ini.
- Bapak Arya Yudhi Wijaya, S.Kom., M.Kom. selaku dosen pembimbing yang telah memberikan ilmu, dan masukan kepada penulis.

- Tim “Young Phoenix” Ryan Nathan dan Ignatius Abraham yang merupakan teman seperjuangan dalam lomba pemrograman.
- Seluruh pihak yang tidak bisa penulis sebutkan satu-persatu yang telah memberikan dukungan selama penulis menyelesaikan Tugas Akhir.

Penulis mohon maaf apabila masih ada kekurangan pada Tugas Akhir ini. Penulis juga mengharapkan kritik dan saran yang membangun untuk pembelajaran dan perbaikan di kemudian hari. Semoga melalui Tugas Akhir ini penulis dapat memberikan kontribusi dan manfaat yang sebaik-baiknya.

Surabaya, Desember 2016

Anton Kristanto

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR KODE SUMBER

Halaman ini sengaja dikosongkan

BAB 1
PENDAHULUAN

Pada bab ini akan dijelaskan latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi dan sistematika penulisan Tugas Akhir.

1.1 Latar Belakang

Permasalahan yang akan dijelaskan berasal dari salah satu permasalahan di website *Sphere Online Judge*(SPOJ) yaitu Moon Safari. Permasalahan tersebut ialah mencari hasil dari rumus berikut ini jika diketahui nilai bilangan bulat n, a, dan r:

$$f(n, a, r) = \sum_{i=1}^{n} a^i i^r$$

Karena hasil perhitungan rumus diatas bisa sangat besar maka output yang dikeluarkan ialah hasil sisa bagi dari $f(n, a, r)$ dengan bilangan prima 1000000007. Terdapat 3 varian masalah Moon Safari yaitu :

- Moon Safari (easy)
  http://www.spoj.com/problems/MOON/
- Moon Safari (medium)
  http://www.spoj.com/problems/MOON1/
- Moon Safari (hard)
  http://www.spoj.com/problems/MOON2/

Ketiga varian masalah diatas hanya memiliki perbedaaan pada batasan nilai bilangan bulat n, a, dan r. Solusi naif yang dapat digunakan untuk menyelesaikan permasalahan diatas ialah *looping* dan algoritma *big modulo.* Solusi naif tersebut memiliki kompleksitas $O(n \log r)$. Dengan kompleksitas tersebut maka solusi ini hanya mampu menyelesaikan masalah Moon Safari

(easy) dengan batasan nilai n yang kecil($1 < n < 10^6$). Sedangkan untuk menyelesaikan masalah Moon Safari (Medium) dan Moon Safari (Hard) akan membutuhkan waktu runtime yang lama dengan batasan nilai n yang besar($1 < n < 10^9$). Solusi untuk Moon Safari (Medium) dan Moon Safari (Hard) akan dijelaskan lebih detail pada Bab 2.

1.2 Rumusan Masalah

Permasalahan yang akan diselesaikan pada tugas akhir ini sebagai berikut :

1. Bagaimana menganalisis dan mendesain algoritma yang efisien dalam menyelesaikan perhitungan $\sum_{i=1}^{n} a^i i^r$?
2. Bagaimana mengimplementasikan algoritma yang sudah didesain untuk menyelesaikan perhitungan $\sum_{i=1}^{n} a^i i^r$?
3. Bagaimana menguji implementasi algoritma yang sudah dirancang untuk mengetahui kinerja dari implementasi yang telah dibuat?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir ini memiliki batasan masalah sebagai berikut :

1. Menghitung hasil sisa bagi persamaan $\sum_{i=1}^{n} a^i i^r$ dengan 1000000007
2. Nilai r merupakan bilangan integer positif kurang dari 1000000007
3. Nilai n merupakan bilangan integer positif yang dapat ditampung dalam tipe data integer 32 bit
4. Nilai a merupakan bilangan integer positif yang dapat ditampung dalam tipe data integer 32 bit
5. Diimplementasikan dengan bahasa pemrograman C++

1.4 Tujuan

Tujuan dari tugas akhir ini antara lain :

1. Menganalisis dan mendesain algoritma yang efisien untuk menyelesaikan masalah perhitungan sisa bagi persamaan $\sum_{i=1}^{n} a^i i^r$
2. Mengimplementasikan algoritma yang sudah didesain untuk menyelesaikan masalah perhitungan sisa bagi persamaan $\sum_{i=1}^{n} a^i i^r$ secara efisien
3. Menguji implementasi dari algoritma yang telah didesain untuk mengetahui kinerja algoritma yang telah dibuat

1.5 Metodologi

Metodologi yang digunakan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir
   Pada tahap ini dilakukan penyusunan proposal tugas akhir yang berisi permasalahan dan gagasan solusi yang akan diteliti pada permasalahan komputasi perhitungan sisa bagi persamaan $\sum_{i=1}^{n} a^i i^r$ dengan 1000000007
2. Studi literatur
   Pada tahap ini dilakukan pencarian informasi dan studi literatur mengenai pengetahuan atau metode yang dapat digunakan dalam penyelesaian masalah. Informasi didapatkan dari materi-materi yang berhubungan dengan algoritma dan struktur data yang digunakan untuk penyelesaian permasalahan ini, materi-materi tersebut didapatkan dari buku maupun internet.
3. Desain
   Pada tahap ini dilakukan desain rancangan algoritma dan struktur data yang digunakan dalam solusi untuk pemecahan masalah komputasi perhitungan sisa bagi persamaan $\sum_{i=1}^{n} a^i i^r$ dengan 1000000007

4. Implementasi perangkat lunak
   Pada tahap ini dilakukan implementasi atau realiasi dari rancangan desain algoritma dan struktur data yang telah dibangun pada tahap desain ke dalam bentuk program.
5. Uji coba dan evaluasi
   Pada tahap ini dilakukan uji coba kebenaran dan uji coba kinerja dari implementasi yang telah dilakukan. Pengujian kebenaran dilakukan pada sistem penilaian daring SPOJ sesuai dengan masalah yang dikerjakan untuk diuji apakah luaran dari program telah sesuai dengan luaran yang seharusnya. Hasil dari uji coba kebenaran dan kinerja pada program digunakan sebagai bahan evaluasi kesalahan dan juga optimasi kinerja agar performa yang didapat lebih optimal.
6. Penyusunan buku Tugas Akhir
   Pada tahap ini dilakukan penyusunan buku Tugas Akhir yang berisi dokumentasi hasil pengerjaan Tugas Akhir.

1.6 Sistematika Penulisan

Berikut adalah sistematika penulisan buku Tugas Akhir ini:

1. BABI: PENDAHULUAN
   Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi dan sistematika penulisan Tugas Akhir.
2. BAB II: DASAR TEORI
   Bab ini berisi dasar teori mengenai permasalahan dan algoritma penyelesaian yang digunakan dalam Tugas Akhir
3. BAB III: DESAIN
   Bab ini berisi desain algoritma dan struktur data yang digunakan dalam penyelesaian permasalahan.
4. BAB IV: IMPLEMENTASI
   Bab ini berisi implementasi berdasarkan desain algortima dan struktur data yang telah dilakukan pada tahap desain.

5. BAB V: UJI COBA DAN EVALUASI
   Bab ini berisi uji coba dan evaluasi dari hasil implementasi yang telah dilakukan pada tahap implementasi.
6. BAB VI: KESIMPULAN
   Bab ini berisi kesimpulan yang didapat dari hasil uji coba yang telah dilakukan.

Halaman ini sengaja dikosongkan

BAB 2
DASAR TEORI

Pada bab ini akan dijelaskan mengenai dasar teori yang menjadi dasar pengerjaan Tugas Akhir ini.

2.1 Deskripsi Permasalahan

Permasalahan yang akan dibahas dalam tugas akhir ini adalah perhitungan rumus (2.1). Semua rumus yang dituliskan dalam buku ini akan dimodulo dengan bilangan prima 1000000007.

$$f(n, a, r) = \sum_{i=1}^{n} a^i i^r \tag{2.1}$$

Batasan inputan untuk nilai bilangan bulat n, a, dan r pada permasalahan Moon Safari (easy) sebagai berikut :

- $1 < n < 10^6$
- $1 < a < 10^9$
- $1 < r < 10^9$

Batasan inputan untuk nilai bilangan bulat n, a, dan r pada permasalahan Moon Safari (medium) sebagai berikut :

- $1 < n < 10^9$
- $1 < a < 10^9$
- $1 < r < 10^3$

Sedangkan batasan inputan untuk nilai bilangan bulat n, a,dan r permasalahan Moon Safari (hard) sebagai berikut :

- $1 < n < 10^9$
- $1 < a < 10^9$
- $1 < r < 10^6$

2.2 Strategi Penyelesaian Naif

Dari rumus (2.1) terlihat bahwa nilai bilangan bulat n relatif kecil hanya 10^6 jika dilihat tetapi nilai bilangan bulat r relatif besar yaitu 10^9. Dengan Nilai n yang relatif kecil maka strategi *brute force* dengan menggunakan loop dan algoritma BigModulo yang mempunyai kompleksitas $O(\log r)$ sudah relatif efisien. Dengan menggunakan solusi ini maka kompleksitas akhir program adalah $O(n \log r)$.

```
1  t = Input() // Total Testcase
2  for t = 1 to t
3      n = Input() // nilai n
4      a = Input() // nilai a
5      r = Input() // nilai r
6      hasil = 0
7      for i = n downto 1
8          hasil = (hasil+BigModulo(i,r))*a;
9      Print(hasil)
```

Gambar 2.1 Pseudocode Strategi Naif

Dapat dilihat pada Gambar 2.1 merupakan desain program untuk solusi naif. Untuk lebih jelas mengenai fungsi BigModulo pada pseudocode tersebut dapat dilihat pada subbab 3.3.

2.3 Strategi Penyelesaian dengan Eulerian Polynomial

Solusi naif dari permasalahan ini telah dijelaskan pada subbab 2.2 yang memiliki kompleksitas $O(n \log r)$. Kompleksitas tersebut masih relatif efisien jika batasan nilai n yang kecil($1 < n < 10^6$), dan batasan nilai r yang besar($1 < n < 10^9$). Batasan ini terdapat pada problem Moon Safari (easy). Pada permasalahan Moon Safari (Medium) dan Moon Safari (Hard) hanya terdapat perbedaan batasan nilai r dan n. Terlihat pada subbab 2.1 batasan nilai n dan r untuk dua permasalahan tersebut memiliki batasan nilai n yang

besar($1 < n < 10^9$), dan batasan nilai r yang kecil($1 < n < 10^6$). Dengan adanya perbedaan batasan ini maka solusi naif tersebut relatif efisien untuk permasalahan Moon Safari (Easy), tetapi masih relatif kurang efisien untuk permasalahan Moon Safari (Medium) dan Moon Safari (Hard).

Untuk mengoptimasi perhitungan pada permasalahan Moon Safari (Medium) dan Moon Safari (Hard), maka dibutuhkan perubahan rumus (2.1) akan diubah menjadi rumus (2.2). Terlihat pada kedua rumus diatas terjadi perubahan dari sigma terhadap n menjadi sigma terhadap r, dengan adanya perubahan ini maka rumus baru tersebut dapat dihitung dengan kompleksitas $O(r^2)$ dan $O(r \log r)$. Untuk penjelasan lebih detail mengenai proses perubahan dari rumus (2.1) menjadi rumus (2.2) dapat dilihat pada lampiran A. Penjelasan lebih detail mengenai perhitungan rumus (2.2) yang memiliki kompleksitas $O(r^2)$ dan $O(r \log r)$ akan dibahas pada subbab-subbab berikutnya.

$$\sum_{i=1}^{n} a^i i^r = \frac{1}{(1-a)^{r+1}} a \sum_{i=0}^{r-1} a^i \left\langle {r \atop i} \right\rangle - a^{n+1} \sum_{i=0}^{r} \frac{1}{(1-a)^{i+1}} \sum_{j=0}^{i} (-1)^j \binom{i}{j} (n-j)^r \quad (2.2)$$

Solusi kedua permasalahan diatas yang berkompleksitas $O(r^2)$ dan $O(r \log r)$ menjadi kurang efisien untuk menyelesaikan permasalahan Moon Safari (Easy) dengan batasan nilai r yang besar($1 < n < 10^9$). Pada subbab 2.3.1 akan dibahas mengenai *eulerian number* yang terdapat pada rumus (2.2). Pada subbab 2.3.2 akan dibahas mengenai *eulerian polynomial* yang juga terdapat pada rumus (2.2). Pada subbab 2.3.3 akan dibahas cara perhitungan keseluruhan rumus (2.2) dengan menggunakan *eulerian number* yang memiliki kompleksitas $O(r^2)$.

2.3.1 Eulerian Number

Bilangan bulat positif r, S_r merupakan himpunan dari permutasi dengan r bilangan berbeda. Untuk semua permutasi $w \in S_r$, dimana w dapat ditulis menjadi $w = w(1)w(2) \dots w(n)$. *Descent* dapat diartikan sebagai posisi ke $1 \leq i < r$ di dalam permutasi w sehingga $w(i) > w(i+1)$. Fungsi *descent* $Des(w)$ dapat ditulis :

$$Des(w) = \{i : w(i) > w(i+1)\}$$

Misal fungsi $des(w)$ merupakan banyaknya *descent* dalam w.

$$des(w) = |Des(w)| = |\{i : w(i) > w(i+1)\}|$$

Contoh jika $w = 3125647$ maka terdapat 2 *descent* dalam permutasi tersebut yaitu di posisi 1 (dimana 3>1) dan 5(dimana 5>1).

Eulerian Number merupakan banyaknya permutasi di dalam himpunan S_r yang mempunyai tepat k *descent*. Eulerian number dapat ditulis dengan notasi $\left\langle {r \atop k} \right\rangle$.

$$\left\langle {r \atop k} \right\rangle = |\{w \in S_r : des(w) = k\}|$$

Ditunjukkan pada Gambar 2.2 Permutasi di dalam himpunan S_4 himpunan dari S_4 yang terdiri dari 24 permutasi. Permutasi tersebut telah dikelompokan berdasarkan jumlah *descents*. Dari gambar tersebut terlihat bahwa banyaknya permutasi yang memiliki 2 *descents* ialah 11 permutasi. Maka dari itu nilai dari *eulerian number* $\left\langle {4 \atop 2} \right\rangle = 11$.

Nilai *Eulerian Number* $\left\langle {n \atop k} \right\rangle$ dimana $0 \leq k < r \leq 10$ dapat dilihat pada Gambar 2.3. Terlihat pada Gambar 2.3 nilai dari $\left\langle {4 \atop 2} \right\rangle$ adalah 11 hal ini juga terlihat pada Gambar 2.2.

$\mathrm{des}(w)=0$	$\mathrm{des}(w)=1$	$\mathrm{des}(w)=2$	$\mathrm{des}(w)=3$
1234	1243	3421	4321
	1324	4231	
	1342	2431	
	1423	3241	
	2134	4312	
	2314	4132	
	2341	1432	
	2413	3142	
	3124	4213	
	3412	2143	
	4123	3214	

Gambar 2.2 Permutasi di dalam himpunan S_4

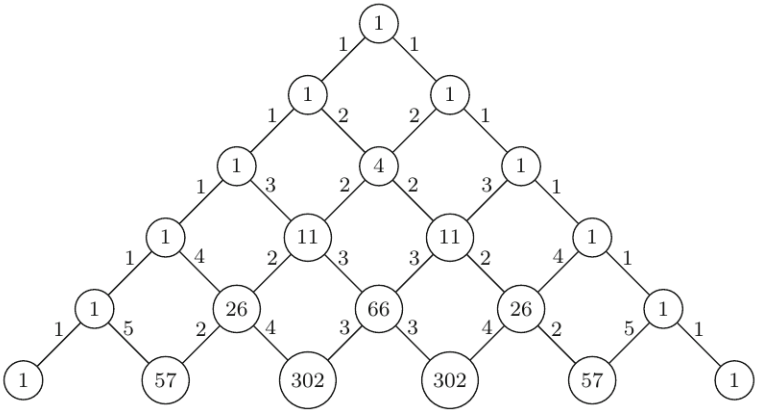
$r\backslash k$	0	1	2	3	4	5	6	7	8	9
1	1									
2	1	1								
3	1	4	1							
4	1	11	11	1						
5	1	26	66	26	1					
6	1	57	302	302	57	1				
7	1	120	1191	2416	1191	120	1			
8	1	247	4293	15619	15619	4293	247	1		
9	1	502	14608	88234	156190	88234	14608	502	1	
10	1	1013	47840	455192	1310354	1310354	455192	47840	1013	1

Gambar 2.3 Eulerian Number$\left\langle {r \atop k} \right\rangle$, $0 \leq k < r \leq 10$

Eulerian number dapat dihitung dengan *recurence relation* sebagai berikut :

$$\left\langle {r \atop 0} \right\rangle = \left\langle {r \atop r-1} \right\rangle = 1$$
$$\left\langle {r \atop k} \right\rangle = (r-k)\left\langle {\mathrm{r}-1 \atop \mathrm{k}-1} \right\rangle + (k+1)\left\langle {r-1 \atop k} \right\rangle \qquad (2.3)$$

Dapat digambarkan *recurrence relation* diatas dengan sebuah *triangle*. Hal ini dapat dilihat pada Gambar 2.4. *triangle* pada Gambar 2.4 ini disebut *Euler Triangle* atau *Euler's Triangle*



Gambar 2.4 Triangle dari Eulerian Number

Selain itu *Eulerian number* juga dapat dihitung dengan rumus sebagai berikut :

$$\left\langle {r \atop k} \right\rangle = \sum_{i=0}^{k} (-1)^i (k+1-i)^r \binom{r+1}{i} \qquad (2.4)$$

2.3.2 Eulerian Polynomials

Eulerian Polynomials $S_r(t)$ merupakan polinomial yang mempunyai derajat $r-1$ yang masing-masing koefisien polinomialnya merupakan nilai dari *Eulerian Number*. Berikut ini merupakan rumus dari *Eulerian Polynomials* :

$$S_r(t) = \sum_{w \in S_r} t^{des(w)} = \sum_{k=0}^{r-1} \left\langle {r \atop k} \right\rangle t^k \qquad (2.5)$$

Dengan melakukan subtitusi rumus (2.4) ke dalam rumus (2.5), maka akan dihasilkan persamaan berikut ini :

$$S_r(t) = \sum_{k=0}^{r-1} t^k \sum_{i=0}^{k} (-1)^i (k-i+1)^r \binom{r+1}{i} \qquad (2.6)$$

Untuk menghitung *euler polynomials* dengan menggunakan rumus diatas dapat digunakan 2 *nested loop* untuk masing masing sigma dan algoritma *big modulo* untuk menghitung $(k-i+1)^r$. Dengan cara perhitungan tersebut maka kompleksitas perhitungan rumus (2.6) adalah $O(r^2 \log r)$. Untuk lebih mengoptimasi perhitungan diatas dapat dilakukan penyimpanan sementara nilai dari x^r dimana x merupakan bilangan bulat lebih besar dari 0. Dengan adanya penyimpanan ini maka perhitungan $(k-i+1)^r$ dapat dihitung dengan kompleksitas $O(1)$ sehingga kompleksitas perhitungan rumus (2.6) menjadi $O(r^2)$. Untuk mengoptimasi perhitungan ini lebih lanjut maka dibutuhkan perubahan rumus sehingga kompleksitas program menjadi $O(r \log r)$.

Berikut ini adalah proses perubahan rumus (2.6) menjadi rumus (2.7). Pertama kali akan dilakukan penjabarkan kedua sigma dari rumus (2.6). Hasil penjabaran rumus tersebut dapat ditulis menjadi berikut ini :

$$\begin{aligned} S_r(t) = {} & t^0 \left((-1)^0 1^r \binom{r+1}{0} \right) \\ & + t^1 \left((-1)^0 2^r \binom{r+1}{0} + (-1)^1 1^r \binom{r+1}{1} \right) \\ & + \cdots \\ & + t^{r-1} \left((-1)^0 r^r \binom{r+1}{0} + (-1)^1 (r-1)^r \binom{r+1}{1} + \cdots \right. \\ & \qquad \left. + (-1)^{r-1} 1^r \binom{r+1}{r-1} \right) \end{aligned}$$

Dari penjabaran diatas dilakukan pengelompokan berdasarkan $(-1)^i\binom{r+1}{i}$ dimana variabel i mulai dari 0 sampai dengan $r-1$. Maka dihasilkan bentuk rumus sebagai berikut :

$$\begin{aligned} S_r(t) = {} & (-1)^0\binom{r+1}{0}(t^0 1^r + t^1 2^r + \cdots + t^{r-1} r^r) \\ & +(-1)^1\binom{r+1}{1}(t^1 1^r + \cdots + t^{r-1}(r-1)^r) \\ & +\cdots \\ & +(-1)^{r-1}\binom{r+1}{r-1}(t^{r-1} 1^r) \end{aligned}$$

Dari hasil pengelompokan di atas, dapat dilihat bahwa terdapat penjumlahan yang memiliki pola yang sama. Dengan adanya pola yang sama tersebut, maka dilakukan perubahan dari bentuk rumus di atas kembali menjadi bentuk sigma yang baru. Perubahan tersebut akan menghasilkan bentuk rumus berikut ini :

$$S_r(t) = \sum_{k=0}^{r-1}(-1)^k\binom{r+1}{k}\sum_{i=1}^{r-k} t^{k+i-1} i^r$$

Untuk menyederhanakan rumus maka dilakukan subtitusi variabel k dengan $r-1-k$. Hal ini dilakukan agar sigma kedua menjadi sigma dari i mulai dari 1 sampai $k+1$. Dengan perubahan sigma tersebut maka rumus ini dapat menghilangkan perhitungan yang *overlapping* dengan perhitungan lain sehingga menjadi lebih cepat dalam proses perhitungan. Berikut ini bentuk rumus yang dihasilkan :

$$S_r(t) = \sum_{k=0}^{r-1}(-1)^{r-1-k}\binom{r+1}{r-1-k}\sum_{i=1}^{r-(r-1-k)} t^{(r-1-k)+i-1} i^r$$

Dari rumus diatas dilakukan penyederhanaan variabel r sehingga berubah menjadi persamaan berikut ini:

$$S_r(t) = \sum_{k=0}^{r-1} (-1)^{r-1-k} \binom{r+1}{k+2} \sum_{i=1}^{k+1} t^{(r-1-k)+i-1} i^r$$

Sigma pertama diubah menjadi sigma k mulai dari 1 sampai r. Berikut ini rumus yang dihasilkan :

$$S_r(t) = \sum_{k=1}^{r} (-1)^{r-k} \binom{r+1}{k+1} \sum_{i=1}^{k} t^{(r-k)+i-1} i^r$$

Karena t^r pada sigma kedua dan $(-1)^r$ pada sigma pertama merupakan konstanta, maka kedua konstanta tersebut dapat dikeluarkan dari sigma. Maka dihasilkan rumus berikut ini:

$$S_r(t) = (-t)^r \sum_{k=1}^{r} (-1)^k \binom{r+1}{k+1} \sum_{i=1}^{k} t^{i-1-k} i^r \qquad (2.7)$$

Dari persamaan rumus (2.7) dapat dimisalkan sebagai berikut :

$$f(k) = \sum_{i=1}^{k} t^{i-1-k} i^r$$

Maka rumus (2.7) dapat ditulis menjadi berikut ini :

$$S_r(t) = (-t)^r \sum_{k=1}^{r} (-1)^k \binom{r+1}{k+1} f(k) \qquad (2.8)$$

Untuk menghitung $S_r(t)$ maka membutuhkan nilai dari $f(1)$ sampai dengan $f(r)$. Jika nilai dari $f(1)$ sampai dengan $f(r)$

sudah diketahui makan perhitungan $S_r(t)$ hanya membutuhkan kompleksitas $O(r)$. Masalah ada pada proses perhitungan nilai dari $f(1)$ sampai dengan $f(n)$. Untuk perhitungan setiap $f(k)$ dibutuhkan kompleksitas program $O(k \log r)$, maka secara naif perhitungan nilai dari $f(1)$ sampai dengan $f(n)$ membutuhkan kompleksitas program $O(r^2 \log r)$. Untuk mengoptimasi perhitungan nilai dari $f(1)$ sampai dengan $f(r)$ maka dibuat fungsi recurence untuk $f(k)$ sebagai berikut :

$$f(0) = 0$$
$$f(k) = \frac{f(k-1) + k^r}{t} \tag{2.9}$$

Dengan adanya *recurrence relation* tersebut maka untuk menghitung nilai dari $f(1)$ sampai dengan $f(r)$ membutuhkan kompleksitas $O(r \log r)$, hal ini dapat dilakukan dengan satu loop dan algoritma *big modulo* dalam setiap loop. Maka komplesitas akhir untuk menghitung $S_r(t)$ ialah $O(r \log r)$

2.3.3 Penjelasan Strategi Penyelesaian

Terlihat pada rumus (2.2) dapat dipecah menjadi 2 bagian yang terdiri dari sigma bagian kiri dan sigma bagian kanan. Masing-masing bagian dalam proses perhitungan membutuhkan kompleksitas $O(r^2)$. Rumus (2.10) dan (2.11) merupakan sigma bagian kiri dan sigma bagian kanan pada rumus (2.2)

$$\frac{1}{(1-a)^{r+1}} a \sum_{i=0}^{r-1} a^i \left\langle {r \atop i} \right\rangle \tag{2.10}$$

$$-a^{n+1} \sum_{i=0}^{r} \frac{1}{(1-a)^{i+1}} \sum_{j=0}^{i} (-1)^j \binom{i}{j} (n-j)^r \tag{2.11}$$

Pertama akan dilakukan perhitungan sigma bagian kiri pada rumus (2.10). Pada rumus (2.10) terdapat notasi $\left\langle {r \atop i} \right\rangle$, notasi ini merupakan notasi *eulerian number*. Penjelasan lebih detail mengenai *eulerian number* dapat dilihat pada subbab 2.2.1. Untuk menghitung rumus (2.10) dibutuhkan nilai dari *eulerian number* $\left\langle {r \atop 0} \right\rangle$, $\left\langle {r \atop 1} \right\rangle$, $\left\langle {r \atop 2} \right\rangle$, …, $\left\langle {r \atop r-1} \right\rangle$. Eulerian number $\left\langle {r \atop 0} \right\rangle$, $\left\langle {r \atop 1} \right\rangle$, $\left\langle {r \atop 2} \right\rangle$, …, $\left\langle {r \atop r-1} \right\rangle$ dapat dihitung dengan *recurrence relation* pada rumus (2.3). Dengan mengunakan rumus (2.3) *eulerian number* dapat dihitung dengan menggunakan *array* dan *nested loop* sehingga kompleksitas perhitungan nilai *eulerian number* adalah $O(r^2)$. Setelah nilai dari *eulerian number* didapat, maka dapat dilakukan perhitungan rumus (2.10) dengan menggunakan *loop* yang memiliki kompleksitas $O(r)$. Sehingga kompleksitas akhir program untuk menghitung rumus (2.10) adalah $O(r^2)$.

Selanjutnya akan dilakukan perhitungan sigma bagian kanan pada rumus (2.11). Pada rumus (2.11) dapat dimisalkan rumus berikut ini :

$$k(i) = \sum_{j=0}^{i} (-1)^j \binom{i}{j} (n-j)^r \qquad (2.12)$$

Dengan adanya permisalan rumus (2.12) maka rumus (2.11) dapat disederhanakan menjadi :

$$-a^{n+1} \sum_{i=0}^{r} \frac{1}{(1-a)^{i+1}} \sum_{j=0}^{i} (-1)^j \binom{i}{j} (n-j)^r = -a^{n+1} \sum_{i=0}^{r} \frac{1}{(1-a)^{i+1}} \, k(i)$$

Untuk menghitung rumus diatas dibutuhkan nilai $k(i)$ mulai dari $k(0)$ s/d $k(r)$. $\left\langle {r \atop r-1} \right\rangle$ dapat dihitung dengan *recurrence relation* berikut ini:

$$c(0,j) = (n-j)^r$$
$$c(i,j) = c(i-1,j) - c(i-1,j+1) \quad (2.13)$$

Dari *recurrence relation* diatas dapat dituliskan rumus sebagai berikut :

$$k(i) = c(i,0). \quad (2.14)$$

Dengan menggunakan *recurrence relation* ini $k(i)$ mulai dari $k(0)$ s/d $k(r)$ dapat dihitung dengan menggunakan *array* dan *nested loop* sehingga kompleksitas perhitungan $k(i)$ adalah $O(r^2)$. Setelah nilai dari $k(i)$ didapat, maka dapat dilakukan perhitungan rumus (2.11) dengan menggunakan *loop* yang memiliki kompleksitas $O(r)$. Sehingga kompleksitas akhir dari perhitungan rumus (2.11) adalah $O(r^2)$.

Setelah mendapatkan hasil dari perhitungan kedua bagian rumus (2.2), dilakukan penjumlahan hasil dari kedua bagian tersebut. Maka kompleksitas program dari perhitungan rumus (2.2) adalah $O(r^2)$. Dengan kompleksitas tersebut maka solusi tersebut dapat digunakan untuk pada permasalahan Moon Safari (medium) yang memiliki batasan $1 < r < 10^3$. Solusi ini masih relatif kurang efisien untuk mengatasi permasalahan Moon Safari (hard) yang memiliki batasan nilai r lebih besar yaitu $1 < r < 10^6$. Maka untuk menyelesaikan permasalahan Moon Safari (hard) solusi ini masih perlu dioptimasi lagi. Pengoptimasian lebih lanjut dengan menggunakan FFT akan dibahas pada subbab 2.4. Dan pada subbab 2.4.3 akan dibahas penggunaan *Fast Polynomial Multiplication* dalam menyelesaikan rumus (2.2).

2.4 Strategi Penyelesaian dengan Multiplication Polynomial

Untuk mengoptimasi solusi sebelumnya maka rumus (2.2) akan dipecah menjadi beberapa bagian sehingga membentuk suatu perkalian dua *polynomial* berderajat r. Perkalian dua *polynomial* berderajat r dapat dihitung algoritma *Fast Multiplication Polynomials* sehingga kompleksitas akhir solusi ini menjadi $O(r \log r)$. Penjelasan lebih lanjut dapat dilihat pada subbab berikutnya. Pada subbab 2.4.1 akan dibahas mengenai algoritma *Fast Fourier Transform* yang akan digunakan untuk *Fast Multiplication Polynomials*. Selanjutnya pada subbab 2.4.2 akan dibahas bagaimana mengunakan *Fast Fourier Transform* untuk mengalikan dua polynomial.

2.4.1 Fast Fourier Transform

Fast fourier transform adalah salah satu algoritma di bidang *signal processing*. *Fast fourier transform* dapat mengubah signal dari *time domain* menjadi *frekuensi domain*. *Fast fourier transform* dalam kehidupan sehari-hari digunakan untuk *compression techniques* pada audio dan video.

Berikut rumus *Discrete Fourier Transform* sebagai dasar *Fast Fourier Transform* :

$$F(j) = \sum_{k=0}^{r-1} f(j) e^{i2\pi jk/r} \tag{2.15}$$

Berikut rumus *Inverse Discrete Fourier Transform* sebagai dasar *Inverse Fast Fourier Transform* :

$$f(j) = \frac{1}{r} \sum_{k=0}^{r-1} F(k) e^{-i2\pi jk/r} \tag{2.16}$$

Perhitungan DFT masih membutuhkan kompleksitas $O(r^2)$ maka dibutuhkan fungsi recurence sehingga perhitungan menjadi lebih efisien dengan kompleksitas $O(r \log r)$, dengan r merupakan bilangan pangkat 2. Berikut ini recurence relation *Fast Fourier Transform* :

$$W_r = e^{i2\pi/r} = \cos\left(\frac{2\pi}{r}\right) + i \sin\left(\frac{2\pi}{r}\right)$$

$$F(j) = \sum_{k=0}^{r-1} f(k) W_r^{jk}$$

$$F(j) = \sum_{k=0}^{r/2-1} f(2k) W_{r/2}^{jk} + W_r^{j} \sum_{k=0}^{r/2-1} f(2j+1) W_{r/2}^{jk}$$

2.4.2 Fast Multiplication Polynomial

Secara naif mengalikan dua polinomial dengan *degree* r membutuhkan kompleksitas $O(r^2)$. Dengan menggunakan algoritma *Fast Fourier Transform*, perkalian dua polinomial dapat dihitung dengan kompleksitasnya menjadi $O(r \log r)$. Polinomial $A(x)$ dan polinomial $B(x)$ dapat direpresentasikan sebagai berikut, dimana a_j dan b_j merupakan koefisien dari x^j dari masing-masing polynomial :

$$A(x) = \sum_{j=0}^{r-1} a_j x^j \qquad\qquad B(x) = \sum_{j=0}^{r-1} b_j x^j$$

Misal $C(x)$ merupakan hasil perkalian polynomial $A(x)$ dan polinomial $B(x)$ yang memiliki derajat yang sama yaitu r. maka

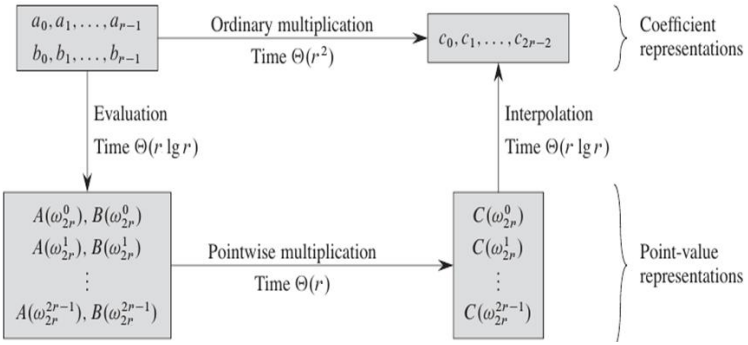
dapat ditulis rumus polinomial $C(x)$ sebagai berikut, dimana c_j merupakan koefisien dari x^j :

$$C(x) = \sum_{j=0}^{2r-2} c_j x^j$$

Dimana c_j dapat dituliskan sebagai :

$$c_j = \sum_{k=0}^{j} a_k b_{j-k}$$

FFT dapat mengubah suatu persamaan polinomial dari *coefficient representation* menjadi *point-value representation* dan sebaliknya. Untuk menghitung perkalian pada *coefficient representation* dapat dilakukan dengan kompleksitas $O(r^2)$, sedangkan pada *point-value representation* hanya membutuhkan kompleksitas $O(r)$. Untuk mempercepat perhitungan perkalian polinomial dibutuhkan konversi dari *coefficient representation* menjadi *point-value representation* dan sebaliknya. Untuk melakukan konversi ini membutuhkan kompleksitas $O(r \log r)$.



Gambar 2.5 Representasi Perkalian Polinomial

Konversi dari *coefficient representation* menjadi *point-value representation* dapat menggunakan rumus (2.15), sedangkan konversi dari *point-value representation* menjadi *coefficient representation* dapat menggunakan rumus (2.16). Proses perkalian polinomial dengan algoritma *Fast Fourier Transform* dapat dilihat pada Gambar 2.5.

2.4.3 Penjelasan Strategi Penyelesaian

Misalkan fungsi $k(i,n,r)$ sebagai berikut :

$$k(i,n,r) = \sum_{j=0}^{i} (-1)^j \binom{i}{j} (n-j)^r$$

Dan fungsi $S_r(t)$ merupakan euler polynomial sebagai berikut :

$$S_r(t) = \sum_{k=0}^{r-1} t^k \left\langle {r \atop k} \right\rangle$$

Maka rumus (2.2) dapat diubah menjadi :

$$\sum_{i=1}^{n} a^i i^r = \frac{1}{(1-a)^{r+1}} a\, S_r(a) - a^{n+1} \sum_{i=0}^{r} \frac{1}{(1-a)^{i+1}} k(i,n,r) \tag{2.17}$$

Dengan pemecahan ini, maka untuk menghitung rumus (2.17) dibutuhkan nilai $S_r(a)$ dan $k(0,r,n)$, $k(1,r,n)$,…, $k(r,r,n)$. Jika nilai tersebut sudah dicari maka komplesitas untuk perhitungan tersebut ialah $O(r)$. Untuk pencarian nilai dari $S_r(a)$ dibutuhkan kompleksitas $O(r \log r)$. Algoritma pencarian $S_r(a)$ sehingga

mempunyai kompleksitas tersebut dijelaskan pada subbab 2.2.2. Sedangkan untuk pencarian masing-masing nilai dari $k(i,r,n)$ dibutuhkan komplesitas $O(q)$ dengan asumsi $(n-j)^r$.dan $\binom{i}{j}$ telah dicari. Maka untuk mencari nilai dari $k(0,r,n), k(1,r,n),\ldots, k(r,r,n)$ secara naif dibutuhkan kompleksitas $O(r^2)$. Tetapi perhitungan ini masih dapat dioptimasi dengan menggunakan algoritma *Fast Multiplication Polynomial* sehingga kompleksitasnya menjadi $O(r \log r)$. Hal ini dapat dihitung dengan algoritma tersebut karena rumus dari $K(0,r,n)/0!, K(1,r,n)/1!,\ldots, K(r,r,n)/r!$ merupakan nilai koefisien dari perkalian dua polynomial. Berikut ini merupakan rumus koefisien dari hasil perkalian 2 polinomial $A(x)$ dan polinomial $B(x)$ yang menghasilkan polynomial $C(x)$ dengan koefisien c_i sebagai berikut :.

$$c_i = \sum_{j=0}^{k} a_j b_{k-j}$$

Penjelasan lebih lanjut mengenai *Fast Polynomial Multiplication* dapat dilihat pada subbab 2.4.2. rumus $k(i,r,n)$ dapat diubah menjadi seperti berikut ini dengan memecah rumus kombinasi :

$$k(i,r,n) = i! \sum_{j=0}^{i} (-1)^j \frac{1}{(i-j)!\,j!} (n-j)^r$$

Koefisien dari polynomial $C(x)$ yaitu c_i dapat dimisalkan dari persamaan diatas, dengan memindahkan $i!$ ke ruas kiri. Dengan adanya permisalan itu dihasilkan rumus sebagai berikut :

$$c_i = \frac{k(i,r,n)}{i!} = \sum_{j=0}^{i} (-1)^j \frac{1}{(i-j)!\,j!} (n-j)^r$$

Dari rumus diatas polinomial $C(x)$ dapat difaktorkan menjadi 2 polinomial $A(x)$ dan polinomial $B(x)$ karena memiliki pola perkalian 2 polynomial. Berikut ini pola yang dimaksud :

$$c_i = \sum_{j=0}^{i} \frac{(-1)^j (n-j)^r}{j!} \frac{1}{(i-j)!}$$

Maka nilai koefisien polynomial A (a_i) ialah :

$$a_i = \frac{(-1)^i (n-i)^r}{i!}$$

Sedangkan nilai koefisien polynomial B (b_i) ialah :

$$b_i = \frac{1}{i!}$$

Maka rumus dari fungsi $k(i, r, n)$ sebagai berikut :

$$k(i, r, n) = i!\, c_i \qquad (2.18)$$

Total dari kompleksitas dari solusi diatas ialah $O(r \log r + r \log r + r)$, maka kompleksitas menjadi $O(r \log r)$. Dengan kompleksitas tersebut solusi ini sudah dapat menyelesaikan permasalahan Moon Safari (hard), Tetapi solusi masih membutuhkan waktu cukup tinggi untuk menyelesaikan permasalahan tersebut. Hal ini disebabkan adanya proses algoritma $O(r \log r)$ yang sering dilakukan beberapa kali sehingga terdapat konstanta yang mengalikan kompleksitas tersebut walaupun angkanya tetap. Maka dibutuhkan solusi lain untuk menyelesaikan permasalahan Moon Safari (hard) dengan waktu yang lebih rendah. Solusi yang lebih optimal menggunakan *Interpolasi Polynomial* akan dijelaskan pada subbab berikutnya.

2.5 Strategi Penyelesaian dengan Interpolation Polynomial

Untuk mengoptimasi problem ini lebih lanjut dapat dilakukan dengan algoritma interpolasi polinomial. Salah satu algoritma interpolasi polinomial yang cukup cepat yaitu *Lagrange Interpolation Polynomial*. Hal ini dapat dilakukan rumus (2.17) dapat diubah menjadi polinomial. Sehingga kompleksitas akhir solusi ini menjadi $O(r \log r)$. Walaupun terlihat sama kompleksitasnya dengan solusi pada subbab 2.3, solusi ini lebih cepat dibanding solusi pada subbab 2.3. Penjelasan lebih lanjut dapat dilihat pada subbab berikutnya. Pada subbab 2.5.1 akan dijelaskan mengenai *Lagrange Interpolation Polynomial* yang akan dipakai untuk menyelesaikan rumus (2.17). Dan pada subbab 2.5.2 akan dibahas bagimana menerapkan *Lagrange Interpolation Polynomial* pada rumus (2.17).

2.5.1 Lagrange Interpolation Polynomial

Interpolasi polynomial berderajat r membutuhkan $r + 1$ titik yang berbeda dalam koordinat kartesius. Terdapat $r + 1$ titik yaitu $(x_0, y_0), (x_1, y_1), \dots, (x_r, y_r)$

Polynomial P(x) yang akan diinterpolasi dapat dituliskan dalam rumus berikut :

$$P(x) = \sum_{j=0}^{r} y_j \, p_j(x) \tag{2.19}$$

Dimana $p_j(x)$ dapat didefinisikan sebagai berikut :

$$p_j(x) = \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m} \tag{2.20}$$

Untuk melakukan menginterpolasi suatu polinomial dapat menggunakan *Interpolation Polynomial Lagrange* yang mempunyai kompleksitas program $O(n)$. Untuk menghitung *Interpolation Polynomial Lagrange* pertama akan dilakukan perhitungan $p_j(x)$. Perhitungan $p_j(x)$ mulai dari *j*=0 sampai dengan *j*=*r* membutuhkan penjabaran fungsi $p_j(x)$ agar dapat dihitung dengan kompleksitas $O(n)$. Berikut ini penjabarannya:

$$p_j(x) = \prod_{0 \le m < j} \frac{x - x_m}{x_j - x_m} \prod_{j < m \le k} \frac{x - x_m}{x_j - x_m}$$

Dari penjabaran diatas fungsi $p_j(x)$ dapat dipecah menjadi 2 fungsi baru sebagai berikut :

$$pLeft_j(x) = \prod_{0 \le m < j} \frac{x - x_m}{x_j - x_m}$$

$$pRight_j(x) = \prod_{j < m \le k} \frac{x - x_m}{x_j - x_m}$$

Maka dapat disusun rumus baru sebagai berikut :

$$p_j(x) = pLeft_j(x)\, pRight_j(x)$$

Perhitungan $pLeft_j(x)$ dan $pRight_j(x)$ mulai dari *j*=0 sampai dengan *j*=*r* dapat menggunakan array sehingga kompleksitas yang dibutuhkan ialah $O(n)$. Setelah nilai $pLeft_j(x)$ dan $pRight_j(x)$ dihitung maka perhitungan $p_j(x)$ mulai dari *j*=0 sampai dengan *j*=*r* juga membutuhkan kompleksitas $O(n)$. Setelah nilai $p_j(x)$ dihitung maka proses perhitungan $P(x)$ juga membutuhkan kompleksitas $O(n)$. Kesimpulannya kompleksitas total yang dibutuhkan untuk melakukan *Interpolation Polynomial Lagrange* adalah $O(n)$

2.5.2 Penjelasan Strategi Penyelesaian

Berikut ini adalah cara merubah rumus (2.17) menjadi bentuk polynomial. Berikut ini adalah persamaan dari rumus (2.17) :

$$\sum_{i=1}^{n} a^i i^r = \frac{1}{(1-a)^{r+1}} a\, S_r(a) - a^{n+1} \sum_{i=0}^{r} \frac{1}{(1-a)^{i+1}} k(i,n,r)$$

Langkah 1 : pindahkan $\frac{1}{(1-a)^{r+1}} a\, S_r(a) - a^{n+1}$ ke ruas kiri

$$\sum_{i=1}^{n} a^i i^r - \frac{1}{(1-a)^{r+1}} a\, S_r(a) = -a^{n+1} \sum_{i=0}^{r} \frac{1}{(1-a)^{i+1}} k(i,n,r)$$

Langkah 2 : bagi kedua ruas dengan a^n

$$\frac{1}{a^n}\left(\sum_{i=1}^{n} a^i i^r - \frac{1}{(1-a)^{r+1}} a\, S_r(a)\right) = -a \sum_{i=0}^{r} \frac{1}{(1-a)^{i+1}} k(i,n,r) \tag{2.21}$$

Ditunjukkan pada setiap fungsi $k(i,r,n)$ merupakan sebuah polinomial n berderajat r. Fungsi $k(i,r,n)$ merupakan hasil sigma dari perkalian $(-1)^j \frac{1}{(i-j)!j!}$ yang merupakan konstanta dan $(n-j)^r$ yang merupakan polinomial. Karena perkalian antara konstanta dan polinomial menghasilkan polinomial baru dan penambahan polinomial dan polinomial juga merupakan polinomial baru maka dapat ditarik kesimpulan bahwa fungsi $k(i,r,n)$ merupakan polinomial. Maka dapat disimpulkan juga

$\sum_{i=0}^{r} \frac{1}{(1-a)^{i+1}} k(i,n,r)$ merupakan polinomial karena merupakan hasil sigma dari perkalian $\frac{1}{(1-a)^{i+1}}$ yang merupakan konstanta dan $k(i,n,r)$ yang merupakan polinomial. Maka dapat disimpulkan juga ruas kanan pada rumus (2.21) tetap merupakan polinomial setelah dikalikan dengan $-a$. Ruas kanan pada rumus (2.21) merupakan polinomial maka dapat disimpulkan ruas kiri pada rumus (2.21) juga merupakan polinomial karena memiliki nilai yang sama. Dari pembuktian ini maka dihasilkan 2 persamaan polinomial baru yaitu yang mempunyai bentuk berbeda. Dari kedua persamaan ini akan dipilih satu polinomial yang akan diinterpolasi. Berikut ini persamaan polinomial yang dihasilkan dari rumus (2.21).

$$poly(n) = \frac{1}{a^n}\left(\sum_{i=1}^{n} a^i i^r - \frac{1}{(1-a)^{r+1}} a\, S_r(a)\right) \tag{2.22}$$

$$poly(n) = -a \sum_{i=0}^{r} \frac{1}{(1-a)^{i+1}} k(i,n,r) \tag{2.23}$$

Terlihat bahwa untuk menghitung rumus (2.22) relatif lebih cepat dibanding menghitung dengan rumus (2.23). Rumus (2.22) dapat dihitung dengan kompleksitas $O(r \log r)$ dan Rumus (2.23) juga dapat dihitung dengan kompleksitas $O(r \log r)$. Mengenai asal usul kompleksitas tersebut telah dijelaskan pada subbab-subbab sebelumnya. Walaupun komplesitasnya terlihat sama, tetapi dibutuhkan waktu lebih untuk menghitung rumus (2.23) dikarenakan pada perhitungan *Fast Multiplication Polynomial* yang membutuhkan perubahan dari *coefficient representation* menjadi *point-value representation* dan perubahan dari *point-value representation* menjadi *coefficient representation*. Proses perubahan representasi ini terdapat operasi bilangan kompleks

sehingga membutuhkan waktu lebih lama dibanding operasi bilangan bulat.

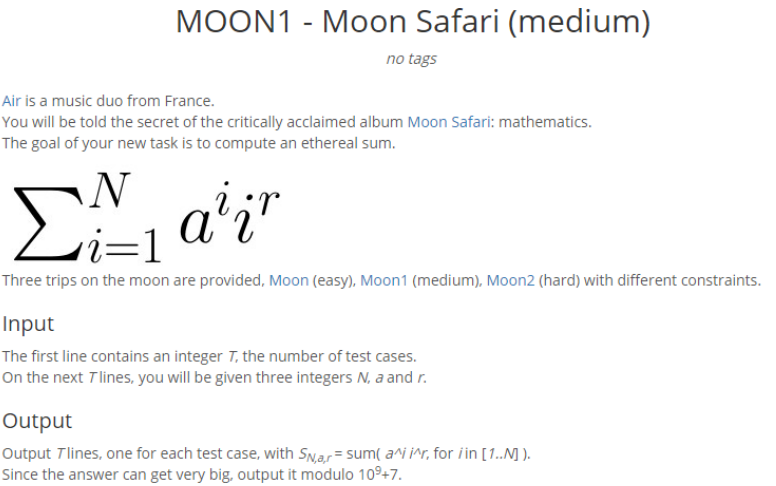
Dengan terbentuknya persamaan polynomial maka dari rumus (2.22) dapat diubah menjadi persamaan berikut dengan mengalikan $-a^n$ pada kedua ruas kemudian memindahkan $\sum_{i=1}^{n} a^i i^r$ ke ruas kiri dan memindahkan $poly(n)$ ke ruas kanan:

$$\sum_{i=1}^{n} a^i i^r = \frac{1}{(1-a)^{r+1}} a\, S_r(a) + poly(n) a^n \qquad (2.24)$$

Pada rumus (2.24) jika $S_r(a)$ dan $poly(n)$ sudah dicari maka kompleksitas untuk menghitungnya nilai $\sum_{i=1}^{n} a^i i^r$ adalah $O(\log n)$. Fungsi $S_r(a)$ dapat dihitung dengan kompleksitas $O(r \log r)$. Cara menghitung fungsi $S_r(a)$ akan dijelaskan lebih lanjut pada subbab 2.5. Sedangkan untuk menghitung $poly(n)$ dapat menggunakan *Lagrange Interpolation Polynomial* yang membutuhkan kompleksitas $O(r)$ jika nilai dari $poly(1)$ sampai dengan $poly(r)$ dimana r merupakan derajat dari polynomial tersebut. Untuk menghitung $poly(1)$ sampai dengan $poly(r)$ dibutuhkan kompleksitas $O(r \log r)$ jika $S_r(a)$ telah dihitung. Penjelasan *Lagrange Interpolation Polynomial* lebih lanjut dapat dilihat pada subbab 2.4.1. Kesimpulannya total kompleksitas dari solusi ini ialah $O(r \log r)$.

2.6 Permasalahan Moon Safari(Medium) pada SPOJ

Pada situs SPOJ ini terdapat permasalahan perhitungan $\sum_{i=1}^{n} a^i i^r$. Deskripsi singkat dari persoalan ini ialah akan diberikan 3 bilangan bulat N, A dan R yang merupakan nilai variabel dari fungsi $\sum_{i=1}^{n} a^i i^r$. Program akan mengeluarkan hasil dari fungsi $\sum_{i=1}^{n} a^i i^r$. Karena hasil dari fungsi tersebut bisa sangat besar maka cukup menampilkan sisa hasil bagi fungsi tersebut dengan 1000000007. Deskripsi permasalahan lebih detail dapat dilihat pada Gambar 2.6.



MOON1 - Moon Safari (medium)

no tags

Air is a music duo from France.
You will be told the secret of the critically acclaimed album Moon Safari: mathematics.
The goal of your new task is to compute an ethereal sum.

$$\sum_{i=1}^{N} a^i i^r$$

Three trips on the moon are provided, Moon (easy), Moon1 (medium), Moon2 (hard) with different constraints.

Input

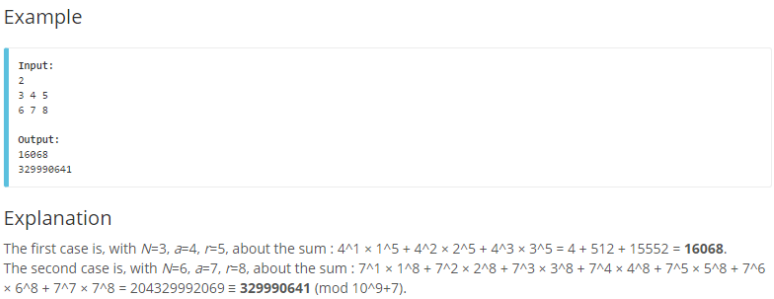
The first line contains an integer *T*, the number of test cases.
On the next *T* lines, you will be given three integers *N*, *a* and *r*.

Output

Output *T* lines, one for each test case, with $S_{N,a,r}$ = sum(*a*^*i* *i*^*r*, for *i* in [*1*..*N*]).
Since the answer can get very big, output it modulo 10^9+7.

Gambar 2.6 Deskripsi permasalahan Moon Safari(Medium)

Format masukan dari permasalahan tersebut dimulai dari baris pertama berisi sebuah bilangan integer T yang merepresentasikan banyaknya kasus uji. T baris berikutnya berisi 3 buah integer yang menyatakan nilai varibel N, A dan R. Format keluaran dari permasalahan tersebut adalah sebuah integer yang merepresentasikan hasil dari fungsi dimodulo 1000000007. Contoh masukan dan keluaran digambarkan pada Gambar 2.7.



Example

```
Input:
2
3 4 5
6 7 8

Output:
16068
329990641
```

Explanation

The first case is, with *N*=3, *a*=4, *r*=5, about the sum : 4^1 × 1^5 + 4^2 × 2^5 + 4^3 × 3^5 = 4 + 512 + 15552 = **16068**.
The second case is, with *N*=6, *a*=7, *r*=8, about the sum : 7^1 × 1^8 + 7^2 × 2^8 + 7^3 × 3^8 + 7^4 × 4^8 + 7^5 × 5^8 + 7^6 × 6^8 + 7^7 × 7^8 = 204329992069 ≡ **329990641** (mod 10^9+7).

Gambar 2.7 Contoh masukan dan keluaran Moon Safari(Medium)

Batasan permasalahan akan dibagi dalam berbagai *subtask*. Berikut ini batasan yang diberikan :

- T<1000 dan r <= 18
- T<100 dan r <= 72
- T<10 dan r <= 256
- T<1 dan r <= 444

Program akan diuji pada cluster Cube (Intel Pentium G860 3GHz) dengan batasan waktu eksekusi 20 s, batasan kapasitas memory sebesar 1536MB dan batasan kode sumber sebesar 50000B.

2.7 Permasalahan Moon Safari(Hard) pada SPOJ

Pada situs SPOJ ini terdapat permasalahan perhitungan $\sum_{i=1}^{n} a^i i^r$. Deskripsi singkat dari persoalan ini ialah akan diberikan 3 bilangan bulat N, A dan R yang merupakan nilai variabel dari fungsi $\sum_{i=1}^{n} a^i i^r$. Program akan mengeluarkan hasil dari fungsi $\sum_{i=1}^{n} a^i i^r$. Karena hasil dari fungsi tersebut bisa sangat besar maka cukup menampilkan sisa hasil bagi fungsi tersebut dengan 1000000007. Deskripsi permasalahan lebih detail dapat dilihat pada Gambar 2.8.

MOON2 - Moon Safari (Hard)

no tags

Air is a music duo from France.
You will be told the secret of the critically acclaimed album Moon Safari: mathematics.
The goal of your new task is to compute an ethereal sum.

$$\sum_{i=1}^{N} a^i i^r$$

Three trips on the moon are provided, Moon (easy), Moon1 (medium), Moon2 (hard) with different constraints.

Input

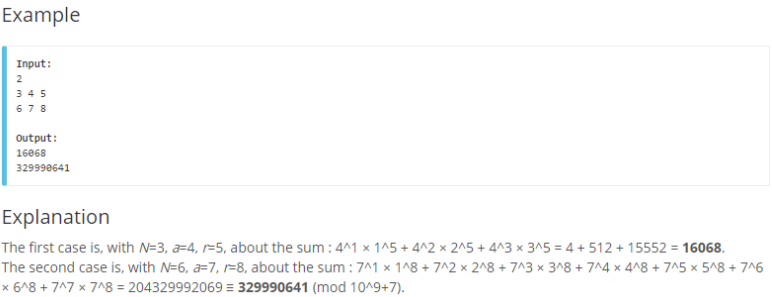
The first line contains an integer *T*, the number of test cases.
On the next *T* lines, you will be given three integers *N*, *a* and *r*.

Output

Output *T* lines, one for each test case, with $S_{N,a,r}$ = sum(*a*^*i* *i*^*r*, for *i* in [*1*..*N*]).
Since the answer can get very big, output it modulo 10^9+7.

Gambar 2.8 Deskripsi permasalahan Moon Safari(Hard)

Format masukan dari permasalahan tersebut dimulai dari baris pertama berisi sebuah bilangan integer T yang merepresentasikan banyaknya kasus uji. T baris berikutnya berisi 3 buah integer yang menyatakan nilai varibel N, A dan R. Format keluaran dari permasalahan tersebut adalah sebuah integer yang merepresentasikan hasil dari fungsi dimodulo 1000000007. Contoh masukan dan keluaran digambarkan pada Gambar 2.9.


Example

Input:
2
3 4 5
6 7 8

Output:
16068
329990641

Explanation

The first case is, with N=3, a=4, r=5, about the sum : 4^1 × 1^5 + 4^2 × 2^5 + 4^3 × 3^5 = 4 + 512 + 15552 = **16068**.
The second case is, with N=6, a=7, r=8, about the sum : 7^1 × 1^8 + 7^2 × 2^8 + 7^3 × 3^8 + 7^4 × 4^8 + 7^5 × 5^8 + 7^6 × 6^8 + 7^7 × 7^8 = 204329992069 ≡ **329990641** (mod 10^9+7).

Gambar 2.9 Contoh masukan dan keluaran Moon Safari(Hard)

Batasan permasalahan akan dibagi dalam berbagai *subtask.* Berikut ini batasan yang diberikan :

- T<20000 dan r <= 128
- T<2000 dan r <= 1250
- T<200 dan r <= 11000
- T<20 dan r <= 100000
- T<2 dan r <= 1000000

Program akan diuji pada cluster Cube (Intel Pentium G860 3GHz) dengan batasan waktu eksekusi 20 s, batasan kapasitas memory sebesar 1536MB dan batasan kode sumber sebesar 50000B

BAB 3
DESAIN

Pada bab ini akan dijelaskan desain sistem yang digunakan untuk menyelesaikan permasalahan pada Tugas Akhir ini.

3.1 Deskripsi Umum Sistem

Pada sistem ini terdapat 2 konstanta yang digunakan yaitu MOD dan MAXR. Konstanta MOD bernilai 1000000007 dan konstanta MAXR bernilai maksimal nilai *r* dari batasan permasalahan. Pada permasalahan Moon Safari(Medium) nilai dari konstanta MAXR adalah 444 sedangkan pada permasalahan Moon Safari(Hard) nilai dari konstanta MAXR adalah 1000000.

```
10 let inv[0..MAXR] be a new array
11 let fact[0..MAXR] be a new array
12 let invFact [0..MAXR] be a new array
13 let power [0..MAXR] be a new array
14 Initialize()
15 t = Input() // Total Testcase
16 for t = 1 to t
17     n = Input() // nilai n
18     a = Input() // nilai a
19     r = Input() // nilai r
20     for i = 0 to r
21         power[i]=BigModulo(i,r);
22     hasil = Calculate(n,a,r) //kalkulasi fungsi
23     Print(hasil)
```

Gambar 3.1 Pseudocode Fungsi Main

Pertama sistem akan melakukan inisialisasi nilai array *inv*, *fact*, dan *invFact*. Inisialisasi ini ada di dalam fungsi Initialize. Penjelasan lebih lanjut mengenai fungsi Initialize dapat dilihat pada subbab 3.2. Initialisasi ini dilakukan sebelum membaca kasus uji karena nilainya sama untuk setiap kasus uji. Sistem selanjutnya akan

menerima masukan berupa jumlah kasus uji. Untuk setiap kasus uji akan diberikan 3 masukkan bilangan bulat yaitu *n*, *a*, dan *r*. Kemudian sistem akan melakukan inisialisasi nilai array *power*. Untuk setiap index ke-i pada array *power* berisi $i^r \% MOD$. *Preprocessing* dan inisialisasi array *power* ini akan akan berguna untuk mempercepat kinerja program sehingga perhitungan yang sama tidak dilakukan berkali-kali. Setelah pembacaan inputan dan inisialisai dilakukan fungsi kalkulasi dalam fungsi Calculate. Ada 3 jenis fungsi Calculate yang merupakan desain dari 3 macam strategi penyelesaian yang telah dijelakan pada subbab 2.3, 2.4, dan 2.5. Setelah melakukan kalkulasi sistem akan menampilkan hasil fungsi Calculate. Pseudocode Fungsi Main ditunjukkan pada Gambar 3.1.

3.2 Desain Fungsi Initialize

Fungsi ini berguna untuk mengisi global array bilangan bulat *inv*, *fact*, dan *invFact.* Masing-masing isi array tersebut bernilai :

1. *inv*[i] dengan $\frac{1}{i} \% MOD$.
2. *fact*[i] dengan $i! \% MOD$.
3. *invfact*[i] dengan $\frac{1}{i!} \% MOD$.

Nilai-nilai dari global array ini akan digunakan pada fungsi-fungsi berikutnya. Pseudocode Fungsi Initialize ditunjukkan pada Gambar 3.2.

```
1  inv[0] = inv[1] = 1
2  fact[0] = fact[1] = 1
3  invFact[0] = invFact[1] = 1
4  for i = 2 to MAXR
5      inv[i] = (inv[MOD%i] * (MOD-MOD/i)) % MOD
6      fact[i] = (fact[i-1] * i) % MOD
7      invFact[i] = (invFact[i-1] * inv[i]) % MOD
```

Gambar 3.2 Pseudocode Fungsi Initialize

3.3 Desain Fungsi BigModulo

Fungsi BigModulo menerima 2 parameter bilangan bulat yaitu *a* dan *b*. Fungsi akan menghitung sisa bagi dari a^b terhadap MOD. Pseudocode Fungsi BigModulo ditunjukkan pada Gambar 3.3.

```
1  power = a
2  hasil = 1
3  while b>0
4      if b and 1
5          hasil = (hasil * power) % MOD
6      power = (power *power) % MOD
7      b = b >> 1
8  return hasil
```

Gambar 3.3 Pseudocode Fungsi BigModulo

3.4 Desain Fungsi InversModulo

Fungsi InversModulo menerima 1 parameter bilangan bulat *n*. Fungsi ini akan mencari nilai *Invers Modulo* dari *n* terhadap MOD dengan menggunakan algoritma *Extended Euclidean*. Pseudocode Fungsi InversModulo ditunjukkan pada Gambar 3.4.

```
1  u = 1
2  x = 0
3  s = n
4  t = MOD
5  while s>0
6      q = t / s
7      x = x - u * q
8      swap(x,u)
9      t = t - s * q
10     swap(t,s)
11 return x/t
```

Gambar 3.4 Pseudocode Fungsi InversModulo

3.5 Desain Fungsi EulerPolynomials

Fungsi EulerPolynomials menerima 2 parameter bilangan bulat yaitu *t* dan *n*. Fungsi ini akan menghitung nilai dari *Euler Polynomials* $S_n(t)$. Cara menghitung nilai dari *Euler Polynomials* telah dijelaskan pada subbab 2.3.2. Rumus (2.7) pada subbab 2.3.2 akan digunakan untuk mendesain perhitungan *Euler Polynomials* tersebut. Pseudocode Fungsi EulerPolynomials ditunjukkan pada Gambar 3.5.

```
1  hasil = 0
2  tmp = 0
3  com = n+1
4  invT = InversModulo(t);
5  for k = 1 to n
6      tmp = ((tmp+power[k])*invT)%MOD
7      com=(com*(n+1-k) *inv[k+1])%MOD
8      if k&1
9          hasil = (hasil-com*tmp)%MOD
10     else
11         hasil = (hasil+com*tmp)%MOD
12 hasil = (hasil*bigMod(-t,n))%MOD
13 return hasil
```

Gambar 3.5 Pseudocode Fungsi Euler Polynomial

3.6 Desain Fungsi Calculation Euler Polynomial

Fungsi Calculation menerima 3 parameter bilangan bulat yaitu *n*, *a* dan *r*. Fungsi Calculation ini merupakan desain dari strategi penyelesaian pada subbab 2.3. Rumus (2.2) akan digunakan untuk mendesain fungsi Calculation ini. Pseudocode Fungsi Calculation ditunjukkan pada Gambar 3.6.

```
1  let eulerianNumber [0..MAXR] [0..MAXR] be new array
2  let konstanta [0..MAXR] be new array
3  hasil1 = hasil2 = 0
4  inv1minA= InversModulo (1-a,MOD);
5  eulerianNumber [0][0]=1;
6  for n = 1 to r
7      eulerianNumber [n][0] = 1;
8      for k = 1 to n-1
9          eulerianNumber [n][k] =
               ((n-k)*eulerianNumber [n-1][k-1] +
               (k+1)*eulerianNumber [n-1][k]) )) %MOD
10 for i = r downto 1
11     hasil1 = ((hasil1 + eulerianNumber [r][i-1])*a)%MOD
12 hasil1 = (hasil1*BigModulo(inv1minA,r+1))%MOD
13 for i = 0 to r
14     konstanta[i]=BigModulo(n-i,r)
15 for i = 0 to r
16     for j = r downto i+1
17         konstanta [j]=(konstanta [j-1]- konstanta [j])%MOD
18 for i = r downto 0
19     hasil2 = ((hasil2 + konstanta [i])*inv1minA)%MOD
20 hasil2 = (hasil2*(-BigModulo(a,n+1))) %MOD
21 return (hasil1+hasil2) %MOD
```

Gambar 3.6 Pseudocode Fungsi Calculation

3.7 Desain Fungsi FastFourierTransform

Fungsi FastFourierTransform menerima 3 parameter yaitu *Array x*, *size* dan *sign*. *Array x* berisi nilai yang akan diubah dari *time domain* menjadi *frekuensi domain* atau sebaliknya. *Array x* merupakan array 1 dimensi dimana nilai real disimpan pada index ganjil(1,3,5,7,…) sedangkan nilai imaginer disimpan pada index genap(2,4,6,8,…). *size* merupakan ukuran *Array x*. *sign* merupakan tanda yang menyatakan proses yang dilakukan fungsi, jika 1 berarti melakukan proses *Fast Fourier Transform*, sedangkan jika bernilai

-1 berarti melakukan proses *Invers Fast Fourier Transform*. Pseudocode Fungsi FastFourierTransform dapat dilihat pada buku Numerical Recipes in C.

3.8 Desain Fungsi MultiplicationPolynomials

Fungsi MultiplicationPolynomial menerima 3 parameter yaitu *Array a*, *Array b* dan *size*. *Array a* berisi koefisien polynomial pertama. *Array b* berisi koefisien polynomial kedua. *size* merupakan ukuran polynomial pertama dan kedua. Fungsi ini mengembalikan *Array c* yang berisi koefisien polynomial hasil. Pseudocode Fungsi MultiplicationPolynomial ditunjukkan pada Gambar 3.7 dan Gambar 3.8.

```
1  let ca1[1..8*MAXR] be new array
2  let ca2[1..8*MAXR] be new array
3  let cb1[1..8*MAXR] be new array
4  let cb2[1..8*MAXR] be new array
5  let cc11[1..8*MAXR] be new array
6  let cb12[1..8*MAXR] be new array
7  let cb22[1..8*MAXR] be new array
8  let c[0..8*MAXR] be new array
9  k=1
10 while (k<size)
11     k=k<<1
12 k=k<<1
13 for i = 0 to k<<1
14     ca1[i] = cb1[i] = ca2[i] = cb2[i] = 0
15 for i = 0 to size-1
16     ca1[(i<<1)+1] = a[i]>>15
17     cb1[(i<<1)+1] = b[i]>>15
18     ca2[(i<<1)+1] = a[i]&32767
19     cb2[(i<<1)+1] = b[i]&32767
```

Gambar 3.7 Pseudocode Fungsi MultiplicationPolynomials Bagian A

```
20 FastFourierTransform(ca1, k, 1);
21 FastFourierTransform(cb1, k, 1);
22 FastFourierTransform(ca2, k, 1);
23 FastFourierTransform(cb2, k, 1);
24 for i = 0 to k-1
25     cc11[(i<<1)+1] = ca1[(i<<1)+1]*cb1[(i<<1)+1]
                        - ca1[(i<<1)+2]*cb1[(i<<1)+2]
26     cc11[(i<<1)+2] = ca1[(i<<1)+1]*cb1[(i<<1)+2]
                        + ca1[(i<<1)+2]*cb1[(i<<1)+1]
27     cc12[(i<<1)+1] = ca1[(i<<1)+1]*cb2[(i<<1)+1]
                        - ca1[(i<<1)+2]*cb2[(i<<1)+2]
                        + ca2[(i<<1)+1]*cb1[(i<<1)+1]
                        - ca2[(i<<1)+2]*cb1[(i<<1)+2]
28     cc12[(i<<1)+2] = ca1[(i<<1)+1]*cb2[(i<<1)+2]
                        + ca1[(i<<1)+2]*cb2[(i<<1)+1]
                        + ca2[(i<<1)+1]*cb1[(i<<1)+2]
                        + ca2[(i<<1)+2]*cb1[(i<<1)+1]
29     cc22[(i<<1)+1] = ca2[(i<<1)+1]*cb2[(i<<1)+1]
                        - ca2[(i<<1)+2]*cb2[(i<<1)+2]
30     cc22[(i<<1)+2] = ca2[(i<<1)+1]*cb2[(i<<1)+2]
31                     + ca2[(i<<1)+2]*cb2[(i<<1)+1]
32 FastFourierTransform(cc11, k, -1);
33 FastFourierTransform(cc12, k, -1);
34 FastFourierTransform(cc22, k, -1);
35 for i = 0 to size
36     c11 = (cc11[(i<<1)+1]/k)%MOD
37     c12 = (cc12[(i<<1)+1]/k)%MOD
38     c22 = (cc22[(i<<1)+1]/k)%MOD;
39     c[i] = ((c11<<30)+(c12<<15)+ c22)%MOD;
40 return c
```

Gambar 3.8 Pseudocode Fungsi MultiplicationPolynomials Bagian B

3.9 Desain Fungsi Calculation Multiplication Polynomial

Fungsi Calculation menerima 3 parameter bilangan bulat yaitu *n*, *a* dan *r*. Fungsi Calculation ini merupakan desain dari strategi penyelesaian pada Subbab 2.4. Rumus (2.17) akan digunakan untuk mendesain fungsi Calculation ini. Pseudocode Fungsi Calculation ditunjukkan pada Gambar 3.9.

```
1  let konstanta [0..MAXR] be new array
2  let polyA [0..MAXR] be new array
3  let polyB [0..MAXR] be new array
4  hasil1 = 0
5  hasil2 = 0
6  inv1minA= InversModulo (1-a,MOD);
7  //Menghitung Bagian Kiri
8  hasil1 = EulerPolynomials(a,r)
9  hasil1 = (hasil1*a*BigModulo(inv1minA,r+1))%MOD
10 //Menghitung  Bagian Kanan
11 for i = 0 to r
12     if (i&1)
13         polyA [i] = (- BigModulo(n-i,r)*invFact[i])%MOD
14     else
15         polyA [i] = (BigModulo(n-i,r)*invFact[i])%MOD
16     polyB [i] = invFact[i]
17 konstanta = MultiplicationPolynomials(polyA, polyB,r+1)
18 for i = 0 to r
19     konstanta[i] = (konstanta[i]*fact[i])%MOD
20 hasil2 = 0;
21 for i = r downto 0
22     hasil2 = ((hasil2 + konstanta [i])*inv1minA)%MOD
23 hasil2 = (hasil2*(-BigModulo(a,n+1))) %MOD
24 return (hasil1+hasil2) %MOD
```

Gambar 3.9 Pseudocode Fungsi Calculation

3.10 Desain Fungsi Lagrange

Fungsi Lagrange menerima 3 parameter yaitu *Array f, r* dan *n*. *Array f* berisi nilai polinomial ke-0 sampai dengan ke-*r* yang akan dilakukan interpolasi. *r* berisi derajat polynomial. *n* merupakan index polynomial yang dicari. Fungsi ini mengembalikan nilai polynomial ke-*n*. Cara melakukan *Interpolation Polynomials Lagrange* telah dijelaskan pada subbab 2.5.1. Rumus (2.19) pada subbab 2.5.1 akan digunakan untuk mendesain perhitungan *Interpolation Polynomials Lagrange* ini. Pseudocode Fungsi Lagrange ditunjukkan pada Gambar 3.10

```
1   let leftFact [0..MAXR] be new array
2   let rightFact [0..MAXR] be new array
3   if n<=r
4       return f[n]%MOD;
5   hasil = 0
6   leftFact[0] = 1
7   rightFact[r] = 1
8   for i = 0 to r-1
9       leftFact[i+1] = (leftFact[i]*(n-i))%MOD;
10  for i = r to 1
11      rightFact[i-1] = (rightFact [i]*(n-i))%MOD;
12  for i = 0 to r
13      tmp=(f[i]*leftFact[i]*rightFact[i]
                    *invFact[i]*invFact[r-i])%MOD
14      if (r-i)&1
15          hasil=(hasil-tmp)%MOD
16      else
17          hasil=(hasil+tmp)%MOD
18  return hasil
```

Gambar 3.10 Pseudocode Fungsi Lagrange

3.11 Desain Fungsi Calculation Interpolation Polynomial

Fungsi Calculation menerima 3 parameter bilangan bulat yaitu *n*, *a* dan *r*. Fungsi Calculation ini merupakan desain dari strategi penyelesaian pada Subbab 2.5. Rumus (2.24) akan digunakan untuk mendesain fungsi Calculation ini. Pseudocode Fungsi Calculation ditunjukkan pada Gambar 3.11

```
1  let poly [0..MAXR] be new array
2  inv1minA = InversModulo (1-a);
3  invA = InversModulo(a);
4  ApowN = BigModulo(a,n);
5  poly[0] = (-BigModulo(inv1minA,r+1)*a
               *EulerPolynomials(a,r))%MOD
6  for i = 1 to r
7      poly[i] = (poly[i-1]*invA+power[i])%MOD
8  hasil = (ApowN*Lagrange(poly,r,n)-poly[0])%MOD
9  return hasil
```

Gambar 3.11 Pseudocode Fungsi Calculation

BAB 4
IMPLEMENTASI

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan yang dilakukan adalah sebagai berikut:

1. Perangkat Keras
   • Processor Intel Core i7-4700HQ CPU @ 2.40GHz.
   • Memory 3.88 GB.
2. Perangkat Lunak
   • Sistem operasi Windows 7 64-bit.
   • Integrated Development Environment Dev C++ 4.9.9.2

4.2 Implementasi Fungsi Main

Fungsi main mengimplementasikan pseudo code pada subbab 3.1. Fungsi Input() adalah *scanf*. Fungsi Print() adalah *printf*. Implementasi Fungsi Main ditunjukkan pada kode sumber 4.1

```
1  int main() {
2      Initialize();
3      int t, n, a, r, hasil;
4      scanf("%d",&t);
5      while (t--) {
6          scanf("%d%d%d",&n,&a,&r);
7          for (int i=0; i<=r; i++) power[i]=BigModulo(i,r);
8          hasil = Calculate(n,a,r);
9          printf("%d\n",hasil);
10     }
11     return 0;
12 }
```

Kode Sumber 4.1 Implementasi Fungsi Main

4.3 Implementasi Variabel Global

Variabel global yang ada pada program telah dijelaskan pada subbab 3.1. Variabel global didefinisikan karena nilainya dibutuhkan oleh banyak fungsi. Implementasi dari Variabel Global ditunjukkan pada kode sumber 4.2.

```
1  #define MOD 1000000007
2  #define MAXR 1000000
3  int inv[MAXR+1];
4  int fact[MAXR+1];
5  int invFact[MAXR+1];
6  int power[MAXR+1];
```

Kode Sumber 4.2 Implementasi Variabel Global

4.4 Implementasi Fungsi Initialize

Fungsi Initialize akan menghitung nilai array *inv*, *fact* dan *invFact*. dengan rumus pada subbab 3.2. Fungsi ini merupakan implementasi dari desain fungsi pada subbab 3.2. Implementasi Fungsi Initialize dapat dilihat pada kode sumber 4.3.

```
1  void Initialize() {
2      inv[0] = inv[1] = 1;
3      fact[0] = fact[1] = 1;
4      iFact[0] = iFact[1] = 1;
5      for (int i=2; i<=MAXR; i++) {
6          inv[i] = ((long long)inv[MOD%i]*
                                (MOD-MOD/i))%MOD;
7          fact[i] = ((long long)fact[i-1]*i)%MOD;
8          invFact[i] = ((long long)invFact[i-1]*inv[i])%MOD;
9      }
10 }
```

Kode Sumber 4.3 Implementasi Fungsi Initialize

4.5 Implementasi Fungsi BigModulo

Fungsi BigModulo akan menghitung sisa bagi dari a^b terhadap MOD. Fungsi ini merupakan implementasi dari desain fungsi pada subbab 3.3. Implementasi Fungsi BigModulo dapat dilihat pada kode sumber 4.4.

```
1  int BigModulo(int a, int b) {
2      int hasil=1, power=a%MOD;
3      while (b) {
4          if (b&1) hasil=((long long)hasil*power)%MOD;
5          power=((long long)power*power)%MOD;
6          b>>=1;
7      }
8      return hasil;
9  }
```

Kode Sumber 4.4 Implementasi Fungsi BigModulo

4.6 Implementasi Fungsi InversModulo

Fungsi InversModulo akan menghitung invers modulo dari n terhadap MOD. Fungsi ini merupakan implementasi dari desain fungsi InversModulo pada subbab 3.4. Implementasi dari Fungsi InversModulo dapat dilihat pada kode sumber 4.5.

```
1  int InversModulo(int n) {
2      int u = 1, x = 0, s = n, t = MOD;
3      while (s) {
4          int q=t/s;
5          swap(x-=u*q,u);
6          swap(t-=s*q,s);
7      }
8      return x/t < 0 ? x/t + MOD : x/t;
9  }
```

Kode Sumber 4.5 Implementasi Fungsi InversModulo

4.7 Implementasi Fungsi EulerPolynomials

Fungsi EulerPolynomials akan menghitung nilai *Euler Polynomials* $S_n(t)$. Cara menghitung nilai dari *Euler Polynomials* telah dijelaskan pada subbab 2.3.2. Fungsi ini merupakan implementasi dari desain fungsi EulerPolynomials pada subbab 3.5. Implementasi dari Fungsi EulerPolynomials dapat dilihat pada kode sumber 4.6.

```
1  int EulerPolynomials(int t, int n) {
2      int hasil=0;
3      int tmp=0;
4      int com=n+1;
5      int invT= InversModulo (t);
6      for(int k=1; k<=n; k++) {
7          tmp= ((long long) (tmp+power[k])*invT)%MOD;
8          com=(((long long)com*(n+1-k))%MOD
9                                      *inv[k+1])%MOD;
10         if (k&1) {
11             hasil-= ((long long)com*tmp)%MOD;
12             if (hasil<0) hasil+=MOD;
13         }
14         else {
               hasil+=((long long)com*tmp)%MOD;
15             if (hasil>=MOD) hasil-=MOD;
16         }
17     }
18     hasil=((long long)hasil*BigModulo(-t,n))%MOD;
19     if (hasil<0) hasil+=MOD;
20     return hasil;
21 }
```

Kode Sumber 4.6 Implementasi Fungsi EulerPolynomials

4.8 Implementasi Fungsi Calculation Euler Polynomials

Fungsi Calculation akan menghitung nilai berdasarkan rumus (2.2). Untuk penjelasan lebih detail mengenai proses perhitungan rumus tersebut dapat dilihat pada subbab 2.3. Fungsi ini merupakan implementasi dari desain fungsi Calculate pada subbab 3.6. Implementasi dari Fungsi Calculation dapat dilihat pada kode sumber 4.7 dan kode sumber 4.8.

```
1  int eulerianNumber [MAXR+1][MAXR+1];
2  int konstanta[MAXR+1];
3  int Calculate(int n, int a, int r) {
4      int hasil1=0, hasil2=0;
5      int inv1minA=InversModulo(1-a);
6      eulerianNumber [0][0]=1;
7      for (int i=1; i<=r; i++) {
8          eulerianNumber[i][0]=1;
9          for (int j=1; j<i; j++)
10             eulerianNumber[i][j] =
                   ((long long) (i-j)*eulerianNumber[i-1][j-1]+
                    (long long) (j+1)*eulerianNumber[i-1][j])
                   %MOD;
11     }
12     for (int i=r; i>0; i--)
13         hasil1 = ((long long) (hasil1 +
                     eulerianNumber[r][i-1])*a)
                     %MOD;
14     hasil1= ((long long) hasil1*BigModulo(inv1minA,r+1))
                %MOD;
15     for (int i=0; i<=r; i++)
16         konstanta[i] = BigModulo(n-i,r);
17     for (int i=0; i<=r; i++)
18         for (int j=r; j>i; j--)
               konstanta[j]=(konstanta[j-1]-konstanta[j])%MOD;
```

Kode Sumber 4.7 Implementasi Fungsi Calculation Bagian A

```
19      for (int i=r; i>=0; i--)
20          hasil2 = ((long long) (hasil2 +
                    konstanta[i])*inv1minA)%MOD;
21      hasil2 = ((long long)hasil2*(-BigModulo(a,n+1)))
                %MOD;
22      int hasil = (hasil1+hasil2) %MOD;
23      if (hasil<0) hasil+=MOD;
24      else if (hasil>=MOD) hasil-=MOD;
25      return hasil;
26  }
```

Kode Sumber 4.8 Implementasi Fungsi Calculation Bagian B

4.9 Implementasi Fungsi MultiplicationPolynomials

Fungsi MultiplicationPolynomials akan menghitung perkalian 2 polynomial. Penjelasan lebih detail mengenai algoritma *Fast Multiplication Polynomial* dapat dilihat pada subbab 2.4.2. Fungsi ini merupakan implementasi dari desain fungsi EulerPolynomials yang terdapat pada subbab 3.7. Implementasi dari Fungsi MultiplicationPolynomials dapat dilihat pada kode sumber 4.9, kode sumber 4.10 dan kode sumber 4.11.

```
1   long double ca1[MAXR*8], ca2[MAXR*8];
2   long double cb1[MAXR*8], cb2[MAXR*8];
3   long double cc11[MAXR*8];
4   long double cc12[MAXR*8];
5   long double cc22[MAXR*8];
6   void MultiplicationPolynomials(int*a,int*b,int*c,int size) {
7       int k=1;
8       while (k<size) k<<=1;
9       k<<=1;
```

Kode Sumber 4.9 Implementasi Fungsi MultiplicationPolynomials Bagian A

```
10    for (int i = 0; i <=(k<<1); i++) {
11        ca1[i] = cb1[i] = ca2[i] = cb2[i] = 0;
12    }
13    for (int i = 0; i <size; i++) {
14        ca1[(i<<1)+1] = (long double)(a[i]>>15);
15        cb1[(i<<1)+1] = (long double)(b[i]>>15);
16        ca2[(i<<1)+1] = (long double)(a[i]&32767);
17        cb2[(i<<1)+1] = (long double)(b[i]&32767);
18    }
19    FastFourierTransform(ca1, k, 1);
20    FastFourierTransform(cb1, k, 1);
21    FastFourierTransform(ca2, k, 1);
22    FastFourierTransform(cb2, k, 1);
23    for (int i = 0; i <k; i++) {
24        cc11[(i<<1)+1] = ca1[(i<<1)+1]*cb1[(i<<1)+1]
                           - ca1[(i<<1)+2]*cb1[(i<<1)+2];
25        cc11[(i<<1)+2] = ca1[(i<<1)+1]*cb1[(i<<1)+2]
                          + ca1[(i<<1)+2]*cb1[(i<<1)+1];
26        cc12[(i<<1)+1] = ca1[(i<<1)+1]*cb2[(i<<1)+1]
                           - ca1[(i<<1)+2]*cb2[(i<<1)+2]
                          + ca2[(i<<1)+1]*cb1[(i<<1)+1]
                           - ca2[(i<<1)+2]*cb1[(i<<1)+2];
27        cc12[(i<<1)+2] = ca1[(i<<1)+1]*cb2[(i<<1)+2]
                          + ca1[(i<<1)+2]*cb2[(i<<1)+1]
                          + ca2[(i<<1)+1]*cb1[(i<<1)+2]
                          + ca2[(i<<1)+2]*cb1[(i<<1)+1];
28        cc22[(i<<1)+1] = ca2[(i<<1)+1]*cb2[(i<<1)+1]
                           - ca2[(i<<1)+2]*cb2[(i<<1)+2];
29        cc22[(i<<1)+2] = ca2[(i<<1)+1]*cb2[(i<<1)+2]
30                        + ca2[(i<<1)+2]*cb2[(i<<1)+1];
31    }
```

Kode Sumber 4.10 Implementasi Fungsi MultiplicationPolynomials Bagian B

```
32      FastFourierTransform(cc11, k, -1);
33      FastFourierTransform(cc12, k, -1);
34      FastFourierTransform(cc22, k, -1);
35      long long c11,c12,c22;
36      for (int i = 0; i <=size; i++) {
37          c11 = ((long long)roundl(cc11[(i<<1)+1]/k))%MOD;
38          c12 = ((long long)roundl(cc12[(i<<1)+1]/k))%MOD;
39          c22 = ((long long)roundl(cc22[(i<<1)+1]/k))%MOD;
40          c[i] = ((c11<<30)+(c12<<15)+ c22)%MOD;
41      }
42  }
```

Kode Sumber 4.11 Implementasi Fungsi MultiplicationPolynomials Bagian C

4.10 Implementasi Fungsi Calculation Multiplication Polynomials

Fungsi Calculation akan menghitung nilai berdasarkan rumus (2.17). Untuk penjelasan lebih detail mengenai proses perhitungan rumus tersebut dapat dilihat pada subbab 2.4. Fungsi ini merupakan implementasi dari desain fungsi Calculate pada subbab 3.9. Implementasi dari Fungsi Calculation dapat dilihat pada kode sumber 4.12 dan kode sumber 4.13

```
1   int konstanta[MAXR+1];
2   int polyA[MAXR+1], polyB[MAXR+1];
3   int Calculate(int n, int a, int r) {
4       int hasil1=0, hasil2=0;
5       int inv1minA=InversModulo(1-a);
6       //Menghitung Bagian Kiri
7       hasil1=((long long)EulerPolynomials(a,r)*a)%MOD;
8       hasil1=((long long)hasil1
                            *BigModulo(inv1minA,r+1))%MOD;
```

Kode Sumber 4.12 Implementasi Fungsi Calculation

```
9       //Menghitung Bagian Kanan
10      for (int i=0; i<=r; i++) {
11          if (i&1)
12              polyA[i]=((long long)-BigModulo(n-i,r)
                                        *invFact[i])%MOD;
13          else
14              polyA[i]=((long long)BigModulo(n-i,r)
                                        *invFact[i])%MOD;
15          polyB[i]=invFact[i];
16      }
17      MultiplicationPolynomials(polyA,polyB,konstanta, r+1);
18      for (int i=0; i<=r; i++)
19          konstanta[i]=((long long)konstanta[i]
                                        *fact[i])%MOD;
20      for (int i=r; i>=0; i--)
21          hasil2=((long long)(hasil2 + konstanta [i])
22                                  *inv1minA)%MOD;
23      hasil2=((long long)hasil2
                            *(-BigModulo(a,n+1)))%MOD;
24      int hasil = (hasil1+hasil2) %MOD;
25      if (hasil<0) hasil+=MOD;
26      else if (hasil>=MOD) hasil-=MOD;
27      return hasil;
28  }
```

Kode Sumber 4.13 Implementasi Fungsi Calculation

4.11 Implementasi Fungsi Lagrange

Fungsi Lagrange akan melakukan *Interpolation Polynomial Lagrange* ke-*n*. Cara menghitung nilai dari *Interpolation Polynomial Lagrange* telah dijelaskan pada subbab 2.5.1. Fungsi ini merupakan implementasi dari desain fungsi Lagrange pada subbab 3.10. Implementasi dari Fungsi Lagrange dapat dilihat pada kode sumber 4.14.

```
1  int leftFact[1000005],rightFact[1000005];;
2  int Lagrange(int*f, int r, int n)
3  {
4      if (n<=r) return f[n]%MOD;
5      int hasil=0;
6      leftFact[0]=1;
7      rightFact[r]=1;
8      for (int i=0; i<r; i++)
9          leftFact[i+1]=((long long)leftFact[i]*(n-i))%MOD;
10     for (int i=r; i>0; i--)
11         rightFact[i-1]=((long long)rightFact[i]*(n-i))%MOD;
12     int tmp;
13     for (int i=0; i<=r; i++)
14     {
15         tmp=(((((long long)f[i]*leftFact[i])%MOD
                                    *rightFact[i])%MOD
                                    *invFact[i])%MOD
                                    *invFact[r-i])%MOD;
16         if ((r-i)&1)
17         {
18             hasil-=tmp;
19             if (hasil<0) hasil+=MOD;
20         }
21         else
22         {
23             hasil+=tmp;
24             if (hasil>=MOD) hasil-=MOD;
25         }
26     }
27     return hasil;
28 }
```

Kode Sumber 4.14 Implementasi Fungsi Lagrange

4.12 Implementasi Fungsi Calculation Interpolation Polynomial

Fungsi Calculation akan menghitung rumus (2.24). Proses perhitungan sudah dijelaskan pada subbab 2.5. Fungsi ini merupakan implementasi dari desain fungsi Calculate pada subbab 3.11. Implementasi dari Fungsi Calculation dapat dilihat pada Kode Sumber 4.15.

```
1  int poly[MAXR+1];
2  int Calculate(int n, int a, int r)
3  {
4      int inv1minA = InversModulo (1-a);
5      int invA = InversModulo(a);
6      int ApowN = BigModulo(a,n);
7      poly[0] = (((long long)-BigModulo(inv1minA,r+1)
                   *a)%MOD*EulerPolynomials(a,r))%MOD;
8      for (int i=1; i<=r; i++)
9          poly[i]=((long long)poly[i-1]*invA
                                  +power[i])%MOD;
10     int hasil = ((long long)ApowN*Lagrange(poly,r,n)
                                   -poly[0])%MOD;
11     if (hasil<0) hasil+=MOD;
12     return hasil;
13 }
```

Kode Sumber 4.15 Implementasi Fungsi Calculation

Halaman ini sengaja dikosongkan

BAB 5
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan tentang uji coba dan evaluasi dari implementasi sistem yang telah dilakukan pada bab 4.

5.1 Lingkungan Uji Coba

Lingkungan uji coba menggunakan sebuah komputer dengan spesifikasi perangkat lunak dan perangkat keras sebagai berikut:

- Perangkat Keras
  - Processor Intel Core i7-4700HQ CPU @ 2.40GHz.
  - Memory 3.88 GB
- Perangkat Lunak
  - Sistem operasi Windows 7 64-bit.
  - Integrated Development Environment Dev C++ 4.9.9.2

5.2 Skenario Uji Coba

Pada subbab ini akan dijelaskan skenario uji coba yang dilakukan. Skenario uji coba terdiri dari uji coba kebenaran dan uji coba kinerja. Uji coba kebenaran dilakukan dengan melakukan analisis penyelesaian sebuah contoh kasus dengan strategi penyelesaian yang telah dijelaskan pada subbab 2.2, 2.3, 2.4, dan 2.5. Uji coba kebenaran akan menggunakan kasus berikut ini:

```
1
6 5 4
```

Gambar 5.1 Contoh Kasus Uji Coba

Kasus diatas terdiri dari sebuah testcase. Tescase tersebut terdiri dari 3 nilai varibel n=6, a=5, dan r=4.

Uji coba kinerja dilakukan dengan membuat komparasi kinerja untuk melihat pengaruh batasan nilai variabel n, a, dan r terhadap pertumbuhan waktu dari strategi penyelesaian yang telah dijelaskan pada subbab 2.2, 2.3, 2.4, dan 2.5.

5.2.1 Uji Coba Kebenaran Naif

Berikut ini adalah analisis strategi penyelesaian yang telah dijelaskan pada subbab 2.2. Kasus yang digunakan dalam analisis ini dapat dilihat pada Gambar 5.1.

Secara garis besar metode ini akan menghitung rumus (2.1) secara langsung menggunakan *Looping* dan *BigModulo*. Pseudocode dari strategi penyelesaian ini dapat dilihat pada Gambar 2.1. Berikut ini contoh perhitungannya :

$$\sum_{i=1}^{n} a^i i^r = 5^1 1^4 + 5^2 2^4 + 5^3 3^4 + 5^4 4^4 + 5^5 5^4 + 5^6 6^4$$
$$= 22373655$$

Dari hasil analisis diatas menghasilkan keluaran yang sesuai. Setelah itu dilakukan juga uji kebenaran dengan mengumpulkan berkas kode sumber ke situs SPOJ. Hasil Uji kebenaran pada situs SPOJ ditunjukkan pada Gambar 5.6

Gambar 5.2 Hasil uji kebenaran pada situs SPOJ

Dari hasil uji coba yang telah dilakukan program mendapat umpan balik *Accepted*. Waktu yang dibutuhkan program adalah 0.95 detik dan memori yang dibutuhkan program adalah 2.7 MB.

5.2.2 Uji Coba Kebenaran Euler Polynomials

Berikut ini adalah analisis strategi penyelesaian yang telah dijelaskan pada subbab 2.3. Kasus yang digunakan dalam analisis ini dapat dilihat pada Gambar 5.1.

Secara garis besar metode ini akan membagi rumus (2.2) menjadi 2 bagian yang terdiri dari sigma bagian kiri dan sigma bagian kanan. Perhitungan kedua bagian, baik bagian kiri maupun bagian membutuhkan kompleksitas $O(r^2)$. Maka kesimpulannya kompleksitas total perhitungan adalah $O(r^2)$.

Langkah pertama adalah menghitung nilai sigma bagian kiri pada rumus (2.2). Rumus yang akan dihitung sebagai berikut :

$$\frac{1}{(1-a)^{r+1}} a \sum_{i=0}^{r-1} a^i \left\langle {r \atop i} \right\rangle$$

Dalam rumus sigma diatas dibutuhkan nilai dari *eulerian number* $\left\langle {r \atop 0} \right\rangle, \left\langle {r \atop 1} \right\rangle, \left\langle {r \atop 2} \right\rangle, \ldots, \left\langle {r \atop r-1} \right\rangle$. *Eulerian number* $\left\langle {r \atop 0} \right\rangle, \left\langle {r \atop 1} \right\rangle, \left\langle {r \atop 2} \right\rangle, \ldots, \left\langle {r \atop r-1} \right\rangle$ dapat dihitung dengan *recurrence relation* pada rumus (2.3). Ilustrasi perhitungan eulerian number dapat dilihat pada Gambar 5.3

$n \setminus k$	0	1	2	3	4	5	6	7
1	1							
2	1	1						
3	1	4	1					
4	1	11	11	1				
5	1	26	66	26	1			
6	1	57	302	302	57	1		
7	1	120	1191	2416	1191	120	1	
8	1	247	4293	15619	15619	4293	247	1

Gambar 5.3 Ilustrasi perhitungan eulerian number $\left\langle {n \atop k} \right\rangle$

Dapat dilihat pada Gambar 5.3 nilai *eulerian number* $\left\langle {n \atop k} \right\rangle$ untuk n=6 dan k=2 mempunyai nilai 302. Berikut ini merupakan contoh kalkulasi nilai *eulerian number* $\left\langle {n \atop k} \right\rangle$ untuk n=6 dan k=2 berdasarkan *recurrence relation* pada rumus (2.3) :

$$\left\langle {6 \atop 2} \right\rangle = (6-2)\left\langle {6-1 \atop 2-1} \right\rangle + (2+1)\left\langle {6-1 \atop 2} \right\rangle = 302$$

Setelah melakukan perhitungan nilai dari eulerian number tersebut, maka selanjutnya akan dilakukan perhitungan berikut ini :

$$\frac{1}{(1-a)^{r+1}} a \sum_{i=0}^{r-1} a^i \left\langle {r \atop i} \right\rangle = \frac{5}{(-4)^5}(5^0.1 + 5^1.11 + 5^2.11 + 5^3.1)$$

$$= \frac{5}{-1024}(1 + 55 + 275 + 125)$$

$$= -\frac{285}{128}$$

Langkah kedua adalah menghitung sigma bagian kanan pada rumus (2.2). Rumus yang akan dihitung sebagai berikut :

$$-a^{n+1} \sum_{i=0}^{r} \frac{1}{(1-a)^{i+1}} \sum_{j=0}^{i} (-1)^j \binom{i}{j} (n-j)^r$$

Terlihat pada rumus diatas terdapat fungsi $k(i)$ yang terdapat pada rumus (2.12). Rumus diatas membutuhkan nilai fungsi $k(i)$ mulai dari $k(0)$ s/d $k(r)$ pada rumus (2.12). Perhitungan fungsi $k(i)$ pada rumus (2.14) membutuhkan nilai $c(i,j)$. Nilai dari $c(i,j)$ dapat dihitung menggunakan *recurrence relation* pada rumus (2.13). Untuk menghitung Ilustrasi perhitungan $c(i,j)$ dapat dilihat pada Gambar 5.4

i/j	0	1	2	3	4
0	1296	625	256	81	16
1	671	369	175	65	
2	302	194	110		
3	108	84			
4	24				

Gambar 5.4 Ilustrasi perhitungan $c(i,j)$

Dapat dilihat pada Gambar 5.4 nilai $c(i,j)$ untuk *i*=1 dan *j*=2 mempunyai nilai 175. Berikut ini merupakan contoh kalkulasi nilai $c(i,j)$ untuk *i*=1 dan *j*=2 berdasarkan *recurrence relation* pada rumus (2.13) :

$$c(1,2) = c(1-1,2) - c(1-1,2+1) = 175$$

Setelah melakukan perhitungan nilai dari $c(i,j)$ dapat dihitung $k(i)$ berdasarkan rumus (2.14). Nilai fungsi $k(i)$ mulai dari $k(0)$ s/d $k(r)$ dapat dilihat pada Gambar 5.5.

i	0	1	2	3	4
$k(i)$	1296	625	256	81	16

Gambar 5.5 Hasil perhitungan $k(i)$

Setelah nilai fungsi $k(i)$ mulai dari $k(0)$ s/d $k(r)$ yang diperlukan dalam rumus tersebut dihitung, maka selanjutnya akan dilakukan perhitungan berikut ini :

$$-a^{n+1}\sum_{i=0}^{r}\frac{1}{(1-a)^{i+1}}k(i)$$

$$= -5^7\left(\frac{1296}{(-4)^1}+\frac{671}{(-4)^2}+\frac{302}{(-4)^3}+\frac{108}{(-4)^4}+\frac{24}{(-4)^5}\right)$$

$$= -78125\left(\frac{1296}{-4}+\frac{671}{16}+\frac{302}{-64}+\frac{108}{256}+\frac{24}{-1024}\right)$$

$$= -78125\left(\frac{-331776 + 42944 - 4832 + 432 - 24}{1024}\right)$$

$$= -78125\left(\frac{-293256}{1024}\right)$$

$$= 78125\left(\frac{36657}{128}\right)$$

$$= \frac{2863828125}{128}$$

Hasil dari rumus (2.2) merupakan penjumlahan sigma bagian kiri dan sigma bagian kanan. Maka hasilnya sebagai berikut :

$$\sum_{i=1}^{n} a^i i^r = -\frac{285}{128} + \frac{2863828125}{128} = 22373655$$

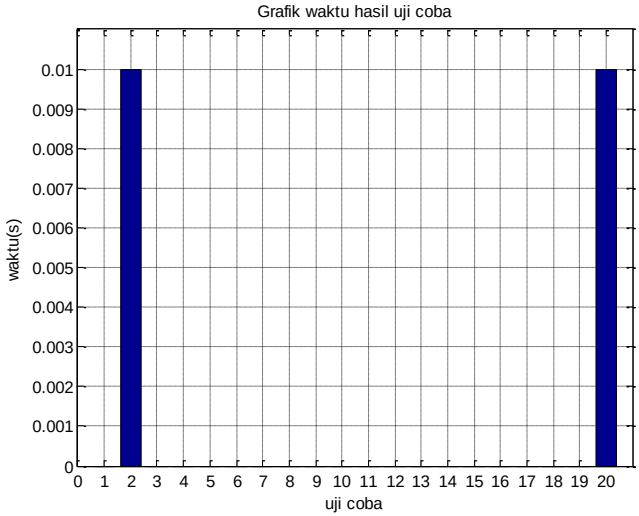
Dari hasil analisis diatas menghasilkan keluaran yang sesuai. Setelah itu dilakukan juga uji kebenaran dengan mengumpulkan berkas kode sumber ke situs SPOJ. Hasil Uji kebenaran pada situs SPOJ ditunjukkan pada Gambar 5.6

18025798	2016-10-26 06:09:44	Moon Safari (medium)	**accepted**	0.00	3.4M	C++ 4.3.2

Gambar 5.6 Hasil uji kebenaran pada situs SPOJ

Dari hasil uji coba yang telah dilakukan program mendapat umpan balik *Accepted*. Waktu yang dibutuhkan program adalah 0.0 detik dan memori yang dibutuhkan program adalah 3.4 MB.

Grafik hasil uji coba pengumpulan sebanyak 20 kali ditunjukkan pada Gambar 5.7. Dari hasil pengumpulan sebanyak 20 kali, didapat waktu rata-rata program yaitu 0.00 detik dan penggunaan memori rata-rata yang dibutuhkan program yaitu 3.4 MB



Gambar 5.7 Grafik Hasil uji pada situs SPOJ sebanyak 20 kali

5.2.3 Uji Coba Kebenaran Multiplication Polynomials

Berikut ini adalah analisis strategi penyelesaian yang telah dijelaskan pada subbab 2.4. Kasus yang digunakan dalam analisis ini dapat dilihat pada Gambar 5.1.

Secara garis besar metode ini akan membagi rumus (2.17) menjadi 2 bagian yang terdiri dari bagian kiri dan bagian kanan. Perhitungan kedua bagian baik bagian kiri maupun bagian membutuhkan kompleksitas $O(r \log r)$. Maka kesimpulannya kompleksitas total perhitungan adalah $O(r \log r)$.

Langkah pertama adalah menghitung nilai bagian kiri pada rumus (2.17). Rumus yang akan dihitung sebagai berikut :

$$\frac{1}{(1-a)^{r+1}} a\, S_r(a)$$

Dalam rumus sigma diatas dibutuhkan nilai dari *eulerian polynomials* $S_r(a)$. Perhitungan tersebut akan menggunakan rumus (2.8). Untuk menghitung $S_r(a)$ maka diperlukan perhitungan $f(k)$ pada rumus (2.8) terlebih dahulu. Nilai $f(k)$ yang dibutuhkan pada rumus (2.8) yaitu $f(1)$ s/d $f(r)$. Perhitungan $f(k)$ mulai dari $f(1)$ s/d $f(r)$ dapat dihitung dengan *recurrence relation* pada rumus (2.9). Ilustrasi perhitungan $f(k)$ dapat dilihat pada Gambar 5.8.

k	$f(k)$
0	0
1	$\frac{0+1^4}{5}=\frac{1}{5}$
2	$\frac{\frac{1}{5}+2^4}{5}=\frac{81}{25}$
3	$\frac{\frac{81}{25}+3^4}{5}=\frac{2106}{125}$
4	$\frac{\frac{2106}{125}+4^4}{5}=\frac{34106}{625}$

Gambar 5.8 Ilustrasi perhitungan $f(k)$

Setelah melakukan perhitungan nilai dari $f(k)$, maka selanjutnya akan dilakukan perhitungan berikut ini :

$$S_r(a) = (-a)^r \sum_{k=1}^{r} (-1)^k \binom{r+1}{k+1} f(k)$$

$$= (-5)^4 \left(-10 \times \frac{1}{5} + 10 \times \frac{81}{25} - 5 \times \frac{2106}{125} + 1 \times \frac{34106}{625}\right)$$

$$= 625 \left(\frac{-1250 + 20250 - 52650 + 34106}{625}\right)$$

$$= 456$$

Setelah perhitungan nilai dari $S_r(a)$, maka selanjutnya akan dilakukan perhitungan berikut ini :

$$\frac{1}{(1-a)^{r+1}} a\, S_r(a) = \frac{1}{(-4)^5} \times 5 \times 456$$

$$= -\frac{285}{128}$$

Langkah kedua adalah menghitung bagian kanan pada rumus (2.17). Rumus yang akan dihitung sebagai berikut :

$$-a^{n+1} \sum_{i=0}^{r} \frac{1}{(1-a)^{i+1}} k(i,n,r)$$

Terlihat pada rumus diatas terdapat fungsi $k(i,n,r)$. fungsi $k(i,n,r)$ yang dibutuhkan nilainya dalam rumus tersebut adalah $k(0,n,r)$, $k(1,n,r)$, …, $k(r,n,r)$. Perhitungan fungsi $k(i,n,r)$ tersebut dapat dihitung dengan menggunakan rumus (2.18). perhitungan polinomial C yang akan digunakan dalam rumus

(2.18) merupakan hasil perkalian 2 polynomial A dan B dengan masing masing koefisien sebagai berikut :

$$a_j = \frac{(-1)^j (n-j)^r}{j!} \qquad b_j = \frac{1}{j!}$$

Perkalian dua polynomial diatas akan menggunakan *Fast Multiplication Polynomial* yang di dalamnya menngunakan *Fast Fourier Transform* untuk mentransformasi *coefficient representation* menjadi *point-value representation*. Hasil transformasi dari *coefficient representation* menjadi *point-value representation* pada polynomial A digambarkan pada Gambar 5.9.

j	a_j	A_j		
0	1296	786.166667	+	0.000000 i
1	-625	803.918734	+	160.473184 i
2	128	862.937537	+	323.487680 i
3	-13.5	978.785560	+	482.415480 i
4	0.666667	1168.666667	+	611.500000 i
5	0	1432.195104	+	662.101483 i
6	0	1727.729130	+	579.487680 i
7	0	1969.100602	+	342.825854 i
8	0	2063.166667	+	0.000000 i
9	0	1969.100602	-	342.825854 i
10	0	1727.729130	-	579.487680 i
11	0	1432.195104	-	662.101483 i
12	0	1168.666667	-	611.500000 i
13	0	978.785560	-	482.415480 i
14	0	862.937537	-	323.487680 i
15	0	803.918734	-	160.473184 i

Gambar 5.9 Transformasi *coefficient representation* menjadi *point-value representation* pada polinomial A

Hasil transformasi dari *coefficient representation* menjadi *point-value representation* pada polynomial B digambarkan pada Gambar 5.10

j	b_j	B_j		
0	1	2.708333	+	0.000000 i
1	1	2.341213	-	0.931883 i
2	0.5	1.547589	-	1.324958 i
3	0.166667	0.875150	-	1.171986 i
4	0.041667	0.541667	-	0.833333 i
5	0	0.417743	-	0.548212 i
6	0	0.369078	-	0.324958 i
7	0	0.365893	-	0.141443 i
8	0	0.375000	+	0.000000 i
9	0	0.365893	+	0.141443 i
10	0	0.369078	+	0.324958 i
11	0	0.417743	+	0.548212 i
12	0	0.541667	+	0.833333 i
13	0	0.875150	+	1.171986 i
14	0	1.547589	+	1.324958 i
15	0	2.341213	+	0.931883 i

Gambar 5.10 Transformasi *coefficient representation* menjadi *point-value representation* pada polinomial B

Setelah mendapatkan *point-value representation* dari polinomial A dan B yang menghasilkan bilangan complex maka dilakukan perkalian kedua *point-value representation* tersebut. Hasil dari perkalian 2 polynomial tersebut dapat dilihat pada Gambar 5.10.

Dari hasil perkalian 2 polynomial diatas akan dilakukan proses *Invers Fast Fourier Transform* yang mentransformasi dari *point-value representation* menjadi *coefficient representation*. Hasil transformasi dari *point-value representation* menjadi *coefficient representation* pada polynomial C digambarkan pada Gambar 5.11

j	C_j			c_j	$k(j,n,r)$
0	2.708333	+	0.000000 i	1296.00	1296
1	2.341213	-	0.931883 i	671.00	671
2	1.547589	-	1.324958 i	151.00	302
3	0.875150	-	1.171986 i	18.00	108
4	0.541667	-	0.833333 i	1.00	24
5	0.417743	-	0.548212 i	-10.79	
6	0.369078	-	0.324958 i	3.42	
7	0.365893	-	0.141443 i	-0.45	
8	0.375000	+	0.000000 i	0.03	
9	0.365893	+	0.141443 i	0.00	
10	0.369078	+	0.324958 i	0.00	
11	0.417743	+	0.548212 i	0.00	
12	0.541667	+	0.833333 i	0.00	
13	0.875150	+	1.171986 i	0.00	
14	1.547589	+	1.324958 i	0.00	
15	2.341213	+	0.931883 i	0.00	

Gambar 5.11 Transformasi *point-value representation* menjadi *coefficient representation* pada polinomial C

Nilai dari $k(i,n,r)$ pada Gambar 5.11 merupakan hasil kalkulasi dengan menggunakan rumus (2.18). Setelah nilai dari $k(i,n,r)$ mulai dari $k(0,n,r)$, $k(1,n,r)$, …, $k(r,n,r)$ yang diperlukan dalam rumus dihitung, maka selanjutnya nilai akan dilakukan perhitungan berikut ini :

$$-a^{n+1}\sum_{i=0}^{r}\frac{1}{(1-a)^{i+1}}k(i,n,r)$$

$$= -5^{7}\left(\frac{1296}{(-4)^{1}}+\frac{671}{(-4)^{2}}+\frac{151}{(-4)^{3}}+\frac{108}{(-4)^{4}}+\frac{24}{(-4)^{5}}\right)$$

$$= -78125\left(\frac{1296}{-4} + \frac{671}{16} + \frac{302}{-64} + \frac{108}{256} + \frac{24}{-1024}\right)$$

$$= -78125\left(\frac{-331776 + 42944 - 4832 + 432 - 24}{1024}\right)$$

$$= -78125\left(\frac{-293256}{1024}\right)$$

$$= 78125\left(\frac{36657}{128}\right)$$

$$= \frac{2863828125}{128}$$

Hasil dari rumus (2.17) merupakan penjumlahan bagian kiri dan bagian kanan. Maka hasilnya sebagai berikut :

$$\sum_{i=1}^{n} a^i i^r = -\frac{285}{128} + \frac{2863828125}{128} = 22373655$$

Dari hasil analisis diatas menghasilkan keluaran yang sesuai. Setelah itu dilakukan juga uji kebenaran dengan mengumpulkan berkas kode sumber ke situs SPOJ. Hasil Uji kebenaran pada situs SPOJ ditunjukkan pada Gambar 5.12

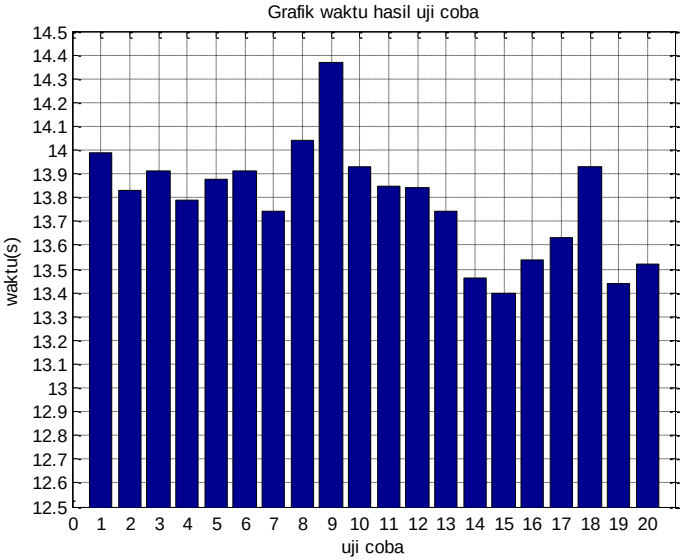
18025814	2016-10-24 06:11:55	Moon Safari (Hard)	**accepted**	13.99	670M	C++ 4.3.2

Gambar 5.12 Hasil uji kebenaran pada situs SPOJ

Dari hasil uji coba yang telah dilakukan program mendapat umpan balik *Accepted*. Waktu yang dibutuhkan program adalah 13.99 detik dan memori yang dibutuhkan program adalah 670 MB.

Grafik hasil uji coba pengumpulan sebanyak 20 kali ditunjukkan pada Gambar 5.13. Dari hasil pengumpulan sebanyak 20 kali,

didapat waktu rata-rata program yaitu 13.79 detik dan penggunaan memori rata-rata yang dibutuhkan program yaitu 670 MB



Gambar 5.13 Grafik Hasil uji pada situs SPOJ sebanyak 20 kali

5.2.4 Uji Coba Kebenaran Interpolation Polynomials

Berikut ini adalah analisis strategi penyelesaian yang telah dijelaskan pada subbab 2.5. Kasus yang digunakan dalam analisis ini dapat dilihat pada Gambar 5.1.

Secara garis besar metode ini akan membagi rumus (2.24) menjadi 2 bagian yang terdiri dari bagian kiri dan bagian kanan. Perhitungan kedua bagian baik bagian kiri maupun bagian membutuhkan kompleksitas $O(r \log r)$. Maka kesimpulannya kompleksitas total perhitungan adalah $O(r \log r)$.

Langkah pertama adalah menghitung nilai bagian kiri pada rumus (2.24). Rumus yang akan dihitung sebagai berikut :

$$\frac{1}{(1-a)^{r+1}} a\, S_r(a)$$

Analisis perhitungan rumus diatas telah dibahas pada subbab 5.2.3. Langkah kedua adalah menghitung bagian kanan pada rumus (2.24). Rumus yang akan dihitung sebagai berikut :

$$poly(n)a^n$$

Untuk menghitung rumus diatas dibutuhkan perhitungan $poly(n)$, maka akan dilakukan interpolasi yang membutuhkan nilai dari $poly(0)$ s/d $poly(r)$. Perhitungan $poly(0)$ s/d $poly(r)$. dapat dihitung dengan menggunakan rumus (2.22). Ilustrasi perhitungan $poly(0)$ s/d $poly(r)$ dapat dilihat pada Gambar 5.14

n	$\frac{1}{a^n}$	$\sum_{i=1}^{n} a^i i^r$	$\frac{1}{(1-a)^{r+1}} a\, S_r(a)$	$poly(n)$
0	$\frac{1}{5^0}$	0	$-\frac{285}{128}$	2.2265625
1	$\frac{1}{5^1}$	5	$-\frac{285}{128}$	1.4453125
2	$\frac{1}{5^2}$	405	$-\frac{285}{128}$	16.2890625
3	$\frac{1}{5^3}$	10530	$-\frac{285}{128}$	84.2578125
4	$\frac{1}{5^4}$	170530	$-\frac{285}{128}$	272.8515625

Gambar 5.14 Perhitungan $poly(0)$ s/d $poly(r)$

Setelah melakukan perhitungan nilai dari $poly(0)$ s/d $poly(r)$, maka akan dicari nilai $poly(n)$. Jika nilai n lebih besar daripada r, maka akan dilakukan interpolasi polynomial unutk mencari nilai $poly(n)$. Ilustrasi interpolasi polynomial dapat dilihat pada Gambar 5.15.

j	$poly(n)$	$p_j(n) = \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{n - x_m}{x_j - x_m}$	$poly(j)\, p_j(n)$
0	2.2265625	$\frac{5}{-1} + \frac{4}{-2} + \frac{3}{-3} + \frac{2}{-4} = 5$	11.1328125
1	1.4453125	$\frac{6}{1} + \frac{4}{-1} + \frac{3}{-2} + \frac{2}{-3} = -24$	-34.6875000
2	16.2890625	$\frac{6}{2} + \frac{5}{1} + \frac{3}{-1} + \frac{2}{-2} = 45$	733.0078125
3	84.2578125	$\frac{6}{3} + \frac{5}{2} + \frac{4}{1} + \frac{2}{-1} = -40$	-3370.3125000
4	272.8515625	$\frac{6}{4} + \frac{5}{3} + \frac{4}{2} + \frac{3}{1} = 15$	4092.7734375
$poly(n) = \sum_{j=0}^{r} poly(j)\, p_j(n)$			1431.9140625

Gambar 5.15 Interpolation Polynomial $poly(n)$

Setelah menghitung nilai $poly(n)$ maka hasil dari rumus (2.24) sebagai berikut :

$$\sum_{i=1}^{n} a^i i^r = -\frac{285}{128} + 5^6 \times 1431.9140625 = 22373655$$

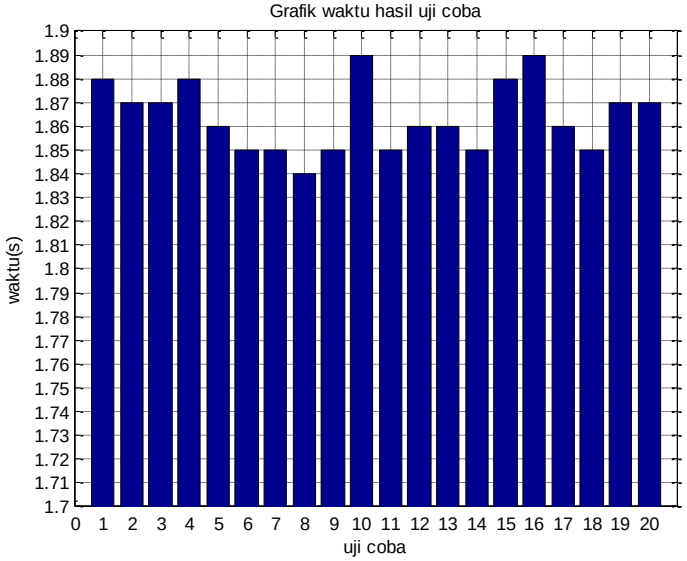
Dari hasil analisis diatas menghasilkan keluaran yang sesuai. Setelah itu dilakukan juga uji kebenaran dengan mengumpulkan berkas kode sumber ke situs SPOJ. Hasil Uji kebenaran pada situs SPOJ ditunjukkan pada Gambar 5.16

18025812	2016-10-24 06:11:40	Moon Safari (Hard)	**accepted**	1.88	29M	C++ 4.3.2

Gambar 5.16 Hasil uji kebenaran pada situs SPOJ

Dari hasil uji coba yang telah dilakukan program mendapat umpan balik *Accepted*. Waktu yang dibutuhkan program adalah 1.88 detik dan memori yang dibutuhkan program adalah 29 MB.

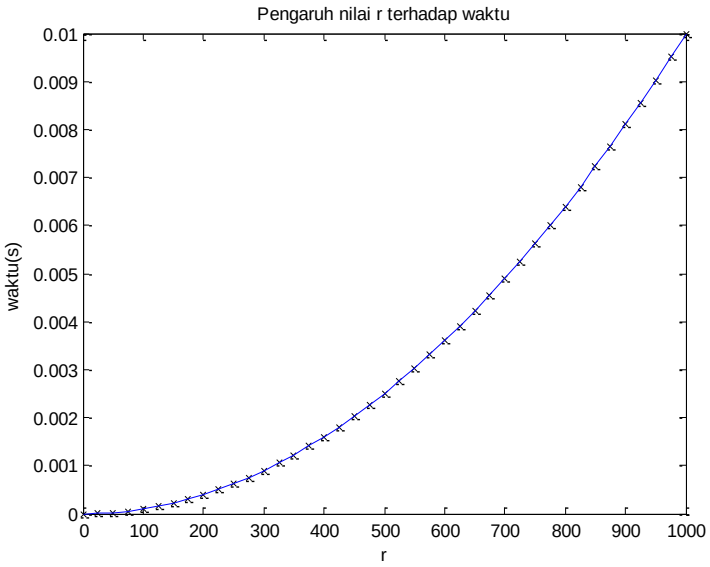
Grafik hasil uji coba pengumpulan sebanyak 20 kali ditunjukkan pada Gambar 5.17. Dari hasil pengumpulan sebanyak 20 kali, didapat waktu rata-rata program yaitu 1.86 detik dan penggunaan memori rata-rata yang dibutuhkan program yaitu 29 MB



Gambar 5.17 Grafik Hasil uji pada situs SPOJ sebanyak 20 kali

5.2.5 Uji Coba Kinerja Euler Polynomial

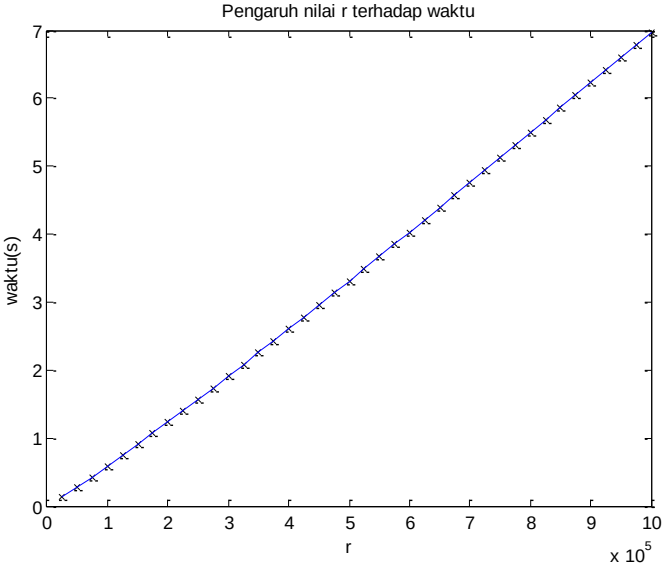
Untuk setiap kasus terdapat 3 parameter bilangan bulat yaitu variabel n, a, dan r. Dari penjelasan strategi penyelesaian pada subbab 2.3 diketahui bahwa kompleksitas waktu program adalah $O(r^2)$. Dari kompleksitas ini dapat disimpulkan kecepatan program dan lama waktu eksekusi program hanya dipengaruhi oleh besarnya nilai variabel r. Semakin besar nilai r maka semakin besar pula lama waktu eksekusi program. Untuk lebih jelas mengenai pengaruh nilai r terhadap waktu untuk Strategi penyelesaian pada subbab 2.3 dapat dilihat pada Gambar 5.18. Dengan kinerja ini maka solusi ini hanya mampu menyelesaikan permasalahan Moon Safari (Medium). Solusi ini masih kurang efektif untuk permasalahan Moon Safari (Hard).



Gambar 5.18 Grafik pengaruh nilai r terhadap waktu pada strategi penyelesaian Eulerian Polynomial

5.2.6 Uji Coba Kinerja Multiplication Polynomial

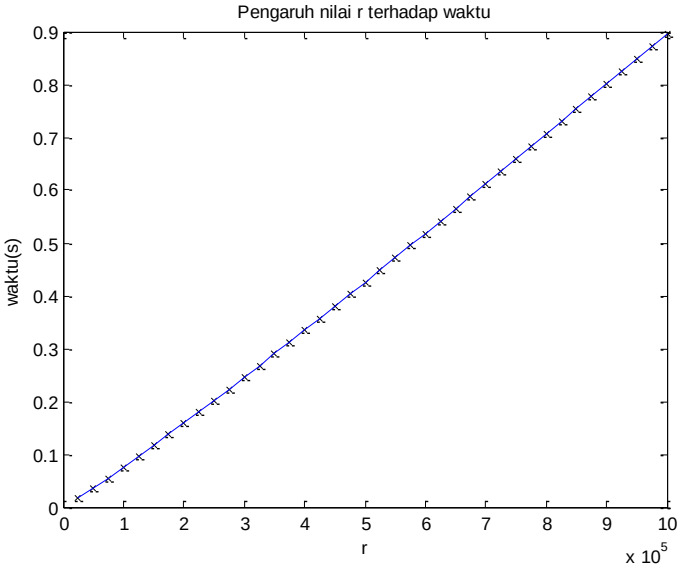
Untuk setiap kasus terdapat 3 parameter bilangan bulat yaitu variabel n, a, dan r. Dari penjelasan strategi penyelesaian pada subbab 2.4 diketahui bahwa kompleksitas waktu program adalah $O(r \log r)$. Dari kompleksitas ini dapat disimpulkan kecepatan program dan lama waktu eksekusi program hanya dipengaruhi oleh besarnya nilai variabel r. Semakin besar nilai r maka semakin besar pula lama waktu eksekusi program. Untuk lebih jelas mengenai pengaruh nilai r terhadap waktu untuk Strategi penyelesaian pada subbab 2.4 dapat dilihat pada Gambar 5.19. Dengan kinerja ini maka solusi ini mampu menyelesaikan permasalahan Moon Safari (Medium). Solusi ini juga masih kurang efektif untuk permasalahan Moon Safari (Hard).



Gambar 5.19 Grafik pengaruh nilai r terhadap waktu pada strategi penyelesaian Multiplication Polynomial

5.2.7 Uji Coba Kinerja Interpolation Polynomial

Untuk setiap kasus terdapat 3 parameter bilangan bulat yaitu variabel n, a, dan r. Dari penjelasan strategi penyelesaian pada subbab 2.5 diketahui bahwa kompleksitas waktu program adalah $O(r \log r)$. Dari kompleksitas ini dapat disimpulkan kecepatan program dan lama waktu eksekusi program hanya dipengaruhi oleh besarnya nilai variabel r. Semakin besar nilai r maka semakin besar pula lama waktu eksekusi program. Untuk lebih jelas mengenai pengaruh nilai r terhadap waktu untuk Strategi penyelesaian pada subbab 2.5 dapat dilihat pada Gambar 5.20. Dengan kinerja ini maka solusi ini mampu menyelesaikan permasalahan Moon Safari (Medium) dan Moon Safari (Hard).



Gambar 5.20 Grafik pengaruh nilai r terhadap waktu pada strategi penyelesaian Interpolation Polynomial

BAB 6
KESIMPULAN

6.1 Kesimpulan

Penyelesaian masalah perhitungan formula ini dapat diselesaikan dengan berbagai solusi disesuaikan dengan batasan dari permasalahan. Berdasarkan tahapan-tahapan yang telah dilakukan antara lain analisis, desain, implementasi dan uji coba maka didapatkan kesimpulan untuk masing masing permasalahan sebagai berikut :

1. Moon Safari (Easy)
   Karena permasalahan mempunya batasan nilai bilangan bulat n yang relatif kecil walaupun batasan nilai bilangan bulat r yang relatif besar maka dengan pendekatan *brute force* permasalahan batasan ini sudah dapat diselesaikan dengan cukup efisien. Kompleksitas akhir program yang dibutuhkan dengan pendekatan ini $O(n \log r)$.
2. Moon Safari (Medium)
   Karena permasalahan ini mempunyai batasan nilai bilangan bulat n yang relatif besar maka pendekatan *brute force* sudah tidak cukup efisien untuk menyelesaikan permasalahan ini. Maka dibutuhkan strategi baru yaitu dengan merubah formula pada permsalahan menjadi formula baru yang dapat dikerjakan dengan lebih cepat. Dengan melakukan pendekatan *Eulerian Polynomial* dihasilkan formula baru yang dapat dihitung dengan kompleksitas akhir program $O(r^2)$
3. Moon Safari (Hard)
   Karena permasalahan ini mempunyai batasan nilai r lebih besar daripada masalah sebelumnya maka perlu dilakukan optimasi lagi dengan melakukan pendekatan lain. Ada dua pendekatan yang dapat digunakan yaitu *Multiplication Polynomial* dan *Interpolation Polynomial*. Kedua pendekatan ini membutuhkan kompleksitas akhir program

$O(r \log r)$. Walaupun memiliki kompleksitas yang sama pendekatan *Interpolation Polynomial* relatif lebih cepat dibandingkan dengan pendekatan *Multiplication Polynomial*. Hal ini disebabkan adanya operasi bilangan kompleks pada pendekatan *Multiplication Polynomial*.

DAFTAR PUSTAKA

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms Third Edition.* Cambridge: The MIT Press.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C The Art of Scientific Computing Second Edition.* Cambridge: Cambridge University Press.

Weisstein, E. W. (1999). *CRC Concise Encyclopedia of Mathematics.* Charlottesvilles: CRC Press LLC.

Halaman ini sengaja dikosongkan

Lampiran A

Pembuktian Rumus

Berikut ini merupakan rumus yang akan digunakan untuk menyelesaikan permasalahan SPOJ MOON Safari

$$f(n,a,r) = \frac{1}{(1-a)^{r+1}} a \sum_{i=0}^{r-1} a^i \left\langle {r \atop i} \right\rangle - a^{n+1} \sum_{i=0}^{r} \frac{1}{(1-a)^{i+1}} \sum_{j=0}^{i} (-1)^j \binom{i}{j} (n-j)^r$$

Pembuktian :

$$f(n,a,0) = \sum_{i=1}^{n} a^i i^0 = \sum_{i=1}^{n} a^i = \frac{a(1-a^n)}{1-a} = \frac{a-a^{n+1}}{1-a}$$

Rumus diatas merupakan rumus deret geometri. Untuk menemukan $f(n,a,r)$ dimana r>0 maka dibutuhkan rumus $f(x,a,0)$. Dimana x bernilai mulai dari 1 sampai n.

Berikut ini penjabaran rumus $f(n,a,1)$:

$$f(n,a,1) = \sum_{i=1}^{n} a^i i^1$$

Persamaan awal

$$f(n,a,1) = \sum_{i=1}^{n} a^i + (2-1)\sum_{i=2}^{n} a^i + (3-2)\sum_{i=3}^{n} a^i + \cdots + (n-(n-1))\sum_{i=n}^{n} a^i$$

Dipecah menjadi beberapa a^i sehingga dapat digunakan rumus geometri

$$f(n,a,1) = \frac{a-a^{n+1}}{1-a} + \frac{a^2-a^{n+1}}{1-a} + \frac{a^3-a^{n+1}}{1-a} + \cdots + \frac{a^n-a^{n+1}}{1-a}$$

Mensubtitusi sigma dengan rumus deret geometri

$$f(n,a,1) = \frac{a+a^2+a^3+\cdots+a^n-na^{n+1}}{1-a}$$

Menyamakan penyebut

$$f(n,a,1) = \frac{\sum_{i=1}^{n} a^i - na^{n+1}}{1-a}$$

Mengubah ke dalam bentuk sigma persamaan $a+a^2+a^3+\cdots+a^n$

$$f(n,a,1) = \frac{\frac{a-a^{n+1}}{1-a} - na^{n+1}}{1-a}$$

Mensubtitusi sigma dengan rumus deret geometri

$$f(n,a,1) = \frac{a-a^{n+1}-(1-a)^1 na^{n+1}}{(1-a)^2}$$

Mengubah penyebut menjadi $(1-a)^2$

Berikut ini penjabaran rumus $f(n,a,2)$:

$$f(n,a,2) = \sum_{i=1}^{n} a^i i^2$$

Persamaan awal

$$f(n,a,2) = \sum_{i=1}^{n} a^i + (2^2-1^2)\sum_{i=2}^{n} a^i + (3^2-2^2)\sum_{i=3}^{n} a^i + \cdots + (n^2-(n-1)^2)\sum_{i=n}^{n} a^i$$

Dipecah menjadi beberapa a^i sehingga dapat digunakan rumus deret geometri untuk perhitungannya

$$f(n,a,2) = \frac{a-a^{n+1}}{1-a} + \frac{3(a^2-a^{n+1})}{1-a} + \frac{5(a^3-a^{n+1})}{1-a} + \cdots + \frac{(2n-1)(a^n-a^{n+1})}{1-a}$$

Mensubtitusi sigma dengan rumus deret geometri

$$f(n,a,2) = \frac{a+3a^2+5a^3+\cdots+(2n-1)a^n-(1+3+5+\cdots+2n-1)a^{n+1}}{1-a}$$

Menyamakan penyebut

$$f(n,a,2) = \frac{a+3a^2+5a^3+\cdots+(2n-1)a^n-n^2a^{n+1}}{1-a}$$

Mengubah penjumlahan $(1+3+5+\cdots+2n-1)$ yang awalnya merupakan n^2

$$f(n,a,2) = \frac{\sum_{i=1}^{n} a^i + (3-1)\sum_{i=2}^{n} a^i + (5-3)\sum_{i=3}^{n} a^i + \cdots + ((2n-1)-(2(n-1)-1)\sum_{i=n}^{n} a^i - n^2a^{n+1}}{1-a}$$

Mengubah ke dalam bentuk sigma persamaan $a+3a^2+5a^3+\cdots+(2n-1)a^n$

$$f(n,a,2) = \frac{\frac{a-a^{n+1}}{1-a} + \frac{2(a^2-a^{n+1})}{1-a} + \frac{2(a^3-a^{n+1})}{1-a} + \cdots + \frac{2(a^n-a^{n+1})}{1-a} - n^2a^{n+1}}{1-a}$$

Mensubtitusi sigma dengan rumus deret geometri

$$f(n,a,2) = \frac{\frac{a+2a^2+2a^3+\cdots+2a^n-(1+2+2+\cdots+2)a^{n+1}}{1-a} - n^2a^{n+1}}{1-a}$$

Menyamakan penyebut

$$f(n,a,2) = \frac{\frac{a+2a^2+2a^3+\cdots+2a^n-(2n-1)a^{n+1}}{1-a} - n^2a^{n+1}}{1-a}$$

Mengubah penjumlahan $(1+2+2+\cdots+2)$ yang awalnya merupakan $(2n-1)$

$$f(n,a,2) = \frac{a + 2a^2 + 2a^3 + \cdots + 2a^n - (2n-1)a^{n+1} - (1-a)^1 n^2 a^{n+1}}{(1-a)^2}$$

Mengubah penyebut menjadi $(1-a)^2$

$$f(n,a,2) = \frac{\sum_{i=1}^{n} a^i + \sum_{i=2}^{n} a^i - (2n-1)a^{n+1} - (1-a)^1 n^2 a^{n+1}}{(1-a)^2}$$

Mengubah $a + 2a^2 + 2a^3 + \cdots + 2a^n$ kedalam bentuk sigma

$$f(n,a,2) = \frac{\frac{a - a^{n+1}}{1-a} + \frac{a^2 - a^{n+1}}{1-a} - (2n-1)a^{n+1} - (1-a)^1 n^2 a^{n+1}}{(1-a)^2}$$

Mensubtitusi sigma dengan rumus deret geometri

$$f(n,a,2) = \frac{\frac{a^1 + a^2 - (1+1)\,a^{n+1}}{1-a} - (2n-1)a^{n+1} - (1-a)^1 n^2 a^{n+1}}{(1-a)^2}$$

Menyamakan penyebut

$$f(n,a,2) = \frac{a^1 + a^2 - 2a^{n+1} - (1-a)^1(2n-1)a^{n+1} - (1-a)^2 n^2 a^{n+1}}{(1-a)^3}$$

Mengubah penyebut menjadi $(1-a)^3$

Berikut ini penjabaran rumus $f(n,a,3)$:

$$f(n,a,3) = \sum_{i=1}^{n} a^i i^3$$

Persamaan awal

$$f(n,a,3) = \sum_{i=1}^{n} a^i + (2^3 - 1^3)\sum_{i=2}^{n} a^i + (3^3 - 2^3)\sum_{i=3}^{n} a^i + \cdots + (n^3 - (n-1)^3)\sum_{i=n}^{n} a^i$$

Dipecah menjadi beberapa a^i sehingga dapat digunakan rumus deret geometri untuk perhitungannya

$$f(n,a,3) = \frac{a - a^{n+1}}{1-a} + \frac{7(a^2 - a^{n+1})}{1-a} + \frac{19(a^3 - a^{n+1})}{1-a} + \cdots + \frac{(3n^2 - 3n + 1)(a^n - a^{n+1})}{1-a}$$

Mensubtitusi sigma dengan rumus deret geometri

$$f(n,a,3) = \frac{a + 7a^2 + 19a^3 + \cdots + (3n^2 - 3n + 1)a^n - (1 + 7 + 19 + \cdots + (3n^2 - 3n + 1))a^{n+1}}{1-a}$$

Menyamakan penyebut

$$f(n,a,3) = \frac{a + 7a^2 + 19a^3 + \cdots + (3n^2 - 3n + 1)a^n - n^3 a^{n+1}}{1-a}$$

Mengubah penjumlahan $1 + 7 + 19 + \cdots + (3n^2 - 3n + 1)$ menjadi n^3

$$f(n,a,3) = \frac{\sum_{i=1}^{n} a^i + 6\sum_{i=2}^{n} a^i + 12\sum_{i=3}^{n} a^i + \cdots + ((3n^2 - 3n + 1) - (3(n-1)^2 - 3(n-1) + 1))\sum_{i=n}^{n} a^i - n^3 a^{n+1}}{1-a}$$

Mengubah ke dalam bentuk sigma persamaan $a + 7a^2 + 19a^3 + \cdots + (3n^2 - 3n + 1)a^n$

$$f(n,a,3) = \frac{\frac{a - a^{n+1}}{1-a} + \frac{6(a^2 - a^{n+1})}{1-a} + \frac{12(a^3 - a^{n+1})}{1-a} + \cdots + \frac{(6n-6)(a^n - a^{n+1})}{1-a} - n^3 a^{n+1}}{1-a}$$

Mensubtitusi sigma dengan rumus deret geometri

$$f(n,a,3) = \frac{\frac{a + 6a^2 + 12a^3 + \cdots + (6n-6)a^n - (1 + 6 + 12 + \cdots + (6n-6))a^{n+1}}{1-a} - n^3 a^{n+1}}{1-a}$$

Menyamakan penyebut

$$f(n,a,3) = \frac{\frac{a + 6a^2 + 12a^3 + \cdots + (6n-6)a^n - (3n^2 - 3n + 1)a^{n+1}}{1-a} - n^3 a^{n+1}}{1-a}$$

Mengubah penjumlahan $1 + 6 + 12 + \cdots + (6n-6)$ menjadi $(3n^2 - 3n + 1)$

$$f(n,a,3) = \frac{a + 6a^2 + 12a^3 + \cdots + (6n-6)a^n - (3n^2 - 3n + 1)a^{n+1} - (1-a)^1 n^3 a^{n+1}}{(1-a)^2}$$

Mengubah penyebut menjadi $(1-a)^2$

$$f(n,a,3) = \frac{\sum_{i=1}^{n} a^i + 5\sum_{i=2}^{n} a^i + 6\sum_{i=3}^{n} a^i + \cdots + 6\sum_{i=n}^{n} a^i - (3n^2 - 3n + 1)a^{n+1} - (1-a)^1 n^3 a^{n+1}}{(1-a)^2}$$

Mengubah ke dalam bentuk sigma persamaan $a + 6a^2 + 12a^3 + \cdots + (6n-6)a^n$

$$f(n,a,3) = \frac{\frac{a-a^{n+1}}{1-a} + \frac{5(a^2-a^{n+1})}{1-a} + \frac{6(a^3-a^{n+1})}{1-a} + \cdots + \frac{6(a^n-a^{n+1})}{1-a} - (3n^2 - 3n + 1)a^{n+1} - (1-a)^1 n^3 a^{n+1}}{(1-a)^2}$$

Mensubtitusi sigma dengan rumus deret geometri

$$f(n,a,3) = \frac{\frac{a + 5a^2 + 6a^3 + \cdots + 6a^n - (1 + 5 + 6 + \cdots + 6)a^{n+1}}{1-a} - (3n^2 - 3n + 1)a^{n+1} - (1-a)^1 n^3 a^{n+1}}{(1-a)^2}$$

Menyamakan penyebut

$$f(n,a,3) = \frac{\frac{a + 5a^2 + 6a^3 + \cdots + 6a^n - (6n-6)a^{n+1}}{1-a} - (3n^2 - 3n + 1)a^{n+1} - (1-a)^1 n^3 a^{n+1}}{(1-a)^2}$$

Mengubah penjumlahan $1 + 5 + 6 + \cdots + 6$ menjadi $(6n-6)$

$$f(n,a,3) = \frac{a + 5a^2 + 6a^3 + \cdots + 6a^n - (6n-6)a^{n+1} - (1-a)^1(3n^2 - 3n + 1)a^{n+1} - (1-a)^2 n^3 a^{n+1}}{(1-a)^3}$$

Mengubah penyebut menjadi $(1-a)^3$

$$f(n,a,3) = \frac{\sum_{i=1}^{n} a^i + 4\sum_{i=2}^{n} a^i + 1\sum_{i=3}^{n} a^i - (6n-6)a^{n+1} - (1-a)^1(3n^2 - 3n + 1)a^{n+1} - (1-a)^2 n^3 a^{n+1}}{(1-a)^3}$$

Mengubah ke dalam bentuk sigma persamaan $a + 5a^2 + 6a^3 + \cdots + 6a^n$

$$f(n,a,3) = \frac{\frac{a - a^{n+1}}{1-a} + \frac{4(a^2 - a^{n+1})}{1-a} + \frac{a^3 - a^{n+1}}{1-a} - (6n-6)a^{n+1} - (1-a)^1(3n^2 - 3n + 1)a^{n+1} - (1-a)^2 n^3 a^{n+1}}{(1-a)^3}$$

Mensubtitusi sigma dengan rumus deret geometri

$$f(n,a,3) = \frac{\frac{a + 4a^2 + a^3 - (1+4+1)a^{n+1}}{1-a} - (6n-6)a^{n+1} - (1-a)^1(3n^2 - 3n + 1)a^{n+1} - (1-a)^2 n^3 a^{n+1}}{(1-a)^3}$$

Menyamakan penyebut

$$f(n,a,3) = \frac{a + 4a^2 + a^3 - 6a^{n+1} - (1-a)^1(6n-6)a^{n+1} - (1-a)^2(3n^2 - 3n + 1)a^{n+1} - (1-a)^3 n^3 a^{n+1}}{(1-a)^4}$$

Mengubah penyebut menjadi $(1-a)^4$

Dari penjabaran rumus diatas dapat disimpulkan berdasarkan polanya sebagai berikut :

$$f(n,a,r) = \frac{1}{(1-a)^{r+1}} a \sum_{i=0}^{r-1} a^i \left\langle {r \atop i} \right\rangle - \frac{a^{n+1}}{(1-a)^{r+1}} \sum_{i=0}^{r} (1-a)^i \sum_{j=0}^{r-i} (-1)^j \binom{r-i}{j} (n-j)^r$$

$$f(n,a,r) = \frac{1}{(1-a)^{r+1}} a \sum_{i=0}^{r-1} a^i \left\langle {r \atop i} \right\rangle - a^{n+1} \sum_{i=0}^{r} (1-a)^{i-r-1} \sum_{j=0}^{r-i} (-1)^j \binom{r-i}{j} (n-j)^r$$

$$f(n,a,r) = \frac{1}{(1-a)^{r+1}} a \sum_{i=0}^{r-1} a^i \left\langle {r \atop i} \right\rangle - a^{n+1} \sum_{i=0}^{r} (1-a)^{(r-i)-r-1} \sum_{j=0}^{r-(r-i)} (-1)^j \binom{r-(r-i)}{j} (n-j)^r$$

$$f(n,a,r) = \frac{1}{(1-a)^{r+1}} a \sum_{i=0}^{r-1} a^i \left\langle {r \atop i} \right\rangle - a^{n+1} \sum_{i=0}^{r} (1-a)^{-i-1} \sum_{j=0}^{i} (-1)^j \binom{i}{j} (n-j)^r$$

$$f(n,a,r) = \frac{1}{(1-a)^{r+1}} a \sum_{i=0}^{r-1} a^i \left\langle {r \atop i} \right\rangle - a^{n+1} \sum_{i=0}^{r} \frac{1}{(1-a)^{i+1}} \sum_{j=0}^{i} (-1)^j \binom{i}{j} (n-j)^r$$

Lampiran B
Hasil Uji Coba Pada Situs SPOJ Sebanyak 20 Kali Menggunakan Strategi Eulerian Polynomial

Berikut ini merupakan lampiran hasil uji coba pengumpulan berkas sumber kode solusi pada situs penilaian daring SPOJ sebanyak 20 kali dengan menggunakan strategi *Eulerian polynomial*.

18063065	☐	2016-10-30 17:26:23	Moon Safari (medium)	**accepted** edit ideone it	0.01	3.4M	C++ 4.3.2
18062995	☐	2016-10-30 17:16:01	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18062876	☐	2016-10-30 16:57:41	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18062733	☐	2016-10-30 16:37:08	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18062247	☐	2016-10-30 15:16:47	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18059285	☐	2016-10-30 06:04:52	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18059169	☐	2016-10-30 05:34:16	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18059050	☐	2016-10-30 04:52:20	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18053109	☐	2016-10-29 11:53:56	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18051367	☐	2016-10-29 06:47:18	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2

Gambar B.1 Hasil uji coba pada situs SPOJ menggunakan strategi eulerian polynomial(1)

18050719	☐	2016-10-29 03:48:25	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18050598	☐	2016-10-29 02:57:20	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18046069	☐	2016-10-28 13:17:31	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18045607	☐	2016-10-28 11:55:15	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18043524	☐	2016-10-28 06:46:15	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18034889	☐	2016-10-27 06:49:12	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18034850	☐	2016-10-27 06:41:46	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18033889	☐	2016-10-27 02:45:03	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2
18030733	☐	2016-10-26 17:23:13	Moon Safari (medium)	**accepted** edit ideone it	0.01	3.4M	C++ 4.3.2
18025798	☐	2016-10-26 06:09:44	Moon Safari (medium)	**accepted** edit ideone it	0.00	3.4M	C++ 4.3.2

Gambar B.2 Hasil uji coba pada situs SPOJ menggunakan strategi eulerian polynomial(2)

Lampiran C
Hasil Uji Coba Pada Situs SPOJ Sebanyak 20 Kali Menggunakan Strategi Multiplication Polynomial

Berikut ini merupakan lampiran hasil uji coba pengumpulan berkas sumber kode solusi pada situs penilaian daring SPOJ sebanyak 20 kali dengan menggunakan strategi *Multiplication polynomial*.

18063082		2016-10-30 17:28:19	Moon Safari (Hard)	**accepted** edit ideone it	13.52	670M	C++ 4.3.2
18062993		2016-10-30 17:15:59	Moon Safari (Hard)	**accepted** edit ideone it	13.44	670M	C++ 4.3.2
18062873		2016-10-30 16:57:38	Moon Safari (Hard)	**accepted** edit ideone it	13.93	670M	C++ 4.3.2
18062731		2016-10-30 16:37:06	Moon Safari (Hard)	**accepted** edit ideone it	13.63	670M	C++ 4.3.2
18059286		2016-10-30 06:04:56	Moon Safari (Hard)	**accepted** edit ideone it	13.54	670M	C++ 4.3.2
18059170		2016-10-30 05:34:18	Moon Safari (Hard)	**accepted** edit ideone it	13.40	670M	C++ 4.3.2
18059049		2016-10-30 04:52:14	Moon Safari (Hard)	**accepted** edit ideone it	13.46	670M	C++ 4.3.2
18053107		2016-10-29 11:53:53	Moon Safari (Hard)	**accepted** edit ideone it	13.74	670M	C++ 4.3.2
18051371		2016-10-29 06:47:24	Moon Safari (Hard)	**accepted** edit ideone it	13.84	670M	C++ 4.3.2
18050721		2016-10-29 03:48:40	Moon Safari (Hard)	**accepted** edit ideone it	13.85	670M	C++ 4.3.2

Gambar C.1 Hasil uji coba pada situs SPOJ menggunakan strategi multiplication polynomial(1)

18050597	☐	2016-10-29 02:57:14	Moon Safari (Hard)	**accepted** edit ideone it	13.93	670M	C++ 4.3.2
18046071	☐	2016-10-28 13:17:43	Moon Safari (Hard)	**accepted** edit ideone it	14.37	670M	C++ 4.3.2
18045605	☐	2016-10-28 11:55:03	Moon Safari (Hard)	**accepted** edit ideone it	14.04	670M	C++ 4.3.2
18043536	☐	2016-10-28 06:48:06	Moon Safari (Hard)	**accepted** edit ideone it	13.74	670M	C++ 4.3.2
18043528	☐	2016-10-28 06:46:52	Moon Safari (Hard)	**accepted** edit ideone it	13.91	670M	C++ 4.3.2
18034896	☐	2016-10-27 06:50:11	Moon Safari (Hard)	**accepted** edit ideone it	13.88	670M	C++ 4.3.2
18034865	☐	2016-10-27 06:44:20	Moon Safari (Hard)	**accepted** edit ideone it	13.79	670M	C++ 4.3.2
18033896	☐	2016-10-27 02:46:03	Moon Safari (Hard)	**accepted** edit ideone it	13.91	670M	C++ 4.3.2
18030743	☐	2016-10-26 17:24:18	Moon Safari (Hard)	**accepted** edit ideone it	13.83	670M	C++ 4.3.2
18025814	☐	2016-10-26 06:11:55	Moon Safari (Hard)	**accepted** edit ideone it	13.99	670M	C++ 4.3.2

Gambar C.2 Hasil uji coba pada situs SPOJ menggunakan strategi multiplication polynomial(2)

Lampiran D
Hasil Uji Coba Pada Situs SPOJ Sebanyak 20 Kali Menggunakan Strategi Interpolation Polynomial

Berikut ini merupakan lampiran hasil uji coba pengumpulan berkas sumber kode solusi pada situs penilaian daring SPOJ sebanyak 20 kali dengan menggunakan strategi *Interpolation polynomial*.

18063085	☐	2016-10-30 17:28:23	Moon Safari (Hard)	**accepted** edit ideone it	1.87	29M	C++ 4.3.2
18062990	☐	2016-10-30 17:15:56	Moon Safari (Hard)	**accepted** edit ideone it	1.87	29M	C++ 4.3.2
18062874	☐	2016-10-30 16:57:38	Moon Safari (Hard)	**accepted** edit ideone it	1.85	29M	C++ 4.3.2
18062246	☐	2016-10-30 15:16:43	Moon Safari (Hard)	**accepted** edit ideone it	1.86	29M	C++ 4.3.2
18059288	☐	2016-10-30 06:05:01	Moon Safari (Hard)	**accepted** edit ideone it	1.89	29M	C++ 4.3.2
18059172	☐	2016-10-30 05:34:22	Moon Safari (Hard)	**accepted** edit ideone it	1.88	29M	C++ 4.3.2
18059047	☐	2016-10-30 04:52:05	Moon Safari (Hard)	**accepted** edit ideone it	1.85	29M	C++ 4.3.2
18053103	☐	2016-10-29 11:53:42	Moon Safari (Hard)	**accepted** edit ideone it	1.86	29M	C++ 4.3.2
18051370	☐	2016-10-29 06:47:22	Moon Safari (Hard)	**accepted** edit ideone it	1.86	29M	C++ 4.3.2
18050722	☐	2016-10-29 03:48:41	Moon Safari (Hard)	**accepted** edit ideone it	1.85	29M	C++ 4.3.2

Gambar D.1 Hasil uji coba pada situs SPOJ menggunakan strategi interpolation polynomial(1)

18050599	☐	2016-10-29 02:57:49	Moon Safari (Hard)	**accepted** edit ideone it	1.89	29M	C++ 4.3.2
18046072	☐	2016-10-28 13:17:46	Moon Safari (Hard)	**accepted** edit ideone it	1.85	29M	C++ 4.3.2
18045606	☐	2016-10-28 11:55:05	Moon Safari (Hard)	**accepted** edit ideone it	1.84	29M	C++ 4.3.2
18043537	☐	2016-10-28 06:48:08	Moon Safari (Hard)	**accepted** edit ideone it	1.85	29M	C++ 4.3.2
18043526	☐	2016-10-28 06:46:29	Moon Safari (Hard)	**accepted** edit ideone it	1.85	29M	C++ 4.3.2
18034893	☐	2016-10-27 06:49:37	Moon Safari (Hard)	**accepted** edit ideone it	1.86	29M	C++ 4.3.2
18034864	☐	2016-10-27 06:44:00	Moon Safari (Hard)	**accepted** edit ideone it	1.88	29M	C++ 4.3.2
18033893	☐	2016-10-27 02:45:42	Moon Safari (Hard)	**accepted** edit ideone it	1.87	29M	C++ 4.3.2
18030734	☐	2016-10-26 17:23:36	Moon Safari (Hard)	**accepted** edit ideone it	1.87	29M	C++ 4.3.2
18025812	☐	2016-10-26 06:11:40	Moon Safari (Hard)	**accepted** edit ideone it	1.88	29M	C++ 4.3.2

Gambar D.2 Hasil uji coba pada situs SPOJ menggunakan strategi interpolation polynomial(2)

BIODATA PENULIS

Anton Kristanto, lahir di Surabaya tanggal 17 September 1994. Penulis merupakan anak kedua dari 2 bersaudara. Penulis telah menempuh pendidikan formal TK Permai Surabaya, SDK Santa Clara Surabaya (2000-2006), SMPK Santa Clara Surabaya (2006-2009), dan SMAK Santa Maria Surabaya (2009-2012). Penulis melanjutkan studi kuliah program sarjana di Jurusan Teknik Informatika ITS.

Selama kuliah di Teknik Informatika ITS, penulis mengambil bidang minat Dasar dan Terapan Komputasi (DTK). Penulis pernah menjadi asisten dosen dan praktikum untuk mata kuliah Pemrograman Terstruktur (2013 dan 2014) dan Algoritma dan Struktur Data (2013). Selama menempuh perkuliahan penulis juga aktif mengikuti kompetisi pemrograman tingkat nasional dan menjadi finalis pada lomba pemrograman COMPFEST UI (2013, dan 2014), INC Bina Nusantara (2013, 2014, dan 2015) dan ICPC Regional Asia-Jakarta (2013, 2014, dan 2015). Penulis dapat dihubungi melalui surel di antonkristanto94@gmail.com