ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - IF184802

IMPLEMENTASI REDUKSI POLIGON MENGGUNAKAN ALGORITMA MELKMAN CONVEX HULL YANG DIMODIFIKASI DENGAN STUDI KASUS LL AND ERBAO(ISUN1) PADA SPHERE ONLINE JUDGE

MICHAEL JULIAN ALBERTUS
NRP 05111640000097

Dosen Pembimbing 1
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing 2
Yudhi Purwananto, S.Kom., M.Kom.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2019

[*Halaman ini sengaja dikosongkan*]

TUGAS AKHIR - IF184802

IMPLEMENTASI REDUKSI POLIGON MENGGUNAKAN ALGORITMA MELKMAN CONVEX HULL YANG DIMODIFIKASI DENGAN STUDI KASUS LL AND ERBAO(ISUN1) PADA SPHERE ONLINE JUDGE

MICHAEL JULIAN ALBERTUS
NRP 05111640000097

Dosen Pembimbing 1
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing 2
Yudhi Purwananto, S.Kom., M.Kom.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2019

[*Halaman ini sengaja dikosongkan*]

UNDERGRADUATE THESES - IF184802

IMPLEMENTATION OF POLYGON REDUCTION USING MODIFIED MELKMAN CONVEX HULL ALGORITHM WITH CASE STUDY LL AND ERBAO FROM SPHERE ONLINE JUDGE

MICHAEL JULIAN ALBERTUS
NRP 05111640000097

Supervisor 1
Rully Soelaiman, S.Kom., M.Kom.

Supervisor 2
Yudhi Purwananto, S.Kom., M.Kom.

INFORMATICS DEPARTMENT
Faculty of Information Technology and Communication
Institut Teknologi Sepuluh Nopember
Surabaya, 2019

[*Halaman ini sengaja dikosongkan*]

LEMBAR PENGESAHAN

IMPLEMENTASI REDUKSI POLIGON MENGGUNAKAN ALGORITMA MELKMAN CONVEX HULL YANG DIMODIFIKASI DENGAN STUDI KASUS LL AND ERBAO(ISUN1) PADA SPHERE ONLINE JUDGE

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Algoritma Pemrograman
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
Michael Julian Albertus
NRP. 05111640000097

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Rully Soelaiman, S.Kom., M.Kom. .....

NIP. 19700213199402100 (Pembimbing 1)

Yudhi Purwananto, S.Kom., M.Kom. .....

NIP. 197007141997031002 (Pembimbing 2)

SURABAYA
KAPAN

[*Halaman ini sengaja dikosongkan*]

ABSTRAK

IMPLEMENTASI REDUKSI POLIGON MENGGUNAKAN ALGORITMA MELKMAN CONVEX HULL YANG DIMODIFIKASI DENGAN STUDI KASUS LL AND ERBAO(ISUN1) PADA SPHERE ONLINE JUDGE

Nama	:	Michael Julian Albertus
NRP	:	05111640000097
Departemen	:	Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, ITS
Pembimbing I	:	Rully Soelaiman, S.Kom., M.Kom.
Pembimbing II	:	Yudhi Purwananto, S.Kom., M.Kom.

Abstrak

[TBA]

Kata Kunci: geometri; convex hull; melkman algorithm; relative poligon;

[*Halaman ini sengaja dikosongkan*]

ABSTRACT

IMPLEMENTATION OF POLYGON REDUCTION USING MODIFIED MELKMAN CONVEX HULL ALGORITHM WITH CASE STUDY LL AND ERBAO FROM SPHERE ONLINE JUDGE

Name	:	Michael Julian Albertus
Student ID	:	05111640000097
Department	:	Informatics Department, Faculty of Information Technology and Communication, ITS
Supervisor I	:	Rully Soelaiman, S.Kom., M.Kom.
Supervisor II	:	Yudhi Purwananto, S.Kom., M.Kom.

Abstract

[TBA]

Keywords: geometry; convex hull; melkman algorithm; relative poligon;

[*Halaman ini sengaja dikosongkan*]

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa. Atas rahmat dan kasih sayangNya, penulis dapat menyelesaikan tugas akhir dan laporan akhir dalam bentuk buku ini.

Pengerjaan buku ini penulis tujukan untuk mengeksplorasi lebih mendalam topik-topik yang tidak diwadahi oleh kampus, namun banyak menarik perhatian penulis. Selain itu besar harapan penulis bahwa pengerjaan tugas akhir sekaligus pengerjaan buku ini dapat menjadi batu loncatan penulis dalam menimba ilmu yang bermanfaat.

Penulis ingin menyampaikan rasa terima kasih kepada banyak pihak yang telah membimbing, menemani dan membantu penulis selama masa pengerjaan tugas akhir maupun masa studi.

1. Bapak Rully Soelaiman S.Kom.,M.Kom., selaku pembimbing penulis. Ucapan terima kasih juga penulis sampaikan atas segala perhatian, didikan, pengajaran, dan nasihat yang telah diberikan oleh beliau selama masa studi penulis.

Penulis menyadari bahwa buku ini jauh dari kata sempurna. Maka dari itu, penulis memohon maaf apabila terdapat salah kata maupun makna pada buku ini. Akhir kata, penulis mempersembahkan buku ini sebagai wujud nyata kontribusi penulis dalam ilmu pengetahuan.

Surabaya, KAPAN

Michael Julian Albertus

[*Halaman ini sengaja dikosongkan*]

DAFTAR ISI

DAFTAR GAMBAR

[*Halaman ini sengaja dikosongkan*]

DAFTAR TABEL

[*Halaman ini sengaja dikosongkan*]

DAFTAR PSEUDOCODE

DAFTAR KODE SUMBER

[*Halaman ini sengaja dikosongkan*]

DAFTAR NOTASI

$\mathbb{Z}$	Himpunan bilangan bulat.
$\mathbb{Z}_n$	Himpunan bilangan bulat positif hingga n eksklusif.
$\mathbb{Z}/p\mathbb{Z}$	Himpunan kongruensi dalam modulo p, dimana p merupakan bilangan prima dalam *multiplicative group*.
$\phi(n)$	*Euler Totient Function* atau *Euler Phi*. Menotasikan banyaknya nilai yang koprima dengan n.
$R[x]/R$	Himpunan *ring* yang merupakan struktur aljabar bilangan pada pertambahan dan perkalian.

[*Halaman ini sengaja dikosongkan*]

BAB I
DASAR TEORI

Pada bab ini, akan dijelaskan dasar teori yang digunakan sebagai landasaan pengerjaan Tugas Akhir ini.

1.1. Deskripsi Permasalahan

Permasalahan yang dibahas pada Tugas Akhir ini adalah perhitungan untuk mencari nilai x yang didefinisikan oleh persamaan (1.1).

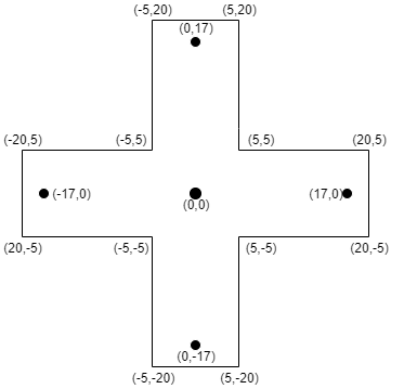
$$x = \sum_{i=0}^{n-1} \mathrm{RCH}_i \tag{1.1}$$

RCH_i pada persamaan (1.1) menyatakan sisi polygon dari RCH yang merupakan *relative convex hull* yang didapatkan dari sekumpulan titik yang dibatasi di dalam polygon sederhana[1]. Permasalahan pada tugas akhir ini adalah mencari *relative convex hull* dari sekumpulan titik yang dibatasi oleh polygon sederhana. Gambar 1.1 dan 1.2 merupakan contoh dari permasalahan ISUN1.

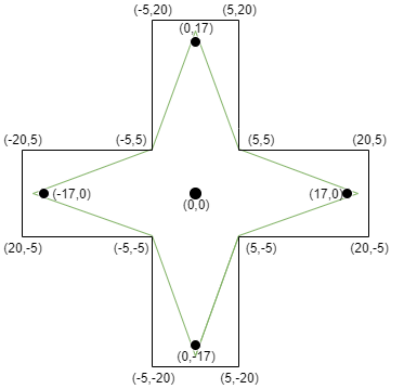
1.2. Convex Polygon

Merupakan sebuah polygon sederhana yang memiliki sudut maksimal 180 derajat pada tiap edgenya. *Convex polygon* memiliki beberapa properti, yaitu:

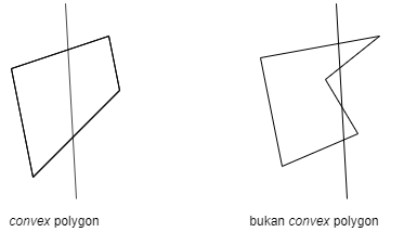
1. sebuah garis lurus yang di gambar melewati sebuah *convex* polygon akan berpotongan maksimal 2 kali. Ilustrasi dapat dilihan pada gambar 1.3
2. Jika dua titik sembarang diambil dan ditarik garis antara keduanya, tidak ada bagian dari garis yang berada di luar polygon. Ilustrasi dapat dilihan pada gambar 1.4



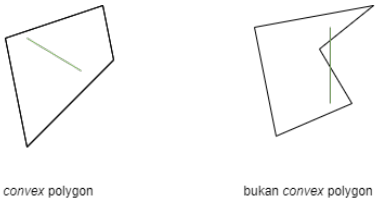
Gambar 1.1: Ilustrasi contoh kasus tanpa solusi



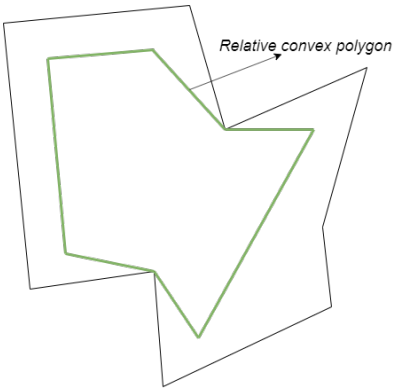
Gambar 1.2: Ilustrasi contoh kasus



Gambar 1.3: Ilustrasi Properti Convex Polygon 1



Gambar 1.4: Ilustrasi Properti Convex Polygon 2



Gambar 1.5: Ilustrasi Relative Convex Polygon

1.2.1. Relative Convex Polygon

Merupakan penurunan dari convex Polygon tetapi ada beberapa sisi dari polygon tersebut berbentuk convace atau cekung kedalam dikarenakan adanya batasan dari luar seperti polygon atau segmen garis lainnya. Ilustrasi relative convex polygon dapat dilihat pada gambar 1.5.

1.3. Strategi Penyelesaian Permasalahan

Pada subbab ini akan dipaparkan mengenai strategi penyelesaian masalah klasik pada daring SPOJ dengan kode ISUN1 menggunakan algoritma reduksi poligon. Secara singkat, strategi penye-

lesaian masalah dari ISUN1 menggunakan algoritma reduksin poligon menjadi 2 bagian besar yaitu :

1. Pemrosesan titik pembentuk polygon yang membentuk *Convex*.
2. *Convex Hull* dari titik yang berada didalam polygon

Sebagai contoh, pada subbab ini akan menggunakan P sebagai poligon luar yang mempunyai n vertex, dimana $P = \langle p_1, p_2, ..., p_n \rangle$ yang mempunyai titik sebanyak m ($S = \langle s_1, s_2, ..., s_m \rangle$), dan $D(A)$ merupakan sebuah deque (*doubly-ended queue*) yang menampung vertex dari polygon P. Reduksi polygon didasari dari algoritma Melkman dengan sedikit modifikasi. Modifikasi yang dilakukan adalah ketika 3 buak titik pembentuk poligon yang konsekutif membuat *convex* maka titik tengan dari ketiga titik tersebut dibuang, dan jika *concave* maka titik tengahnya tetap disimpan. Pada saat sebuah titik dibuang, maka luas dari polygon akan tereduksi. Langkah - langkah reduksi dilakukan dengan mengulangi 2 langkah yang akan dijelaskan pada subbab 1.3.1 dan 1.3.2

1.3.1. Pemrosesan Titik Pembentuk Polygon yang Membentuk Convex

Pemrosesan titik pembentuk poligon dapat dilakukan dengan cara melakukan *traversing* terhadap semua vertex pembentuk poligon. Untuk setiap vertex p_i yang di cek, hitung orientasi(secara berlawanan jarum jam) titik p_i dengan p_{i-1} dan p_{i+1}. Jika orientasinya membentuk *convex* maka titik p_i akan dibuang.

Sebelum membuang titik p_i, kita akan membuat sebuah segitiga ABC dimana $A = p_i$, $B = p_{i-1}$, dan $C = p_{i+1}$ karena triangulation of polygon(Teorema 1).

Teorema 1 (Triangulation of Polygon) *Semua polygon dapat di buat dari beberapa segitiga.*

Kemudian cari $T(ABC)$ dimana $T(ABC)$ merupakan semua titik S yang berada di dalam segitiga ABC dengan menggunakan algoritma *Point inside Polygon* (dapat dilihat pada subbab 1.5). Pencarian titik yang berada di dalam segitiga ABC berguna untuk mencari pengganti vertex p_i sebagai pembentuk poligon luarnya.

1.3.2. Convex Hull dari Titik yang Berada di Dalam Polygon

Melanjutkan dari subbab 1.3.1, ketika sudah mendapatkan T, lakukan pencarian *Convex Hull* dari titik - titik tersebut menggunakan *monotone chain* (dapat dilihat pada subbab 1.4.2.2). Kemudian sisipkan semua titik yang membentuk *Convex Hull* diantara vertex p_{i-1}, p_{i+1} untuk me-rekonstruksi poligon luar yang sudah di reduksi.

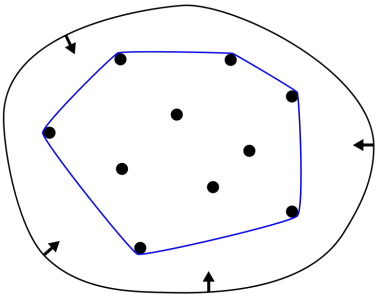
1.4. Convex Hull

Convex Hull dari sekumpulan titik S adalah sebuah set dari semua kombinasi *convex* dari titik - titik tersebut. Setiap titik s_i pada S diberikan sebuah koefisien a_i dimana a_i merupakan bialangan non negatif dan jika semua a_i dijumlahkan hasilnya satu. Dan koefisien ini digunakan untuk menghitung berat rata - rata untuk setiap titik. Untuk setiap koefisien yang dipilih akan dikombinasikan dan menghasilkan *convex hull*. Set *convex hull* ini dapat di ekspresikan dengan formula (1.2) dan ilustrasi *convex hull* ada pada gambar 1.6.

$$Conv(S) = \left\{ \sum_{i=1}^{|S|} a_i s_i \Big| (\forall i : a_i \geq 0 \wedge \sum_{i=1}^{|S|} a_i = 1) \right\} \tag{1.2}$$

1.4.1. Relative Convex Hull

Relative convex hull merupakan penurunan dari *convex hull*. *Relative convex hull* merupakan *convex hull* yang mempunyai



Gambar 1.6: Ilustrasi Convex Cull

cavity (cekungan kedalam) yang diakibatkan atau relatif terhadap sesuatu yang membatasi *convex hull tersebut*. ilustrasi *relative convex hull* dapat dilihat pada gambar 1.7.

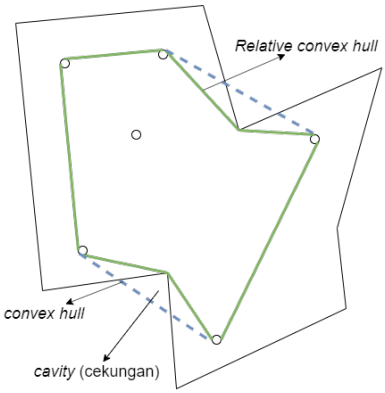
Penentuan untuk mengetahui sebuah polygon merupakan *convex* atau *concave* dapat menggunakan orientasi. Apabila orientasi dari tiga titik yang berurutan adalah positif berlawanan jarum jam maka tiga titik tersebut adalah *convex*. Sebaliknya apabila negatif maka tiga titik tersebut adalah *concave*. Untuk mencari orientasi antara tiga titik dapat digunakan persamaan 1.3.

$$\begin{aligned} \vec{u} &= (B_x - A_x)x + (B_y - A_y)y \\ \vec{v} &= (C_x - A_x)x + (C_y - A_y)y \end{aligned} \tag{1.3}$$

$$Orientasi = u_x * v_y - u_y * v_x$$

1.4.2. Algoritma Convex Hull

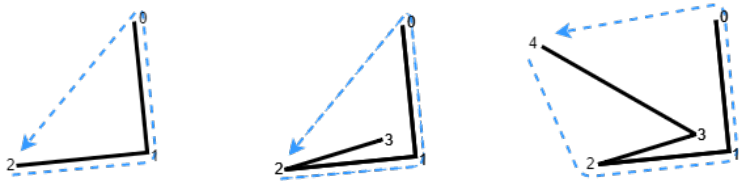
Ada beberapa algoritma yang dapat digunakan untuk mencari sebuah *convex hull*, untuk melihat perbandingan dari beberapa algoritma dapat dilihat pada tabel 1.1. Berdasarkan tabel 1.1, penulis memilih 2 algoritma yang akan digunakan pada buku ini.



Gambar 1.7: Ilustrasi Relative Convex Hull

Tabel 1.1: Tabel Perbandingan Algoritma Convex Hull

Algoritma Convex Hull	Implementasi	Kompleksitas	Kode Sumber	Jenis Input
Jarvis's Algorithm	Mudah	$\mathcal{O}(n^2)$	Singkat	Kumpulan Titik
Graham's Scan	Sedikit Mudah	$\mathcal{O}(n \log(n))$	Singkat	Kumpulan Titik
Quick Hull	Kompleks	$\mathcal{O}(n \log(n))$	Panjang	Kumpulan Titik
Monotone Chain	Mudah	$\mathcal{O}(n \log(n))$	Singkat	Kumpulan Titik
Melkman's Algorithm	Mudah	$\mathcal{O}(n)$	Singkat	Poligon Sederhana



Gambar 1.8: Ilustrasi Algoritma Melkman

1.4.2.1. Algoritma Melkman Convex Hull

Merupakan algoritma untuk menghitung rantai polygonal ataupun polygon sederhana dengan waktu linear $\mathcal{O}(n)$[2]. Asumsikan sebuah poligon sederhana P, dengan vertex p_i dan edge $p_i p_{i+1}$. Algoritma ini menggunakan deque, $D = \langle d_1, d_2, ..., d_n \rangle$, untuk mereprentasikan *convex hull* , $Q_i = CH(P_i)$, dimana $P_i = (p_0, p_1, ..., p_i)$. Deque mempunyai fungsi *push* dan *pop* dari atas/depan dan *insert* dan *remove* dari bawah/belakang. Secara spesifiknya yang dilakukan *push* v ke deque melakukan $(l \leftarrow l+1; d_t \leftarrow v)$, untuk *pop* d_t dari deque melakukan $(t \leftarrow t-1)$, untuk insert v ke deque melakukan $(b \leftarrow b-1; d_b \leftarrow v)$, dan *remove* d_b dari deque melakukan $(b \leftarrow b+1)$.

Algoritma ini menggunakan konvensi dimana vertexnya berurutan secara berlawanan jarum jam di sekitar *convex hull* Q.

Setiap d_t dan d_b mengacu kepada vertex yang sama pada rantai polygon C, dan vertex ini akan selalu menjadi vertex yang kita tambahkan terakhir pada *convex hull* . Pseudocode Melkman Convex Hull dapat dilihat pada pseudocode 1.1.

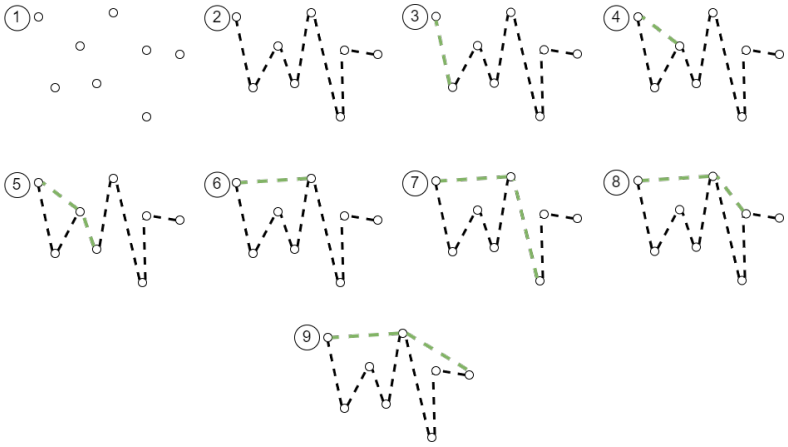
1.4.2.2. Algoritma Monotone Chain

Algoritma *monotone chain* merupakan proses pembentukan *convex hull* dari sekumpulan titik dengan kompleksitas $\mathcal{O}(n \log(n))$[3]. Asumsikan sekumpulan titik S sejumlah n ,$S = \langle s_1, s_2, ..., s_n \rangle$ algoritma ini menggunakan list untuk membentuk

Pseudocode 1.1: Melkman Convex Hull

Input: P
Output: Q

Inisialisasi: D
if LEFT(p_0, p_1, p_2) **then**
    $D \leftarrow \langle p_2, p_0, p_1, p_2 \rangle$
else
    $D \leftarrow \langle p_2, p_1, p_0, p_2 \rangle$
end if
$i = 3$
while $i < n$ **do**
    while LEFT(d_{t-1}, d_t, p_i) dan LEFT(d_b, d_{b+1}, p_i)) **do**
        $i \leftarrow i + 1$
    end while
    while !LEFT(d_{t-1}, d_t, p_i) **do**
        pop d_t
    end while
    push p_i
    while !LEFT(p_i, d_b, d_{b+1}) **do**
        remove d_b
    end while
    insert p_i
    $i \leftarrow i + 1$
end while



Gambar 1.9: Ilustrasi Algoritma Monotone Chain

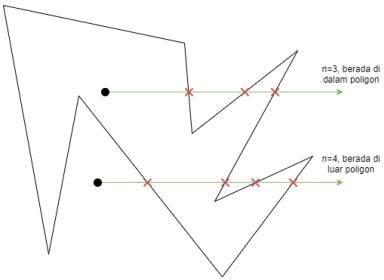
sebuah rantai (*monotone chain*), dimana list $L(S)$ menampung semua titik yang ada di S yang terurut berdasarkan nilai koordinatnya terhadap sumbu x. algoritma ini memeriksa setiap 3 vertex yang berurutan, jika 3 vertex tersebut membuat *convex* makan ketiga vertex tersebut disimpan, dan sebaliknya jika ketiga vertex tersebut membuat *concave* maka vertex ke 2 akan dibuang dari vertex penyusun *convex hull*. lalu lakukan hal yang sama dengan membalikan urutan pada L untuk mendapatkan *lower hull*. Pseudocode algoritma *Monotone Chain* dapat dilihat pada pseudocode 1.2.

1.5. Point Inside Polygon

Point Inside Polygon merupakan algoritma untuk menentukan apakah suatu polygon berada di dalam sebuah polygon atau tidak [4]. Ide utama dari algoritma ini adalah dengan cara menarik garis sejajar dengan sumbu x dimana garis tersebut berujung pada titik yang ingin dicari lokasinya kemudian hitung ada berapa edge dari poligon yang berpotongan dengan garis tersebut. Jika jumlah edge polygon yang berpotongan adalah ganjil, maka titik tersebut bera-

Pseudocode 1.2: Monotone Chain Algorithm

Input: S
Output: $CH(S)$
1: Inisialisasi: L
2: Sort S
3: $L \leftarrow S$
4: Inisialisasi $CH(S)$
5: **for** $i = 0; i < 2; i++$ **do**
6: **for** $j = 0; j < Size(L); j++$ **do**
7: **while** $Size(CH) \geq 2$ and $right(CH[Size(CH) - 1], CH[Size(CH) - 2], S[j])$ **do**
8: Delete CH last element
9: **end while**
10: push pt to CH
11: **end for**
12: reverse L
13: **end for**



Gambar 1.10: Ilustrasi Algoritma Point Inside Polygon

da dalam polygon, dan sebaliknya, jika jumlahnya genap maka titik tersebut berada di luar polygon.

BAB II
DESAIN

Pada bab ini akan dijelaskan desain algoritma yang akan digunakan untuk menyelesaikan permasalahan.

2.1. Desain Umum Sistem

Pada subbab ini akan dijelaskan mengenai gambaran secara umum dari algoritma yang dirancang. Sistem akan diawali dengan menerima masukan berupa nilai T yang menyatakan banyaknya kasus uji. T baris selanjutnya berisi nilai dari N dan P. Nilai masukan ini mengikuti batasan pada subbab **??**. Setelah menerima masukan, maka masukan tersebut akan diolah untuk menghitung nilai dari $N!$ dalam modulo P dan hasilnya ditampilkan pada layar berupa nilai dari $N!$ dalam modulo P. Pada tahap pencarian nilai dari $N!$ dalam modulo P, sistem menggunakan dua metode yang berbeda yaitu *Multipoint Evaluation* dan *Shifting Evaluation Values*. Masing-masing metode akan digunakan untuk dibandingkan kinerja satu sama lainnya. Pseudocode fungsi MAIN ditunjukkan pada pseudocode 2.1. Pada pseudocode 2.1, fungsi MAIN merupakan bagian fungsi utama yang menerima masukan, memproses masukan tersebut, dan menampilkan masukan tersebut. Fungsi INPUT merupakan fungsi untuk menerima masukan, dan fungsi PRINT merupakan fungsi untuk menampilkan hasil.

Pseudocode 2.1: Fungsi MAIN

```
1: T ← INPUT()
2: for i ← 1, T do
3:     N, P ← INPUT()
4:     ans ← SOLVE(N, P)
5:     PRINT(ANS)
6: end for
```

Nama Variabel	Fungsi Variabel
mod	Menyimpan nilai modulo yang akan digunakan dalam perkalian modular
inv	Menyimpan nilai inverse dari mod
r	Menyimpan nilai r pada persamaan (**??**)
$n\_$	Menyimpan nilai bilangan *Montgomery*

Tabel 2.1: Nama dan Fungsi Variabel dalam class MOD64

2.1.1. Desain Class Montgomery Multiplication

Class MOD64 adalah class bilangan 64-bit yang menerapkan perkalian modular *Montgomery* didalamnya. Semua variabel yang memiliki tipe data MOD64 akan dikonvert ke dalam *Montgomery World*. Class ini merupakan class terpenting dikarenakan semua perhitungan FFT dan polinomial dilakukan melibatkan perkalian modular. Pseudocode 2.2 merupakan pseudocode dari class MOD64. Nantinya pada implementasi, class ini akan melakukan override terhadap operator perkalian, penambahan, dan pengurangan.

Pseudocode 2.2: Class MOD64

```
1:  mod, inv, r, n_ ← integer
2:  constructor MOD64()
3:  constructor MOD64(n)
4:  procedure SETMOD(m)
5:  function REDUCE(x)
6:  function INIT(w)
7:  function POW(e)
8:  function INVERSE()
9:  function MULT(other)
10: function GET()
```

Fungsi-fungsi yang terkandung dalam class ini adalah SETMOD, REDUCE, INIT, POW, INVERSE, MULT, dan GET. Tabel 2.1

Masukan	Proses	Keluaran
Suatu bilangan bulat m yang menyatakan modulo yang digunakan	Menginisialisasi variabel mod, inv dan m	-

Tabel 2.2: Masukan, Proses, dan Keluaran dari Fungsi SETMOD Class MOD64

menjelaskan variabel dan kegunaannya dalam class MOD64. Pseudocode 2.2 memberikan gambaran mengenai desain class MOD64.

Fungsi *constructor* dari class ini terdiri dari dua jenis. Fungsi *constructor* yang pertama adalah fungsi dengan tanpa parameter, pada *constructor* ini, bilangan yang akan dikonversi kedalam *Montgomery World* adalah 0. Fungsi *constructor* yang kedua adalah fungsi dengan parameter n, menyatakan bilangan yang akan dikonversi kedalam *Montgomery World*.

Fungsi SETMOD, fungsi ini bertanggung jawab untuk menginisialiasi modulo yang digunakan pada seluruh tipe data MOD64, variabel yang akan diinisialisasi adalah variabel mod, inv dan r. Masukan, proses dan keluaran dari fungsi ini tercantum pada tabel 2.2. Pseudocode fungsi ini dapat dilihat pada pseudocode 2.3.

Pseudocode 2.3: Prosedur SETMOD pada class MOD64

Input: m

1: $mod \leftarrow m, inv \leftarrow$INIT($m$)$\rightarrow$INVERSE()
2: $r \leftarrow 2^{128} - m \pmod m$

Masukan	Proses	Keluaran
Suatu bilangan bulat x yang menyatakan bilangan dari *Montgomery World* yang merupakan bilangan *integer* 128-bit	Mengembalikan bilangan bulat x kedalam nilai yang seharusnya	Bilangan bulat x dengan nilai yang seharusnya

Tabel 2.3: Masukan, Proses, dan Keluaran dari Fungsi REDUCE Class MOD64

Fungsi REDUCE, fungsi ini akan mengembalikan bilangan yang menjadi masukkan fungsi dari *Montgomery World* kembali menjadi bilangan yang seharusnya. Terdapat sedikit perbedaan dengan langkah pada pseudocode **??**, langkah reduksi diubah menjadi $y = (x - (x \times n^{-1} \times mod)$ agar menjadi lebih efisien, karena langkah diatas akan menghasilkan hasil yang sama dengan pseudocode **??**. Pada desain fungsi ini akan menggunakan masukan *integer* 128-bit, yang kemudian akan direduksi menjadi *integer* 64 bit dengan dibagi 2^{64} karena akan memberikan hasil yang sama dengan modulo r, hal ini dilakukan karena komputer akan lebih cepat dalam memproses dalam bentuk bitwise. Masukan, proses dan keluaran dari fungsi ini tercantum pada tabel 2.3. Pseudocode fungsi ini dapat dilihat pada pseudocode 2.4.

Pseudocode 2.4: Fungsi REDUCE pada class MOD64

Input: x

1: **return** $y = (x - (x \times mod^{-1})\%2^{64} \times mod)/2^{64}$

Masukan	Proses	Keluaran
Suatu bilangan bulat w yang menyatakan bilangan yang akan diinisialisasi dalam tipe data MOD64	Mengkonversi bilangan masukan kedalam *Montgomery World*	-

Tabel 2.4: Masukan, Proses, dan Keluaran dari Fungsi INIT Class MOD64

Fungsi INIT, fungsi ini dipanggil oleh fungsi constructor untuk menginisialisasi bilangan pada tipe data MOD64, Fungsi ini mengkonversi bilangan masukan kedalam *Montgomery World*. Masukan, proses dan keluaran dari fungsi ini tercantum pada tabel 2.4. Pseudocode fungsi ini dapat dilihat pada pseudocode 2.5.

Pseudocode 2.5: Fungsi INIT pada class MOD64

Input: w

1: **return** REDUCE($w \times r$)

Masukan	Proses	Keluaran
Suatu bilangan bulat e yang menyatakan pangkat	Menghitung hasil pemangkatan dengan algoritma *fast modular exponentiation*	Bilangan hasil pemangkatan dalam *Montgomery World*

Tabel 2.5: Masukan, Proses, dan Keluaran dari Fungsi POW Class MOD64

Fungsi POW, fungsi ini akan menerima input e dan mengembalikan nilai bilangan sekarang pangkat e dalam *Montgomery World*. Fungsi ini menerapkan *fast modular exponentiation* seperti pada pseudocode **??**. Masukan, proses dan keluaran dari fungsi ini tercantum pada tabel 2.5. Pseudocode fungsi ini dapat dilihat pada pseudocode 2.6

Pseudocode 2.6: Fungsi POW pada class MOD64

Input: e

1: $res \leftarrow$ INIT(1) , $base \leftarrow n\_$
2: **for** $e \leftarrow 1, e/ = 2$ **do**
3: **if** e ganjil **then**
4: $res \leftarrow$MULT($res, base$)
5: **end if**
6: $base \leftarrow$MULT($base, base$)
7: **end for**
8: **return** res

Masukan	Proses	Keluaran
-	Menghitung hasil modulo inverse dengan theorema **??**	Bilangan hasil modulo inverse dalam *Montgomery World*

Tabel 2.6: Masukan, Proses, dan Keluaran dari Fungsi INVERSE Class MOD64

Fungsi INVERSE, fungsi ini akan mengembalikan nilai modulo inverse dari bilangan sekarang dengan mengadaptasi theorema **??**. Masukan, proses dan keluaran dari fungsi ini tercantum pada tabel 2.6. Pseudocode fungsi ini dapat dilihat pada pseudocode 2.7.

Pseudocode 2.7: Fungsi INVERSE pada class MOD64

1: **return** POW($mod - 2$)

Fungsi MULT, fungsi ini akan mengembalikan nilai perkalian modulo dua bilangan dengan tipe MOD64. Masukan, proses dan keluaran dari fungsi ini tercantum pada tabel 2.7. Pseudocode fungsi ini dapat dilihat pada pseudocode 2.8.

Pseudocode 2.8: Fungsi MULT pada class MOD64

1: **return** REDUCE($n\_$) $\times other$

Masukan	Proses	Keluaran
Suatu bilangan $other$ dengan tipe data MOD64 yang menyatakan bilangan yang akan dikali	Menghitung hasil perkalian dengan theorema *Montgomery World*	Bilangan hasil perkalian dalam *Montgomery World*

Tabel 2.7: Masukan, Proses, dan Keluaran dari Fungsi MULT Class MOD64

Masukan	Proses	Keluaran
-	Menghitung hasil reduksi bilangan dalam *Montgomery World*	Bilangan seharusnya

Tabel 2.8: Masukan, Proses, dan Keluaran dari Fungsi GET Class MOD64

Fungsi GET, fungsi ini akan mengembalikan nilai seharusnya dengan mereduksi variabel $n\_$. Masukan, proses dan fungsi ini tercantum pada tabel 2.8. Pseudocode fungsi ini dapat dilihat pada pseudocode 2.9.

Pseudocode 2.9: Fungsi GET pada class MOD64

```
1: return REDUCE(n_)
```

Nama Variabel	Fungsi Variabel
N	Menyimpan nilai batas maksimal panjang polinomial yang akan dikalikan yaitu 316230 dikarenakan derajat polinomial terbesar adalah $\sqrt{10^{11}}$.
MOD64_1, MOD64_2	Menyimpan *instance* dari class MOD64 dengan bilangan prima yang ditentukan pada tabel **??**.
$f1, f2, g1, g2$	Array *temporary* yang digunakan untuk mengalikan polinomial dengan NTT.

Tabel 2.9: Nama dan Fungsi Variabel dalam Namespace NTT

2.1.2. Desain Namespace Perhitungan Perkalian Polinomial dengan NTT

Class dan fungsi yang digunakan dalam perhitungan perkalian polinomial dengan NTT akan dikelompokkan menjadi satu *namespace* NTT. Namespace NTT ini digunakan sebagai dasar perkalian polinomial menggunakan NTT dengan tipe data MOD64. Pseudocode 2.10 merupakan pseudocode dari namespace NTT. Fungsi-fungsi yang terkandung dalam namespace ini adalah CONVOLVE, REVPERMUTE dan NTTDIT4. Tabel 2.9 menjelaskan variabel dan kegunaannya dalam namespace NTT.

Pseudocode 2.10: Namespace NTT

```
1: N ← 316230
2: Mod64_1 ← MOD64 → prima 709143768229478401
3: Mod64_2 ← MOD64 → prima 711416664922521601
4: f1, f2, g1, g2 ← array [1..N] integer
5: function CONVOLVE(x)
6: function REVPERMUTE(w)
7: function NTTDIT4(e)
```

Masukan	Proses	Keluaran
Berupa array A, ukuran array A, array B dan ukuran array B	Konvolusi array A dan B menggunakan NTT	Polinomial hasil konvolusi pada array A

Tabel 2.10: Masukan, Proses, dan Keluaran dari Fungsi CONVOLVE Namespace NNT

Fungsi CONVOLVE, fungsi ini bertanggung jawab untuk mengalikan dua polinomial yang menjadi input dalam *montgomery form* menggunakan langkah langkah pada subbab **??**. Pada fungsi ini akan dilakukan preproses agar derajat polinomial yang dihasilkan adalah dua pangkat i. Pada fungsi ini terdapat parameter masukan $cyclic$ yang menyatakan perkalian polinomial biasa apabila bernilai $false$ dan perkalian polinomial *middle product* apabila bernilai $true$. Setelah preproses selesai, langkah selanjutnya adalah mengecek masukan polinomial, apabila kedua polinomial tersebut identik, maka akan dilakukan proses NTT pada salah satu polinomial saja untuk menghemat waktu. Masukan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.10. Pseudocode fungsi ini dapat dilihat pada pseudocode 2.11.

Pseudocode 2.11: Fungsi CONVOLVE pada namespace NTT

Input: $a, s1, b, s2, cylic$

1: $S \leftarrow$ MAX($s1, s2$)
2: **if** !$cyclic$ **then**
3: $\quad S \leftarrow s1 + s2 - 1$
4: **end if**
5: $size \leftarrow 1$
6: **while** $size < s$ **do**
7: $\quad size = size * 2$;
8: **end while**
9: $roots \leftarrow Array(Mod64 :: level)$
10: $roots[0] \leftarrow Mod64 :: omega$
11: **for** $i \leftarrow 1$ **to** $Mod64 :: level - 1$ **do**
12: $\quad roots[i] = roots[i-1] * roots[i-1]$
13: **end for**
14: NTTDIT4(A, size, 1, roots)
15: NTTDIT4(B, size, 1, roots)
16: **for** $i \leftarrow 1$ **to** $size$ **do**
17: $\quad A[i] = A[i] * B[i]$
18: **end for**
19: NTTDIT4(A, size, -1, roots)
20: $inv \leftarrow Mod64(size).Inverse()$
21: **for** $i \leftarrow 1$ **to** $size$ **do**
22: $\quad A[i] = A[i] * inv$
23: **end for**
24: **return** A

Masukan	Proses	Keluaran
Berupa array A dan ukuran array A	Merestruktur Array A dengan bit-reverse	Array A yang telah di bit-reverse

Tabel 2.11: Masukan, Proses, dan Keluaran dari Fungsi REVPERMUTE Namespace NNT

Fungsi REVPERMUTE, fungsi ini bertanggung jawab untuk melakukan pengurutan data atau pengelompokan data dengan metode *bit-reverse*. Kompleksitas dari fungsi ini adalah $\mathcal{O}(N \log N)$. Masukan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.11. Pseudocode fungsi ini dapat dilihat pada pseudocode 2.12.

Pseudocode 2.12: Fungsi REVPERMUTE pada namespace NTT

Input: a, n

```
1:  r ← 0, nh ← n/2
2:  for i ← 1 to n do
3:      for h ← nh do
4:          if (r ← r xor h) and h then
5:              break
6:          end if
7:      end for
8:      h ← h/2
9:      if r > i then
10:         SWAP(A[i], A[r])
11:     end if
12: end for
13: return A
```

Masukan	Proses	Keluaran
Berupa array A, ukuran array A, sign atau tanda, dan array *root of unity*	NTT pada Array A sesuai dengan sign atau tanda	Array A yang telah melalui proses NTT

Tabel 2.12: Masukan, Proses, dan Keluaran dari Fungsi NTTDIT4 Namespace NNT

Fungsi NTTDIT4, fungsi ini bertanggung jawab untuk melakukan *Number Theoretic Transform* maupun *Inverse Number Theoretic Transform* pada polinomial yang menjadi input. NTT yang digunakan dalam tugas akhir ini adalah NTT-DIT 4, yang kompatibel dengan polinomial dengan derajat 4 pangkat i, karena memiliki kompleksitas lebih rendah dari NTT-DIT 2 yang biasanya digunakan. Sehingga langkah pertama adalah jika derajat polinomial bukan 4 pangkat i, maka polinomial tersebut akan di konvolusikan satu kali terlebih dahulu sebelum masuk kedalam proses NTT. Masukan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.12. Pseudocode fungsi ini agak berbeda dari NTT-DIT 2, pseudocode lengkap dapat dilihat pada pseudocode 2.13.

Pseudocode 2.13: Fungsi NTTDIT4 pada namespace NTT

Input: $A, n, sign, roots$

```
1: REVPERMUTE(A, n)
2: if log(n)&1 then
3:     for i ← 0; i < n; i ← i + 2 do
4:         a ← A[i], b ← A[i+1], A[i] ← a+b, A[i+1] ← a−b
5:     end for
6: end if
7: imag ← roots[Mod64 :: level − 2]
8: if sign < 0 then imag ← imag.inverse()
9: end if
10: for e ← 2 + log(n)&1); e < log(n) + 1; e ← e + 2 do
11:     m ← 2^e, m4 ← m/2
12:     dw ← roots[Mod64 :: level − e]
13:     if sign < 0 then dw ← dw.inverse()
14:     end if
15:     block_size ← min(n, max(m, 2^20/size_int))
16:     for k ← 0; k < n; k ← k + block_size do
17:         w, w2, w3 ← 1
18:         for j ← 0; j < m4; j ← j + 1 do
19:             for i ← k + j; i < k + block_size; i ← i + m do
20:                 a0 ← A[i+m4∗0]∗one, a2 ← A[i+m4∗1]∗w2
21:                 a1 ← A[i+m4∗2]∗w, a3 ← A[i+m4∗3]∗w3
22:                 t02 ← a0 + a2, t13 ← a1 + a3
23:                 A[i + m4 ∗ 0] = t02 + t13
24:                 A[i + m4 ∗ 2] = t02 − t13
25:                 t02 = a0 − a2, t13(a1 − a3) ∗ imag
26:                 A[i + m4 ∗ 1] = t02 + t13
27:                 A[i + m4 ∗ 3] = t02 − t13
28:             end for
29:             w ← w ∗ dw, w2 ← w ∗ w, w3 ← w2 ∗ w
30:         end for
31:     end for
32: end for
33: return A
```

2.2. Desain Penyelesaian Perhitungan Faktorial dengan Multipoint Evaluation

Pada subbab ini akan dijelaskan mengenai desain dan algoritma Multipoint Evaluation dalam menyelesaikan perhitungan Faktorial dengan modulo prima. Algoritma Multipoint Evaluation memiliki banyak sekali operasi polinomial, sehingga diperlukan *class* polinomial. Pada subbab ini akan dipaparkan dua *class* atau fungsi yang digunakan dalam algoritma *Multipoint Evaluation*.

2.2.1. Desain Class Polinomial

Class POLY adalah class yang mengandung semua algoritma yang berkaitan dengan polinomial. Class ini merupakan bagian paling penting dalam algoritma *Multipoint Evaluation*. Pseudocode 2.14 merupakan pseudocode dari class POLY. Nantinya pada implementasi, class ini akan melakukan override terhadap operator penambahan, pengurangan dan perkalian. Semua bilangan di class POLY akan menggunakan class MOD64.

Pada class POLY didefinisikan 4 variabel static yaitu NTT_THRESHOLD yang berperan sebagai batas derajat pada perkalian polinomial untuk menggunakan algoritma *NTT*, QUOTIENT_THRESHOLD yang berperan sebagai batas derajat pada pembagian polinomial untuk menggunakan algoritma pembagian, DIVREM_THRESHOLD berperan sebagai batas derajat pada perkalian polinomial untuk menggunakan algoritma pembagian dengan sisa, dan EVALUATE_THRESHOLD yang berperan sebagai batas derajat pada evaluasi polinomial menggunakan *Multipoint Evaluation*. Batas derajat ini diperlukan, karena terkadang algoritma yang cepat memiliki batas tertentu agar dia bisa lebih cepat dari pada cara naif.

Fungsi *constructor* dari class ini terdiri dari dua jenis. Fungsi *constructor* yang pertama adalah fungsi dengan tanpa parameter dan kedua adalah dengan parameter n, yang digunakan untuk mem-

Pseudocode 2.14: Class POLY

```
1:  public :
2:  static NTT_THRESHOLD ← 900
3:  static QUOTIENT_THRESHOLD ← 700
4:  static DIVREM_THRESHOLD ← 700
5:  static EVALUATE_THRESHOLD ← 16
6:  procedure VECTORADD(res, s, f, c)
7:  procedure VECTORSUB(res, s, f, c)
8:  constructor POLY()
9:  constructor POLY(n)
10: function MULBASECASE(f, g)
11: function QUOTIENTBASECASE(f, g)
12: function DIVREMBASECASE(f, g)
13: function MUL(f, g)
14: function MULCYCLICALLY(f, g)
15: function MIDDLEPRODUCT(f, g)
16: function INVERSE(n)
17: function QUOTIENT(f, g)
18: function SUBMUL(f, g, d)
19: function DIVREM(f, g)
20: function EVALUATE(x)
21: function MULPOINT(f)
22: function BUILDSUBPRODUCTTREE(beg, end, k)
23: function FASTEVAL(g, beg, end, k)
24: function MULTIPOINTEVALUATION(f, points)
25: function FACTMOD(N, mod)
26: private :
27: function MULCRT(beg, end)
28: function MULCONVOLVE(f, g, cyclic)
```

Masukan	Proses	Keluaran
Berupa hasil, ukuran array, array penambah/pengurang, dan multiplier	Melakukan operasi penambahan. pengurangan	Array hasil yang telah ditambah

Tabel 2.13: Masukan, Proses, dan Keluaran dari Fungsi VECTORADD / VECTORSUB class POLY

buat polinomial kosong dengan derajat paling besar n. Pada implementasi nya, mungkin akan bertambah *construct* yang dibuat untuk menyesuaikan dengan kebutuhan.

Fungsi VECTORADD dan VECTORSUB, fungsi ini bertanggung jawab utuk melakukan operasi pertambahan ataupun pengurangan dengan suatu konstanta tertentu dikalian nilai pada vector yang menjadi input. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.15. Pseudocode fungsi ini dapat dilihat pada pseudocode 2.13 dan pseudocode 2.16.

Pseudocode 2.15: Fungsi VECTORADD pada class POLY

Input: res, s, f, c

1: **for** $i \leftarrow 0; i < s; i \leftarrow i + 1$ **do**
2: $\quad res[i] \leftarrow res[i] + (f[i] * c)$
3: **end for**

Pseudocode 2.16: Fungsi VECTORSUB pada class POLY

Input: res, s, f, c

1: **for** $i \leftarrow 0; i < s; i \leftarrow i + 1$ **do**
2: $\quad res[i] \leftarrow res[i] - (f[i] * c)$
3: **end for**

Masukan	Proses	Keluaran
Berupa polinomial f dan polinomial g	Mengalikan polinomial f dan polinomial g dengan cara naif	Hasil perkalian polinomial f dan g

Tabel 2.14: Masukan, Proses, dan Keluaran dari Fungsi MULBASECASE Class POLY

Fungsi MULBASECASE, bertanggung jawab untuk melakukan perkalian polinomial dengan cara naif. Kompleksitas dari fungsi ini adalah $\mathcal{O}(NM)$, dengan N, M adalah derajat dari polinomial yang akan dikalikan. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.14. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.17.

Pseudocode 2.17: Fungsi MULBASECASE pada namespace NTT

Input: f, g
Output: $f \times g$
1: $size \leftarrow |f| + |g| - 1$
2: $temp, ret \leftarrow poly(0)$
3: **for** $i \leftarrow 0$ **to** |g| **do**
4: **if** $g[i]$ **then**
5: VECTORADD(temp + i, |f|, f, g[i])
6: **end if**
7: **end for**
8: **for** $i \leftarrow 0$ **to** |f| **do**
9: $ret[i] = f[i]$
10: **end for**
11: **return** ret

Fungsi QUOTIENTBASECASE, bertanggung jawab untuk melakukan pembagian polinomial dengan cara naif. Kompleksitas dari fungsi ini adalah $\mathcal{O}(NM)$, dengan N adalah derajat polinomial numerator dan M adalah derajat dari polinomial denumerator. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.15. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.18.

Pseudocode 2.18: Fungsi QUOTIENTBASECASE pada namespace NTT

Input: f, g
Output: f/g

1: $size \leftarrow |f| - |g| + 1$
2: $q \leftarrow poly(0)$
3: $temp \leftarrow f$
4: **for** $i \leftarrow 0$ **to** |f| **do**
5: **if** $temp[i]$ **then**
6: VECTORSUB($temp + i + 1, min(|f| - i, |g|) - 1, g + 1, temp[i]$)
7: **end if**
8: **end for**
9: **for** $i \leftarrow 0$ **to** |f| **do**
10: $q[i] = temp[i]$
11: **end for**
12: **return** q

Masukan	Proses	Keluaran
Berupa polinomial f dan polinomial g	Membagi polinomial f dengan polinomial g dengan cara naif	Hasil pembagian polinomial f dengan g

Tabel 2.15: Masukan, Proses, dan Keluaran dari Fungsi QUOTIENTBASECASE Class POLY

Masukan	Proses	Keluaran
Berupa polinomial f dan polinomial g	Membagi polinomial f dengan polinomial g dengan cara naif	Hasil pembagian dan sisa dari pembagian polinomial f dengan g

Tabel 2.16: Masukan, Proses, dan Keluaran dari Fungsi DIVREMBASECASE Class POLY

Fungsi DIVREMBASECASE, bertanggung jawab untuk melakukan pembagian polinomial serta sisa dari pembagian polinomial dengan cara naif. Kompleksitas dari fungsi ini adalah $\mathcal{O}(NM)$, dengan N adalah derajat polinomial numerator dan M adalah derajat dari polinomial denumerator. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.16. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.19.

Pseudocode 2.19: Fungsi DIVREMBASECASE pada namespace NTT

Input: f, g
Output: $f/g, f\%g$

1: $size \leftarrow |f| - |g| + 1$
2: $q \leftarrow poly(0)$
3: $r \leftarrow poly(0)$
4: $temp \leftarrow f$
5: $inv \leftarrow g[0]^{-1}$
6: **for** $i \leftarrow 0$ **to** |g| **do**
7: $\quad c \leftarrow temp[i] * inv$
8: $\quad$ **if** c **then**
9: $\quad\quad$ VECTORSUB($temp + i + 1, |g| - 1, g + 1, c$)
10: $\quad$ **end if**
11: $\quad q[i] \leftarrow c$
12: **end for**
13: **for** $i \leftarrow 0$ **to** |f| **do**
14: $\quad r[i - size] = temp[i]$
15: **end for**
16: **return** $[q, r]$

Masukan	Proses	Keluaran
Berupa polinomial f dan polinomial g	Pergantian antara perkalian polinomial dengan cara naif maupun menggunakan NTT	Hasil perkalian polinomial f dan g

Tabel 2.17: Masukan, Proses, dan Keluaran dari Fungsi MUL Class POLY

Fungsi MUL, bertanggung jawab untuk melakukan pergantian untuk mengalikan polinomial dengan naif maupun menggunakan NTT. Pada fungsi ini digunakan variabel NTT_THRESHOLD yang merupakan batas menggunakan algoritma naif maupun NTT. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.17. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.20.

Pseudocode 2.20: Fungsi MUL pada namespace NTT

Input: f, g
Output: fxg

```
1: if |f| = 0  or  |g| = 0 then
2:     return Poly(0)
3: end if
4: if |f| + |g| ≤ NTT_THRESHOLD then
5:     return MULBASECASE(f, g)
6: end if
7: MULCONVOLVE(f, g, false)
8: return MULCRT(0, |f|+|g|-1)
```

Masukan	Proses	Keluaran
Berupa polinomial f dan polinomial g	Mengalikan polinomial f dan g dengan cyclic	Hasil perkalian cyclic polinomial f dan g

Tabel 2.18: Masukan, Proses, dan Keluaran dari Fungsi MULCYCLICALLY Class POLY

Fungsi MULCYCLICALLY, bertanggung jawab untuk melakukan perkalian polinomial yang memiliki sifat *cyclic* seperti Middle Product. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.18. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.21.

Pseudocode 2.21: Fungsi MULCYCLICALLY pada namespace NTT

Input: f, g
Output: $fxg(cyclic)$

```
1:  if |f| = 0  or  |g| = 0 then
2:      return Poly(0)
3:  end if
4:  MULCONVOLVE(f, g, true)
5:  s ← 1MAX(|f|, |g|)
6:  size ← 1
7:  while size < s do size ← size * 2
8:  end while
9:  if |f| + |g| ≤ NTT_THRESHOLD then
10:     return MULBASECASE(f, g)
11: end if
12: return MULCRT(0, size)
```

Masukan	Proses	Keluaran
Berupa polinomial f dan polinomial g	Mengalikan polinomial f dan g dengan *Middle Product Optimization*	Hasil perkalian *Middle Product* polinomial f dan g

Tabel 2.19: Masukan, Proses, dan Keluaran dari Fungsi MIDDLEPRODUCT Class POLY

Fungsi MIDDLEPRODUCT, bertanggung jawab untuk melakukan perkalian polinomial yang mengembalikan *middle product* dari polinomial hasil perkalian. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.19. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.22.

Pseudocode 2.22: Fungsi MIDDLEPRODUCT pada namespace NTT

Input: f, g
Output: $f \times g(\textit{middle product})$
1: **if** $|f| = 0$ or $|g| = 0$ **then**
2: **return** Poly(0)
3: **end if**
4: MULCONVOLVE($f, g, true$)
5: **return** MULCRT($|f|, |g|$)

Masukan	Proses	Keluaran
Berupa polinomial f dan pangkat modulo n	Mencari *power inversion* n dari polinomial f	Hasil *power inversion* n dari polinomial f

Tabel 2.20: Masukan, Proses, dan Keluaran dari Fungsi INVERSE Class POLY

Fungsi INVERSE, bertanggung jawab untuk melakukan power inverse dari sebuah polinomial menggunakan Interasi Newton. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.20. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.23.

Pseudocode 2.23: Fungsi INVERSE pada namespace NTT

Input: f, n
Output: $f^{-1} \bmod x^n$

1: $ret \leftarrow Poly(1)$
2: **for** $e \leftarrow 1, ne; e < n; e = ne$ **do**
3: $\quad ne \leftarrow \text{MIN}(2 * e, n)$
4: $\quad h \leftarrow ret_{[ne-e \cdots n]} * -\text{MIDDLEPRODUCT}(ret, f)$
5: $\quad [ret] \leftarrow [ret, h]$
6: **end for**
7: **return** ret

Masukan	Proses	Keluaran
Berupa polinomial f dan g	Membagi polinomial f dengan polinomial g	Hasil bagi polinomial f dari polinomial g

Tabel 2.21: Masukan, Proses, dan Keluaran dari Fungsi QUOTIENT Class POLY

Fungsi QUOTIENT, bertanggung jawab untuk melakukan pembagian yang tidak melibatkan sisa hasil bagi, serta melakukan pergantian untuk membagi polinomial dengan cara naif maupun menggunakan algoritma yang telah didefinisikan sebelumnya. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.21. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.24.

Pseudocode 2.24: Fungsi QUOTIENT pada namespace NTT

Input: f, g
Output: f/g
1: **if** $|g| \leq$ QUOTIENT_THRESHOLD **then**
2: **return** QUOTIENTBASECASE(f, g)
3: **end if**
4: $s \leftarrow |f|/2 + 1$
5: $inv \leftarrow b^{-1}_{[0\cdots n-s-1]}$
6: $q1 \leftarrow inv \times a_{[0\cdots n-s-1]}$
7: $lo \leftarrow$ MIDDLEPRODUCT$(q1, b)$
8: $q2 \leftarrow [inv_0, \cdots inv_{s-1}] \times [b_{n-s} - lo_0, \cdots, b_{n-1} - lo_{s-1}]$
9: **return** $[q1_0, \cdots, q1_{n-s-1}, q2_0, \cdots, q2_{s-1}]$.

Masukan	Proses	Keluaran
Berupa polinomial f, q dan d	Menghitung $f - q.d$	Hasil dari $f - q.d$

Tabel 2.22: Masukan, Proses, dan Keluaran dari Fungsi SUBMUL Class POLY

Fungsi SUBMUL, bertanggung jawab untuk melakukan operasi $f - d.q$. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.22. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.25.

Pseudocode 2.25: Fungsi SUBMUL pada namespace NTT

Input: f, q, d
Output: $f - q.d$

1: $sq \leftarrow |q|$
2: $q \leftarrow$ MULCYCICALLY(d)
3: $mask \leftarrow |p| - 1$
4: **for** $i \leftarrow 0$ **to** sq **do**
5: $\quad p[i\&mask] \leftarrow p[i\&mask] - f[i\&mask]$
6: **end for**
7: $r \leftarrow f$
8: **for** $i \leftarrow 0$ **to** |f| - sq **do**
9: $\quad r[i] \leftarrow r[i] - p[(sq + i)\&mask]$
10: **end for**
11: **return** r.

Masukan	Proses	Keluaran
Berupa polinomial f dan g	Membagi polinomial f dengan polinomial g dengan sisa	Hasil bagi dan sisa hasil bagi polinomial f dari polinomial g

Tabel 2.23: Masukan, Proses, dan Keluaran dari Fungsi DIVREM Class POLY

Fungsi DIVREM, bertanggung jawab untuk melakukan pembagian yang melibatkan sisa hasil bagi, serta melakukan pergantian untuk membagi polinomial dengan cara naif maupun menggunakan algoritma yang telah didefinisikan sebelumnya. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.23. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.26.

Pseudocode 2.26: Fungsi DIVREM pada namespace NTT

Input: f, g
Output: f/g
1: **if** $|f| < |g|$ **then**
2: **return** $[Poly(0), f]$
3: **end if**
4: **if** $|f| \leq$ DIVREM_THRESHOLD **then**
5: **return** DIVREMBASECASE(f, g)
6: **end if**
7: $sq \leftarrow |f| - |g| + 1$
8: $q =$ QUOTIENT(f, g)
9: $r =$ SUBMUL(f, q, g)
10: **return** $[q, r]$.

Masukan	Proses	Keluaran
Berupa polinomial f dan g	Membagi polinomial f dengan polinomial g dengan sisa	Hasil bagi dan sisa hasil bagi polinomial f dari polinomial g

Tabel 2.24: Masukan, Proses, dan Keluaran dari Fungsi EVALUATE Class POLY

Fungsi EVALUATE, bertanggung jawab untuk menghitung hasil evaluasi polinomial pada titik tertentu menggunakan metode horner. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.24. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.27.

Pseudocode 2.27: Fungsi EVALUATE pada namespace NTT

Input: f, x
Output: $f(x)$
1: $ret \leftarrow 0$
2: **for** $i \leftarrow 0$ **to** sq **do**
3: $\quad ret \leftarrow ret * x + f[i]$
4: **end for**
5: **return** ret.

Masukan	Proses	Keluaran
Berupa titik awal beg dan akhir end	Mengalikan polinomial dari titik beg sampai end	Hasil perkalian polinomial dari titik beg sampai end

Tabel 2.25: Masukan, Proses, dan Keluaran dari Fungsi MULPOINT Class POLY

Fungsi MULPOINT, bertanggung jawab untuk melakukan perkalian pada polinomial titik. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.25. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.28.

Pseudocode 2.28: Fungsi MULPOINT pada namespace NTT

Input: beg, end
Output: $(x - f(beg)) \times \cdots \times (x - f(end))$

```
1: if beg − end = 1 then
2:     return Poly(1, f[beg])
3: end if
4: mid ← (beg + end)/2
5: return MULPOINT(beg, mid) × MULPOINT(mid, end)
```

Masukan	Proses	Keluaran
Berupa index mulai , berakhir, serta posisi node saat ini	Membuat pohon polinomial	-

Tabel 2.26: Masukan, Proses, dan Keluaran dari Fungsi BUILDSUBPRODUCTTREE Class POLY

Fungsi BUILDSUBPRODUCTTREE, bertanggung jawab untuk membuat pohon polinomial dari titik titik pada vector points. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.26. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.29.

Pseudocode 2.29: Fungsi BUILDSUBPRODUCTTREE pada namespace NTT

Input: beg, end, k

1: **if** $end - beg = 1$ **then**
2: $\quad tree[k] \leftarrow Poly(1, -points[beg])$
3: **else**
4: $\quad mid \leftarrow (beg + end)/2$
5: $\quad$ BUILDSUBPRODUCTTREE$(beg, mid, 2 * k + 1)$
6: $\quad$ BUILDSUBPRODUCTTREE$(mid, end, 2 * k + 2)$
7: $\quad tree[k] \leftarrow tree[2 * k + 1] \times tree[2 * k + 2]$
8: **end if**

Masukan	Proses	Keluaran
Berupa polinomial r, index mulai , berakhir, serta posisi node saat ini	Evaluasi polinomial r pada titik pada array dengan index mulai sampai index berakhir	-

Tabel 2.27: Masukan, Proses, dan Keluaran dari Fungsi FASTEVAL Class POLY

Fungsi FASTEVAL, bertanggung jawab untuk evaluasi terhadap pohon polinomial. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel . Pseudocode dari fungsi ini dapat dilihat pada pseudocode .

Pseudocode 2.30: Fungsi FASTEVAL pada namespace NTT

Input: g, beg, end, k

```
1: if end − beg ≤ EVALUATE_THRESHOLD then
2:     for i ← beg to end-1 do
3:         res[i] ←EVALUATE(r, points[i])
4:     end for
5: else
6:     mid ← (beg + end)/2
7:     FASTEVAL(r, beg, mid, 2 ∗ k + 1)
8:     FASTEVAL(r, mid, end, 2 ∗ k + 2)
9: end if
```

Masukan	Proses	Keluaran
Berupa polinomial f dan titik titik evaluasi	Multipoint Evaluation pada polinomial f dan titik titik evaluasi	Array hasil evaluasi polinomial f pada titik titik evaluasi

Tabel 2.28: Masukan, Proses, dan Keluaran dari Fungsi MULTIPOINTEVALUATION Class POLY

Fungsi MULTIPOINTEVALUATION, bertanggung jawab untuk melakukan *multipoint evaluation* pada polinomial f pada beberapa titik. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.28. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.31.

Pseudocode 2.31: Fungsi MULTIPOINTEVALUATION pada namespace NTT

Input: $f, points$
Output: res

1: $s \leftarrow |s|$
2: $res \leftarrow Array()$
3: $tree\_size \leftarrow 2 \times 2^{2 \log S - 1}$
4: $tree \leftarrow Array(tree\_size)$
5: BUILDSUBPRODUCTTREE$(0, s, 0)$
6: FASTEVAL$(f, 0, s, 0)$
7: **return** res

Masukan	Proses	Keluaran
Berupa N dan P	Perhitungan $N!$ mod P	Hasil perhitungan $N!$ mod P

Tabel 2.29: Masukan, Proses, dan Keluaran dari Fungsi FACTMOD Class POLY

Masukan	Proses	Keluaran
Berupa index awal beg dan akhir end	Perhitungan nilai modulo sesung-guhnya pada index beg sampai end	Hasil nilai modulo sesungguhnya pada index beg sampai end

Tabel 2.30: Masukan, Proses, dan Keluaran dari Fungsi MULCRT Class POLY

Fungsi FACTMOD, bertanggung jawab untuk menghitung nilai dari $N!$ mod P. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.29. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.32.

Fungsi MULCRT, bertanggung jawab untuk menghitung nilai modulo sesungguhnya dari dua buah array hasil konvolusi yang sebelumnya dihitung dengan dua buah bilangan prima berbeda. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.30. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.33.

Pseudocode 2.32: Fungsi FACTMOD pada namespace NTT

Input: N, P
Output: $N!$ mod P

SETMOD(P)
if $N \leq P - N - 1$ **then**
    $v \leftarrow \left\lfloor \sqrt{N} \right\rfloor, ret = 1, points \leftarrow Array()$
    for $i \leftarrow 0$ **to** $v - 1$ **do** $points[i] \leftarrow (i * v + 1)$
    end for
    $f \leftarrow$ MULPOINT(points)
    for $i \leftarrow 0$ **to** $v - 1$ **do** $points[i] \leftarrow i$
    end for
    $eval \leftarrow$ MULTIPOINTEVALUATION$(f, point)$
    for $i \leftarrow 0$ **to** $v - 1$ **do** $ret \leftarrow ret * eval[i]$
    end for
    for $i \leftarrow v \times v + 1$ **to** N **do** $ret \leftarrow ret * i$
    end for
    return ret
else
    $v \leftarrow \left\lfloor \sqrt{P - N - 1} \right\rfloor, ret = 1, points \leftarrow Array()$
    for $i \leftarrow 0$ **to** $v - 1$ **do** $points[i] \leftarrow (i * v + 1 + N)$
    end for
    $f \leftarrow$ MULPOINT(points)
    for $i \leftarrow 0$ **to** $v - 1$ **do** $points[i] \leftarrow i$
    end for
    $eval \leftarrow$ MULTIPOINTEVALUATION$(f, point)$
    for $i \leftarrow 0$ **to** $v - 1$ **do** $ret \leftarrow ret * eval[i]$
    end for
    for $i \leftarrow v \times v + 1 + N$ **to** $P - 1$ **do** $ret \leftarrow ret * i$
    end for
    $ret = ret^{-1}$
    if $v * v + 1 + N > P - 1$ **then** $ret = ret * (P - 1)$
    end if
    return ret
end if

Pseudocode 2.33: Fungsi MULCRT pada namespace NTT

Input: beg, end
Output: Evaluasi CRT pada array dengan index beg sampai end

```
1: inv ← Mod64_2(Mod64_1 :: modulus).Inverse()
2: mod ← Mod64 :: modulus
3: mod1 ← Mod64_1 :: modulus % mod
4: ret ← Poly()
5: for i ← 0 to end − beg do
6:     r1 ← Mod64_2(f1[i + beg])
7:     r2 ← f2[i + beg]
8:     ret[i] ← r1% mod + ((r2 − r1) ∗ inv)% mod ∗ mod1
9: end for
10: return ret
```

Masukan	Proses	Keluaran
Berupa polinomial f, g dan boolean cyclic	Melakukan konvolusi pada polinomial f dan g dengan modulo 2 bilangan prima berbeda	Hasil konvolusi pada polinomial f dan g dengan modulo 2 bilangan prima berbeda

Tabel 2.31: Masukan, Proses, dan Keluaran dari Fungsi MULCONVOLVE Class POLY

Fungsi MULCONVOLVE, bertanggung jawab untuk menghitung nilai konvolusi dengan modulo dua buah bilangan prima berbeda. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.31. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.34.

Pseudocode 2.34: Fungsi MULCONVOLVE pada namespace NTT

Input: $f, g, cyclic$

for $i \leftarrow 0$ **to** $|f|$ **do**
    $f1[i] \leftarrow f[i]$
end for
for $i \leftarrow 0$ **to** $|g|$ **do**
    $g1[i] \leftarrow g[i]$
end for
CONVOLVE($f1, |f|, g1, |g|, cyclic$)
for $i \leftarrow 0$ **to** $|f|$ **do**
    $f2[i] \leftarrow f[i]$
end for
for $i \leftarrow 0$ **to** $|g|$ **do**
    $g2[i] \leftarrow g[i]$
end for
CONVOLVE($f2, |f|, g2, |g|, cyclic$)

2.2.2. Desain Fungsi Solve

Pada subbab ini akan dijelaskan mengenai fungsi Solve untuk menyelesaikan permasalahan. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.35.

Pseudocode 2.35: Fungsi SOLVE

Input: N, P

return POLY::FACTMOD(N, P)

Masukan	Proses	Keluaran
Berupa bilangan n yang menyatakan batas perhitungan *inverse* faktorial	Melakukan perhitungan *inverse* faktorial	Array hasil *precompute inverse* faktorial

Tabel 2.32: Masukan, Proses, dan Keluaran dari Fungsi PRECOMPUTEIFACTORIALS

2.3. Desain Penyelesaian Perhitungan Faktorial dengan Shifting Evaluation Values

Pada subbab ini akan dijelaskan mengenai desain dan algoritma Shifting Evaluation Values dalam menyelesaikan perhitungan Faktorial dengan modulo prima. selanjutnya akan dipaparkan beberapa fungsi yang digunakan dalam algoritma *Shifting Evaluation Values*.

2.3.1. Desain Fungsi Perhitungan Inverse Factorial

Pada persamaan (**??**), diperlukan perhitungan *inverse* faktorial. Nilai *inverse* faktorial ini dapat dihitung terlebih dahulu agar efisien. Fungsi PRECOMPUTEIFACTORIALS, bertanggung jawab untuk melakukan perhitungan ini. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.32. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.36.

Pseudocode 2.36: Fungsi PRECOMPUTEIFACTORIALS

Input: n

```
1: iter ← 1
2: ret ← Array(1, n + 1)
3: for i ← 1 to n do
4:     ret[i] ← iter ∗ ret[i − 1]
5:     iter ← iter + 1
6: end for
7: ret[n] ← ret[n].inverse()
8: iter ← iter − 1
9: if n = 0 then return ret
10: end if
11: for i ← n − 1 to 0 do
12:     ret[i] ← iter ∗ ret[i + 1]
13:     iter ← iter − 1
14: end for
15: return ret
```

Masukan	Proses	Keluaran
Berupa array f yang nilai awal	Melakukan proses agar nilai awal tersebut menjadi polinomial $\tilde{P}$	Polinomial $\tilde{P}$

Tabel 2.33: Masukan, Proses, dan Keluaran dari Fungsi CONV

2.3.2. Desain Fungsi Convolution

Konvolusi digunakan untuk menghitung polinomial $\tilde{P}$ dilakukan oleh fungsi CONV. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.33. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.37.

Pseudocode 2.37: Fungsi CONV

Input: f

1: $n \leftarrow |f|$
2: $ret \leftarrow f$
3: **for** $i \leftarrow 0$ **to** $n-1$ **do**
4: $\quad d \leftarrow ifactorials[i] \times ifactorials(n-1-i) \times (-1)^{n-1-i}$
5: $\quad ret[i] \leftarrow ret[i] * d$
6: **end for**
7: **return** ret

Masukan	Proses	Keluaran
Berupa polinomial f dan g	Melakukan perkalian *Middle Product* polinomial f dan g	*Middle Product* dari perkalian polinomial f dan g

Tabel 2.34: Masukan, Proses, dan Keluaran dari Fungsi MIDDLEPRODUCT

Masukan	Proses	Keluaran
Berupa polinomial $\tilde{P}$ dan delta perubahan	Melakukan proses *shifting* pada polinomial $\tilde{P}$ sesuai dengan delta perubahan	Polinomial $\tilde{P}$ baru hasil *shifting*

Tabel 2.35: Masukan, Proses, dan Keluaran dari Fungsi SHIFT

2.3.3. Desain Fungsi Middle Product

Middle Product Optimization digunakan untuk mengalikan polinomial $\tilde{P}$ dan polinomial S, seperti pada teorema **??**. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.34. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.38.

2.3.4. Desain Fungsi Shifting Evaluation Values

Shifting Evaluation Values akan digunakan untuk menyelesaikan grid. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.35. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.39.

Pseudocode 2.38: Fungsi MIDDLEPRODUCT

Input: f, g

Middle Product $f \times g$
for $i \leftarrow 0$ **to** $|f|$ **do**
    $f1[i] \leftarrow f[i]$
end for
for $i \leftarrow 0$ **to** $|g|$ **do**
    $g1[i] \leftarrow g[i]$
end for
CONVOLVE($f1, |f|, g1, |g|, true$)
for $i \leftarrow 0$ **to** $|f|$ **do**
    $f2[i] \leftarrow f[i]$
end for
for $i \leftarrow 0$ **to** $|g|$ **do**
    $g2[i] \leftarrow g[i]$
end for
CONVOLVE($f2, |f|, g2, |g|, true$)
$beg \leftarrow |f| - 1, end \leftarrow |g|$
$inv \leftarrow Mod64\_2(Mod64\_1 :: modulus).Inverse()$
$mod \leftarrow Mod64 :: modulus$
$mod1 \leftarrow Mod64\_1 :: modulus$ % mod
$ret \leftarrow Poly()$
for $i \leftarrow 0$ **to** $end - beg$ **do**
    $r1 \leftarrow Mod64\_2(f1[i + beg])$
    $r2 \leftarrow f2[i + beg]$
    $ret[i] \leftarrow r1\%$ mod $+ ((r2 - r1) * inv)\%$ mod $* mod1$
end for
return ret

Pseudocode 2.39: Fungsi SHIFT

Input: f, dx

1: $n \leftarrow |f|, deg \leftarrow n-1, a = \frac{dx}{\sqrt{N}}, g = Array(0)$
2: $r \leftarrow a - deg$
3: **for** $i \leftarrow 0$ **to** $|g|$ **do**
4: $\quad g[i] \leftarrow r$
5: $\quad r \leftarrow r + 1$
6: $\quad$ **if** $r = mod$ **then** $r = 0$
7: $\quad$ **end if**
8: **end for**
9: **for** $i \leftarrow 1$ **to** $|g|$ **do** $g[i] \leftarrow g[i] \times g[i-1]$
10: **end for**
11: $inv \leftarrow g[|g|-1].inverse()$
12: $iter \leftarrow |g| - 1$
13: **for** $i \leftarrow |g|$ **to** 0 **do**
14: $\quad g[i] \leftarrow g[i-1] \times inv$
15: $\quad inv \leftarrow inv * (a + iter - deg)$
16: $\quad iter \leftarrow iter - 1$
17: **end for**
18: $g[0] \leftarrow inv$
19: $ret \leftarrow$ MIDDLEPRODUCT(f, g)
20: $prod \leftarrow 1, iter \leftarrow 0$
21: **for** $i \leftarrow 0$ **to** $n-1$ **do**
22: $\quad prod \leftarrow prod \times (a + deg - iter), iter \leftarrow iter + 1$
23: **end for**
24: $iter \leftarrow iter - 1$
25: **for** $i \leftarrow n-1$ **to** 0 **do**
26: $\quad ret[i] \leftarrow ret[i] \times prod$
27: $\quad prod \leftarrow (prod \times g[n-1+i]) \times (a + iter - deg - 1)$
28: $\quad iter \leftarrow iter - 1$
29: **end for**
30: **return** ret

2.3.5. Desain Fungsi Solve Grid

Fungsi Solve Grid digunakan untuk mengimplementasikan *Shifting Evaluation Values* kepada permasalahan faktorial. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.36. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.40.

Pseudocode 2.40: Fungsi SOLVEGRID

Input: n

1: **if** n = 1 **then return** $[1, 1 + \sqrt{N}]$
2: **end if**
3: $halfN \leftarrow n/2$
4: $f11 \leftarrow$ SOLVEGRID($halfN$)
5: $f \leftarrow$ CONV($f11$)
6: $f12 \leftarrow$ SHIFT($f, halfN$)
7: $f21 \leftarrow$ SHIFT($f, halfN * \sqrt{N}$)
8: $f22 \leftarrow$ SHIFT($f, halfN * \sqrt{N} + halfN$)
9: **for** $i \leftarrow 0$ **to** $halfN$ **do** $f11[i] \leftarrow f11[i] \times f12[i]$
10: **end for**
11: **for** $i \leftarrow 1$ **to** $halfN$ **do** $f11[i + halfN] \leftarrow f21[i] \times 212[i]$
12: **end for**
13: **if** n&1 **then**
14: **for** $i \leftarrow 0$ **to** n **do** $f11[i] \leftarrow f11[i] \times (\sqrt{N} \times i + n)$
15: **end for**
16: **for** $i \leftarrow 1$ **to** n **do** $prod \leftarrow prod \times (\sqrt{N} \times n + i + 1)$
17: **end for**
18: $f11[|f11| - 1] \leftarrow prod$
19: **end if**
20: **return** $f11$

Masukan	Proses	Keluaran
Berupa polinomial $\tilde{P}$ dan delta perubahan	Melakukan proses *shifting* pada polinomial $\tilde{P}$ sesuai dengan delta perubahan	Polinomial $\tilde{P}$ baru hasil *shifting*

Tabel 2.36: Masukan, Proses, dan Keluaran dari Fungsi SOLVEGRID

Masukan	Proses	Keluaran
Berupa N dan P	Perhitungan $N!$ mod P	Hasil perhitungan $N!$ mod P

Tabel 2.37: Masukan, Proses, dan Keluaran dari Fungsi FACTMOD

2.3.6. Desain Fungsi FactMod

Fungsi FACTMOD, digunakan untuk menghitung nilai dari $N!$ mod P. Masukkan, proses, dan keluaran dari fungsi ini tercantum pada tabel 2.37. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.41.

2.3.7. Desain Fungsi Solve

Pada subbab ini akan dijelaskan mengenai fungsi Solve untuk menyelesaikan permasalahan. Pseudocode dari fungsi ini dapat dilihat pada pseudocode 2.42.

Pseudocode 2.41: Fungsi FACTMOD

Input: n

1: **if** $n/2 >= mod$ **then**
2: $m \leftarrow mod - n - 1$
3: $ret \leftarrow$ FACTMOD(M)
4: **if** M&1 = 0 **then**
5: $ret \leftarrow mod - ret$
6: **end if**
7: **return** $ret.inverse()$
8: **end if**
9: $k \leftarrow$ MIN($\sqrt{n}, \frac{n}{\sqrt{n}}$)
10: $ret \leftarrow f[k-1]$
11: $t \leftarrow k * \sqrt{N}$
12: **for** $i \leftarrow 0$ **to** n **do**
13: $ret \leftarrow ret \times (t + i + 1)$
14: **end for**
15: **return** ret

Pseudocode 2.42: Fungsi SOLVE

Input: N, mod

1: **if** $N \geq mod$ **then return** 0
2: **end if**
3: **if** $N = mod$ **then return** 1
4: **end if**
5: **if** $N + 1 = mod$ **then return** N
6: **end if**
7: $x \leftarrow N$
8: **if** $N * 2 > mod$ **then**
9: $\quad x \leftarrow mod - N$
10: **end if**
11: MOD64::SETMOD(mod)
12: $ifactorials \leftarrow$ PRECOMPUTEIFACTORIALS$(\sqrt{N}/2)$
13: $f \leftarrow$ SOLVEGRID$(\sqrt{N})$
14: **for** $i \leftarrow 1$ **to** $\sqrt{N} - 1$ **do**
15: $\quad f[i] \leftarrow f[i] \times f[i-1]$
16: **end for**
17: **return** FACTMOD(N)

[*Halaman ini sengaja dikosongkan*]

DAFTAR PUSTAKA

[1] SPOJ. (2009). LL and ErBao, **url**: https://www.spoj.com/problems/ISUN1/.

[2] A. A. Melkman, **?**On-line construction of the convex hull of a simple polyline**?**, *Information Processing Letters 25*, **pages** 11–12, 1987.

[3] A. Andrew., **?**Another Efficient Algorithm for Convex Hulls in Two Dimensions**?**, *Information Processing Letters 9*, **pages** 216–219, 1979.

[4] geeksforgeeks. (2019). How to check if a given point lies inside or outside a polygon, **url**: https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/.