Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - IF184802

IMPLEMENTASI REDUKSI POLIGON MENGGUNAKAN ALGORITMA MELKMAN CONVEX HULL YANG DIMODIFIKASI DENGAN STUDI KASUS SPHERE ONLINE JUDGE 5637 LL AND ERBAO

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 SPHERE ONLINE JUDGE 5637 LL AND ERBAO

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 SPHERE ONLINE JUDGE 5637 LL AND ERBAO

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 SPHERE ONLINE JUDGE 5637 LL AND ERBAO

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
6 November 2019

[*Halaman ini sengaja dikosongkan*]

ABSTRAK

IMPLEMENTASI REDUKSI POLIGON MENGGUNAKAN ALGORITMA MELKMAN CONVEX HULL YANG DIMODIFIKASI DENGAN STUDI KASUS SPHERE ONLINE JUDGE 5637 LL AND ERBAO

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 SPHERE ONLINE JUDGE 5637 LL AND ERBAO

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, 6 November 2019

Michael Julian Albertus

[*Halaman ini sengaja dikosongkan*]

DAFTAR ISI

[*Halaman ini sengaja dikosongkan*]

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR PSEUDOCODE

[*Halaman ini sengaja dikosongkan*]

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.

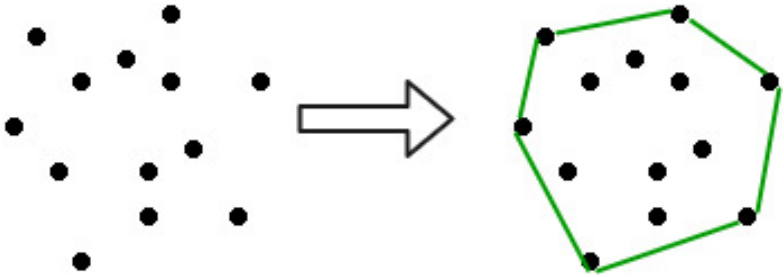
[*Halaman ini sengaja dikosongkan*]

BAB I
PENDAHULUAN

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

1.1. Latar Belakang

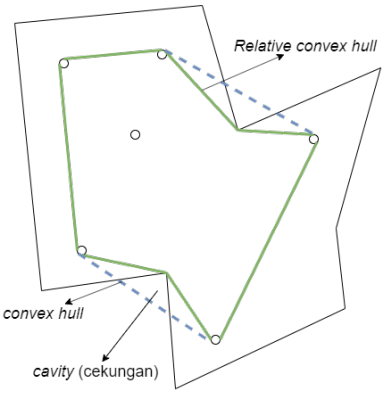
Computational geometry adalah cabang dari ilmu komputer yang dikhususkan untuk mempelajari algoritma yang dapat dinyatakan dalam suatu geometri. Salah satu algoritma yang sering dipakai pada *computational geometry* adalah algoritma *convex hull* . *Convex hull* adalah sebuah set poligon dari titik pada bidang Euclidean atau ruang Euclidean, atau dapat disebut himpunan cembung terkecil yang berisi titik. Sebagai contoh, ketika suatu kumpulan titik merupakan bagian yang dibatasi dalam sebuah bidang, *convex hull* dapat divisualisasikan sebagai bentuk yang tertutup oleh karet gelang yang membentang di sekitar titik - titik tersebut. Berikut merupakan contoh dari *convex hull* :



Gambar 1.1: Ilustrasi Convex Hull

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

cavity (cekungan kedalam) yang diakibatkan atau relatif terhadap sesuatu yang membatasi *convex hull* tersebut. ilustrasi *relative convex hull* dapat dilihat pada gambar 2.7.



Gambar 1.2: Ilustrasi Relative Convex Hull

Pada topik Tugas Akhir ini akan dijelaskan algoritma penyelesaian untuk mencari *relative convex hull* dari sekumpulan titik yang berada di dalam sebuah polygon sederhana dengan menggunakan algoritma *Melkman Convex Hull* yang dimodifikasi dengan studi kasus pada Sphere Online Judge 5637 LL and ErBao.

1.2. Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut :

1. Bagaimana mencari *relative convex hull* dari kumpulan titik di dalam sebuah poligon?
2. Bagaimana algoritma *Melkman convex hull* yang dimodifikasi menyelesaikan masalah *relative convex hull* dari kumpulan titik?

1.3. Batasan Masalah

Permalsahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut :

1. Implementasi algoritma *Melkman convex hull* yang di modifikasi sebagai penyelesaian permasalahan *relative convex hull* pada soal ISUN1.
2. Algoritma *relative convex hull* terbatas pada analisis intuitif yang logis.

Berikut merupakan batasan pada situs Sphere Online Judge:

1. Implementasi dilakukan menggunakan bahasa pemrograman C++.
2. Banyaknya sisi pada poligon pembatas (n) diantara 3 sampai 500.
3. banyaknya pohon yang berada dalam taman (m) diantara 0 sampai 500.
4. Batas maksimum untuk tiap vertex memenuhi(x, y) dimana nilai $|x|, |y| \leq 10000$.
5. Banyak soal tidak diketahui karena program berhenti sampai EOF.
6. Batas waktu yang diberikan adalah 0.142 detik.
7. Batas memori yang diberikan adalah 1.536 MB.
8. Batas kode sumber yang diberikan adalah 50.000 B.

1.4. Tujuan

Tujuan Tugas Akhir ini adalah sebagai berikut :

1. Mengevaluasi kinerja algoritma *Melkman Convex Hull* yang dimodifikasi untuk menyelesaikan permasalahan komputasi *relative convex hull* pada LL and ErBao.

1.5. Manfaat

Tugas Akhir ini mampu memberikan pemahaman algoritma yang tepat untuk menyelesaikan permasalahan komputasi *relative convex hull* dengan efisien.

1.6. Metodologi

Metodologi pengerjaan yang digunakan pada Tugas Akhir ini memiliki beberapa tahapan. Tahapan-tahapan tersebut yaitu :

1. Penyusunan proposal
   Pada tahapan ini penulis memberikan penjelasan mengenai apa yang penulis akan lakukan dan mengapa Tugas Akhir ini dilakukan. Penjelasan tersebut dituliskan dalam bentuk proposal Tugas Akhir.
2. Studi literatur
   Pada tahapan ini penulis mengumpulkan referensi yang diperlukan guna mendukung pengerjaan Tugas Akhir. Referensi yang digunakan dapat berupa hasil penelitian yang sudah pernah dilakukan, buku, artikel internet, atau sumber lain yang bisa dipertanggungjawabkan.
3. Implementasi algoritma
   Pada tahapan ini penulis mulai mengembangkan algoritma yang digunakan untuk menyelesaikan permasalahan komputasi *relative convex hull*.
4. Pengujian dan evaluasi
   Pada tahapan ini penulis menguji performa algoritma yang digunakan. Hasil pengujian kemudian dievaluasi untuk kemudian dipertimbangkan apakah algoritma masih bisa ditingkatkan lagi atau tidak.
5. Penyusunan buku
   Pada tahapan ini penulis menyusun hasil pengerjaan Tugas Akhir mengikuti format penulisan Tugas Akhir.

1.7. Sistematika Penulisan

Sistematika laporan Tugas Akhir yang akan digunakan adalah sebagai berikut :

1. BAB I : PENDAHULUAN
   Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, 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 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 : PENUTUP
   Bab ini berisi kesimpulan dan saran yang didapat dari hasil uji coba yang telah dilakukan.

[*Halaman ini sengaja dikosongkan*]

BAB II
DASAR TEORI

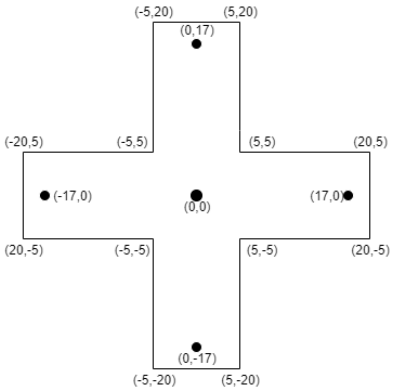
Pada bab ini, akan dijelaskan dasar teori yang digunakan sebagai landasaan pengerjaan Tugas Akhir ini.

2.1. Deskripsi Permasalahan

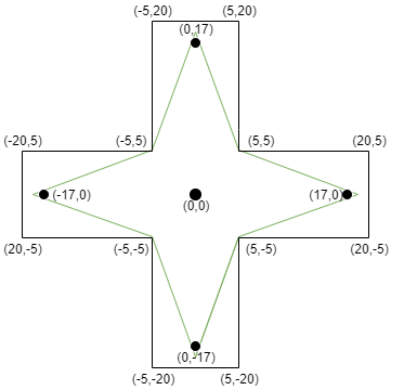
Permasalahan yang dibahas pada Tugas Akhir ini adalah perhitungan untuk mencari nilai x yang didefinisikan oleh persamaan (2.1).

$$x = \sum_{i=0}^{n-1} \text{RCH}_i \tag{2.1}$$

RCH_i pada persamaan (2.1) 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 2.1 dan 2.2 merupakan contoh dari permasalahan ISUN1.



Gambar 2.1: Ilustrasi Contoh Kasus Tanpa Solusi



Gambar 2.2: Ilustrasi Contoh Kasus

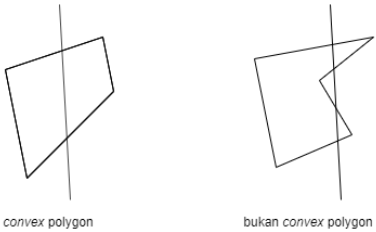
2.2. Convex Polygon

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

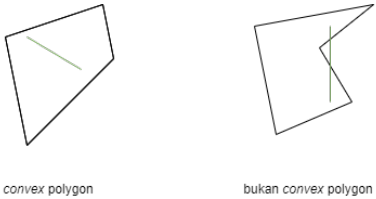
1. Sebuah garis lurus yang di gambar melewati sebuah *convex* polygon akan berpotongan maksimal 2 kali. Ilustrasi dapat dilihat pada gambar 2.3.
2. Jika dua titik sembarang diambil dan ditarik garis antara keduanya, tidak ada bagian dari garis yang berada di luar polygon. Ilustrasi dapat dilihat pada gambar 2.4.

2.2.1. Relative Convex Polygon

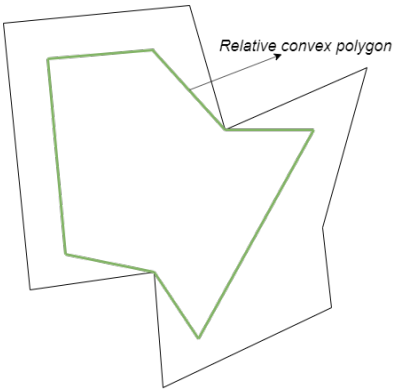
Relative convex polygon merupakan penurunan dari *convex polygon* tetapi ada beberapa sisi dari polygon tersebut berbentuk *convace* atau cekung kedalam dikarenakan adanya batasan dari luar seperti polygon atau segmen garis lainnya. Ilustrasi *relative convex polygon* dapat dilihat pada gambar 2.5.



Gambar 2.3: Ilustrasi Properti Convex Polygon 1



Gambar 2.4: Ilustrasi Properti Convex Polygon 2



Gambar 2.5: Ilustrasi Relative Convex Polygon

2.3. Strategi Penyelesaian Permasalahan

Pada subbab ini akan dipaparkan mengenai strategi penyelesaian masalah klasik pada daring SPOJ dengan kode ISUN1 meng-

gunakan algoritma reduksi polygon. Secara singkat, strategi penyelesaian masalah dari ISUN1 menggunakan algoritma reduksi polygon 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 polygon 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 convex hull* dengan sedikit modifikasi. Modifikasi yang dilakukan adalah ketika 3 buah titik pembentuk polygon yang konsekutif membuat *convex* maka titik tengah 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 2.3.1 dan 2.3.2.

2.3.1. Pemrosesan Titik Pembentuk Polygon yang Membentuk Convex

Pemrosesan titik pembentuk polygon dapat dilakukan dengan cara melakukan *traversing* terhadap semua vertex pembentuk polygon. Untuk setiap vertex p_i yang di periksa, hitung orientasi(secara berlawanan arah 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 2.5). Pencarian titik yang berada di dalam segitiga ABC berguna untuk mencari pengganti vertex p_i sebagai pembentuk polygon luarnya.

2.3.2. Convex Hull dari Titik yang Berada di Dalam Polygon

Melanjutkan dari subbab 2.3.1, ketika sudah mendapatkan $T(ABC)$, lakukan pencarian *Convex Hull* dari titik - titik tersebut menggunakan algoritma *monotone chain* (dapat dilihat pada subbab 2.4.2.2). Kemudian sisipkan semua titik yang membentuk *Convex Hull* diantara vertex p_{i-1}, p_{i+1} untuk me-rekonstruksi polygon luar yang sudah di reduksi.

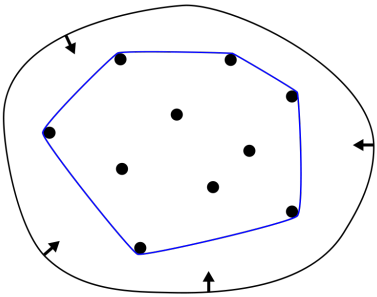
2.4. Convex Hull

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 (2.2) dan ilustrasi *convex hull* ada pada gambar 2.6.

$$Conv(S) = \left\{ \sum_{i=1}^{|S|} a_i s_i \Big| (\forall i : a_i \geq 0 \wedge \sum_{i=1}^{|S|} a_i = 1) \right\} \tag{2.2}$$

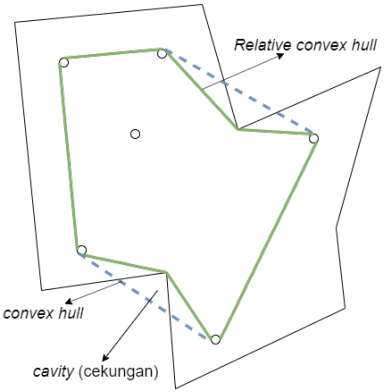
2.4.1. Relative Convex Hull

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



Gambar 2.6: Ilustrasi Convex Cull

ty (cekungan kedalam) yang diakibatkan atau relatif terhadap sesuatu yang membatasi *convex hull* tersebut. Ilustrasi *relative convex hull* dapat dilihat pada gambar 2.7.



Gambar 2.7: Ilustrasi Relative Convex Hull

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

Tabel 2.1: 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	Polygon Sederhana atau Polyline

orientasi antara tiga titik dapat digunakan persamaan 2.3.

$$\begin{aligned} \vec{u} &= (B_x - A_x)x + (B_y - A_y)y \\ \vec{v} &= (C_x - A_x)x + (C_y - A_y)y \end{aligned} \tag{2.3}$$

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

2.4.2. Algoritma Convex Hull

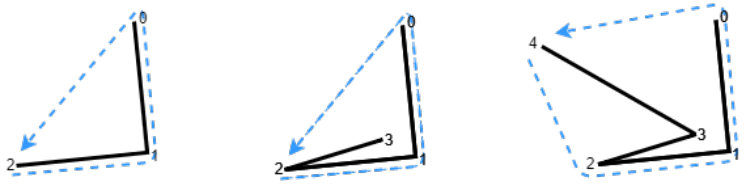
Ada beberapa algoritma yang dapat digunakan untuk mencari sebuah *convex hull*, untuk melihat perbandingan dari beberapa algoritma dapat dilihat pada tabel 2.1. Berdasarkan tabel 2.1, penulis memilih 2 algoritma yang akan digunakan pada buku ini.

2.4.2.1. Algoritma Melkman Convex Hull

Algoritma melkman *convex hull* merupakan algoritma untuk menghitung rantai polygonal ataupun polygon sederhana dengan waktu linear $\mathcal{O}(n)$[2]. Asumsikan sebuah polygon 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 arah 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 2.1.



Gambar 2.8: Ilustrasi Algoritma Melkman

2.4.2.2. Algoritma Monotone Chain

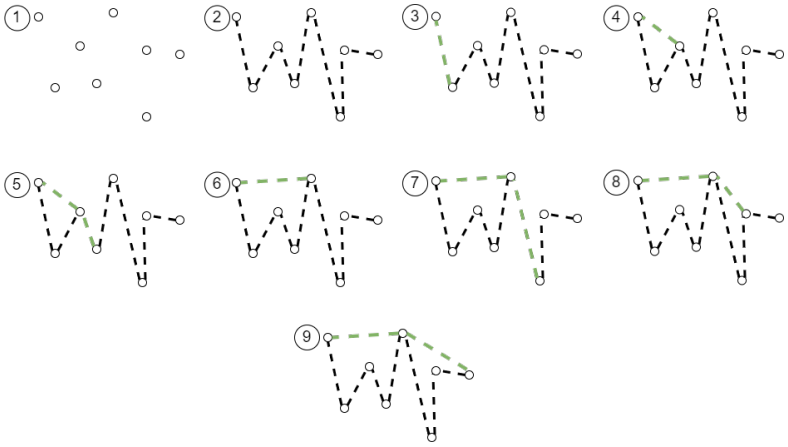
Algoritma *monotone chain* merupakan proses pembentukan *convex hull* dari sekumpulan titik dengan kompleksitas $\mathcal{O}(n \log(n))$[3]. Asumsikan sekumpulan titik S sejumlah n ,$S = \langle s_1, s_2, ..., s_n \rangle$ algoritma ini menggunakan list untuk membentuk

Pseudocode 2.1: 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

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* maka 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 2.2.



Gambar 2.9: Ilustrasi Algoritma Monotone Chain

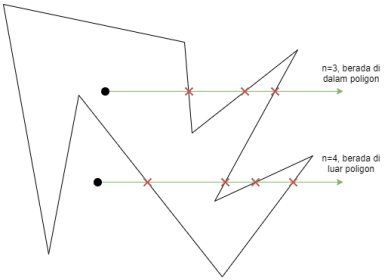
2.5. Point Inside Polygon

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 polygon yang berpotongan dengan garis tersebut. Jika jumlah edge polygon yang berpotongan adalah ganjil, maka titik tersebut berada

Pseudocode 2.2: 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**

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



Gambar 2.10: Ilustrasi Algoritma Point Inside Polygon

BAB III
DESAIN

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

3.1. Desain Umum Sistem

Pada subbab ini akan dijelaskan mengenai gambaran secara umum dari algoritma yang dirancang. Sistem diawali dengan menerima masukan 2 buah bilangan bulat N yang merupakan banyaknya vertex pembentuk polygon luar dan M yang merupakan banyaknya titik yang ada di dalam polygon tersebut. N baris berikutnya berisikan 2 buah bilangan bulat x_i, y_i yang merupakan koordinat dari vertex pembentuk polygon luar terurut berlawanan arah jarum jam. M baris berikutnya berisikan dua buah bilangan bulat $x1_i$, $y1_i$ yang merupakan koordinat dari titik yang ada di dalam polygon.

3.2. Desain Fungsi Main

Fungsi MAIN merupakan fungsi yang bertanggung jawab untuk menerima masukan yang sudah dijelaskan pada 3.1 untuk dilakukan proses sekanjutnya. Pseudocode fungsi MAIN dapat dilihat pada pseudocode 3.1. Fungsi INPUT merupakan fungsi untuk menerima masukan, dan fungsi PRINT merupakan fungsi untuk menampilkan hasil.

3.3. Desain Class Point

Class POINT adalah class untuk menyimpan titik dalam diagram Kartesius. Pseudocode 3.2 merupakan pseudocode dari class POINT. Nantinya pada implementasi, class ini akan melakukan *override* terhadap operator perbandingan.

Pseudocode 3.1: Fungsi MAIN

```
1:  while (N ← INPUT()) and N ≠ EOF do
2:      M ← INPUT()
3:      perimeter ← POLYGON
4:      trees ← Array POINT
5:      for i ← 1, N do
6:          x_i, y_i ← INPUT()
7:          perimeter.P[i] ← POINT(x_i, y_i, false)
8:      end for
9:      if M = 1 or M = 0 then
10:         PRINT(0)
11:         CONTINUE
12:     end if
13:     for i ← 1, M do
14:         x1_i, y1_i ← INPUT()
15:         trees ← POINT(x1_i, y1_i, true)
16:     end for
17:     ans ←SOLVE(perimeter, trees)
18:     PRINT (ans)
19: end while
```

Nama Variabel	Fungsi Variabel
x	Menyimpan ordinat dari titik tersebut
y	Menyimpan absis dari titik tersebut
$fixed$	untuk membedakan antara titik pembentuk polygon P dan titik yang ada di dalam kumpulan titik S

Tabel 3.1: Nama dan Fungsi Variabel dalam Class POINT

Pseudocode 3.2: Class POINT

1: $x, y \leftarrow$ **double**
2: $fixed \leftarrow$ **boolean**
3: **constructor** POINT()
4: **constructor** POINT($\_x, \_y, \_fixed$)

Nama Variabel	Fungsi Variabel
x	Menyimpan arah vektor absis
y	Menyimpan arah vektor ordinat

Tabel 3.2: Nama dan Fungsi Variabel dalam Class VEC

Class POINT tidak memiliki fungsi karena class ini memang hanya untuk menyimpan suatu titik yang akan digunakan nanti.

Fungsi *Constructor* dari class ini terdiri dari dua jenis. Fungsi *constructor* yang pertama adalah fungsi dengan tanpa parameter, pada *constructor* ini, semua variabel yang ada di dalam class POINT akan di set 0. Fungsi *constructor* kedua adalah fungsi dengan parameter $\_x, \_y, \_fixed$, menyatakan nilai $x, y, fixed$ secara berurutan.

3.4. Desain Class Vec

Class VEC merupakan class yang menyimpan vector dari dua buah titik pada diagram kartesian. Pseudocode 3.3 merupakan pseudocode dari class VEC.

Pseudocode 3.3: Class VEC

1: $x, y \leftarrow$ **double**
2: **constructor** VEC()
3: **constructor** VEC($\_x, \_y$)
4: **constructor** VEC(A, B)

Nama Variabel	Fungsi Variabel
a	Menyimpan nilai a pada persamaan $ax+by+c=0$
b	Menyimpan nilai b pada persamaan $ax+by+c=0$
c	Menyimpan nilai c pada persamaan $ax+by+c=0$

Tabel 3.3: Nama dan Fungsi Variabel dalam class LINE

Class VEC tidak memiliki fungsi karena class ini hanya untuk menyimpan vector dari dua titik yang akan digunakan nanti.

Fungsi *Constructor* dari class ini terdiri dari 3 jenis. Fungsi *constructor* yang pertama adalah fungsi dengan tanpa parameter, pada *constructor* ini, semua variabel yang ada di dalam class VEC akan di set 0. Fungsi *constructor* kedua adalah fungsi dengan parameter $\_x, \_y$, menyatakan nilai x, y secara berurutan. Fungsi *constructor* ketiga adalah fungsi dengan parameter A, B, menyatakan POINT dari titik A dan POINT dari titik B, dimana nantinya nilai x dan y akan didapatkan dari pengurangan koordinat dari POINT A dan POINT B.

3.5. Desain Class Line

Class LINE merupakan class yang bertanggung jawab untuk melakukan operasi-operasi pada garis dalam diagram kartesian. Pseudocode 3.4 merupakan pseudocode dari Class LINE.

Pseudocode 3.4: Class LINE

```
1: a, b, c ← double
2: constructor LINE()
3: constructor LINE(_a, _b, _c)
4: constructor LINE(A, B)
```

Nama Variabel	Fungsi Variabel
P	Menyimpan POINT yang merupakan ujung awal dari sebuah segment garis
Q	Menyimpan POINT yang merupakan ujung akhir dari sebuah segment garis
L	Menyimpan fungsi dari garis yang melalui dua titik tersebut

Tabel 3.4: Nama dan Fungsi Variabel dalam Class SEGMENT

Class LINE tidak memiliki fungsi karena class ini hanya untuk menyimpan nilai dari fungsi $ax + by + c = 0$ yang akan digunakan nanti.

Fungsi *Constructor* dari class ini terdiri dari 3 jenis. Fungsi *constructor* yang pertama adalah fungsi dengan tanpa parameter, pada *constructor* ini, semua variabel yang ada di dalam class LINE akan di set 0. Fungsi *constructor* kedua adalah fungsi dengan parameter $\_a, \_b, \_c$, menyatakan nilai a, b, c secara berurutan. Fungsi *constructor* ketiga adalah fungsi dengan parameter A, B, menyatakan POINT dari titik A dan POINT dari titik B, dimana nantinya nilai a, b dan c akan didapatkan dengan mencari fungsi garis yang melewati POINT A dan POINT B.

3.6. Desain Class Segment

Class SEGMENT merupakan class yang bertanggung jawab untuk menyimpan dan melakukan operasi-operasi pada segment garis dalam diagram kartesian. Pseudocode 3.5 merupakan pseudocode dari class SEGMENT.

Class SEGMENT tidak memiliki fungsi karena class ini hanya untuk menyimpan data dari sebuah segmen garis yang akan digunakan nanti.

Fungsi *Constructor* dari class ini terdiri dari 2 jenis. Fung-

Pseudocode 3.5: Class SEGMENT

```
1: P, Q ← POINT
2: L ← LINE
3: constructor SEGMENT()
4: constructor SEGMENT(_P, _Q)
```

Nama Variabel	Fungsi Variabel
P	Menyimpan array dari POINT yang membentuk polygon tersebut

Tabel 3.5: Nama dan Fungsi Variabel dalam Class POLYGON

si *constructor* yang pertama adalah fungsi dengan tanpa parameter, pada *constructor* ini, semua variabel yang ada di dalam class SEGMENT akan di set 0. Fungsi *constructor* kedua adalah fungsi dengan parameter $\_P, \_Q$, menyatakan POINT dari titik P dan POINT dari titik Q, yang merupakan titik POINT A dan POINT B secara berturut, dan LINE L didapar dengan menggunakan *constructor* LINE dengan parameter P dan Q.

3.7. Desain Class Polygon

Class POLYGON merupakan class yang bertanggung jawab untuk menyimpan dan melakukan operasi-operasi pada polygon pada diagram kartesian. Pseudocode 3.6 merupakan pseudocode dari class POLYGON.

Fungsi-fungsi yang terkandung dalam class ini adalah PREV, NEXT, PERIMETER. Tabel 3.5 menjelaskan variabel dan kegunaannya dalam class POLYGON.

Fungsi *Constructor* dari class ini terdiri dari 2 jenis. Fungsi *constructor* yang pertama adalah fungsi dengan tanpa parameter, pada *constructor* ini, variabel P yang ada di dalam class POLYGON akan di inisialisasi. Fungsi *constructor* kedua adalah fungsi dengan

Pseudocode 3.6: Class POLYGON

```
1: P ← Array POINT
2: constructor POLYGON()
3: constructor POLYGON(_P)
4: function PREV(idx)
5: function NEXT(idx)
6: function PERIMETER()
```

Masukan	Proses	Keluaran
Suatu bilangan bulat idx yang menyatakan index saat ini	mencari index selanjutnya	Suatu bilangan bulat yang menyatakan index selanjutnya

Tabel 3.6: Masukan, Proses, dan Keluaran dari Fungsi NEXT Class POLYGON

parameter $\_P$, menyatakan array POINT dari titik pembentuk polygon tersebut.

Fungsi *next* bertanggung jawab untuk mencari index selanjutnya dari titik yang membentuk polygon. Masukan, proses dan keluaran dari fungsi ini tercantum pada tabel 3.6. Pseudocode fungsi ini dapat dilihat pada pseudocode 3.7.

Pseudocode 3.7: Fungsi NEXT pada class POLYGON

```
Input: idx
1: if idx =SIZE(P)−1 then
2:     return 0
3: else
4:     return idx + 1
5: end if
```

Fungsi *prev* bertanggung jawab untuk mencari index sebe-

Masukan	Proses	Keluaran
Suatu bilangan bulat idx yang menyatakan index saat ini	mencari index sebelumnya	Suatu bilangan bulat yang menyatakan index sebelumnya

Tabel 3.7: Masukan, Proses, dan Keluaran dari Fungsi PREV Class POLYGON

lumnya dari titik yang membentuk polygon. Masukan, proses dan keluaran dari fungsi ini tercantum pada tabel 3.7. Pseudocode fungsi ini dapat dilihat pada pseudocode 3.8.

Pseudocode 3.8: Fungsi PREV pada class POLYGON

Input: idx
1: **if** $idx = 0$ **then**
2: **return** SIZE(P)-1
3: **else**
4: **return** $idx - 1$
5: **end if**

Fungsi *perimeter* bertanggung jawab untuk mencari keliling dari sebuah polygon. Masukan, proses, dan keluaran dari fungsi ini tercantum pada tabel 3.8. Pseudocode funsi ini dapat dilihat pada pseudocode 3.9.

Pseudocode 3.9: Fungsi PERIMETER pada class POLYGON

1: $ret \leftarrow 0$
2: **for** $i \leftarrow 0$ **to** SIZE(P)-1 **do**
3: $ret \leftarrow ret$ EDIST($P[i]$, $P[$ NEXT(i)$]$)
4: **end for**
5: **return** ret

Masukan	Proses	Keluaran
-	Mencari keliling dengan mencari jarak eulerian dari semua titik pembentuk polygon	Suatu bilangan berkoma yang menyatakan keliling dari polygon tersebut

Tabel 3.8: Masukan, Proses, dan Keluaran dari Fungsi PERIMETER Class POLYGON

Masukan	Proses	Keluaran
tiga buah bilangan x, l, r, dimana bilangan x merupakan bilangan yang ingin diketahui apakah berada diantara titik l dan r	Mencari tahu apakan bilangan x berada diantara bilangan l dan r	Sebuah boolean yang menyatakan apakah x berada diantara l dan r

Tabel 3.9: Masukan, Proses, dan Keluaran dari Fungsi BETWEEND

3.8. Fungsi BetweenD

Fungsi BETWEEND bertanggung jawab untuk mencari tahu apakah suatu bilangan x berada diantara bilangan l dan bilangan$r$. Pseudocode dari fungsi BETWEEND dapat dilihat pada pseudocode 3.10. Masukan, proses, dan keluaran dari fungsi ini tercantum pada tabel 3.9.

3.9. Fungsi EDist

Fungsi EDIST bertanggung jawab untuk mencari jarak antara dua titik POINT A dan POINT B dengan menggunakan rumus Pythagoras. Rumus Pythagoras dapat di lihat pada persamaan 3.1. Pseu-

Pseudocode 3.10: Fungsi BETWEEND

Input: x, l, r
1: **if** MIN$(l, r) \leq x + EPS$ and $x \leq$ MAX$(l, r) + EPS$ **then**
2: **return** TRUE
3: **else**
4: **return** FALSE
5: **end if**

Masukan	Proses	Keluaran
Dua buah POINT A dan POINT B yang akan dicari jaraknya	Mencari jarak antara POINT A dan POINT B	Sebuah bilangan berkoma yang menyatakan jarak POINT A dan POINT B

Tabel 3.10: Masukan, Proses, dan Keluaran dari Fungsi EDIST

docode fungsi EDIST dapat dilihat pada pseudocode 3.11. Masukan, proses, dan keluaran dari fungsi ini tercantum pada tabel 3.10.

$$C = \sqrt{A^2 + B^2} \tag{3.1}$$

Pseudocode 3.11: Fungsi EDIST

Input: A, B
1: **return** SQRT$((A * A) + (B * B))$

3.10. Fungsi Cross

Fungsi CROSS bertanggung jawab untuk mencari nilai perkalian *cross* dari dua buah vector. Rumus Pythagoras dapat di lihat pada persamaan 3.2. Pseudocode fungsi CROSS dapat dilihat pada pseudocode 3.12. Masukan, proses, dan keluaran dari fungsi ini

Masukan	Proses	Keluaran
Dua buah VEC U dan VEC V yang akan dicari hasil perkalian silangnya	Mencari nilai perkalian silang dari VEC U dan VEC V	Sebuah bilangan yang menyatakan hasil perkalian silang VEC U dan VEC V

Tabel 3.11: Masukan, Proses, dan Keluaran dari Fungsi CROSS

tercantum pada tabel 3.11.

$$C = (u_x * v_y) - (u_y * v_x) \tag{3.2}$$

Pseudocode 3.12: Fungsi CROSS

Input: U, V

1: **return** $(U.x * V.y) - (U.y * V.x)$

3.11. Fungsi Orientation

Fungsi ORIENTATION bertanggung jawab untuk mencari orientasi dari tiga titik. Pseudocode fungsi ORIENTATION dapat dilihat pada pseudocode 3.13. Masukan, proses, dan keluaran dari fungsi ini tercantum pada tabel 3.12.

Pseudocode 3.13: Fungsi ORIENTATION

Input: O, P, Q

1: $OP \leftarrow$ VEC(O, P)

2: $OQ \leftarrow$ VEC(O, Q)

3: **return** CROSS(OP, OQ)

Masukan	Proses	Keluaran
Tiga buah POINT O, POINT P dan POINT Q yang akan dicari orientasinya	Mencari orientasi antara POINT O, POINT P dan POINT Q	Sebuah bilangan yang menyatakan orientasi antara POINT O, POINT P dan POINT Q

Tabel 3.12: Masukan, Proses, dan Keluaran dari Fungsi ORIENTATION

Masukan	Proses	Keluaran
Dua buah POINT P dan SEGMENT S yang akan dicari tahu apakah POINT P berada di SEGMENT S	Mencari tahu apakah POINT P berada di SEGMENT S	Sebuah boolean yang menyatakan apakah POINT P berada di dalam SEGMENT S

Tabel 3.13: Masukan, Proses, dan Keluaran dari Fungsi ONSEGMENT

3.12. Fungsi OnSegment

Fungsi ONSEGMENT bertanggung jawab untuk mencari tahu apakah sebuah titik POINT P bersentuhan dengan sebuah segment garis SEGMENT S atau tidak. Pseudocode fungsi ONSEGMENT dapat dilihat pada pseudocode 3.14. Masukan, proses, dan keluaran dari fungsi ini tercantum pada tabel 3.13.

3.13. Fungsi ConvexHull

Fungsi CONVEXHULL bertanggung jawab untuk mencari *convex hull* dari sekumpulan titik pts. Algoritma yang digunakan oleh fungsi ini adalah algoritma *Monotone Chain* yang dapat dilihat pada bagian 2.4.2.2. Pseudocode dari fungsi CONVEXHULL yang digu-

Pseudocode 3.14: Fungsi ONSEGMENT

Input: P, S

```
1: if ORIENTATION(S.P, S.Q, P) then
2:     return FALSE
3: else
4:     return (BETWEEND(P.x, S.P.x, S.Q.x) and
   BETWEEND(P.y, S.P.y, S.Q.y))
5: end if
```

Masukan	Proses	Keluaran
Sebuah array POINT pts yang merupakan sekumpulan titik	Mencari POLYGON dengan keliling terkecil dari sekumpulan titik	Sebuah POLYGON yang mengelilingin kumpulan POINT pts

Tabel 3.14: Masukan, Proses, dan Keluaran dari Fungsi CONVEXHULL

nakan dapat dilihat pada Pseudocode 3.15. Masukan, proses, dan keluaran dari fungsi ini tercantum pada tabel 3.14.

3.14. Fungsi InSimplePolygon

Fungsi INSIMPLEPOLYGON bertanggung jawab untuk mencari tahu apakah sebuah titik POINT P berada di dalam POLYGON A atau tidak. Algoritma yang digunakan pada fungsi ini adalah algoritma *point inside polygon* yang dapat dilihat pada bagian 2.5. Pada desain fungsi INSIMPLEPOLYGON ada 3 macam keluaran yaitu -1 untuk menandakan bahwa POINT P berada di dalam POLYGON A, 0 untuk menandakan bahwa POINT P berada di salah satu sisi POLYGON A, dan 1 untuk menandakan bahwa POINT P berada di luar POLYGON A. Pseudocode fungsi INSIMPLEPOLYGON dapat dilihat pada pseudocode 3.16. Masukan, proses, dan keluaran dari

Pseudocode 3.15: Fungsi CONVEXHULL

Input: pts

```
1: SORT(pts)
2: hull ← Array POINT
3: for i ← 0 to 2 do
4:     start ← SIZE(hull)
5:     for pt in pts do
6:         while (SIZE(hull)≥ start + 2) and (ORIENTATION(hull[SIZE(hull)−1], hull[SIZE(hull)−2], pt)≤ 0) do
7:             hull.ERASE(hull[SIZE(hull)−1])
8:         end while
9:         hull.INSERT(pt)
10:    end for
11:    hull.ERASE(hull[SIZE(hull)−1])
12:    REVERSE(pts)
13: end for
14: return POLYGON(hull);
```

Masukan	Proses	Keluaran
Sebuah POINT A dan sebuah POLYGON P yang akan dicari tahu apakah POINT A berada di dalam POLYGON P	Mencari tahu apakah POINT A berada di dalam POLYGON P	Sebuah bilangan yang apakah POINT A berada di dalam POLYGON P

Tabel 3.15: Masukan, Proses, dan Keluaran dari Fungsi INSIMPLEPOLYGON

fungsi ini tercantum pada tabel 3.15.

3.15. Fungsi GetBetween

Fungsi GETBETWEEN bertanggung jawab untuk mencari list POINT yang akan menggantikan POINT yang akan dibuang. Pseudocode fungsi GETBETWEEN dapat dilihat pada pseudocode 3.17. Masukan, proses, dan keluaran dari fungsi ini tercantum pada tabel 3.16.

3.16. Fungsi Solve

Fungsi SOLVE bertanggung jawab untuk mencari *relative convex polygon* yang mengelilingi semua titik yang ada di dalam polygon luar. Pseudocode fungsi SOLVE dapat dilihat pada pseudocode 3.18. Masukan, proses, dan keluaran dari fungsi ini tercantum pada tabel 3.17.

Pseudocode 3.16: Fungsi INSIMPLEPOLYGON

Input: P, A

```
1: ret ← INTEGER
2: for i ← 0 to SIZE(A.P) do
3:     if P = A.P[i] then
4:         return 0
5:     end if
6:     j ← A.NEXT(i)
7:     if ONSEGMENT(P, SEGMENT(A.P[i], A.P[j])) then
8:         return 0
9:     end if
10:    below ← (A.P[i].y < P.y)
11:    if below ≠ (A.P[j].y < P.y) then
12:        o ← ORIENTATION(P, A.P[i], A.P[j])
13:        if o = 0 then
14:            return 0
15:        end if
16:        if below = (o > 0) and below = TRUE then
17:            ret+ = 1
18:        else
19:            if below = (o > 0) and below = FALSE then
20:                ret− = 1
21:            end if
22:        end if
23:    end if
24: end for
25: if ret = 0 then
26:    return 1
27: else
28:    return −1
29: end if
```

Masukan	Proses	Keluaran
Sebuah POLYGON $triangle$ yang merupakan segitiga, QUEUE POINT q yang merupakan pembentuk polygon luar, dan array POINT $trees$ yang merupakan titik yang berada di dalam polygon luar	Mencari POINT mana saja yang akan menggantikan POINT $triangle[1]$ yang akan dibuang	Sebuah LIST POINT yang berisikan daftar POINT yang akan menggantikan POINT $triangle[1]$

Tabel 3.16: Masukan, Proses, dan Keluaran dari Fungsi GETBETWEEN

Masukan	Proses	Keluaran
Sebuah POLYGON $perimeter$ yang merupakan polygon sederhana yang merupakan polygon pembatas, dan array POINT $trees$ yang merupakan titik yang berada di dalam polygon pembatas	Mencari *relative convex hull* dari semua titik $trees$ di dalam POLYGON $perimeter$	Sebuah POLYGON yang merupakan *relative convex hull* dari semua titik $trees$

Tabel 3.17: Masukan, Proses, dan Keluaran dari Fungsi SOLVE

Pseudocode 3.17: Fungsi GETBETWEEN

Input: $triangle, q, trees$

```
points ← Array POINT
while q not EMPTY do
    if INSIMPLEPOLYGON(q.FRONT(), triangle) ≠ 1 then
        points.INSERT(q.FRONT())
    end ifq.POP
end while
for pt in trees do
    if INSIMPLEPOLYGON(pt, triangle) ≠ 1 then
        points.INSERT(pt)
    end if
end for
P ← CONVEXHULL(points)
pts ← Array POINT
i ← 0
while TRUE do
    if P.P[i] = triangle.P[0] then
        if P.P[P.NEXT(i)] = triangle.P[2] then
            i ← P.PREV(i)
            while P.P[i] ≠ triangle.P[2] do
                pts.INSERT(P.P[i])
                i = P.PREV(i)
            end while
        else
            i ← P.NEXT(i)
            while P.P[i] ≠ triangle.P[2] do
                pts.INSERT(P.P[i])
                i = P.NEXT(i)
            end while
        end if
        BREAK
    end if
    i ← P.NEXT(i)
end while
return pts
```

Pseudocode 3.18: Fungsi SOLVE

Input: $perimeter, trees$

```
1: q ← QUEUE
2: for pt in perimeter.P do
3:     q.PUSH pt
4: end for
5: bef ← perimeter.P[perimeter.SIZE() − 1]
6: while TRUE do
7:     erased ← FALSE
8:     count ← q.SIZE()
9:     while count– do
10:        cur ← q.FRONT() q.POP()
11:        if cur.fixed =FALSE and (FINE(q, cur) or cur = bef
   or cur = q.FRONT()) then
12:            erased ← TRUE
13:            triangle ← POLYGON
14:            triangle.P.INSERT(bef, cur, q.FRONT())
15:            points ← GETBETWEEN(triangle, q, trees)
16:            for pt in points do
17:                q.PUSH(pt); bef ← pt
18:            end for
19:        else
20:            q.PUSH(cur); bef ← cur
21:        end if
22:    end while
23:    if erased = FALSE then BREAK
24:    end if
25: end while
26: hull ← array of POINT
27: while q not empty do
28:    hull.INSERT(q.FRONT())
29:    q.POP()
30: end while
31: return POLYGON(hull)
```

[*Halaman ini sengaja dikosongkan*]

BAB IV
IMPLEMENTASI

4.1. Lingkungan implementasi

Lingkungan implementasi dan pengembangan yang dilakukan adalah sebagai berikut.

1. Perangkat Keras
   (a) Processor Intel® Core™ i7-6500U CPU @ 2.50GHz (4 CPUs), 2.6GHz
   (b) Random Access Memory 8192MB
2. Perangkat Lunak
   (a) Sistem Operasi Windows 10 Home Single Language 64-bit
   (b) Dev C++
   (c) Bahasa Pemrograman C++
   (d) Kompiler GCC 7.4.0 (Ubuntu 7.4.0-1ubuntu1 18.04.1) untuk Windows Subsystem Linux

4.2. Implementasi Program Utama

Subbab ini menjelaskan implementasi proses algoritma secara Keseluruhan berdasarkan desain yang telah dijelaskan pada Bab 3. Program ini merupakan program yang digunakan untuk menyelesaikan permasalahan *LL and ErBao*.

4.2.1. Header yang diperlukan

Implementasi algoritma ini membutuhkan lima buah *header* yaitu `iostream`, `vector`, `cmat`, `functional` dan `algorithm` seperti yang terlihat pada Kode Sumber 4.1.

```
1 #pragma GCC optimize("O3")
```

```
2 #pragma GCC target("avx")
3
4 #include <iostream>
5 #include<vector>
6 #include<cmath>
7 #include<map>
8 #include<queue>
9 #include<algorithm>
```

Kode Sumber 4.1: *Header* yang Diperlukan

Selain header, terdapat juga preprocessor *pragma*, digunakan untuk mengganti flag kompiler yang digunakan pada daring SPOJ. *Header* `iostream` berisi fungsi standar input output operasi yang digunakan oleh bahasa C++. *Header* `vector` berisi struktur data yang digunakan untuk menyimpan data array secara dinamis. *Header* `cmath` berisi fungsi-fungsi untuk operasi matematika seperti fungsi `hypot`. *Header* `queue` berisi struktur data yang digunakan untuk menyimpan antrian data. *Header* `algorithm` berisi modul yang memiliki fungsi-fungsii yang sangat berguna dalam membantu mengimplementasi algoritma yang telah dibangun, contohnya adalah fungsi *reverse* dan *sort*. *Header* `map` berisi struktur data untuk menyimpan data *key value*.

4.2.2. Preprocessor

Pre-processor seperti `using` digunakan untuk membuat alias dari tipe data sesungguhnya. Terdapat tiga alias yang digunakan yaitu `push_back(x)` sebagai `pb(x)`, `pop_back(x)` sebagai `pob(x)`, `getchar(x)` sebagai `gc(x)`, dan `for (int i = 0; i < n; i++)` sebagai `FOR(i,n)`. Pre-processor dapat dilihat pada Kode Sumber 4.2.

```
1 #define pb(x) push_back(x)
2 #define pob(x) pop_back(x)
3 #define FOR(i, n) for (int i = 0; i < n; i++)
4 #define gc(x) getchar(x)
```

```
5 using namespace std;
```

Kode Sumber 4.2: *Preprocessor* yang Diperlukan

4.2.3. Variabel Global

Variabel global digunakan untuk memudahkan dalam mengakses data yang digunakan lintas fungsi/struct. Kode sumber implementasi variabel global dapat dilihat pada kode sumber 4.3. Variabel tersebut didefinisikan secara global agar dapat digunakan pada setiap fungsi.

```
1
2 const double EPS = 0.0;
3 const double INF = 1E9;
4 map<point, int> pool;
```

Kode Sumber 4.3: Variabel Global yang Didefinisikan Setelah Definisi Struct Mod64

4.2.4. Implementasi Fungsi Main

Fungsi main adalah implementasi algoritma yang dirancang pada pseudocode 3.1. Implementasi fungsi main dapat dilihat pada Kode Sumber 4.4.

```
 1 int main(){
 2   int kase = 1;
 3   int n, m;
 4   while (cin >> n){
 5     pool.clear();
 6     cin >> m;
 7     polygon perimeter;
 8     vector<point> trees;
 9
10     for (int i = 0; i < n; i++){
11       double a = readint(), b = readint();
12       pool[point(a, b, false)]++;
```

```
13        perimeter.P.push_back(point(a, b, false));
14      }
15
16      if (m == 0 || m == 1){
17        printf("Case #%d: %.3lf\n", kase++, 0.0);
18        continue;
19      }
20
21      for (int i = 0; i < m; i++){
22        double a = readint(), b = readint();
23        pool[point(a, b, true)]++;
24        trees.push_back(point(a, b, true));
25      }
26
27      polygon hasil = solve(perimeter, trees);
28
29      printf("Case #%d: %.3lf\n", kase++,
            hasil.perimeter());
30    }
31  }
```

Kode Sumber 4.4: Fungsi Main

4.2.5. Implementasi Class Point

Pada subbab ini akan dijelaskan mengenai implementasi dari class `Point` pada subbab 3.3 dan pseudocode 3.2. Implementasi dari class `Point` dapat dilihat pada Kode Sumber 4.5.

```
1  struct point{
2    double x, y;
3    bool fixed;
4    point(){
5      x = y = 0.0;
6      fixed = 0;
7    }
8    point(double _x, double _y, bool _fixed =
         false){
9      x = _x;
10     y = _y;
```

```
11      fixed = _fixed;
12    }
13    bool operator<(point other) const{
14      if (y < other.y + EPS)
15        return true;
16      if (y + EPS > other.y)
17        return false;
18      return x < other.x + EPS;
19    }
20    bool operator==(point other) const{
21      return same_d(x, other.x) && same_d(y,
            other.y);
22    }
23  };
```

Kode Sumber 4.5: Struct Point

4.2.6. Implementasi Class Vec

Pada subbab ini akan dijelaskan mengenai implementasi dari class `Vec` pada subbab 3.4 dan pseudocode 3.3. Implementasi dari class `Vec` dapat dilihat pada Kode Sumber 4.6.

```
 1  struct vec{
 2    double x, y;
 3    vec(){
 4      x = y = 0.0;
 5    }
 6    vec(double _x, double _y){
 7      x = _x;
 8      y = _y;
 9    }
10    vec(point A){
11      x = A.x;
12      y = A.y;
13    }
14    vec(point A, point B){
15      x = B.x - A.x;
16      y = B.y - A.y;
17    }
```

```
18 };
```

Kode Sumber 4.6: Struct Vec

4.2.7. Implementasi Class Line

Pada subbab ini akan dijelaskan mengenai implementasi dari class `Line` pada subbab 3.5 dan pseudocode 3.4. Implementasi dari class `Line` dapat dilihat pada Kode Sumber 4.7.

```
 1 struct line{
 2   double a, b, c;
 3   line(){
 4     a = b = c = 0.0;
 5   }
 6   line(double _a, double _b, double _c){
 7     a = _a;
 8     b = _b;
 9     c = _c;
10   }
11   line(point P1, point P2){
12     if (P2 < P1){
13       point T;
14       T = P1;
15       P1 = P2;
16       P2 = T;
17     }
18     if (same_d(P1.x, P2.x))
19       a = 1.0, b = 0.0, c = -P1.x;
20     else
21       a = -(P1.y - P2.y) / (P1.x - P2.x), b =
             1.0, c = -(a * P1.x) - P1.y;
22   }
23   line(point P, double slope){
24     if (same_d(slope, INF))
25       a = 1.0, b = 0.0, c = -P.x;
26     else
27       a = -slope, b = 1.0, c = -(a * P.x) - P.y;
28   }
29   bool operator==(line other) const{
```

```
30      return same_d(a, other.a) && same_d(b,
            other.b) && same_d(c, other.c);
31    }
32  };
```

Kode Sumber 4.7: Struct Line

4.2.8. Implementasi Class Segment

Pada subbab ini akan dijelaskan mengenai implementasi dari class `Segment` pada subbab 3.6 dan pseudocode 3.5. Implementasi dari class `Segment` dapat dilihat pada Kode Sumber 4.8.

```
 1 struct segment{
 2   point P, Q;
 3   line L;
 4   segment(){
 5     point T1;
 6     P = Q = T1;
 7     line T2;
 8     L = T2;
 9   }
10   segment(point _P, point _Q){
11     if (_Q < _P){
12       point T1 = _P;
13       _P = _Q;
14       _Q = T1;
15     }
16     P = _P;
17     Q = _Q;
18     line T2(_P, _Q);
19     L = T2;
20   }
21   bool operator==(segment other) const{
22     return P == other.P && Q == other.Q;
23   }
24 };
```

Kode Sumber 4.8: Struct Segment

4.2.9. Implementasi Class Polygon

Pada subbab ini akan dijelaskan mengenai implementasi dari class `Polygon` pada subbab 3.7 dan pseudocode 3.6. Implementasi dari class `Polygon` dapat dilihat pada Kode Sumber 4.9.

```
 1 struct polygon{
 2   vector<point> P;
 3   polygon(){
 4     P.clear();
 5   }
 6   polygon(vector<point> &_P){
 7     P = _P;
 8   }
 9   int prev(int idx){
10     return (idx == 0 ? P.size() - 1 : idx - 1);
11   }
12   int next(int idx){
13     return (idx == P.size() - 1 ? 0 : idx + 1);
14   }
15   double perimeter(){
16     double ret = 0;
17     FOR(i, P.size()){
18       ret += e_dist(P[i], P[next(i)]);
19     }
20     return ret;
21   }
22 };
```

Kode Sumber 4.9: Struct Polygon

4.2.10. Implementasi Fungsi BetweenD

Pada subbab ini akan dijelaskan mengenai implementasi dari fungsi `BetweenD` pada pseudocode 3.10. Implementasi dapat dilihat pada Kode Sumber 4.10.

```
1 double between_d(double x, double l, double r){
2   return (min(l, r) <= x + EPS && x <= max(l, r)
        + EPS);
3 }
```

Kode Sumber 4.10: Fungsi BetweenD

4.2.11. Implementasi Fungsi EDist

Pada subbab ini akan dijelaskan mengenai implementasi dari fungsi `EDist` pada pseudocode 3.11. Implementasi dapat dilihat pada Kode Sumber 4.11.

```
1 double e_dist(point P1, point P2){
2   return hypot(P1.x - P2.x, P1.y - P2.y);
3 }
```

Kode Sumber 4.11: Fungsi EDist

4.2.12. Implementasi Fungsi Cross

Pada subbab ini akan dijelaskan mengenai implementasi dari fungsi `Cross` pada pseudocode 3.12. Implementasi dapat dilihat pada Kode Sumber 4.12.

```
1 double cross(vec u, vec v){
2   return (u.x * v.y - u.y * v.x);
3 }
```

Kode Sumber 4.12: Fungsi Cross

4.2.13. Implementasi Fungsi Orientation

Pada subbab ini akan dijelaskan mengenai implementasi dari fungsi `Orientation` pada pseudocode 3.13. Implementasi dapat dilihat pada Kode Sumber 4.13.

```
1 double orientation(point O, point P, point Q){
2   vec OP(O, P), OQ(O, Q);
3   double c = cross(OP, OQ);
4   return c;
```

```
5 }
```

Kode Sumber 4.13: Fungsi Orientation

4.2.14. Implementasi Fungsi OnSegment

Pada subbab ini akan dijelaskan mengenai implementasi dari fungsi `OnSegment` pada pseudocode 3.14. Implementasi dapat dilihat pada Kode Sumber 4.14.

```
1 bool onSegment(point P, segment S){
2   if (orientation(S.P, S.Q, P) != 0.0)
3     return false;
4   return between_d(P.x, S.P.x, S.Q.x) &&
        between_d(P.y, S.P.y, S.Q.y);
5 }
```

Kode Sumber 4.14: Fungsi OnSegment

4.2.15. Implementasi Fungsi ConvexHull

Pada subbab ini akan dijelaskan mengenai implementasi dari fungsi `ConvexHull` pada pseudocode 3.15. Implementasi dapat dilihat pada Kode Sumber 4.15.

```
1 polygon convexHull(vector<point> &pts){
2   sort(pts.begin(), pts.end());
3   vector<point> hull;
4   for (int i = 0; i < 2; i++){
5     int start = (int)hull.size();
6     for (int i = 0; i < pts.size(); i++){
7       point pt = pts[i];
8       while ((int)hull.size() >= start + 2 &&
            orientation(hull[(int)hull.size() - 1],
            hull[(int)hull.size() - 2], pt) <= 0.0)
9         hull.pob();
10      hull.pb(pt);
11    }
12    hull.pop_back();
```

```
13      reverse(pts.begin(), pts.end());
14    }
15    if ((int)hull.size() == 2 && hull[0] == hull[1])
16      hull.pob();
17    return polygon(hull);
18  }
```

Kode Sumber 4.15: Fungsi ConvexHull

4.2.16. Implementasi Fungsi InSimplePolygon

Pada subbab ini akan dijelaskan mengenai implementasi dari fungsi `inSimplePolygon` pada pseudocode 3.16. Implementasi dapat dilihat pada Kode Sumber 4.16.

```
 1 int inSimplePolygon(point P, polygon &A){
 2   int ret = 0;
 3   FOR(i, A.P.size()){
 4     if (P == A.P[i])
 5       return 0;
 6     int j = A.next(i);
 7     if (onSegment(P, segment(A.P[i], A.P[j])))
 8       return 0;
 9     bool below = (A.P[i].y < P.y);
10     if (below != (A.P[j].y < P.y)){
11       double o = orientation(P, A.P[i], A.P[j]);
12       if (o == 0.0)
13         return 0;
14       if (below == (o > 0.0))
15         ret += below ? 1 : -1;
16     }
17   }
18   return ret == 0 ? 1 : -1;
19 }
```

Kode Sumber 4.16: Fungsi InSimplePolygon

4.2.17. Implementasi Fungsi GetBetween

Pada subbab ini akan dijelaskan mengenai implementasi dari fungsi `GetBetween` pada pseudocode 3.17. Implementasi dapat dilihat pada Kode Sumber 4.17.

```
 1 vector<point> getBetween(polygon &triangle,
      queue<point> q, vector<point> trees){
 2   vector<point> points;
 3
 4   while (!q.empty()){
 5     if (inSimplePolygon(q.front(), triangle) !=
         1){
 6       points.pb(q.front());
 7     }
 8     q.pop();
 9   }
10   for (int i = 0; i < trees.size(); i++){
11     if (inSimplePolygon(trees[i], triangle) != 1){
12       points.pb(trees[i]);
13     }
14   }
15
16   polygon P = convexHull(points);
17
18   vector<point> pts;
19   int i = 0;
20   while (1){
21     if (P.P[i] == triangle.P[0]){
22       if (P.P[P.next(i)] == triangle.P[2]){
23         i = P.prev(i);
24         while (!(P.P[i] == triangle.P[2])){
25           pts.pb(P.P[i]);
26           i = P.prev(i);
27         }
28       }
29       else{
30         i = P.next(i);
31         while (!(P.P[i] == triangle.P[2])){
32           pts.pb(P.P[i]);
33           i = P.next(i);
```

```
34          }
35        }
36        break;
37      }
38      i = P.next(i);
39    }
40    return pts;
41  }
```

Kode Sumber 4.17: Fungsi GetBetween

4.2.18. Implementasi Fungsi Solve

Pada subbab ini akan dijelaskan mengenai implementasi dari fungsi `Solve` pada pseudocode 3.18. Implementasi dapat dilihat pada Kode Sumber 4.18.

```
 1 polygon solve(polygon &perimeter, vector<point>
       &trees){
 2   queue<point> q;
 3   for (int i = 0; i < perimeter.P.size(); i++){
 4     q.push(perimeter.P[i]);
 5   }
 6
 7   point bef = perimeter.P[perimeter.P.size() - 1];
 8   while (1){
 9     bool erased = 0;
10     int count = q.size();
11     while (count--){
12       point cur = q.front();
13       q.pop();
14       pool[cur]--;
15       if (!cur.fixed && (!find(q, cur) || cur ==
             bef || cur == q.front()) &&
             orientation(cur, bef, q.front()) <=
             0.0){
16         erased = true;
17         polygon triangle;
18         triangle.P.pb(bef);
19         triangle.P.pb(cur);
```

```
20          triangle.P.pb(q.front());
21          vector<point> points =
                getBetween(triangle, q, trees);
22          for (int i = 0; i < points.size(); i++){
23            q.push(points[i]);
24            pool[points[i]]++;
25            bef = points[i];
26          }
27        }
28        else{
29          q.push(cur);
30          pool[cur]++;
31          bef = cur;
32        }
33      }
34      if (!erased)
35        break;
36    }
37
38    vector<point> hull;
39    while (!q.empty()){
40      hull.pb(q.front());
41      q.pop();
42    }
43    return polygon(hull);
44  }
```

Kode Sumber 4.18: Fungsi Solve

BAB V
UJI COBA DAN EVALUASI

Pada bab ini dijelaskan tentang uji coba dan evaluasi dari implementasi yang telah dilakukan pada tugas akhir ini.

5.1. Lingkungan Uji Coba

Lingkungan uji coba digunakan untuk uji coba kebenaran adalah salah satu sistem yang digunakan situs penilaian daring Sphere Online Judge, yaitu kluster *Cube* dengan spesifikasi sebagai berikut :

1. Perangkat Keras
   (a) Processor Intel Xeon E3-1220 v5 (5 CPUs)
   (b) Random Access Memory 1536MB
2. Perangkat Lunak
   (a) Kompiler GCC 6.3.0

Lingkungan uji coba yang digunakan untuk melakukan uji coba kinerja menggunakan komputer pribadi penulis yang memiliki spesifikasi sebagai berikut

1. Perangkat Keras
   (a) Processor Intel® Core™ i7-6500U CPU @ 2.50GHz (4 CPUs), 2.6GHz
   (b) Random Access Memory 8192MB
2. Perangkat Lunak
   (a) Sistem Operasi Windows 10 Home Single Language 64-bit
   (b) Dev C++
   (c) Bahasa Pemrograman C++

(d) Kompiler GCC 7.4.0 (Ubuntu 7.4.0-1ubuntu1 18.04.1) untuk Windows Subsystem Linux

5.2. Skenario Uji Coba

Subbab ini akan menjelaskan hasil pengujian program untuk menyelesaikan permasalahan LL and ErBao. Metode pengujian yang dilakukan adalah sebagai berikut.

1. Pengujian luar. Pengujian ini menggunakan Online Judge untuk menguji kebenaran dan kinerja program.
2. Pengujian lokal. Pengujian ini menggunakan mesin yang digunakan dalam pengembangan untuk mengukur kinerja program.

Dalam pengujian lokal, dibuat beberapa kasus uji berdasarkan batasan yang ada pada soal.

Untuk pengujian luar, uji coba dilakukan dengan mengirimkan kode program dengan algoritma *Melkman convex hull* yang dimodifikasi ke situs penilaian daring Sphere Online Judge sebanyak 10 kali.

5.3. Uji Coba Kebenaran

Pada subbab ini akan dibahas mengenai uji coba kebenaran yang dilakukan dengan mengirim kode sumber terkait ke dalam situs penilaian daring Sphere Online Judge. Bukti hasil pengujian dapat dilihat pada gambar 5.1.

Gambar 5.1: Hasil Uji Coba Kebenaran Situs Penilaian Sphere Online Judge

5.4. Uji Coba Kinerja Lokal

Pada subbab ini akan ditampilkan hasil uji coba kinerja dari algoritma *Melkman convex hull* yang di modifikasi untuk menyelesaikan permasalahan LL and ErBao. Pengujian dilakukan terhadap kelompok masukan yang telah dijelaskan pada subbab 5.2. Detil masukan dapat dilihat pada Lampiran A. Langkah pengujian kinerja dilakukan dengan langkah sebagai berikut:

1. Rekam waktu tepat sebelum komputasi penyelesaian masalah.
2. Melakukan komputasi penyelesaian masalah untuk masukan kasus uji yang sama sebanyak 10 kali secara berturut-turut.
3. Rekam waktu tepat setelah komputasi dengan mengurangi waktu selesai komputasi dengan waktu sebelum komputasi.
4. Ulangi untuk semua kasus uji.

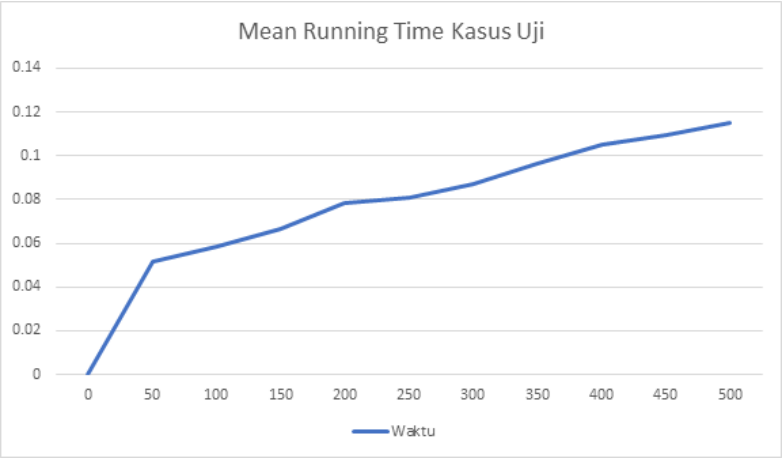
Grafik 5.2 menampilkan rata-rata kinerja masing-masing metode. Pada grafik tersebut dapat dilihat bahwa rata-rata *running time* reduksi polygon dengan algoritma *Melkman convex hull* berbanding lurus dengan banyaknya titik di dalam polygon tersebut.

5.5. Evaluasi Kebenaran Uji Coba Lokal

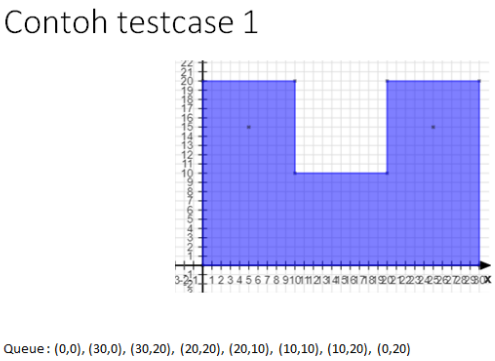
Evaluasi dilakukan dengan memeriksa hasil keluaran program apakah sama dengan contoh keluaran pada Sphere Online Judge 5637 LL and ErBao. Tabel kebenaran dapat dilihat pada gambar 5.1

Gambar 5.3 menunjukan ilustrasi kondisi awal program sebelum iterasi dimulai.

Pada awal iterasi ke-1, memeriksa titik $(0, 0)$. Titik tersebut akan dibuang karena titik tersebut merupakan titik luar dan orientasi terhadap titik sebelumnya dan sesudahnya membentuk *convex*. Sebelum membuang titik tersebut, program membuat segitiga dengan titik sebelumnya, titik tersebut dan titik sesudahnya. Kemudian pro-



Gambar 5.2: Grafik Mean Running Time Kasus Uji



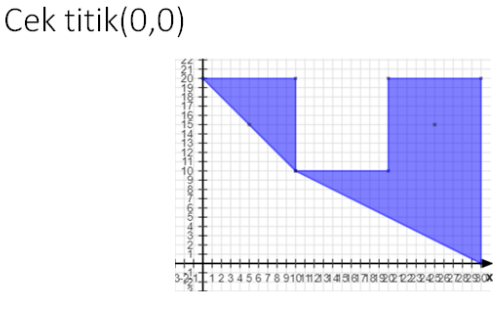
Gambar 5.3: Ilustrasi Kondisi Awal

gram mencari semua titik yang berada di dalam segitiga tersebut. Selanjutnya program mencari *convex hull* dari semua titik yang didapatkan dan disisipkan ke queue polygon luar untuk menggantikan

Data Masukan	Hasil Keluaran
8 2 0 0 30 0 30 20 20 20 20 10 10 10 10 20 0 20 5 15 25 15	Case #1: 48.284

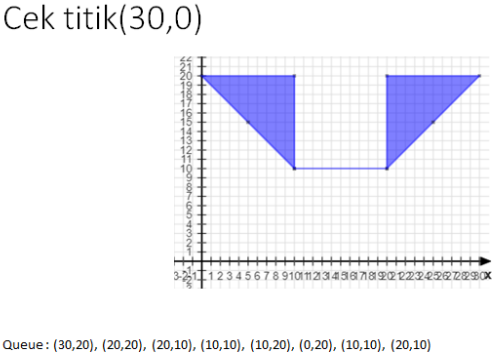
Tabel 5.1: Tabel Data Uji Coba Kebenaran Lokal dengan Sampel Data

titik yang dibuang. kondisi setelah iterasi ke-1 dapat dilihat pada gambar 5.4.



Gambar 5.4: Ilustrasi Iterasi 1

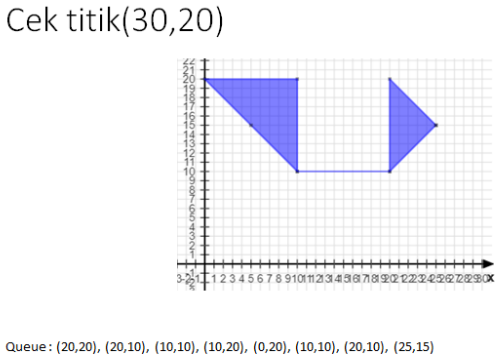
Pada awal iterasi ke-2, memeriksa titik $(30, 0)$. Titik tersebut akan dibuang karena titik tersebut merupakan titik luar dan orientasi terhadap titik sebelumnya dan sesudahnya membentuk *convex*. Sebelum membuang titik tersebut, program membuat segitiga dengan titik sebelumnya, titik tersebut dan titik sesudahnya. Kemudian program mencari semua titik yang berada di dalam segitiga tersebut. Selanjutnya program mencari *convex hull* dari semua titik yang didapatkan dan disisipkan ke queue polygon luar untuk menggantikan titik yang dibuang. kondisi setelah iterasi ke-2 dapat dilihat pada gambar 5.5.



Gambar 5.5: Ilustrasi Iterasi 2

Pada awal iterasi ke-3, memeriksa titik $(30, 20)$. Titik tersebut akan dibuang karena titik tersebut merupakan titik luar dan orientasi terhadap titik sebelumnya dan sesudahnya membentuk *convex*. Sebelum membuang titik tersebut, program membuat segitiga dengan titik sebelumnya, titik tersebut dan titik sesudahnya. Kemudian program mencari semua titik yang berada di dalam segitiga tersebut. Selanjutnya program mencari *convex hull* dari semua titik yang didapatkan dan disisipkan ke queue polygon luar untuk

menggantikan titik yang dibuang. kondisi setelah iterasi ke-3 dapat dilihat pada gambar 5.6.



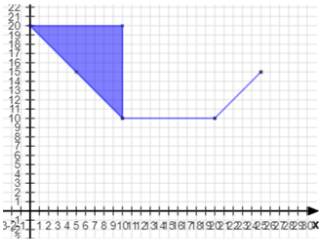
Gambar 5.6: Ilustrasi Iterasi 3

Pada awal iterasi ke-4, memeriksa titik $(20, 20)$. Titik tersebut akan dibuang karena titik tersebut merupakan titik luar dan orientasi terhadap titik sebelumnya dan sesudahnya membentuk *convex*. Sebelum membuang titik tersebut, program membuat segitiga dengan titik sebelumnya, titik tersebut dan titik sesudahnya. Kemudian program mencari semua titik yang berada di dalam segitiga tersebut. Selanjutnya program mencari *convex hull* dari semua titik yang didapatkan dan disisipkan ke queue polygon luar untuk menggantikan titik yang dibuang. kondisi setelah iterasi ke-4 dapat dilihat pada gambar 5.7.

Pada awal iterasi ke-5, memeriksa titik $(20, 10)$. Titik tersebut akan tidak akan dibuang karena titik tersebut merupakan titik luar tetapi orientasi terhadap titik sebelumnya dan sesudahnya membentuk *concave*. kondisi setelah iterasi ke-5 dapat dilihat pada gambar 5.8.

Pada awal iterasi ke-6, memeriksa titik $(10, 10)$. Titik ter-

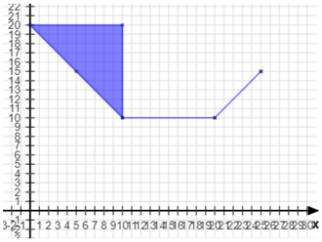
Cek titik(20,20)



Queue : (20,10), (10,10), (10,20), (0,20), (10,10), (20,10), (25,15)

Gambar 5.7: Ilustrasi Iterasi 4

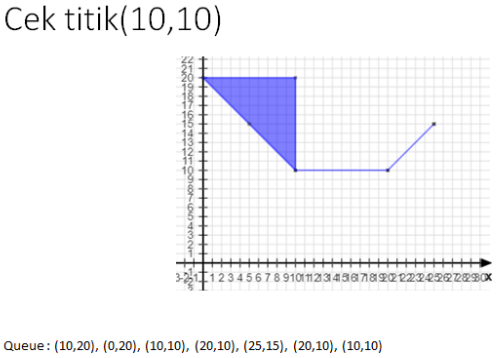
Cek titik(20,10)



Queue : (10,10), (10,20), (0,20), (10,10), (20,10), (25,15), (20,10)

Gambar 5.8: Ilustrasi Iterasi 5

sebut akan tidak akan dibuang karena titik tersebut merupakan titik luar tetapi orientasi terhadap titik sebelumnya dan sesudahnya membentuk *concave*. kondisi setelah iterasi ke-6 dapat dilihat pada gambar 5.9.

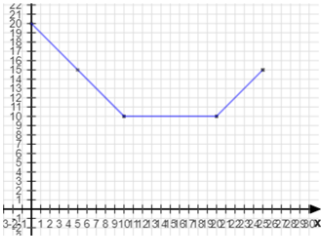


Gambar 5.9: Ilustrasi Iterasi 6

Pada awal iterasi ke-7, memeriksa titik $(10, 20)$. Titik tersebut akan dibuang karena titik tersebut merupakan titik luar dan orientasi terhadap titik sebelumnya dan sesudahnya membentuk *convex*. Sebelum membuang titik tersebut, program membuat segitiga dengan titik sebelumnya, titik tersebut dan titik sesudahnya. Kemudian program mencari semua titik yang berada di dalam segitiga tersebut. Selanjutnya program mencari *convex hull* dari semua titik yang didapatkan dan disisipkan ke queue polygon luar untuk menggantikan titik yang dibuang. kondisi setelah iterasi ke-7 dapat dilihat pada gambar 5.10.

Pada awal iterasi ke-8, memeriksa titik $(0, 20)$. Titik tersebut akan dibuang karena titik tersebut merupakan titik luar dan orientasi terhadap titik sebelumnya dan sesudahnya membentuk *convex*. Sebelum membuang titik tersebut, program membuat segitiga dengan titik sebelumnya, titik tersebut dan titik sesudahnya. Kemudian program mencari semua titik yang berada di dalam segitiga tersebut. Selanjutnya program mencari *convex hull* dari semua titik yang didapatkan dan disisipkan ke queue polygon luar untuk menggantikan

Cek titik(10,20)

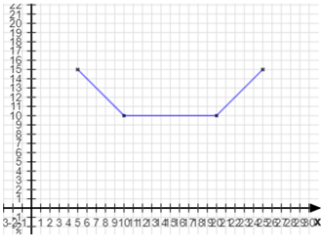


Queue: (0,20), (10,10), (20,10), (25,15), (20,10), (10,10)

Gambar 5.10: Ilustrasi Iterasi 7

titik yang dibuang. kondisi setelah iterasi ke-8 dapat dilihat pada gambar 5.11.

Cek titik(0,20)

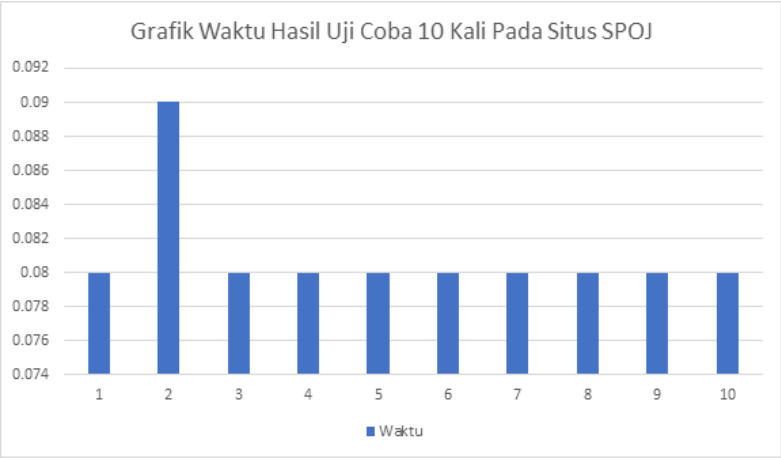


Queue: (10,10), (20,10), (25,15), (20,10), (10,10), (5,15)

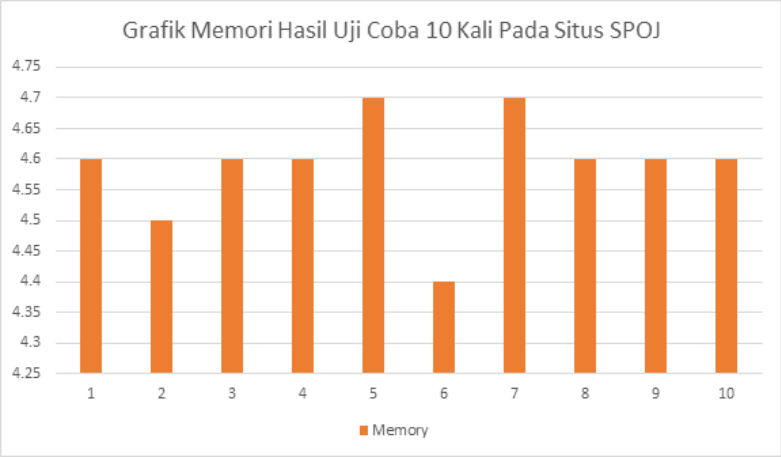
Gambar 5.11: Ilustrasi Iterasi 8

5.6. Uji Coba Kinerja Luar

Pada subbab ini akan ditampilkan hasil uji coba kinerja dari algoritma *Melkman Convex Hull* yang dimodifikasi. Pengujian dilakukan dengan cara mengirimkan kode program ke situs penilaian daring Sphere Online Judge. Detil mengenai hasil uji kinerja dapat dilihat pada Lampiran. Rata-rata hasil pengumpulan kode berkas dengan algoritma *Melkman Convex Hull* adalah 0.08 detik dengan memori 4.6 MB. Hasil uji coba pada situs Sphere Online Judge dapat dilihat pada gambar 5.12 dan 5.13.



Gambar 5.12: Grafik Waktu Uji Coba 10 Kali pada Situs SPOJ



Gambar 5.13: Grafik Memori Uji Coba 10 Kali pada Situs SPOJ

BAB VI
KESIMPULAN

Pada bab ini dijelaskan mengenai kesimpulan dari hasil uji coba yang telah dilakukan serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

6.1. Kesimpulan

Berdasarkan penjabaran di bab-bab sebelumnya, dapat disimpulkan beberapa poin terkait penyelesaian permasalahan LL and Erbao.

1. Permasalahan LL and ErBao dapat diselesaikan dengan melakukan reduksi polygon luar terhadap titik didalamnya.
2. Permasalahan LL and Erbao dapat diselesaikan dengan batasan pada soal dapat diselesaikan dengan reduksi polygon dengan waktu minimum 0.08 detik, waktu maksimum 0.08 detik, dan memori minimum 4.4 MB, memori maksimum 4.7 MB.
3. Algoritma *Melkman convex hull* terbukti efektif untuk melakukan reduksi polygon untuk mencari *relative convex polygon*.

6.2. Saran

Pada Tugas Akhir kali ini tentunya terdapat kekurangan derta nilai-nilai yang dapat penulis ambil. Berikut adalah saran-saran yang dapat diambil melalui Tugas Akhir ini:

1. Untuk kedepannya, algoritma pada tugas akhir ini dapat menjadi bahan riset untuk mencari optimasi lebih lanjut.
2. Metode reduksi polygon dengan menggunakan algoritma

Melkman convex hull yang dimodifikasi dapat digunakan untuk mencari *relative convex hull* dengan polygon yang membatasi segment garis ataupun polygon sederhana.

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/.