**Penyelesaian** *Convex Hull* **dengan Algoritma** *Divide and Conquer*

**SEMESTER I TAHUN 2017/2018**

Diajukan untuk memenuhi Tugas Kecil I IF2211 Strategi Algoritma



Oleh:

**Yasya Rusyda Aslina 13516091**

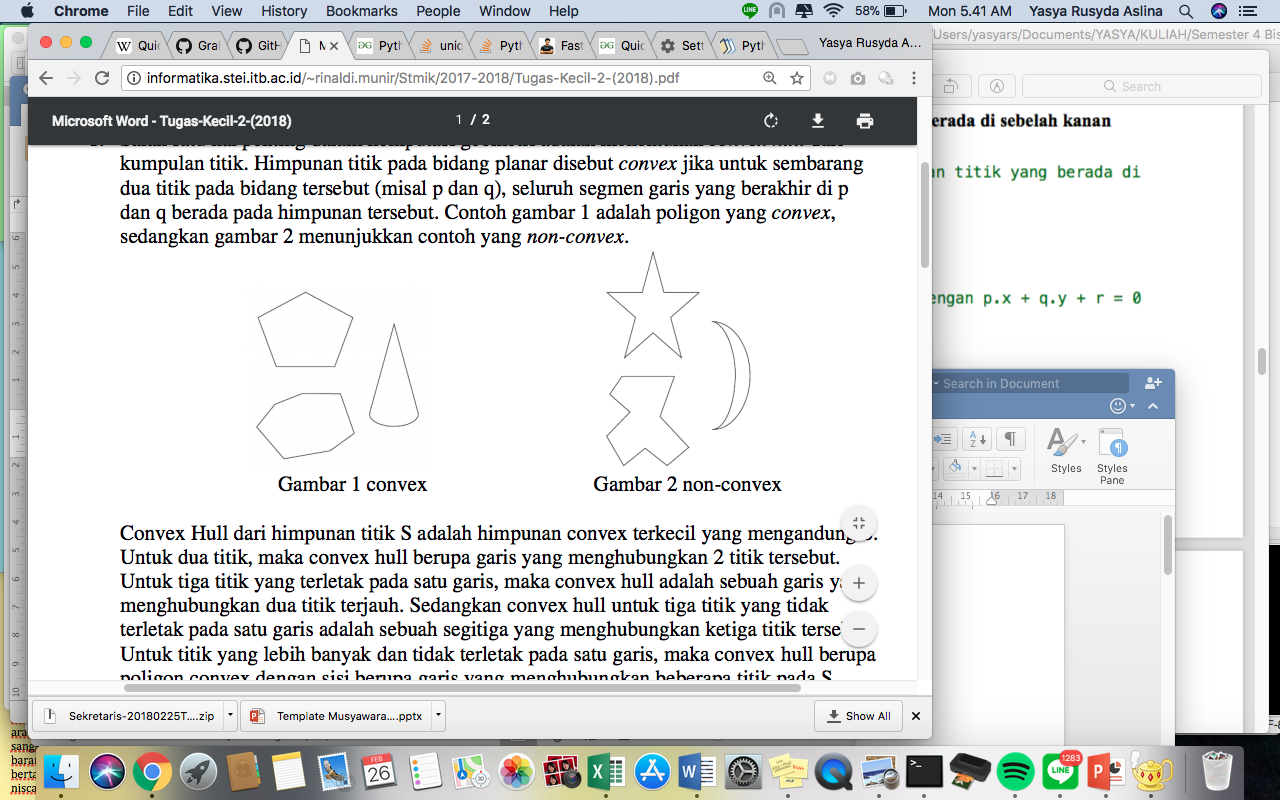
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
BANDUNG**

**2018**

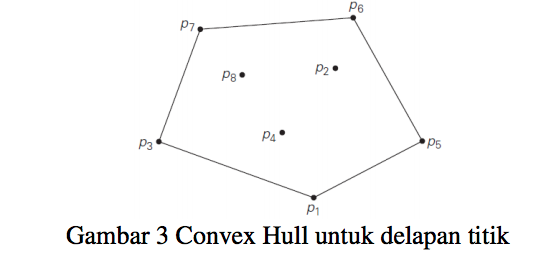
# **BAB 1**

**DESKRIPSI MASALAH**

1. Salah satu hal penting dalam komputasi geometri adalah menentukan convex hull dari kumpulan titik. Himpunan titik pada bidang planar disebut convex jika untuk sembarang dua titik pada bidang tersebut (misal p dan q), seluruh segmen garis yang berakhir di p dan q berada pada himpunan tersebut. Contoh gambar 1 adalah poligon yang convex, sedangkan gambar 2 menunjukkan contoh yang non-convex.



Convex Hull dari himpunan titik S adalah himpunan convex terkecil yang mengandung S. Untuk dua titik, maka convex hull berupa garis yang menghubungkan 2 titik tersebut. Untuk tiga titik yang terletak pada satu garis, maka convex hull adalah sebuah garis yang menghubungkan dua titik terjauh. Sedangkan convex hull untuk tiga titik yang tidak terletak pada satu garis adalah sebuah segitiga yang menghubungkan ketiga titik tersebut. Untuk titik yang lebih banyak dan tidak terletak pada satu garis, maka convex hull berupa poligon convex dengan sisi berupa garis yang menghubungkan beberapa titik pada S. Contoh convex hull untuk delapan titik dapat dilihat pada gambar 3.



Pemanfaatan dari convex hull ini cukup banyak. Pada animasi komputer, pemindahan suatu objek akan lebih mudah dengan memindahkan convex hull objek untuk collision detection. Pada bidang statistik, convex hull juga dapat mendeteksi outliers pada kumpulan data. Conves hull juga dapat digunakan dalam persoalan optimasi, karena penentuan titik ekstrimnya dapat membatasi kandidat nilai optimal yang diperiksa.

1. Mahasiswa ditugaskan untuk membuat sebuah aplikasi sederhana dalam Bahasa C++/Java/Pyhton (pilih salah satu) untuk menentukan convex hull dari kumpulan titik yang diberikan dalam dua dimensi, dengan pendekatan quick hull (divide and conquer). Masukan dari program adalah banyaknya titik (yaitu n), dan kemudian titik sebanyak n dibangkitkan secara acak oleh program. Setiap titik dinyatakan dengan korordinta (x, y). Luaran program adalah himpunan titik yang membentuk convex hull, dan waktu yang diperlukan (tuliskan spesifikasi komputer yang digunakan). Bonus (10) jika dapat menampilkan semua titik dan convex hull yang terbentuk. Khusus untuk proses penggambaran dapat menggunakan library yang tersedia bebas (freeware).

# **BAB 2**

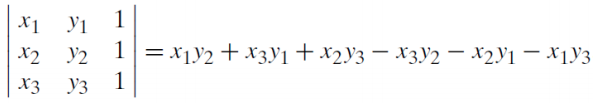
**ALGORITMA DIVIDE AND CONQUER**

* 1. **Definisi**

*Divide and Conquer*  *Divide*: membagi masalah menjadi beberapa upa- masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama),  *Conquer*: memecahkan (menyelesaikan) masing- masing upa-masalah (secara rekursif), dan  *Combine:* mengabungkan solusi masing-masing upa- masalah sehingga membentuk solusi  masalah semula.  Obyek permasalahan yang dibagi adalah masukan (*input*) atau *instances* yang berukuran *n*: tabel (larik), matriks, eksponen, dan sebagainya, bergantung pada masalahnya.  Tiap-tiap upa-masalah mempunyai karakteristik yang sama (*the same type*) dengan karakteristik masalah asal,  sehingga metode *Divide and Conquer* lebih natural diungkapkan dalam skema rekursif.

* 1. **Strategi Algoritma Divide and Conquer**

Strategi algoritma divide and conquer yang digunakan yaitu dengan memanfaatkan partisi titik yang berada pada bagian terluar. Langkah pertama yang dilakukan yaitu membagi titik-titik menjadi dua bagian. Langkah ini dikerjakan dengan memeriksa posisi titik-titik tersebut dengan garis yang terbentuk dari titik dengan absis terkecil dan titik dengan absis terbesar. Titik-titik yang membentuk garis tersebut terlebih dahulu disimpan ke dalam List Convex Hull. Berikut rumus yang digunakan untuk memeriksa posisi tersebut:

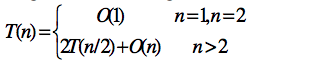
****

Tititk (x3,y3) berada di sebelah kiri dari garis ((x1,y1),(x2,y2)) jika hasil formula positif.

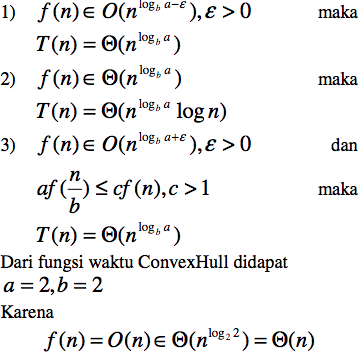
Kemudian untuk setiap partisi, dicari titik yang memiliki jarak maksimum terhadap garis yang menjadi acuan. Titik tersebut juga akan disimpan sebagai elemen Convex Hull. Setelah itu akan diperiksa titik-titik yang berada pada luar area yang dibentuk oleh ketiga titik sebelumnya. Pada proses ini, digunakan rekursif mencari titik dengan jarak maksimum terhadap sebuah garis untuk mengulangi langkah yang dibentuk oleh setiap partisi yang berada pada bagian luar area. Langkah terakhir yang harus dilakukan adalah menghubungkan titik-titik yang telah disimpan di dalam Convex Hull untuk dibentuk menjadi polygon.

Proses pengurutan pada algoritma ini dapat menggunakan pengurutan quicksort yang mempunyai kompleksitas waktu O(n log n) Untuk membagi dua sebuah himpunan dibutuhkan O(n) karena hanya mengiterasi seluruh titik yang sudah terurut pada Q. Demikian juga menggabungkan 2 buah himpunan membutuhkan waktu asimtotik linier karena dalam convex hull sudah otomatis terurut berdasarkan letaknya.

Fungsi waktu untuk ConvexHull adalah



Dimana O(n) adalah kompleksitas waktu untuk membagi himpunan menjadi dua sub himpunan dan mengabungkan dua buah convex hull yang telah dicari. Untuk mencari kompleksitas waktu asimptotik dari bentuk rekurens seperti ini dapat digunakan cara Master Theorem. Pada bentuk T(n) =aT(n/b)+ f(n), jika



**Berdasarkan Teorema Master**

T (n) = Θ(nlog n)

**Kompleksitas** waktu rata-rata menggunakan algoritma ini:

* 1. **Pseudocode**

input(){

Masukan: n yang merupakan jumlah titik-titik yang akan diperiksa

Randomisasi n buah titik-titik dalam area yang telah ditentukan

}

QuickHull(){

//Prosedur ini digunakan untuk mencari convex hull pada sebuah himpunan titik-titik

Convex Hull= [] //Convex Hull diinisialisasi dengan sebuah list kosong

Mencari titik minimum dan maksimum (titik dengan absis terkecil dan absis terbesar) anggap A dan B. Kemudian memasukkan A dan B ke dalam list Convex Hull

Membagi titik-titik yang tersedia ke dalam dua partisi yang dipisahkan oleh garis AB. Misalnya partisi S1 dan S2

S1 merupakan titik-titik yang berada pada bagian kiri garis dan S2 merupakan ttik-titik yang berada pada kanan garis.

Mencari posisi dapat dilakukan dengan pemanggilan fungsi determinan yang sudah disesuaikan rumusnya

CariHull(S1,A,B)

CariHull(S2,B,A)

}

CariHull(S,P,Q){

//Prosedur ini digunakan untuk mencari titik-titik yang masuk ke dalam element ConvexHull dari bagian yang berada pada sebelah kanan garis

Mencari titik yang memiliki jarak terjauh dari garis yang dibentuk oleh titik P dan Q kemudian memasukkan titik maksimum tersebut ke dalam list ConvexHull. Anggap titik bernama pmax

Mencari jarak dapat dilakukan melalui pemanggilan fungsi jarak yang telah disesuaikan rumusnya

Membagi titik-titik yang tersedia ke dalam 3 segment yaitu S1,S2,dan S0

S0 yang merupakan titik-titik yang berada di dalam area segitiga yang dibentuk oleh titik P, pmax, Q dapat diabaikan karena sudah pasti tidak akan masuk ke dalam Convex

S1 merupakan titik-titik yang berada pada sebelah kiri dari garis yang dibentuk oleh titik P dan pmax

S2 merupakan titik-titik yang berada pada sebelah kanan dari garis yang dibentuk oleh titik pmax dan Q

Melakukan rekursi terhadap himpunan S1 dan S2

CariHull(S1,P,pmax)

CariHull(S2,pmax,Q)

}

drawHull{

Membagi titik-titik yang masih berantakan ke dalam dua partisi yaitu bagian atas dan bawah (berdasarkan garis yang dibentuk oleh titik terkiri dan titik terkanan)

Mengurutkan titik pada bagian atas mulai dari absis terkecil

Mengurutkan titik pada bagian bawah mulai dari absis terbesar

Membuat titik-titik convexhull unik agar pada saat ditampilkan lebih baik

menyimpan titik-titik x ConvexHull pada sebuah list xhull

menyimpan titik-titik y ConvexHull pada sebuah list yhull

Menghubungkan semua titik sesuai urutannya untuk membuat sebuah poligon

}

# 

# **BAB 3**

**IMPLENTASI PROGRAM**

* 1. **Inisialisasi**

Meng-*import* hal hal yang akan diperlukan pada program

import sys

import random

import matplotlib.pyplot as plt #keperluan visualisasi convexhull

import math

* 1. **Masukan Awal**

Pada langkah pertama, program akan menerima input n yang merupakan jumlah dari titik yang akan dirandom

def inputs():

global n

global listpoint

global size

global pointx

global pointy

size=100

listpoint=[]

pointx=[]

pointy=[]

n= eval(raw\_input('Masukkan jumlah poin: '))

for x in range(n):

listpoint.append([random.randint(0,size),random.randint(0,size)])

for x in range(len(listpoint)):

pointx.append(listpoint[x][0])

pointy.append(listpoint[x][1])

* 1. **Fungsi Determinan**

Fungsi ini digunakan untuk mencari determinan untuk menentukan posisi dari sebuah titik terhadap garis

def Determinan(P1,P2,P3):

det=((P1[0]\*P2[1])+(P3[0]\*P1[1])+(P2[0]\*P3[1])-(P3[0]\*P2[1])-(P2[0]\*P1[1])-(P1[0]\*P3[1]))

return det

* 1. **Prosedur untuk Menentukan Convex Hull**

Prosedur ini akan mencari titik terkanan dan terkiri lalu membagi titik-titik ke dalam dua daerah dan memanggil fungsi CariHull untuk mencari titik-titik lain yang masuk ke dalam Convex Hull

def QuickHull():

global leftmost

global rightmost

global ConvexHull

#========================= cari paling kiri sama paling kanan =====================+#

leftmost=[size,50]

rightmost=[0,50]

ConvexHull=[]

for x in range(len(listpoint)):

if listpoint[x][0]<leftmost[0]:

leftmost=listpoint[x]

for x in range(len(listpoint)):

if listpoint[x][0]>rightmost[0]:

rightmost=listpoint[x]

print ('leftmost: ',leftmost)

print ('rightmost: ',rightmost)

ConvexHull.append(leftmost) #memasukkan titik terkiri ke dalam convexhull

ConvexHull.append(rightmost) #memasukkan titik terkanan ke dalam convexhull

#=========================== cari atas bawah =======================================#

global S1

global S2

S1=[]

S2=[]

for x in range(len(listpoint)):

if Determinan(leftmost,rightmost,listpoint[x])>0: #kalau >0 berarti di kiri

if listpoint[x] not in S1:

S1.append(listpoint[x])

elif Determinan(leftmost,rightmost,listpoint[x])<0: #kalau <0 berarti di kanan

if listpoint[x] not in S2:

S2.append(listpoint[x])

print ('S1 :',S1)

print('S2 :',S2)

CariHull(S1,leftmost,rightmost) #cari hull buat bagian atas

CariHull(S2,rightmost,leftmost) #cari hull buat bagian bawah

* 1. **Fungsi Jarak**

def jarak(P1,P2,Point): #fungsi cari jarak

pembilang= abs(((P2[1]-P1[1])\*Point[0])-((P2[0]-P1[0])\*Point[1])+(P2[0]\*P1[1])-(P2[1]\*P1[0]))

return pembilang

* 1. **Prosedur untuk Mencari Convex Hull**

Prosedur ini akan digunakan untuk mencari titik-titik yang masuk ke dalam ConvexHull dari sebuah himpunan titik.

def CariHull(S,P,Q):

if S==[]: #kondisi berhenti jika tidak ada titik di himpunan

return

else: #kondisi rekursif

maks=0

pmax=[]

A=[]

B=[]

for x in range (len(S)):

if (jarak(P,Q,S[x])>maks): #jika menemukan jarak maksimum akan disimpan ke dalam pmax

del pmax[:]

maks=jarak(P,Q,S[x])

pmax.append(S[x][0]),

pmax.append(S[x][1])

ConvexHull.insert(1,pmax) #memasukkan pmax ke dalam convexhull

for x in range(len(S)): #membagi titik-titik ke dalam dua partisi yang ada di luar

if Determinan(P,pmax,S[x])>0: #segitiga yang dibetnuk oleh P pmax dan Q

A.append(S[x])

if Determinan(pmax,Q,S[x])>0:

B.append(S[x])

CariHull(A,P,pmax) #melakukan rekursif

CariHull(B,pmax,Q)

* 1. **Main program**

inputs()

QuickHull()

plt.scatter(pointx,pointy, c='orange',edgecolors='none')

drawHull(ConvexHull)

plt.scatter([leftmost[0],rightmost[0]],[leftmost[1],rightmost[1]], c='red',edgecolors='none')

plt.show()

exit()

* 1. **Prosedur untuk Menampilkan Output**

def drawHull(Q):

global xhull

global yhull

global atas

global bawah

atas=[leftmost,rightmost]

bawah=[rightmost,leftmost]

for x in range(len(Q)): #partisi titik menjadi dua bagian

if Determinan(leftmost,rightmost,Q[x])>0:

atas.insert(1,Q[x])

else:

bawah.insert(1,Q[x])

def takeA(elem):

return elem[0]

del Q[:]

atas.sort(key=takeA) #mengurutkan list sesuai partisinya

bawah.sort(key=takeA,reverse=True)

Q= atas+bawah

sets=[] #membuat list unik dari convex hull

for x in range(len(Q)):

if Q[x] not in sets:

sets.append(Q[x])

for x in range(len(sets)): #menampilkan set convex hull yang sudah rapi

print 'ConvexHull: ', sets[x]

xhull=[]

yhull=[]

for x in range(len(Q)): #memasukkan list poin x dan poin y convex hull

xhull.append(Q[x][0])

yhull.append(Q[x][1])

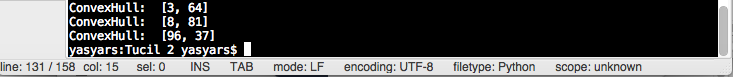
plt.plot(xhull,yhull,c='green') #visualisasi convex hull

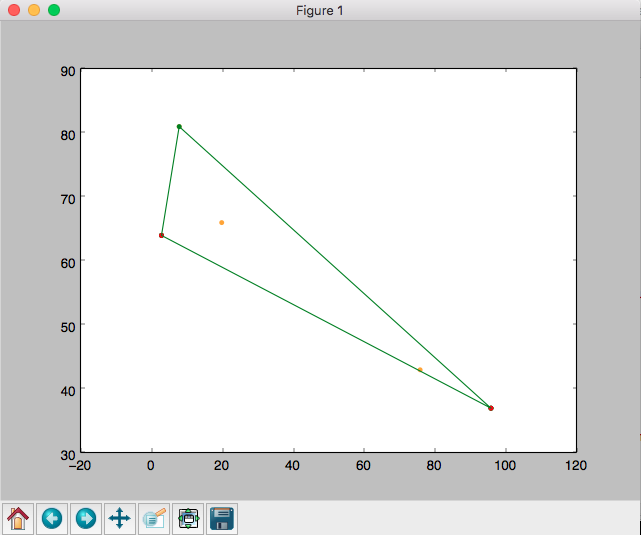
plt.scatter(xhull,yhull,c='green',edgecolors='none')

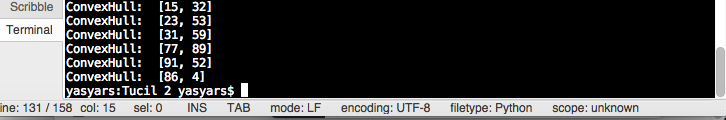
# **BAB 4**

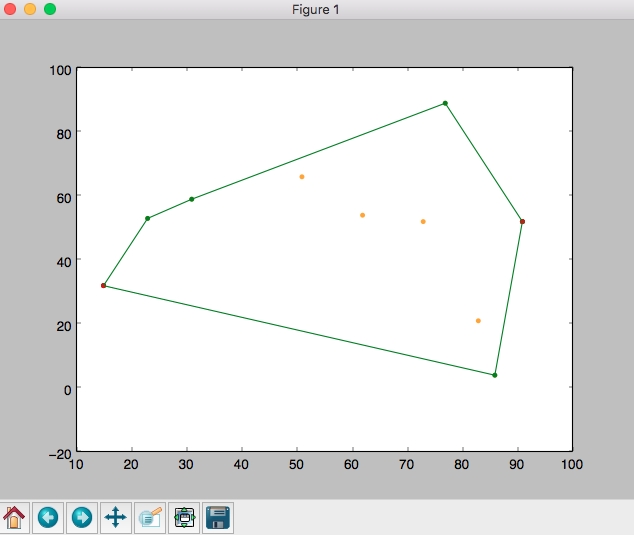
**INPUT OUTPUT**

**4.2 Input N=5**

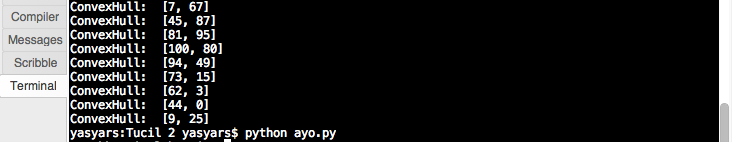
****

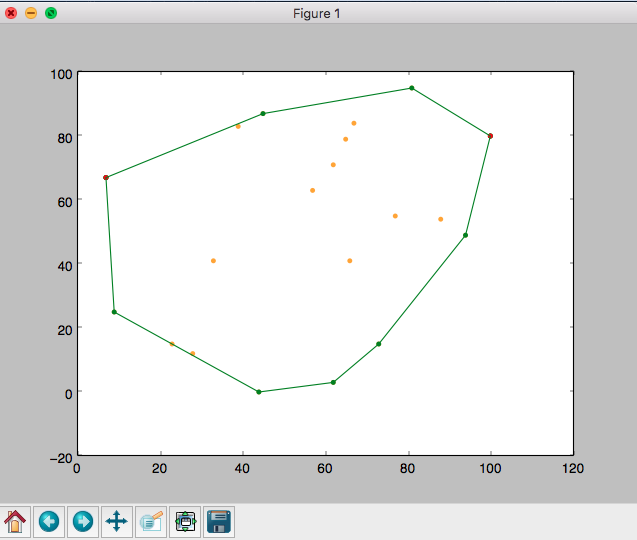
****

**4.2 Input N=10**

****

**4.3 Input N=20**

****

****

# **BAB 5**

**HASIL**

|  |  |  |
| --- | --- | --- |
| **POIN** | **YA** | **TIDAK** |
| Program berhasil dikompilasi | ✓ |  |
| Program berhasil *running* | ✓ |  |
| Program dapat menerima input dan menuliskan output. | ✓ |  |
| Luaran sudah benar untuk semua n | ✓ |  |

**Daftar Pustaka**

Munir, Rinaldi. “Diktat Kuliah Strategi Algoritma”. Bandung :Department Teknologi Informatika Institut Teknologi Bandung, 2004

Slide PPT Rinaldi Munir, “Convex Hull”

Novandi, Petra. Makalah “Analisis Kompleksitas Algoritma Pencarian Convex Hull Pada Bidang Planar”. 2004