Tugas Kecil 2 IF2211 Strategi Algoritma Semester II tahun 2022/2023

Mencari Pasangan Titik Terdekat 3D dengan Algoritma Divide and Conquer

Disusun oleh:

Muhammad Haidar Akita Tresnadi
13521025
M. Malik I. Baharsyah
13521029

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

Daftar Isi

BAB I

Deskripsi Masalah

Pencarian pasangan dua titik terdekat pada euclidean space merupakan permasalahan yang cukup umum pada grafika komputasi, sistem informasi grafis maupun pemrosesan citra. Kasus dari permasalahan ini adalah diberikan n buah titik dalam euclidean space dan kita perlu mencari pasangan dua titik yang memiliki jarak terdekat. Solusi dari permasalahan ini dapat diselesaikan dengan menggunakan algoritma komputasi modern. Terdapat berbagai macam cara untuk menyelesaikan persoalan ini, beberapa contoh diantaranya adalah dengan algoritma brute force dan algoritma divide and conquer.

Dalam laporan ini, penulis akan melakukan analisis serta membandingkan perbandingan eksekusi waktu menggunakan kedua algoritma berbeda tersebut. Dikarenakan algoritma brute force memiliki kompleksitas algoritma yang berbeda dengan algoritma divide and conquer, maka besar kemungkinan waktu eksekusi programnya pun berbeda. Algoritma brute force mencari seluruh kemungkinan solusi yang ada dengan menghitung selisih jarak dua titik dengan semua kombinasi titik yang ada. Sedangkan pada algoritma divide and conquer, digunakan teknik atau metode pemecahan suatu masalah besar menjadi beberapa upa permasalahan yang lebih kecil dan terpisah.

BAB II

Algoritma

2.1 Algoritma Brute Force

Algoritma brute force menggunakan metode pencarian seluruh kemungkinan pasangan titik dan menghitung jarak masing-masing pasangan. Setelah itu, nilai minimum dari semua jarak yang dihitung akan menjadi jarak minimum antara dua titik. Pada algoritma brute force untuk mencari dua titik terdekat, proses yang dilakukan adalah sebagai berikut:

1. Ambil dua titik dari himpunan titik yang diberikan
2. Hitung jarak antara kedua titik tersebut
3. Ulangi langkah 1 dan 2 untuk semua pasangan titik yang mungkin
4. Cari nilai minimum dari semua jarak yang dihitung pada langkah 2

Algoritma brute force cukup sederhana sehingga mudah dipahami, namun sangat disayangkan karena memiliki kompleksitas waktu yang sangat tinggi. Karena mencari seluruh kemungkinan pasangan titik membutuhkan waktu yang proporsional dengan kuadrat jumlah titik yang ada. Oleh karena itu, algoritma ini tidak efisien untuk masalah dengan jumlah titik yang sangat besar.

2.2 Algoritma Divide and Conquer

Algoritma Divide and Conquer adalah sebuah algoritma yang memecah suatu masalah menjadi beberapa sub-masalah yang lebih kecil, menyelesaikan setiap sub-masalah secara rekursif, dan kemudian menggabungkan solusi sub-masalah untuk mendapatkan solusi akhir dari masalah asli.

Proses pemecahan masalah dengan divide and conquer terdiri dari tiga langkah utama: divide, conquer, dan combine. Pertama, masalah awal dibagi menjadi beberapa sub-masalah yang lebih kecil. Kemudian, setiap sub-masalah dipecahkan secara rekursif menggunakan metode yang sama. Akhirnya, solusi dari setiap sub-masalah digabungkan untuk membentuk solusi akhir dari masalah asli. Algoritma divide and conquer sering digunakan dalam pengurutan data, pencarian elemen, pemecahan persamaan, dan berbagai masalah komputasi lainnya.

Algoritma divide and conquer dapat digunakan untuk mencari pasangan titik terdekat dalam himpunan titik-titik pada bidang. Langkah-langkah algoritma tersebut adalah sebagai berikut:

1. Bagi himpunan titik menjadi dua bagian yang. Garis pembagi dapat dibuat secara vertikal atau horizontal.
2. Cari pasangan titik terdekat pada masing-masing bagian himpunan secara rekursif. Jarak antara pasangan titik terdekat ini dapat dihitung menggunakan rumus Euclidean distance.
3. Tentukan jarak terdekat antara kedua pasangan titik terdekat yang ditemukan pada langkah 2. Jarak ini dapat dinyatakan dalam bentuk jarak antara dua titik atau hanya sebagai nilai numerik.
4. Pilih pasangan titik terdekat yang jaraknya lebih kecil dari nilai yang ditemukan pada langkah 3. Pasangan titik ini adalah pasangan titik terdekat pada seluruh himpunan titik.

BAB III

Source Code Program

3.1 Struktur Data Program

Program yang kami buat dikembangkan menggunakan bahasa python, dengan menggunakan library plotly sebagai visualisasi dari titik yang ditemukan sebagai solusi. Program kami dibagi menjadi 6 bagian file yang berbeda yaitu main.py sebagai program utama, point.py sebagai deklarasi fungsi yang berkaitan dengan titik, plotting.py untuk melakukan visualisasi, solver.py sebagai tempat deklarasi fungsi dari algoritma brute force dan algoritma divide and conquer, UI.py sebagai deklarasi fungsi dan prosedur interaksi dengan pengguna, dan getSpec.py yang menampung prosedur untuk mendapatkan detail spesifikasi komputer yang digunakan pengguna.

3.2 Source Code

point.py

```
import random



def randomPoint(numberOfPoint, dimension):
    matriksOfpoint = [[0 for _ in range(dimension)] for _ in range(numberOfPoint)]
    for i in range(numberOfPoint):
        for j in range(dimension):
            matriksOfpoint[i][j] = random.randint(0,1000)


    return matriksOfpoint


def sortPoint(matriksofPoint):
    return sorted(matriksofPoint, key=lambda a_entry: a_entry[0])
```

```
def calculateDistance(point1, point2):
    sum = 0
    for i in range(len(point1)):
        sum += (point1[i] - point2[i])**2
    return sum**(1/2)

def viewPoints(matrixOfPoints):
    if (len(matrixOfPoints) > 5):
        for i in range(5):
            print(matrixOfPoints[i], end=' ')
        print("...")
    else:
        print(matrixOfPoints)
```

solver.py

```
from point import *
import time

def bruteforce(matriksofPoint):
    global countBF
    global timeBF
    countBF = 0
```

```
    startTime = time.time()
    minDistance = calculateDistance(matriksofPoint[0], matriksofPoint[1])
    countBF += 1
    for i in range(len(matriksofPoint)):
        for j in range(i+1, len(matriksofPoint)):
            countBF += 1
            if calculateDistance(matriksofPoint[i], matriksofPoint[j]) <= minDistance:
                minDistance = calculateDistance(matriksofPoint[i], matriksofPoint[j])
                countBF += 1
                point1 = matriksofPoint[i]
                point2 = matriksofPoint[j]
    endTime = time.time()
    timeBF = endTime - startTime
    return minDistance, point1, point2


def divideAndConquer(matriksofPoint):
    global countDaC
    global timeDaC
    countDaC = 0
    startTime = time.time()
    if len(matriksofPoint) == 3:
        endTime = time.time()
        timeDaC = endTime - startTime
```

```
    res = bruteforce(matriksofPoint)
    countDaC += countBF
    return res
elif len(matriksofPoint) == 2:
    endTime = time.time()
    timeDaC = endTime - startTime
    countDaC += 1
    return calculateDistance(matriksofPoint[0], matriksofPoint[1]), matriksofPoint[0], matriksofPoint[1]
else:
    if len(matriksofPoint) % 2 == 0:
        mid = len(matriksofPoint) // 2
        left = matriksofPoint[:mid]
        right = matriksofPoint[mid:]
        leftmin = divideAndConquer(left)
        rightmin = divideAndConquer(right)
        if leftmin[0] <= rightmin[0]:
            distance, p1, p2 = leftmin
        else:
            distance, p1, p2 = rightmin
        inMid = []
        xMid = (matriksofPoint[mid-1][0] + matriksofPoint[mid][0]) / 2
        # mengecek jika terdapat point yang berada dalam jarak mid + distance
        for point in matriksofPoint:
```

```
            countDaC += 1
            if abs(point[0] - xMid) <= distance:
                inMid.append(point)
        # mencari dua poin yang jaraknya kurang dari distance
        for i in range(len(inMid)):
            for j in range(i+1, len(inMid)):
                countDaC += 1
                d = calculateDistance(inMid[i], inMid[j])
                if d < distance:
                    p1, p2 = inMid[i], inMid[j]
                    distance = d
        endTime = time.time()
        timeDaC = endTime - startTime
        return (distance, p1, p2)


    else:
        mid = len(matriksofPoint) // 2
        left = matriksofPoint[:mid]
        right = matriksofPoint[mid:]
        leftmin = divideAndConquer(left)
        rightmin = divideAndConquer(right)
        if leftmin[0] <= rightmin[0]:
            distance, p1, p2 = leftmin
```

```
        else:
            distance, p1, p2 = rightmin

        inMid = []
        xMid = (matriksofPoint[mid-1][0] + matriksofPoint[mid][0]) / 2
        # mengecek jika terdapat point yang berada dalam jarak mid + distance
        for point in matriksofPoint:
            countDaC += 1
            if abs(point[0] - xMid) <= distance:
                inMid.append(point)
        # mencari dua poin yang jaraknya kurang dari distance
        for i in range(len(inMid)):
            for j in range(i+1, len(inMid)):
                countDaC += 1
                d = calculateDistance(inMid[i], inMid[j])
                if d < distance:
                    p1, p2 = inMid[i], inMid[j]
                    distance = d
        endTime = time.time()
        timeDaC = endTime - startTime
        return (distance, p1, p2)

def getTimeBF():
```

```
    return timeBF


def getTimeDaC():
    return timeDaC


def getNOperationBF():
    return countBF


def getNOperationDaC():
    return countDaC
```

Plotting.py

```
import plotly.graph_objs as go
import numpy as np


def plot3D(matriksOfPoint, n, result):


        x = [matriksOfPoint[i][0] for i in range(n)]
        y = [matriksOfPoint[i][1] for i in range(n)]
        z = [matriksOfPoint[i][2] for i in range(n)]


        # create a 3D scatter plot
```

```
    fig = go.Figure(data=[go.Scatter3d(x=x, y=y, z=z, mode='markers', marker=dict(
            size=5,
            color='blue',
            colorscale='Viridis',
            opacity=0.8
    ))])

    fig.add_trace(go.Scatter3d(x=[result[1][0]], y=[result[1][1]], z=[result[1][2]], mode='markers',
marker=dict(
            size=10,
            color='red',
            symbol='square-open'
    )))

    fig.add_trace(go.Scatter3d(x=[result[2][0]], y=[result[2][1]], z=[result[2][2]], mode='markers',
marker=dict(
            size=10,
            color='red',
            symbol='square-open'
    )))

    fig.update_layout(scene=dict(xaxis_title='X', yaxis_title='Y', zaxis_title='Z'))

    fig.show()
```

getSpec.py

```
import platform, psutil



def getPCSpec():

    print("  OS:",platform.system())

    print("  Processor:",platform.processor())

    print("  CPU Core:",psutil.cpu_count(logical=True))

    print("  Memory:",round(psutil.virtual_memory().total / (1024*1024*1024), ndigits=2), "GB")
```

UI.py

```
from Plotting import *

from getSpec import *

from solver import *



def getNPoints():

    n = int(input("Please enter the number of points:\n  > "))

    return n



def getDimension():

    d = int(input("Enter the dimension:\n  > "))

    return d
```

```
def plotting(matriksOfPoint, n, result):
    if (len(matriksOfPoint[0])) == 3:
        p = str(input("\nIt seems like the points are built on 3D, do you want to plot the points? (y/n)\n  >
"))
        if p == "y":
            plot3D(matriksOfPoint, n, result)
    else:
        print("Your points are not built on 3D, we couldn't provide the graph")



def exit():
    print("Thank you for using our program ^v^")
    print("I hope u have a great day")


def asciiart():
    print("""
              ////////
         ////////&@///////,
    (///////%%&///////%,#///////
    //#//#//////#*////////////, .
    /////((////%////(////(..,..
    /(//////((,//////( .    ... .          Pair of Closest Points solver
    (///%*//(////#     .   *, ...
```

```
    #//////#*///(    .,. .  ...

    ///#////////( .... ... *....

    //(///%////(...../.........

        (/////#/(...........

            (////(.......

                (/(...

    """)


def program():

    asciiart()

    n = getNPoints()

    dim = getDimension()

    matrixOfPoints = randomPoint(n, dim)

    matrixOfPoints = sortPoint(matrixOfPoints)

    print("Random points generated, preview:")

    viewPoints(matrixOfPoints)

    resBF = bruteforce(matrixOfPoints)

    print("\nClosest pair using Bruteforce algorithm found! Here's the result:")

    print("  Distance between the two:", resBF[0])

    print("  Point 1:", resBF[1])

    print("  Point 2:", resBF[2])

    print("  Number of operations executed:", getNOperationBF())

    print("  Elapsed time:", round(getTimeBF(), ndigits=2), "second(s)")
```

```
    resDaC = divideAndConquer(matrixOfPoints)

    print("\nClosest pair using Divide and Conquer algorithm found! Here's the result:")

    print("  Distance between the two:", resDaC[0])

    print("  Point 1:", resDaC[1])

    print("  Point 2:", resDaC[2])

    print("  Number of operations executed:", getNOperationDaC())

    print("  Elapsed time:", round(getTimeDaC(), ndigits=2), "second(s)")

    print("\nAbove results are obtained using your PC specification below:")

    getPCSpec()

    plotting(matrixOfPoints, n, resDaC)

    exit()
```

main.py

```
from UI import *



if __name__ == '__main__':

    program()
```

BAB IV

Test Case

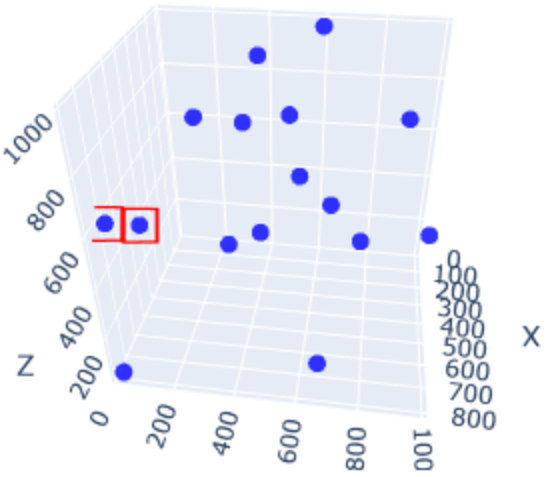
n = 16

```
Please enter the number of points:
  > 16
Enter the dimension:
  > 3
Random points generated, preview:
[22, 913, 647] [50, 592, 985] [74, 227, 45] [149, 142, 665] [289, 406, 964] ...

Closest pair using Bruteforce algorithm found! Here's the result:
  Distance between the two: 99.54396013822235
  Point 1: [706, 118, 585]
  Point 2: [726, 21, 595]
  Number of operations executed: 127
  Elapsed time: 0.0 second(s)

Closest pair using Divide and Conquer algorithm found! Here's the result:
  Distance between the two: 99.54396013822235
  Point 1: [706, 118, 585]
  Point 2: [726, 21, 595]
  Number of operations executed: 59
  Elapsed time: 0.0 second(s)

Above results are obtained using your PC specification below:
  OS: Windows
  Processor: Intel64 Family 6 Model 78 Stepping 3, GenuineIntel
  CPU Core: 4
  Memory: 11.88 GB
```



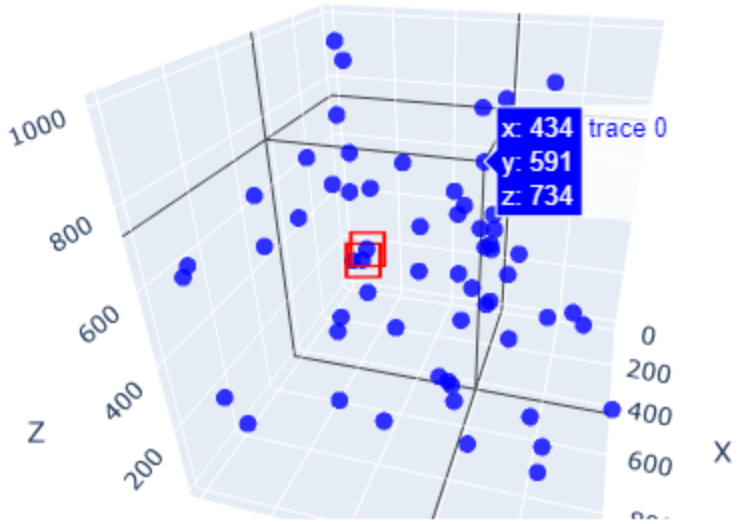
n = 64

```
Please enter the number of points:
  > 64
Enter the dimension:
  > 3
Random points generated, preview:
[10, 19, 116] [46, 291, 134] [52, 764, 23] [82, 446, 169] [86, 46, 937] ...

Closest pair using Bruteforce algorithm found! Here's the result:
  Distance between the two: 37.603191353926334
  Point 1: [918, 443, 692]
  Point 2: [928, 458, 725]
  Number of operations executed: 2027
  Elapsed time: 0.01 second(s)

Closest pair using Divide and Conquer algorithm found! Here's the result:
  Distance between the two: 37.603191353926334
  Point 1: [918, 443, 692]
  Point 2: [928, 458, 725]
  Number of operations executed: 231
  Elapsed time: 0.01 second(s)

Above results are obtained using your PC specification below:
  OS: Windows
  Processor: Intel64 Family 6 Model 78 Stepping 3, GenuineIntel
  CPU Core: 4
  Memory: 11.88 GB
```



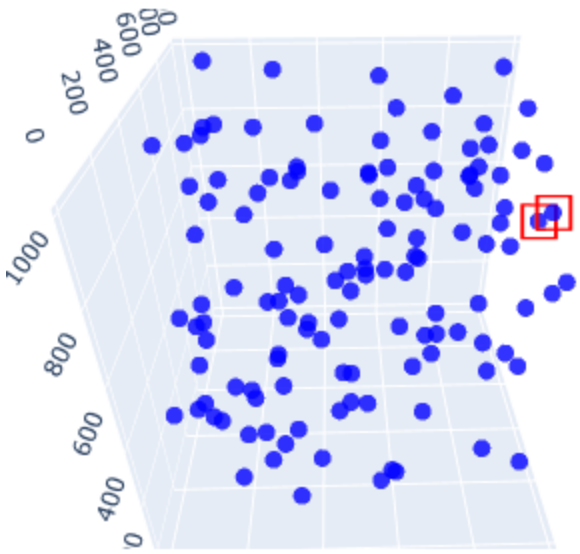
n = 128

```
Please enter the number of points:
  > 128
Enter the dimension:
  > 3
Random points generated, preview:
[9, 914, 483] [21, 863, 488] [30, 225, 896] [36, 524, 572] [36, 835, 447] ...

Closest pair using Bruteforce algorithm found! Here's the result:
  Distance between the two: 32.93933818400121
  Point 1: [104, 128, 925]
  Point 2: [109, 152, 903]
  Number of operations executed: 8132
  Elapsed time: 0.02 second(s)

Closest pair using Divide and Conquer algorithm found! Here's the result:
  Distance between the two: 32.93933818400121
  Point 1: [104, 128, 925]
  Point 2: [109, 152, 903]
  Number of operations executed: 763
  Elapsed time: 0.01 second(s)

Above results are obtained using your PC specification below:
  OS: Windows
  Processor: Intel64 Family 6 Model 78 Stepping 3, GenuineIntel
  CPU Core: 4
  Memory: 11.88 GB
```



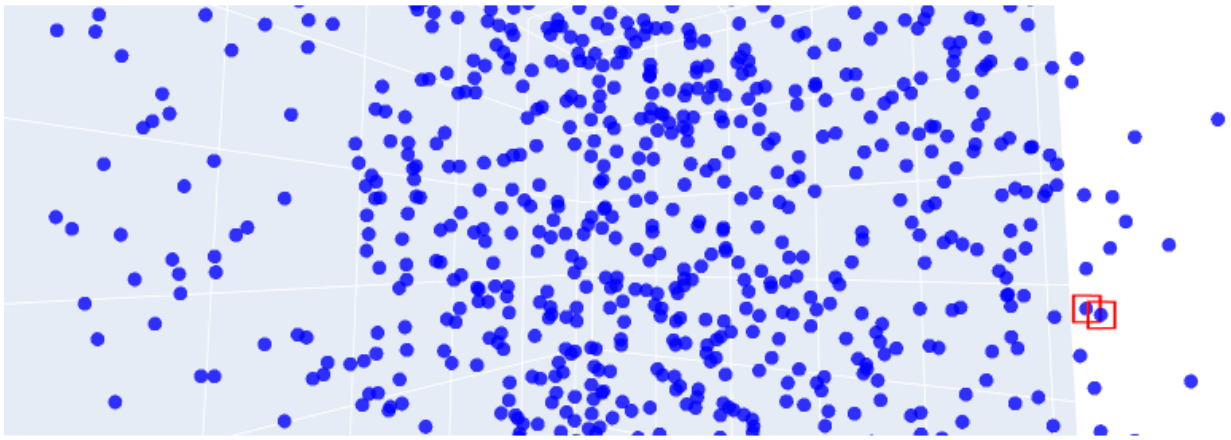
n = 1000

```
Please enter the number of points:
  > 1000
Enter the dimension:
  > 3
Random points generated, preview:
[1, 344, 709] [1, 216, 109] [2, 166, 33] [2, 716, 100] [6, 379, 0] ...

Closest pair using Bruteforce algorithm found! Here's the result:
  Distance between the two: 6.0
  Point 1: [160, 93, 427]
  Point 2: [164, 97, 429]
  Number of operations executed: 499520
  Elapsed time: 1.35 second(s)

Closest pair using Divide and Conquer algorithm found! Here's the result:
  Distance between the two: 6.0
  Point 1: [160, 93, 427]
  Point 2: [164, 97, 429]
  Number of operations executed: 3947
  Elapsed time: 0.14 second(s)

Above results are obtained using your PC specification below:
  OS: Windows
  Processor: Intel64 Family 6 Model 78 Stepping 3, GenuineIntel
  CPU Core: 4
  Memory: 11.88 GB
```



Berdasarkan data yang diperoleh dari 4 percobaan dengan jumlah titik yang berbeda, terlihat bahwa algoritma bruteforce memiliki perbedaan waktu yang cukup signifikan jika dipakai untuk menghitung kasus dengan banyak titik. Hal ini dikarenakan algoritma brute force mencoba seluruh kemungkinan solusi sehingga jumlah dari operasi yang dilakukan pun berbeda cukup jauh dengan algoritma divide and conquer. Oleh karena itu, untuk permasalahan pencarian dua titik dengan jarak terdekat dapat disimpulkan bahwa algoritma divide and conquer lebih optimal jika dibandingkan dengan algoritma bruteforce.

BAB V

Checklist dan Pranala

5.1 Pranala Github

https://github.com/Darmodar/Tucil2_13521025_13521029

5.2 Checklist

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa ada kesalahan	✓	
2. Program berhasil running	✓	
3. Program dapat menerima masukan dan dan menuliskan luaran	✓	
4. Luaran program sudah benar (solusi closest pair benar)	✓	
5. Bonus 1 dikerjakan	✓	
6. Bonus 2 dikerjakan	✓	