LAPORAN TUGAS KECIL 2

IF2211/Strategi Algoritma

Finding Closest Pair of Point with Divide and Conquer

Dipersiapkan oleh:

Angger Ilham A / 13521001

Christophorus Dharma Winata / 13521009

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

Bagian 1
Latar belakang

Tugas kecil 2 mata kuliah IF2211 Strategi Algoritma semester genap meminta penulis untuk membuat program untuk mencari pasangan titik dengan jarak yang terdekat pada ruang titik tiga dimensi. Pencarian pasangan titik mengimplementasikan strategi *Divide and Conquer* untuk mencari solusinya dan membandingkan hasil dari pencarian tersebut dengan pencarian yang sama namun menggunakan strategi *Bruteforce*.

1.1. Algoritma Divide and Conquer

Divide and Conquer dulunya adalah strategi militer yang dikenal dengan nama divide ut imperes. Strategi tersebut memiliki unsur dasar yang dapat diterapkan di ilmu komputer sehingga nama Divide and Conquer dirasa cocok untuk mewakili strategi tersebut. Secara prosedur, Strategi DnC (Divide and Conquer) dibagi menjadi tiga bagian: Divide yang berarti membagi-bagi masalah menjadi sub-masalah, Conquer yang berarti memecahkan seluruh sub-masalah yang sudah dibuat, dan Combine yaitu proses menggabung-gabungkan solusi-solusi dari semua sub-masalah untuk menjadi solusi permasalahan besar di awal.

1.2. Pencarian *closest pair of points*

Pada tugas kecil 2 IF2211 Strategi Algoritma ini, mahasiswa diminta untuk mencari sepasang titik yang terletak paling berdekatan pada suatu ruang tiga dimensi. Pencarian jarak ini dapat dilakukan dengan rumus *Euclidean Distance* yang dimana berisi sebagai berikut:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 + \ldots}$$

d = jarak dari suatu titik $P_1(x_1,y_1,z_1)$ dari titik $P_2(x_2,y_2,z_2)$

Rumus *Euclidean Distance* juga dapat dikembangkan untuk vektor yang terletak pada dimensi R^n dimana setiap vektor dinyatakan dalam bentuk $x = (x_1, x_2, \ldots, x_n)$.

Pada tugas kecil kali ini, rumus *Euclidean Distance* tersebut digunakan untuk langkah *Conquer* pada implementasi strategi *Divide and Conquer* dan juga digunakan untuk menghitung jarak antara 2 titik pada implementasi strategi *Bruteforce* pada kasus ini.

Bagian 2
Penjelasan Program

Algoritma yang kami gunakan untuk penyelesaian masalah ini adalah sebagai berikut,

1. Urutkan titik - titik tersebut menurut absis.
2. Lalu anggap ruang tempat titik - titik itu berada sebagai R. R dibagi menjadi 2 yaitu R1 dan R2 dimana keduanya berisi setengah atau setengah + 1 titik dari jumlah titik yang ada di R.
3. R1 dan R2 itu dibagi lagi menjadi ruang yang lebih kecil hingga terdapat hanya 2 atau 3 titik pada tiap ruang.
4. Jika pada ruangan kecil terdapat 2 titik, maka langsung hitung euclidean distancenya. Jika terdapat 3 titik, dilakukan *bruteforce* mencari pasangan titik terdekat dengan cara membandingkan jarak antara tiap pasangan yang mungkin dari titik tersebut.
5. Bandingkan jarak minimum yang didapat dari pasangan titik terdekat dari 2 ruang dan ambil jarak paling minimum antara 2 ruang tersebut.
6. Gabung kedua ruangan tersebut.
7. Semisal jarak paling minimum antara 2 ruang tersebut adalah d. Terdapat kemungkinan dimana pasangan titik terdekat ada di titik yang dibatasi oleh garis pembagi.
8. Dari garis pembagi tadi kita mencari titik - titik yang berada pada daerah dengan lebar 2*d dengan pusat garis pembagi tersebut.
9. Lalu, cari pasangan titik yang memiliki jarak lebih kecil daripada jarak minimal yang telah didapatkan sebelumnya. Misalkan diminta membuat n-dimensi, kita dapat mencari jarak minimalnya dengan membuat garis pembagi dari sumbu ke-n hingga sumbu y.

10. Rekursikan langkah langkah tersebut hingga menjadi satu kesatuan ruang S dan akan didapatkan pasangan titik terdekat dan jaraknya.

Bab 3
Source Code

Berikut screenshot dari source code program yang dibuat dalam bahasa Python:

main.py

```
elif(long == 3) : #bruteforce 3 titik
    p1 = euclideanDistance(titik[0], titik[1])
    p2 = euclideanDistance(titik[0], titik[2])
    p3 = euclideanDistance(titik[1], titik[2])

    if(p1 < p2) :
        if(p1 < p3) :
            pasangan = (titik[0], titik[1])
            jarak = p1
            return pasangan, jarak
        else :
            pasangan = (titik[1], titik[2])
            jarak = p3
            return pasangan, jarak
    else :
        if(p2 < p3) :
            pasangan = (titik[0], titik[2])
            jarak = p2
            return pasangan, jarak
        else :
            pasangan = (titik[1], titik[2])
            jarak = p3
            return pasangan, jarak

else : # lebih dari 3 maka dilakukan divide conquer
    if (dimensi>3):
        sortArr(titik,sumbudivide)
    kiri =  titik[:long//2]
    kanan = titik[long//2:]

    # ini buat minimum kiri sama minimum kanan
    # saat dimensi lebih dari 3, perlakuan divide dilakukan dilakukan pada sumbu-sumbu selain absis
    if (dimensi>3):
            titikkiri, jarakkiri = closestPair(kiri,dimensi,sumbudivide+1 if sumbudivide+1 < dimensi else 0)
            titikkanan, jarakkanan = closestPair(kanan,dimensi,sumbudivide+1 if sumbudivide+1 < dimensi else 0)
    else:
        titikkiri, jarakkiri = closestPair(kiri, dimensi)
        titikkanan, jarakkanan = closestPair(kanan, dimensi)

    # ambil minimum antara kiri atau kanan
    if(jarakkiri < jarakkanan) :
        pasangan, jarak = titikkiri, jarakkiri
    else : pasangan, jarak = titikkanan, jarakkanan
```

```
        # titik titik didekat garis tengah
        d = titik[long//2][0]
        midpoint = []
        for i in range(len(kiri)) :
            if(kiri[i][0] >= d - jarak) : midpoint.append(kiri[i])
        for i in range(len(kanan)) :
            if(kanan[i][0] <= d + jarak) : midpoint.append(kanan[i])
        for k in range(dimensi-1,0,-1) :
            midpoint = sortArr(midpoint,k)
            for i in range(len(midpoint)) :
                for j in range(i+1, len(midpoint)) :
                    if(abs(midpoint[i][0] - midpoint[j][0]) < jarak or abs(midpoint[i][k] - midpoint[j][k]) < jarak) :
                        pasangan2, jarak2 = (midpoint[i], midpoint[j]), euclideanDistance(midpoint[i], midpoint[j])
                        if jarak2 < jarak : pasangan, jarak = pasangan2, jarak2
        return pasangan, jarak


def closestPair_bruteforce(titik) :
    long = len(titik)
    jarakminimum = float("inf")

    for i in range(long) :
        for j in range(i+1, long) :
            # print(i,"  ", j)
            # print(len(titik[2]))
            jar = euclideanDistance(titik[i],titik[j])
            if(jar < jarakminimum) :
                jarakminimum = jar
                titikterdekat = (titik[i], titik[j])
    return titikterdekat,jarakminimum
```

```python
# main program
print("="*40, "WELCOME", "="*40)
while True:
    mauDimensi = input("apakah ingin menggunakan dimensi lain [default 3 detik] ? (Y/N) ")
    if (mauDimensi in "YyNn"): break
    else : print("Invalid!")
if (mauDimensi in "Yy"):
    dimensi = int(input("oke! masukkan dimensi: "))
elif (mauDimensi in "Nn"):
    print("baik! default ke 3")
    dimensi = 3
n = int(input("Masukkan jumlah titik : "))
if (n <= 1):
    # exception handle: titik yang dimasukkan tidak bisa dibuat pasangan
    while n<2:
        n = int(input("Invalid: Minimal titik harus 2. Input: "))
else :
    # inisialisasi titik-titik
    titik = []
    for i in range(n):
        point = [random.randint(0,1000) for j in range(dimensi)]
        titik.append(point)
    titik = sortArr(titik,0)
    #inisialisasi penghitungan operasi euclidean
    euclideancount = 0
    startTime = time.time()
    titikterdekat, jaraktitikterdekat = closestPair(titik, dimensi, 0)
    stopTime = time.time()
    print("="*40, "Strategi Divide and Conquer", "="*40)
    print("Pasangan titik terdekat : ", titikterdekat)
    print("Jaraknya : ", jaraktitikterdekat)
    print("banyak perhitungan euclidean : ", euclideancount)
    print(f"Lama program berjalan : {(stopTime-startTime)*(1000):.6f} ms")
```

```python
    #inisialisasi penghitungan operasi euclidean
    euclideancount = 0
    startTimeB = time.time()
    titikterdekat_bruteforce, jaraktitikterdekat_bruteforce = closestPair_bruteforce(titik)
    stopTimeB = time.time()
    print("="*40, "Strategi Bruteforce", "="*40)
    print("Pasangan titik terdekat : ", titikterdekat_bruteforce)
    print("Jaraknya : ", jaraktitikterdekat_bruteforce)
    print("banyak perhitungan euclidean : ", euclideancount)
    print(f"Lama program berjalan : {(stopTimeB-startTimeB)*(1000):.6f} ms")
    if (dimensi <= 3 and dimensi > 0):
        vis = input("Visualisasikan hasil? (Y/N) ")
        if (vis in "Yy"):
            print("Visualisasi hasil Divide and Conquer")
            bonus1.showcartesian(titikterdekat,titik)
        elif (vis in "Nn"):
            print("Tidak divisualisasikan")
    else:
        print("Dimensi abstrak, tidak divisualisasikan")
getSpecs.sysSpec()
```

bonus1.py

```python
import matplotlib.pyplot as plt

def showcartesian(titikterdekat,titik) :
    x = []
    y = []
    z = []
    xclose = []
    yclose = []
    zclose = []

    # masukin yg pasangan terdekat
    for i in range(len(titikterdekat)) :
        for j in range(0,3) :
            if(j == 0) :
                xclose.append(titikterdekat[i][j])
            elif(j == 1) :
                yclose.append(titikterdekat[i][j])
            else :
                zclose.append(titikterdekat[i][j])

    # masukin yg lain
    for i in range(len(titik)) :
        for j in range(0,3) :
            if(j == 0) :
                x.append(titik[i][j])
            elif(j == 1) :
                y.append(titik[i][j])
            elif(j == 2) :
                z.append(titik[i][j])

    fig = plt.figure()
    ax = fig.add_subplot(projection = '3d')

    for i in range(len(titik)) :
        ax.plot(x[i], y[i], z[i], 'ok')

    for i in range(len(titikterdekat)) :
```

```
# masukin yg lain
for i in range(len(titik)) :
    for j in range(0,3) :
        if(j == 0) :
            x.append(titik[i][j])
        elif(j == 1) :
            y.append(titik[i][j])
        elif(j == 2) :
            z.append(titik[i][j])

fig = plt.figure()
ax = fig.add_subplot(projection = '3d')

for i in range(len(titik)) :
    ax.plot(x[i], y[i], z[i], 'ok')

for i in range(len(titikterdekat)) :
    ax.plot(xclose[i], yclose[i], zclose[i], 'or')



ax.set_title('Diagram kartesian')
ax.set_xlabel('Sumbu X')
ax.set_ylabel('Sumbu Y')
ax.set_zlabel('Sumbu Z')

plt.show()
```

getSpec.py

```python
import platform
import psutil

def get_size(bytes, suffix="B"):
    factor = 1024
    for unit in ["", "K", "M", "G", "T", "P"]:
        if bytes < factor:
            return f"{bytes:.2f}{unit}{suffix}"
        bytes /= factor

def sysSpec():
    print("="*40, "Spesifikasi perangkat", "="*40)
    my_system = platform.uname()
    print(f"System: {my_system.system}")
    print(f"Node Name: {my_system.node}")
    print(f"Release: {my_system.release}")
    print(f"Version: {my_system.version}")
    print(f"Machine: {my_system.machine}")
    print(f"Processor: {my_system.processor}")
    # let's print CPU information
    print("="*40, "CPU Info", "="*40)
    # number of cores
    print("Physical cores:", psutil.cpu_count(logical=False))
    print("Total cores:", psutil.cpu_count(logical=True))
    # CPU frequencies
    cpufreq = psutil.cpu_freq()
    print(f"Max Frequency: {cpufreq.max:.2f}Mhz")
    print(f"Min Frequency: {cpufreq.min:.2f}Mhz")
    print(f"Current Frequency: {cpufreq.current:.2f}Mhz")
    # CPU usage
    print("CPU Usage Per Core:")
    for i, percentage in enumerate(psutil.cpu_percent(percpu=True, interval=1)):
        print(f"Core {i}: {percentage}%")
    print(f"Total CPU Usage: {psutil.cpu_percent()}%")
    # Memory Information
    print("="*40, "Memory Information", "="*40)
    # get the memory details
    svmem = psutil.virtual_memory()
    print(f"Total: {get_size(svmem.total)}")
    print(f"Available: {get_size(svmem.available)}")
    print(f"Used: {get_size(svmem.used)}")
    print(f"Percentage: {svmem.percent}%")
```

```python
print("="*20, "SWAP", "="*20)
# get the swap memory details (if exists)
swap = psutil.swap_memory()
print(f"Total: {get_size(swap.total)}")
print(f"Free: {get_size(swap.free)}")
print(f"Used: {get_size(swap.used)}")
print(f"Percentage: {swap.percent}%")
```

Bab 4
Hasil Pengujian

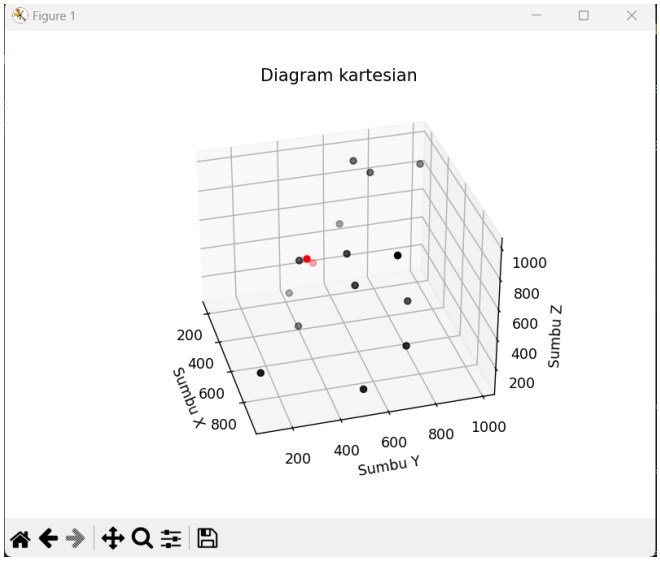
Untuk hasil pengujian yang akan ditampilkan akan menggunakan spesifikasi tugas utama yaitu pada bidang 3 dimensi

16 titik

```
==================================== WELCOME ====================================
apakah ingin menggunakan dimensi lain [default 3 dimensi] ? (Y/N) n
baik! default ke 3
Masukkan jumlah titik : 16
================================== Strategi Divide and Conquer ==================================
Pasangan titik terdekat :  ([794, 325, 912], [799, 348, 886])
Jaraknya :  35.07135583350036
banyak perhitungan euclidean :  83
Lama program berjalan : 0.000000 ms
================================== Strategi Bruteforce ==================================
Pasangan titik terdekat :  ([794, 325, 912], [799, 348, 886])
Jaraknya :  35.07135583350036
banyak perhitungan euclidean :  120
Lama program berjalan : 0.000000 ms
Visualisasikan hasil? (Y/N) y
Visualisasi hasil Divide and Conquer
```

Untuk hasil program yang 16 titik ini lama program berjalan tertulis 0 ms mungkin dikarenakan program berjalan terlalu cepat hingga tidak ke-*display* lama jalannya proses *divide and conquer* maupun *bruteforce*-nya.

visualisasi 16 titik :



64 titik

```
/Documents/ngoding/Sem 4/Stima/Tucil2_13521001_13521009/src/main.py
========================================= WELCOME =========================================
apakah ingin menggunakan dimensi lain [default 3 dimensi] ? (Y/N) n
baik! default ke 3
Masukkan jumlah titik : 64
====================================== Strategi Divide and Conquer ======================================
Pasangan titik terdekat :  ([279, 541, 247], [321, 533, 270])
Jaraknya :  48.54894437575342
banyak perhitungan euclidean :  1500
Lama program berjalan : 14.780998 ms
========================================= Strategi Bruteforce ==========================================
Pasangan titik terdekat :  ([279, 541, 247], [321, 533, 270])
Jaraknya :  48.54894437575342
banyak perhitungan euclidean :  2016
Lama program berjalan : 1.997232 ms
Visualisasikan hasil? (Y/N) n
Tidak divisualisasikan
```

128 titik

```
====================================== Strategi Divide and Conquer ======================================
Pasangan titik terdekat :  ([48, 380, 222], [67, 388, 237])
Jaraknya :  25.495097567963924
banyak perhitungan euclidean :  3323
Lama program berjalan : 7.015467 ms
========================================= Strategi Bruteforce ==========================================
Pasangan titik terdekat :  ([48, 380, 222], [67, 388, 237])
Jaraknya :  25.495097567963924
banyak perhitungan euclidean :  8128
Lama program berjalan : 5.000114 ms
Visualisasikan hasil? (Y/N) n
Tidak divisualisasikan
```

1000 titik

```
Python/Python311/python.exe C:/Users/Lenovo/Documents/ngoding/Sem 4/Stima/Tucil2_13521001_13521009/src/m
========================================= WELCOME =========================================
apakah ingin menggunakan dimensi lain [default 3 dimensi] ? (Y/N) n
baik! default ke 3
Masukkan jumlah titik : 1000
====================================== Strategi Divide and Conquer ======================================
Pasangan titik terdekat :  ([194, 997, 522], [199, 998, 522])
Jaraknya :  5.0990195135927845
banyak perhitungan euclidean :  81083
Lama program berjalan : 67.513466 ms
========================================= Strategi Bruteforce ==========================================
Pasangan titik terdekat :  ([194, 997, 522], [199, 998, 522])
Jaraknya :  5.0990195135927845
banyak perhitungan euclidean :  499500
Lama program berjalan : 277.294397 ms
Visualisasikan hasil? (Y/N) y
Visualisasi hasil Divide and Conquer
```

visualisasi 1000 titik :

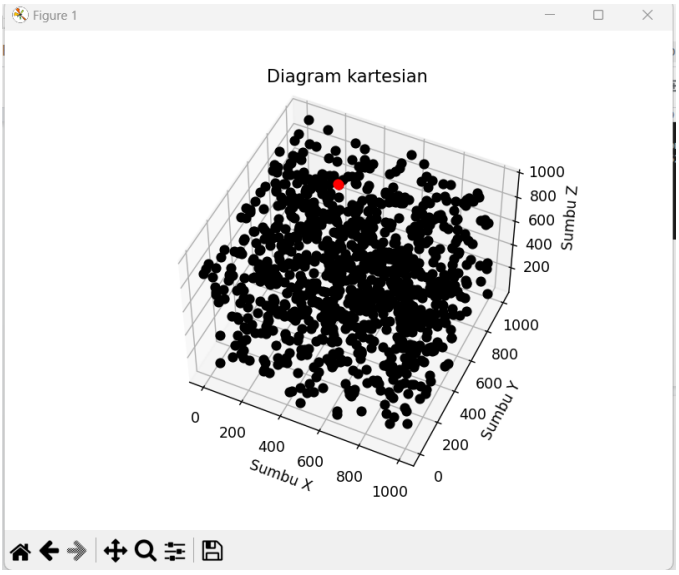


Figure 1
Diagram kartesian
Sumbu X
Sumbu Y
Sumbu Z
0
200
400
600
800
1000

Bagian 2 Lampiran

Berikut link repository GitHub source code yang dikerjakan : [Tucil2_13521001_13521009]

Ceklist untuk asisten:

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa 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	√	