PENCARIAN PASANGAN TITIK TERDEKAT MENGGUNAKAN ALGORITMA DIVIDE AND CONQUER

LAPORAN TUGAS KECIL
IF2211
STRATEGI ALGORITMA

Oleh

Varraz Hazzandra Abrar
13521020
K03

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
INSTITUT TEKNOLOGI BANDUNG
2022

DESKRIPSI TEORI

Mencari sepasang titik terdekat dengan Algoritma Divide and Conquer sudah dijelaskan di dalam kuliah. Persoalan tersebut dirumuskan untuk titik pada bidang datar (2D). Pada Tucil 2 kali ini Anda diminta mengembangkan algoritma mencari sepasang titik terdekat pada bidang 3D. Misalkan terdapat n buah titik pada ruang 3D. Setiap titik P di dalam ruang dinyatakan dengan koordinat P = (x, y, z). Carilah sepasang titik yang mempunyai jarak terdekat satu sama lain. Jarak antara dua titik dihitung dengan rumus Euclidian.

DASAR TEORI

1. Divide and Conquer

Algoritma divide and conquer merupakan teknik menyelesaikan suatu persoalan dengan memecah persoalan menjadi beberapa upa-persoalan yang lebih kecil sampai upa-persoalan tidak bisa dipecah lagi. Algoritma divide and conquer terbagi menjadi 3 langkah, yaitu divide, conquer, dan combine.

1. Langkah divide : memecah persoalan menjadi beberapa upa-persoalan yang punya kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya berukuran hampir sama).

2. Langkah conquer (solve) menyelesaikan masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar).

3. Langkah combine mengabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

2. Penerapan Divide and Conquer pada Pencarian *Closest Pair* di Ruang 3 dimensi

Pertama, titik-titik akan diurutkan berdasarkan koordinat x, y, dan z secara terpisah tiap koordinatnya. Titik-titik yang sudah terurut akan dimasukkan ke dalam *list* untuk masing-masing koordinat. Lalu, akan dilakukan pemanggilan fungsi yang menerapkan algoritma *divide and conquer*.

Algoritma *divide and conquer* akan diterapkan dengan membuat fungsi sendiri. Pertama, akan dicek banyak titik-titik pada *list* untuk koordinat x. Jika banyak titik <= 3, algoritma *brute force* akan diterapkan pada titik-titik tersebut untuk mencari *closest pair*. Jika lebih dari 3, langkah pertama adalah membagi titik-titik pada *list* untuk koordinat x menjadi 2 bagian, yaitu bagian kiri dan kanan dengan indeks tengah berupa *mid*. Fungsi akan memanggil dirinya sendiri untuk mencari *closest pair* di kedua bagian. *Closest pair* keduanya akan dibandingkan dan didapatkan *closest pair* terkecil di antara kedua sisi dan disimpan di variable minDist. Lalu, fungsi akan memeriksa area sekitar *mid* dengan jangkauan sejauh minDist ke kanan dan ke kiri dari indeks *mid*. Titik-titik yang berada di dalam area jangkauan akan disimpan dalam *list* strip. Akan dicari *closest pair* di strip, lalu dibandingkan dengan minDist yang menyimpan *closest pair* hasil perbandingan bagian kiri dan kanan. Hasil perbandingan terbaru akan disimpan di variable minDist. Akan didapat berupa jarak terdekat dan *closest pair*

IMPLEMENTASI

Penulis membuat dua *file* untuk menyelesaikan persoalan *closest pair* di ruang 3 dimensi.

1. algoClPair.py

Program ini berisi penerapan algoritma *brute force* dan *divide and conquer* dalam menemukan *closest pair*. Program ini terdiri dari beberapa fungsi

a. fungsi eucDist : fungsi ini mencari jarak Euclidian antara dua titik.

```
def eucDist(p1, p2):
    return ((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2 + (p1[2]-p2[2])**2)**0.5
```

b. fungsi bfClosestPair : fungsi ini mencari *closest pair* dengan algoritma brute force.

```
# Algoritma brute force
def bfClosestPair(points):
    minDist = float('inf')
    clPair = None
    countEuc = 0
    totPoints = len(points)
    for i in range(totPoints):
        for j in range(i+1, totPoints):
            dist = eucDist(points[i], points[j])
            countEuc += 1
            if dist < minDist:
                clPair = (points[i], points[j])
                minDist = dist
    return clPair, minDist, countEuc
```

c. fungsi quicksort : menerapkan algoritma quick sort yang menggunakan pendekatan divide and conquer untuk mengurutkan titik-titik

```
def quicksort(points, dim):
    if len(points) <= 1:
        return points
    else :
        pivot = points[len(points) // 2][dim]
        left, right, equal = [], [], []
        for point in points:
            if point[dim] < pivot:
                left.append(point)
            elif point[dim] > pivot:
                right.append(point)
            else:
                equal.append(point)
        return quicksort(left, dim) + equal + quicksort(right, dim)
```

d. fungsi sorting : fungsi ini mengurutkan titik-titik berdasarkan koordinat x, y, dan z.

```
def sorting(points):
    pointsX = sorted(points, key=lambda x: x[0])
    pointsY = sorted(points, key=lambda x: x[1])
    pointsZ = sorted(points, key=lambda x: x[2])
    return pointsX, pointsY, pointsZ
```

e. fungsi dcClosestPair : fungsi ini mencari *closest pair* dengan algoritma *divide and conquer.*

```
def dcClosestPair(pointsX, pointsY, pointsZ):
    totPointsX = len(pointsX)
    if totPointsX <= 3:
        return bfClosestPair(pointsX)
    else:
        mid = totPointsX // 2
        midPoint = pointsX[mid]
        leftPointsY = [p for p in pointsY if p[0] <= midPoint[0]]
        rightPointsY = [p for p in pointsY if p[0] > midPoint[0]]
```

```
        leftPointsZ = [p for p in pointsZ if p[0] <= midPoint[0]]
        rightPointsZ = [p for p in pointsZ if p[0] > midPoint[0]]
        leftPair, leftMinDist, leftCountEuc = dcClosestPair(pointsX[:mid],
leftPointsY, leftPointsZ)
        rightPair, rightMinDist, rightCountEuc = dcClosestPair(pointsX[mid:],
rightPointsY, rightPointsZ)
        if leftMinDist < rightMinDist:
            minDist = leftMinDist
            clPair = leftPair
        else:
            minDist = rightMinDist
            clPair = rightPair
        strip = [p for p in pointsY if abs(p[0] - midPoint[0]) < minDist]
        totPointsStrip = len(strip)
        for i in range(totPointsStrip):
            j = i + 1
            while j < totPointsStrip and strip[j][1] - strip[i][1] < minDist:
                dist = eucDist(strip[i], strip[j])
                if dist < minDist:
                    minDist = dist
                    clPair = (strip[i],strip[j])
                j += 1
        countEuc = leftCountEuc + rightCountEuc + j - i
        return clPair, minDist, countEuc
```

2. eksekutorAlgoClPair.py

Program ini merupakan program utama yang akan mengeluarkan keluaran hasil pencarian *closest pair*.

a. *import library* dan *file* algoClPair.py untuk memanggil fungsi di dalamnya

```
import random
import time
import algoClPair
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

b. Pemrosesan masukan titik-titik secara acak sejumlah n dan visualisasi titik-titik

```python
# Masukan berupa titik-titik sejumlah n yang dibangkitkan secaran acak dan validasi masukan agar n>=2
while True:
    n = int(input("Masukkan jumlah titik: "))
    if n >= 2:
        break
    else:
        print("Jumlah titik harus >= 2. Silakan coba lagi.")

points = [(random.uniform(0, 2000), random.uniform(0, 2000), random.uniform(0, 2000)) for i in range(n)]

print("Titik-titik yang dibangkitkan : ")

for p in points:
    print(p)

visual = plt.figure()
ax = visual.add_subplot(111, projection='3d')
xs = [p[0] for p in points]
ys = [p[1] for p in points]
zs = [p[2] for p in points]
ax.scatter(xs, ys, zs, c='black')
```

c. Pemrosesan pencarian *closest pair* dengan algoritma *brute force*

```python
# Menggunakan algoritma brute force
starting = time.time()
clPairBF, minDistBF, countEucBF = algoClPair.bfClosestPair(points)
ending = time.time()
durationBF = ending - starting
```

d. Pemrosesan pencarian *closest pair* dengan algoritma *divide and conquer*

```python
# Menggunakan algoritma divide and conquer
starting = time.time()
pointsX, pointsY, pointsZ = algoClPair.sorting(points)
clPairDC, minDistDC, countEucDC = algoClPair.dcClosestPair(pointsX, pointsY, pointsZ)
ending = time.time()
durationDC = ending - starting
```

e. Pemberian warna pada titik-titik dan penanda pada *closest pair*

```python
# Mewarnai titik-titik pada pasangan terdekat dengan warna kuning dan garis penghubung antara pasangan terdekat deng
for p in clPairDC:
    ax.scatter(p[0], p[1], p[2], c='yellow')
    x = [p[0], clPairDC[0][0]]
    y = [p[1], clPairDC[0][1]]
    z = [p[2], clPairDC[0][2]]
    ax.plot(x, y, z, c='red')
```

f. Mencetak keluaran dari program dan pemanggilan fungsi yang menampilkan visualisasi hasil program

```
print("\nHasil pencarian pasangan titik terdekat dengan algoritma brute force : ")
print("Pasangan titik terdekat :", clPairBF)
print("Jarak terdekat:", minDistBF)
print("Jumlah operasi euclidean terpakai:", countEucBF)
print("Lama waktu pencarian:", durationBF, "detik")

print("\nHasil pencarian pasangan titik terdekat dengan algoritma divide and conquer:")
print("Pasangan titik terdekat:", clPairDC)
print("Jarak terdekat:", minDistDC)
print("Jumlah operasi euclidean terpakai:", countEucDC)
print("Lama waktu pencarian:", durationDC, "detik")

plt.show()
```

EKSPERIMEN

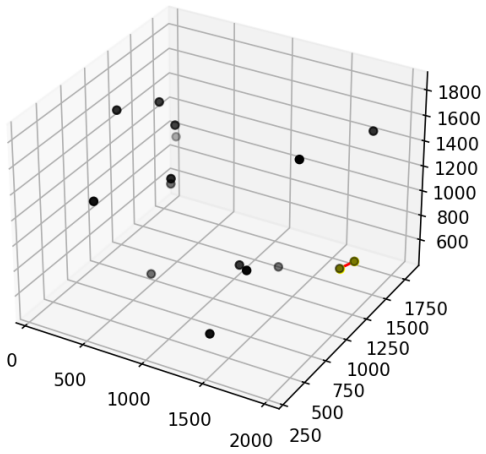
Eksperimen dilakukan di laptop Acer Swift SF314-510G

n = 16

```
PS D:\kodinh> python3 eksekutorAlgoClPair.py
Masukkan jumlah titik: 16
Titik-titik yang dibangkitkan :
(1642.2389627069028, 500.31788682995517, 1121.5703118604501)
(367.70876693381837, 782.6160053376364, 1839.6350700749497)
(537.3273232318398, 316.47187276128454, 1455.7569739987355)
(511.66821506803603, 938.6097101284863, 483.3479800917708)
(1792.1404792063367, 872.833381726698, 1781.2230641623078)
(637.1450109813281, 1063.876008735738, 1635.7204275354704)
(1976.964100752217, 1481.1597176757518, 1697.2964665111008)
(1349.3712262105844, 827.0693672779232, 870.498535978146)
(1292.9520569658382, 1372.1111876809123, 491.52395883313346)
(60.65480043050453, 1823.7192154975164, 966.1021966204175)
```

```
Hasil pencarian pasangan titik terdekat dengan algoritma brute force :
Pasangan titik terdekat : ((1723.584294572711, 1500.8560332256297, 520.2202489377461), (1769.6198397714734, 1617.926821637726, 518.2465503639124))
Jarak terdekat: 125.81230626308968
Jumlah operasi euclidean terpakai: 120
Lama waktu pencarian: 0.0 detik

Hasil pencarian pasangan titik terdekat dengan algoritma divide and conquer:
Pasangan titik terdekat: ((1723.584294572711, 1500.8560332256297, 520.2202489377461), (1769.6198397714734, 1617.926821637726, 518.2465503639124))
Jarak terdekat: 125.81230626308968
Jumlah operasi euclidean terpakai: 15
Lama waktu pencarian: 0.0 detik
```

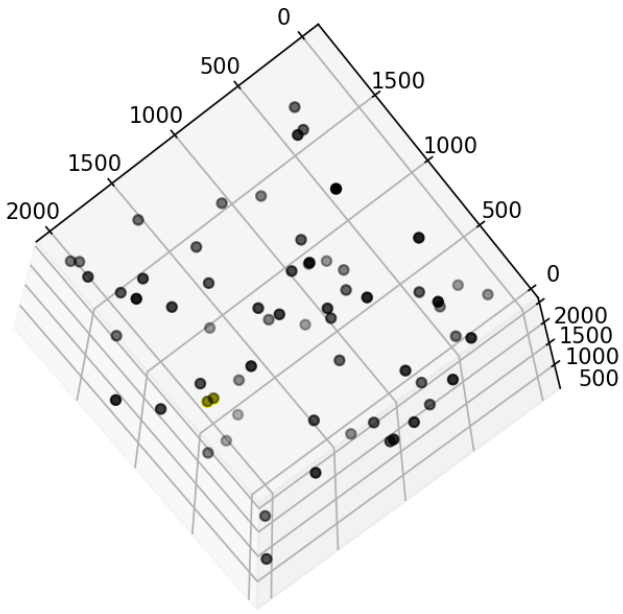


n = 64

```
Masukkan jumlah titik: 64
Titik-titik yang dibangkitkan :
(1488.088071039365, 1716.5551099738923, 180.13201576021686)
(654.0461405616036, 89.74266497966954, 841.9461235992429)
(787.696594725297, 1191.4071484188307, 864.995066519979)
(1966.421156068008, 1340.337325130375, 143.15177770493736)
(1755.7157055302796, 1871.261250874538, 858.3399715062995)
(969.4501865003333, 1084.5744121806829, 602.5503858548933)
(1614.8942183019346, 1029.0540618345897, 653.5857630526726)
```

```
Hasil pencarian pasangan titik terdekat dengan algoritma brute force :
Pasangan titik terdekat : ((1787.655222341668, 782.7876483515803, 1355.442469519685), (1749.7017331441368, 772.0321686776576, 1377.9847973527724))
Jarak terdekat: 45.43461488136181
Jumlah operasi euclidean terpakai: 2016
Lama waktu pencarian: 0.0 detik

Hasil pencarian pasangan titik terdekat dengan algoritma divide and conquer:
Pasangan titik terdekat: ((1749.7017331441368, 772.0321686776576, 1377.9847973527724), (1787.655222341668, 782.7876483515803, 1355.442469519685))
Jarak terdekat: 45.43461488136181
Jumlah operasi euclidean terpakai: 63
Lama waktu pencarian: 0.0 detik
```

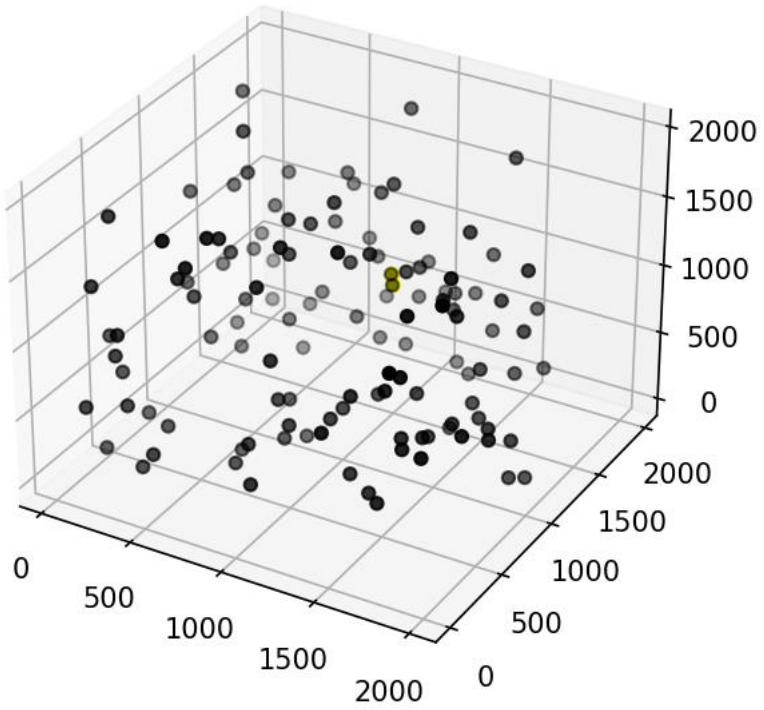


n = 128

```
PS D:\kodinh> python3 eksekutorAlgoClPair.py
Masukkan jumlah titik : 128
Titik-titik yang dibangkitkan :
(1861, 926, 1310)
(1785, 82, 752)
(1254, 1952, 163)
(1505, 66, 689)
```

```
Hasil pencarian pasangan titik terdekat dengan algoritma brute force :
Pasangan titik terdekat : ((985.2904484251794, 1508.578075819561, 824.2748705399712), (947.9018229937286, 1562.1765388256117, 857.120644916976))
Jarak terdekat: 73.14061418042247
Jumlah operasi euclidean terpakai: 8128
Lama waktu pencarian: 0.0 detik

Hasil pencarian pasangan titik terdekat dengan algoritma divide and conquer:
Pasangan titik terdekat: ((985.2904484251794, 1508.578075819561, 824.2748705399712), (947.9018229937286, 1562.1765388256117, 857.120644916976))
Jarak terdekat: 73.14061418042247
Jumlah operasi euclidean terpakai: 127
Lama waktu pencarian: 0.015752792358398438 detik
```

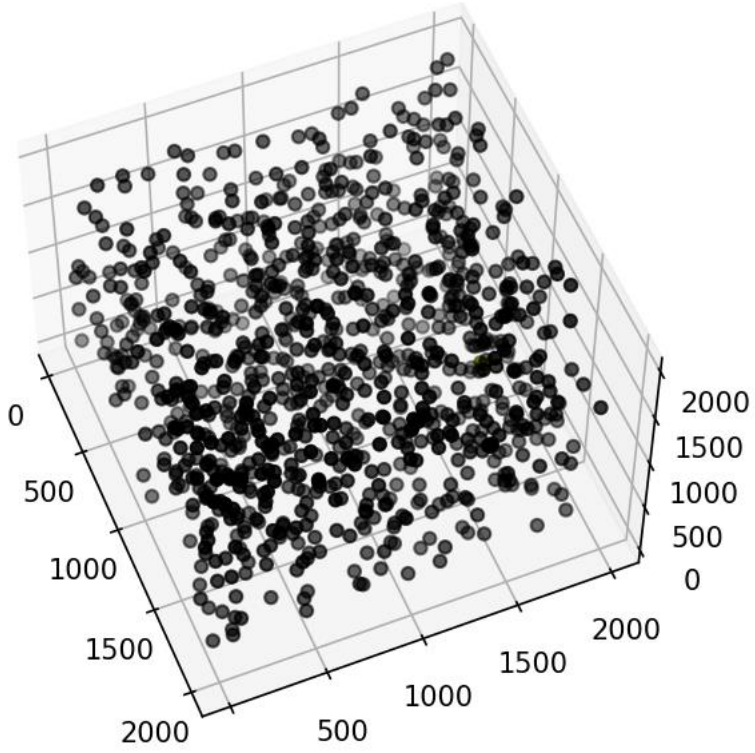


n = 1000

```
PS D:\kodinh> python3 eksekutorAlgoClPair.py
Masukkan jumlah titik : 1000
```

```
Hasil pencarian pasangan titik terdekat dengan algoritma brute force :
Pasangan titik terdekat : ((886.1173098213824, 1843.8275070319255, 137.62464622761604), (902.1982852823478, 1854.6355784678653, 135.51682092216043
))
Jarak terdekat: 19.489871920022193
Jumlah operasi euclidean terpakai: 499500
Lama waktu pencarian: 0.3791213035583496 detik

Hasil pencarian pasangan titik terdekat dengan algoritma divide and conquer:
Pasangan titik terdekat: ((886.1173098213824, 1843.8275070319255, 137.62464622761604), (902.1982852823478, 1854.6355784678653, 135.51682092216043)
)
Jarak terdekat: 19.489871920022193
Jumlah operasi euclidean terpakai: 1023
Lama waktu pencarian: 0.0 detik
```



LAMPIRAN

Pranala repositori : https://github.com/varrazha/Tucil2_13521020

POIN	YA	TIDAK
Program berhasil dikompilasi tanpa kesalahan	✓	
Program berhasil running	✓	
Program dapat menerima masukan dan memberikan luaran	✓	
Luaran program sudah benar (solusi closest pair benar)	✓	
Bonus 1 dikerjakan	✓	
Bonus 2 dikerjakan		✓