**LAPORAN TUGAS KECIL 2 IF2211 STRATEGI ALGORITMA**

**MENCARI PASANGAN TITIK TERDEKAT 3D DENGAN ALGORITMA DIVIDE AND CONQUER**

**Dosen : Dr. Ir. Rinaldi, M.T.**

****

**Disusun oleh :**

**Vieri Fajar Firdaus 13521099**

**Sekolah Teknik Elektro dan Informatika**

**Institut Teknologi Bandung**

**2022/2023**

**Bab 1 Deskripsi Masalah**

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 dua buah titk *P*1 = (*x*1, *y*1, *z*1) dan *P*2 = (*x*2, *y*2, *z*2) dihitung dengan rumus Euclidean berikut:

Buatlah program dalam dalam Bahasa C/C++/Java/Python/Golang/Ruby/Perl (pilih salah  
satu) untuk mencari sepasang titik yang jaraknya terdekat datu sama lain dengan menerapkan  
algoritma *divide and conquer* untuk penyelesaiannya, dan perbandingannya dengan  
Algoritma *Brute Force*.  
Masukan program:

* *n*
* titik-titik (dibangkitkan secara acak) dalam koordinat (x, y, z)

Luaran program :

* sepasang titik yang jaraknya terdekat dan nilai jaraknya
* banyaknya operasi perhitungan rumus Euclidian
* waktu riil dalam detik (spesifikasikan komputer yang digunakan)
* Bonus 1 (Nilai = 7,5) penggambaran semua titik dalam bidang 3D, sepasang titik  
  yang jaraknya terdekat ditunjukkan dengan warna yang berbeda dari titik lainnya.
* Bonus 2 (nilai = 7,5): Generalisasi program anda sehingga dapat mencari sepasang titik terdekat untuk sekumpulan vektor di Rn, setiap vektor dinyatakan dalam bentuk **x** = (x1, x2,…, xn)

**Bab 2 Algoritma**

Landasar teori yang akan digunakan pada penerapan strategi algoritma dalam mencari jarak minimum dari titik pada 3 dimesi yaitu: algoritma *Divide and Conquer* dan *Brute Force*

1. Definisi Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* merupakan algoritma umum yang digunakan untuk memecahkan suatu masalah besar menjadi beberapa masalah kecil dan menyelesaikan masalah kecil secara terpisah, kemudian menggabungkan solusi masalah kecil dan memecahkan masalah asli. Dari namanya Divide and Conquer dibagi menjadi tiga langkah dasar:

1. Divide: membagi masalah menjadi masalah-masalah yang lebih kecil. Biasanya  
   menjadi setengahnya atau mendekati setengahnya
2. Conquer: ketika sebuah masalah sudah cukup kecil untuk diselesaikan, langsung  
   selesaikan masalah tersebut.
3. Combine: ketika sebuah masalah sudah cukup kecil untuk diselesaikan, langsung  
   selesaikan masalah tersebut.

Algoritma *Divide and Conquer* digunakan dalam banyak aplikasi, termasuk komputasi paralel, optimasi dan pembuatan keputusan. Beberapa contoh dari masalah yang dapat dipecahkan dengan penedakat ini meliputi *sorting,* pencarian elemen terntentu, penghitungan nilai fungsi matematika. Dalam implementasinya, algoritma *Divide and* *Conquer* dapat memberikan solusi yang efektif daripada pendekatan algoritma lainnya.

1. Implementasi *Divide and Conquer* pada algoritma *quicksort*

Terdapat algoritma pengurutan yang memiliki kompleksitas cukup cepat yaitu *quicksort*. *Quicksort* menggunakan prinsip *divide and conquer* dalam pengurutannya. Tahapan dari pengurutan *quicksort* adalah sebagai berikut :

1. Divide: pilih suatu elemen yang kita sebut *pivot,* kemudian kita bagi *array* menjadi dua sehingga *object* dalam array *pivot* dan yang lainnya selalu *pivot.*
2. Conquer: ketika *array* hanya memiliki satu elemen, *array* tersebut sudah terurut
3. Combine: gabunglah *subarray* dengan menempelkan hasil quicsort bagian kiri dan bagian kanan

Pada implementasi yang dilakukan sekarang ini, pivot secara awal dipilih dari indeks ke-0

1. Implementasi algoritma *closest pair*

Algoritma closest pair merupakan algortima pencarian pasangan titik dari himpunan semesta sehingga jarak pasangan titik tersebut minimum. Dengan menggunakan algoritma *brute force*, pencarian *closest pair* akan mudah dilakukan dengan cara melakukan pemilihan dua titik dari himpunan semesta, sehingga apabila terdapat titik maka akan ada kemungkinan, dengan mengecek kemungkinan maka didapatkan *closest pair* dengan algoritma *brute force*. Namun algoritma ini dirasa cukup lama dibandingkan algoritma *Divide and Conquer*, untuk kompleksitas dari algoritma *brute force* adalah . Namun pada kali ini akan dicoba dengan menggunakan algoritma *divide and conquer,* untuk implementasinya pada pencarian *closest pair* sebagai berikut:

1. Divide: Sebelum melakukan algortima *DnC* pastika titik sudah terurut berdasarkan koordinat x, jika sudah terurut bagi kedua himpunan titik menjadi 2 bagian, dan cari closest pair dari kedua titik tersebut, serta pilih titik tengah sebagai median
2. Conquer: ketika jumlah titik maka lakukanlah *brute force* untuk mendapatkan pasangan titik terpendek
3. Combine: cari closest pair dari himpunan titik satu dan himpunan titik dua (himpunan titik satu dan dua telah dilakukan pada bagian divide), pilih jarak terpendek dari kedua himpunan dan simpan dalam *temp\_min*. Lakukan *traversal* sebanyak jumlah titik untuk mencari titik dengan selisih koordinat x terhadap titik median kurang dari *temp\_min* lalu kumpulkan titik titik tersebut ke dalam array *temp\_solution*. Setelah didapat array *temp\_solution,* lakukan brute force pada *temp\_solution* untuk mencari selisih yang lebih kecil dari *temp\_min,* untuk mengoptimalisasi pencarian dilakukan pengecekan apabila selisih koordinat lebih dari temp\_min maka tidak akan dilakukan penghitungan jarak

Untuk kompleksitas pada dimensi 3 yaitu dan untuk kompleksitas pada dimensi-d yaitu

1. Algoritma Penggambaran titik pada 3D dan 2D

Untuk penggambaran titik pada 3D dan 2D digunakan library matplotlib untuk menggambarnya, untuk menandakan pasangan titik terdekat digambarkan dengan titik merah dan dihubungkan oleh garis berwarna merah.

Secara singkat kode akan mengelompokkkan koordinat x, koordinat y, dan koordinat z lalu akan dilakukan plot pada bidang 3D,untuk menandai pasangan titik terdekat akan digambarkan garis dengan warna merah setelah itu akan dilakukan pelabelan sumbu x, sumbu y, dan sumbu z. Untuk lebih lengkapnya kdoe dapat dilihat pada bagian *source code*

**Bab 4 Kode Program dalam Bahasa Python**

1. Kode main.py

|  |
| --- |
| from input import \*  from point import \*  from sorting import \*  from plot import \*  import time  def main() :      point=[]      (derajatcnt,titikcnt)=inp()      point=pointinput(derajatcnt,titikcnt)      sorting(point,0,len(point)-1,0)      dncawal=time.time()      iDNC,jDNC,solutionDNC,cntDNC=Closest(point,0,titikcnt-1)      dncakhir=time.time()      iBF,jBF,solutionBF,cntBF=brute\_force(point,0,titikcnt-1)      bfakhir=time.time()      aa=(dncakhir-dncawal)      bb=(bfakhir-dncakhir)      print("Solusi koordinat titik berdasarkan DNF: ")      print(point[iDNC])      print(point[jDNC])      print("Dengan jarak :",solutionDNC)      print("Solusi koordinat titik berdasarkan BF: ")      print(point[iBF])      print(point[jBF])      print("Dengan jarak :",solutionBF)      print("DNC :",aa\*1000,"ms")      print("BF :",bb\*1000,"ms")      print("Banyak proses DNC:",cntDNC)      print("Banyak proses BF:",cntBF)      if(derajatcnt==2 or derajatcnt==3) :          tampilkan=input("Apakah ingin menampilkan plot dalam 2D atau 3D? (y/n): ")          if(tampilkan=="y") :              if(derajatcnt==3) :                  plot3d(point,iDNC,jDNC)              if(derajatcnt==2) :                  plot2d(point,iDNC,jDNC)  main() |

1. Kode input.py

|  |
| --- |
| from function import \*  from sorting import \*  import random  def inp() :      derajatcnt=checkkint("Masukkan banyak derajat : ")      titikcnt=checkkint("Masukkan banyak titik : ")      return (derajatcnt,titikcnt)  def pointinput(derajatcnt,titikcnt) :      arr=[]      for i in range(titikcnt) :          arr.append([])          for j in range(derajatcnt) :              arr[i].append(random.uniform(-100,100))      return arr |

1. Kode sorting.py

|  |
| --- |
| def swap(a,b) :      return (b,a)  def partition(arr,l,r,param) :      pivot=arr[r][param]        i=l-1        for j in range(l,r) :          if arr[j][param]<pivot :              i+=1              arr[i],arr[j]=swap(arr[i],arr[j])        arr[i+1],arr[r]=swap(arr[i+1],arr[r])      return i+1  def sorting(arr,l,r,param) :      if l<r :          pi=partition(arr,l,r,param)          sorting(arr,l,pi-1,param)          sorting(arr,pi+1,r,param) |

1. Kode point.py

|  |
| --- |
| from function import \*  from brute\_force import \*  def Closest(point,start,end) :      point\_i=-1      point\_j=-1      solution=float('inf')      cnt\_distance=0      if(end-start<=3) :          return brute\_force(point,start,end)      mid=(end+start)//2      min\_left=Closest(point,start,mid)      min\_right=Closest(point,mid+1,end)      cnt\_distance=min\_left[3]+min\_right[3]      if(min\_left[2]<min\_right[2]) :          point\_i=min\_left[0]          point\_j=min\_left[1]          solution=min\_left[2]      else :          point\_i=min\_right[0]          point\_j=min\_right[1]          solution=min\_right[2]      temp\_solution=[mid]      for i in range(start,end+1) :          if(abs(point[i][0]-point[mid][0])<solution and i!=mid) :  *# if(distance(point[i],point[mid])<solution and i!=mid) :*              temp\_solution.append(i)      for i in range(len(temp\_solution)) :          for j in range(i+1,len(temp\_solution)) :              if(deltacondition(point,temp\_solution[i],temp\_solution[j],solution)) :                  cnt\_distance+=1                  if(distance(point[temp\_solution[i]],point[temp\_solution[j]])<solution) :                      solution=distance(point[temp\_solution[i]],point[temp\_solution[j]])                      point\_i=temp\_solution[i]                      point\_j=temp\_solution[j]        return (point\_i,point\_j,solution,cnt\_distance) |

1. Kode brute\_force.py

|  |
| --- |
| from function import \*  def brute\_force(point,start,end) :      cnt\_distance=0      solution=float('inf')      point\_i=start      point\_j=end        for i in range(start,end+1) :          for j in range(i+1,end+1) :              cnt\_distance+=1              if(distance(point[i],point[j])<solution) :                  solution=distance(point[i],point[j])                  point\_i=i                  point\_j=j      return (point\_i,point\_j,solution,cnt\_distance) |

1. Kode plot.py

|  |
| --- |
| import matplotlib.pyplot as plt  from mpl\_toolkits.mplot3d import Axes3D  import numpy as np  def plot3d(arr,i,j) :      xs=[]      ys=[]      zs=[]      for ii in range(len(arr)) :          xs.append(arr[ii][0])          ys.append(arr[ii][1])          zs.append(arr[ii][2])      fig = plt.figure()      ax = fig.add\_subplot(111, projection='3d')      ax.scatter(xs,ys,zs)      ax.scatter([arr[i][0],arr[j][0]],[arr[i][1],arr[j][1]],[arr[i][2],arr[j][2]],c='red',s=30)      ax.plot([arr[i][0],arr[j][0]],[arr[i][1],arr[j][1]],[arr[i][2],arr[j][2]],color='red')      ax.set\_xlabel("Sumbu X")      ax.set\_ylabel("Sumbu Y")      ax.set\_zlabel("Sumbu Z")      plt.show()  def plot2d(arr,i,j) :      xs=[]      ys=[]      for ii in range(len(arr)) :          xs.append(arr[ii][0])          ys.append(arr[ii][1])      plt.scatter(xs,ys)      plt.scatter([arr[i][0],arr[j][0]],[arr[i][1],arr[j][1]],c='red',s=30)      plt.plot([arr[i][0],arr[j][0]],[arr[i][1],arr[j][1]],color='red')      plt.show() |

1. Kode function.py

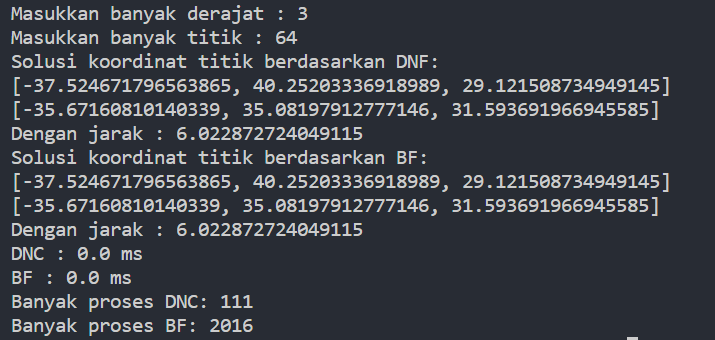
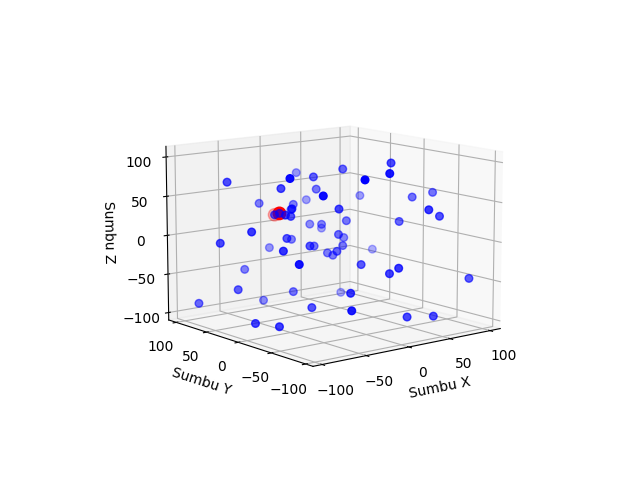
|  |
| --- |
| import random  def swap(a,b) :      return (b,a)  def distance(a,b) :      res=0.0      for i in range(len(a)) :          res+=(a[i]-b[i])\*\*2      return res\*\*0.5  def checkkint(msg):      while 1:          try:              n = input(msg)              return(int(n))          except ValueError:              print("Masukkan sebuah bilangan bulat")  def deltacondition(point,i,mid,temp\_min) :      res=True      for j in range(len(point[0])) :          if(abs(point[i][j]-point[mid][j])<temp\_min) :              res=True          else :              return False      return res |

**Bab 5 Dokumentasi**

1. **Chart, scatter chart

   Description automatically generated**Text

   Description automatically generatedBanyak titik 16 dengan banyak derajat tiga
2. Banyak titik 64 dengan banyak derajat tiga



1. Chart, scatter chart

   Description automatically generatedText

   Description automatically generatedBanyak titik 128 dengan banyak derajat tiga
2. Banyak titik 1000 dengan banyak derajat tiga

Text

Description automatically generatedChart, scatter chart

Description automatically generated

1. Text

   Description automatically generatedBanyak titik 16 dengan banyak derajat empat
2. Banyak titik 100 dengan banyak derajat 10

Text

Description automatically generated

1. Banyak titik 200 dengan banyak derajat 20

Text

Description automatically generated

1. Banyak titik 1000 dengan derajat empat

**Graphical user interface, text

Description automatically generated**

**Bab 6 Tabel Penilaian**

|  |  |  |  |
| --- | --- | --- | --- |
| No | Poin | Ya | Tidak |
| 1 | Program berhasil dikompilasi tanpa ada kesalahan | V |  |
| 2 | Program berhasil running | V |  |
| 3 | Program dapat menerima masukan dan menuliskan luaran | V |  |
| 4 | Luaran program sudah benar (solusi *closest pair* sudah benar) | V |  |
| 5 | Bonus 1 dikerjakan | V |  |
| 6 | Bonus 2 dikerjakan | V |  |

**Bab 7 Kesimpulan**

Dalam melakukan pencari *closest pair* terdapat berbagai macam metode yang dapat diaplikasikan, diantaranya *brute force* dan *divide and conquer*. Salah satunya adalah dengan menggunakan metode *divide and conquer:* dengan metode divide, conquer dan combine. Namun untuk beberapa kasus penggunaan brute force dirasa lebih cepat dibandingkan dnc untuk kasus dengan derajat tinggi (lebih dari 20)

Pada Tugas Kecil 2 Strategi Algoritma ini, penulis membuat aplikasi pencarian *closest pair*. Aplikasi dibuat dengan menggunakan bahasa python, bahasa ini dipilih karena terdapat library matplotlib untuk menggambarkan titik pada bangun 3 dimensi. Program ini dirasa cukup efektif namun masih belum begitu baik, penulis berharap bisa meningkatkan efektivitas program

**Bab 8 Referensi**

* <http://www.niser.ac.in/~aritra/CG/ClosestPairProblem.pdf>
* <http://people.csail.mit.edu/indyk/6.838-old/handouts/lec17.pdf>
* <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf>

Link Repository Github : <https://github.com/vierifirdaus/Tucil2_13521099>