TUGAS EKSPLORASI MANDIRI
PERANCANGAN DAN ANALISIS ALGORITMA
" THE CLOSEST-PAIR PROBLEMS "

DOSEN PENGAMPU:

Randi Proska Sandra, S.Pd, M.Sc

OLEH:

Fadhli Akbar Sahendra

23343036

Informatika

PROGRAM STUDI INFORMATIKA

DEPARTEMEN ELEKTRONIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

2024

PENJELASAN PROGRAM / ALGORITMA

The Closest-Pair Problem adalah permasalahan dalam komputasi geometris yang bertujuan untuk menemukan pasangan titik dengan jarak terdekat di antara sekumpulan titik pada bidang 2D. Penyelesaian masalah ini memiliki banyak aplikasi dalam berbagai bidang, seperti pengenalan pola, robotika, sistem navigasi, dan analisis data spasial.
Untuk menyelesaikan masalah ini, terdapat dua pendekatan utama, yaitu **Brute Force** dan **Divide and Conquer**:

1. **Pendekatan Brute Force**
   Pendekatan **Brute Force** bekerja dengan cara menghitung jarak antara semua pasangan titik yang mungkin, lalu memilih pasangan dengan jarak terpendek. Meskipun sederhana dan mudah diimplementasikan, metode ini memiliki kompleksitas waktu **O(n2)O(n^2)O(n2)** karena harus membandingkan setiap pasangan titik, sehingga tidak efisien untuk jumlah titik yang besar.
2. **Pendekatan Divide and Conquer**
   Untuk meningkatkan efisiensi, digunakan pendekatan **Divide and Conquer** yang memiliki kompleksitas waktu **O(nlogn)O(n \log n)O(nlogn)**. Algoritma ini bekerja dengan langkah-langkah berikut:
   - **Pengurutan Awal**: Titik-titik dalam himpunan diurutkan berdasarkan koordinat **x**.
   - **Pembagian (Divide)**: Himpunan titik dibagi menjadi dua bagian yang hampir sama besar berdasarkan koordinat **x**.
   - **Penyelesaian Rekursif (Conquer)**: Algoritma secara rekursif mencari pasangan titik terdekat di kedua bagian.
   - **Penggabungan (Combine)**: Hasil dari kedua bagian dibandingkan untuk menentukan jarak terpendek sementara. Selain itu, dibuat sebuah **strip** yang berisi titik-titik di sekitar garis tengah dengan jarak lebih kecil dari jarak minimum yang ditemukan. Strip ini kemudian diurutkan berdasarkan koordinat **y**, lalu diperiksa apakah ada pasangan titik dengan jarak lebih kecil.

Pemeriksaan dalam **strip** dilakukan dengan membandingkan setiap titik dengan **maksimal 7 titik berikutnya**. Hal ini berdasarkan fakta matematis bahwa dalam area kecil dengan lebar **2d**, tidak akan ada lebih dari 7 titik yang lebih dekat dari jarak minimum yang telah ditemukan sebelumnya.

Keunggulan Algoritma Divide and Conquer

- Memiliki kompleksitas waktu **O(nlogn)O(n \log n)O(nlogn)** yang jauh lebih cepat dibandingkan metode **Brute Force**.
- Lebih efisien dalam menangani jumlah titik yang besar.
- Mampu menangani berbagai aplikasi dunia nyata, seperti dalam pemetaan geografis, analisis data spasial, dan pencarian tetangga terdekat dalam kecerdasan buatan.

Dengan strategi **Divide and Conquer**, algoritma **Closest-Pair Problem** dapat menyelesaikan pencarian pasangan titik terdekat dengan efisien dan optimal, sehingga menjadi pilihan utama dalam permasalahan komputasi geometris.

PSEUDOCODE ALGORITMA

```
Algoritma PasanganTerdekat(P)
Masukan: P[] = Array berisi n titik dengan koordinat (x, y)
Keluaran: Sepasang titik terdekat beserta jaraknya

1. Urutkan P berdasarkan koordinat x
2. Kembalikan hasil dari PasanganTerdekatRekursif(P, 0, n-1)

Fungsi PasanganTerdekatRekursif(P, kiri, kanan)
3. Jika kanan - kiri ≤ 3 maka
4.     Kembalikan BruteForceTerdekat(P, kiri, kanan)  // Gunakan
brute force jika titik sedikit
5. Selain itu
6.     tengah = (kiri + kanan) / 2
7.     TitikTengah = P[tengah]
8.
9.     (p1, q1, d1) = PasanganTerdekatRekursif(P, kiri, tengah)
// Cari di bagian kiri
10.    (p2, q2, d2) = PasanganTerdekatRekursif(P, tengah+1, kanan)
// Cari di bagian kanan
11.
12.    d = min(d1, d2)
13.    Jika d1 < d2 maka
14.        PasanganTerbaik = (p1, q1)
15.    Selain itu
16.        PasanganTerbaik = (p2, q2)
17.
18.    Strip = Semua titik dalam P yang memiliki |x -
TitikTengah.x| < d
19.    Urutkan Strip berdasarkan koordinat y
20.
21.    Untuk i = 0 hingga panjang(Strip) - 1 lakukan
22.        Untuk j = i+1 hingga min(i+7, panjang(Strip)) lakukan
// Cek hanya hingga 7 titik
23.            Jika Jarak(Strip[i], Strip[j]) < d maka
24.                d = Jarak(Strip[i], Strip[j])
25.                PasanganTerbaik = (Strip[i], Strip[j])
26.
27.    Kembalikan (PasanganTerbaik, d)

Fungsi BruteForceTerdekat(P, kiri, kanan)
28. MinJarak = ∞
29. Untuk i = kiri hingga kanan lakukan
30.     Untuk j = i+1 hingga kanan lakukan
31.         Jika Jarak(P[i], P[j]) < MinJarak maka
32.             MinJarak = Jarak(P[i], P[j])
33.             PasanganTerbaik = (P[i], P[j])
34. Kembalikan (PasanganTerbaik, MinJarak)

Fungsi Jarak(p1, p2)
35. Kembalikan akar((p1.x - p2.x)² + (p1.y - p2.y)²)
```

SOURCE CODE

```python
import math

def jarak(t1, t2):
    return math.sqrt((t1[0] - t2[0]) ** 2 + (t1[1] - t2[1]) ** 2)

def brute_force_terdekat(titik):
    min_jarak = float('inf')
    pasangan_terdekat = None
    n = len(titik)
    for i in range(n):
        for j in range(i + 1, n):
            d = jarak(titik[i], titik[j])
            if d < min_jarak:
                min_jarak = d
                pasangan_terdekat = (titik[i], titik[j])
    return pasangan_terdekat, min_jarak

def strip_terdekat(strip, d):
    min_jarak = d
    pasangan_terdekat = None
    strip.sort(key=lambda titik: titik[1])  # Urutkan berdasarkan koordinat y

    for i in range(len(strip)):
        for j in range(i + 1, min(i + 7, len(strip))):
            d = jarak(strip[i], strip[j])
            if d < min_jarak:
                min_jarak = d
                pasangan_terdekat = (strip[i], strip[j])
    return pasangan_terdekat, min_jarak

def pasangan_terdekat_rekursif(titik):
    n = len(titik)
    if n <= 3:
        return brute_force_terdekat(titik)

    tengah = n // 2
    titik_tengah = titik[tengah]

    kiri_pasangan, kiri_jarak = pasangan_terdekat_rekursif(titik[:tengah])
    kanan_pasangan, kanan_jarak = pasangan_terdekat_rekursif(titik[tengah:])

    if kiri_jarak < kanan_jarak:
        pasangan_min, jarak_min = kiri_pasangan, kiri_jarak
    else:
        pasangan_min, jarak_min = kanan_pasangan, kanan_jarak

    strip = [t for t in titik if abs(t[0] - titik_tengah[0]) < jarak_min]
    strip_pasangan, strip_jarak = strip_terdekat(strip, jarak_min)

    if strip_pasangan and strip_jarak < jarak_min:
        return strip_pasangan, strip_jarak
    else:
        return pasangan_min, jarak_min

def pasangan_terdekat(titik):
    titik.sort()  # Urutkan berdasarkan koordinat x
    return pasangan_terdekat_rekursif(titik)

# Contoh penggunaan
data_titik = [(2, 3), (12, 30), (40, 50), (5, 1), (12, 10), (3, 4)]
pasangan, jarak_min = pasangan_terdekat(data_titik)
print("Pasangan titik terdekat:", pasangan)
print("Jarak minimum:", jarak_min)
```

```
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python312/python.exe
itma/Algoritma The Closest-Pair Problem.py"
Pasangan titik terdekat: ((2, 3), (3, 4))
Jarak minimum: 1.4142135623730951
```

ANALISIS KEBUTUHAN WAKTU

1) Analisis Operasi/Instruksi yang Dieksekusi

Pendekatan ini menganalisis jumlah eksekusi dari berbagai operasi utama dalam kode, seperti operator penugasan (=, +=, *=) dan operator aritmatika (+, -, *, /).

- **Fungsi jarak(t1, t2)**:
  - Menghitung jarak antara dua titik menggunakan **3 operasi aritmatika** (-, *, +) dan **1 operasi akar kuadrat** (math.sqrt()).
  - Kompleksitas waktu: **O(1)**
- **Fungsi brute_force_terdekat(titik)**:
  - Dua loop bersarang (for i in range(n), for j in range(i+1, n)) menghasilkan **O(n^2)** perbandingan.
  - Setiap iterasi melakukan:
    - 1 kali pemanggilan jarak(), yaitu **4 operasi utama**.
    - 1 kali perbandingan (if d < min_jarak).
    - 2 kali assignment jika kondisi terpenuhi.
  - Kompleksitas waktu: **O(n^2)**
- **Fungsi strip_terdekat(strip, d)**:
  - Mengurutkan strip berdasarkan koordinat y → **O(n log n)**
  - Dua loop bersarang untuk mencari pasangan titik dalam strip, tetapi terbatas hingga 7 titik terdekat.
  - Kompleksitas waktu: **O(n log n)** dalam pengurutan, tetapi perbandingan tetap **O(n)** dalam skenario umum.
- **Fungsi pasangan_terdekat_rekursif(titik)**:
  - **Divide**: Membagi titik menjadi dua → **O(1)**
  - **Conquer**: Memanggil rekursi dua kali pada bagian kiri dan kanan → **2T(n/2)**
  - **Combine**: Memeriksa strip → **O(n)**
  - Kompleksitas waktu rekursif: **T(n) = 2T(n/2) + O(n)**, yang menghasilkan **O(n log n)**.

2) Analisis Berdasarkan Jumlah Operasi Abstrak

Pendekatan ini melihat operasi utama seperti perbandingan, perkalian, atau pemanggilan fungsi.

Fungsi	Operasi Utama	Kompleksitas Waktu
`jarak()`	3 aritmatika, 1 sqrt	O(1)
`brute_force_terdekat()`	n(n-1)/2 perbandingan dan operasi	O(n^2)
`strip_terdekat()`	Sort O(n log n), perbandingan O(n)	O(n log n)
`pasangan_terdekat_rekursif()`	Divide O(1), Conquer 2T(n/2), Combine O(n)	O(n log n)
Total (Divide & Conquer)	Rekursi + kombinasi	**O(n log n)**

3) Analisis Best-Case, Worst-Case, dan Average-Case
Pendekatan ini mempertimbangkan skenario terbaik, terburuk, dan rata-rata dari algoritma.

- **Best-Case (Kasus Terbaik)**:
  - Jika titik-titik sudah dalam posisi ideal untuk menemukan pasangan terdekat dengan cepat tanpa perlu banyak perhitungan.
  - Terjadi ketika pasangan terdekat ditemukan di tahap awal rekursi.
  - Kompleksitas waktu: **O(n log n)**.
- **Worst-Case (Kasus Terburuk)**:
  - Jika titik-titik tersebar merata sehingga algoritma harus menjalankan semua langkah rekursi dan memeriksa banyak kemungkinan pasangan.
  - Kompleksitas waktu: **O(n log n)** karena setiap rekursi tetap memeriksa **strip** yang mengandung titik dalam jumlah besar.
- **Average-Case (Kasus Rata-Rata)**:
  - Secara umum, algoritma tetap bekerja dengan **O(n log n)** karena pembagian titik dan pemrosesan dalam strip tetap membutuhkan pengurutan dan pencarian optimal.

REFERENSI

Bentley, J. L. (1979). "Data Structures for Range Searching". *ACM Computing Surveys*, 397-409.

Cormen, T. H. (2009). *Introduction to Algorithms (3rd ed.).* MIT Press.

Levitin, A. (2012). *Introduction to the Design and Analysis of Algorithms (3rd Edition).* Pearson.

Preparata, F. P. (1985). *Computational Geometry: An Introduction.* Springer-Verlag.

LAMPIRAN LINK GITHUB

https://github.com/fadhliakbar333/Perancangan-dan-Analisis-Algoritma