TUGAS EKSPLORASI MANDIRI

PERANCANGAN DAN ANALISIS ALGORITMA

ALGORITMA THE CLOSEST- PAIR PROBLEMS

DOSEN PENGAMPU:

Randi Proska Sandra, S.Pd, M.Sc

OLEH:

Aisyah Farhanah

23343003

Informatika(NK)

PROGRAM STUDI INFORMATIKA

DEPARTEMEN ELEKTRONIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

2025

A. RANGKUMAN MATERI

Masalah pasangan terdekat adalah tantangan komputasional yang meminta untuk menemukan dua titik dengan jarak terdekat dalam himpunan titik berdimensi n. Secara matematis, diberikan himpunan P = {p1, p2, ..., pn} dengan pi = (xi, yi), tujuannya adalah menemukan sepasang titik (pi, pj) yang meminimalkan jarak Euclidean d(pi, pj).

Algoritma divide and conquer untuk masalah pasangan terdekat merupakan pendekatan canggih yang secara signifikan meningkatkan efisiensi komputasional dibandingkan metode brute force. Proses algoritma dimulai dengan membagi kumpulan titik menjadi dua himpunan bagian yang sama besar menggunakan garis vertikal melalui median koordinat x. Dengan cara ini, setengah titik ditempatkan di sebelah kiri garis dan setengah lainnya di sebelah kanan, menciptakan dua subset yang dapat diselesaikan secara rekursif.

Pada setiap tahap rekursi, algoritma menghitung jarak terdekat untuk masing-masing subset (dl untuk subset kiri dan dr untuk subset kanan), kemudian mengambil jarak minimum di antara keduanya. Namun, algoritma tidak dapat langsung menyimpulkan bahwa jarak minimum ini adalah jarak terdekat keseluruhan, karena dimungkinkan terdapat pasangan titik yang lebih dekat dan terletak di sisi yang berbeda dari garis pembagi.

Untuk mengatasi hal ini, algoritma menggunakan teknik cerdik dengan membuat strip vertikal simetris selebar 2d di sekitar garis pemisah, di mana d adalah jarak minimum yang ditemukan sebelumnya. Titik-titik dalam strip ini diurutkan berdasarkan koordinat y, dan algoritma secara sistematis memeriksa titik-titik potensial yang mungkin membentuk pasangan terdekat. Kunci utama dalam tahap ini adalah pembatasan jumlah titik yang perlu diperiksa, dengan bukti matematis menunjukkan bahwa tidak lebih dari enam titik yang perlu dibandingkan.

Kompleksitas waktu algoritma ini sangat mengesankan, yaitu O(n log n), yang jauh lebih efisien dibandingkan pendekatan brute force $O(n^2)$. Analisis matematis menggunakan Teorema Master membuktikan kompleksitas waktu ini, dan telah dibuktikan bahwa ini adalah kelas efisiensi terbaik yang mungkin dicapai untuk masalah pencarian pasangan terdekat. Kebutuhan untuk mensortir titik input terlebih dahulu tidak mempengaruhi kelas efisiensi keseluruhan, terutama jika digunakan algoritma sorting efisien seperti merge sort.

Implementasi algoritma ini memerlukan dua array input: satu diurutkan berdasarkan koordinat x dan satu lagi diurutkan berdasarkan koordinat y. Proses rekursif dimulai dengan kasus basis untuk kumpulan titik kecil ($n \leq 3$), di mana algoritma brute force digunakan. Untuk kumpulan titik yang lebih besar, algoritma membagi, memecahkan secara rekursif, dan kemudian menggabungkan solusi dengan pemeriksaan strip vertikal yang cerdik. Pendekatan ini tidak hanya efisien secara komputasional, tetapi juga memberikan wawasan mendalam tentang struktur geometris dari kumpulan titik.

B. PSEUDOCODE

```
Fungsi JarakEuclidean(titik1, titik2):
    Kembalikan akar kuadrat dari ((titik1.x - titik2.x)^2 + (titik1.y - titik2.y)^2)

Fungsi PasanganTerdekatBruteForce(titik):
    jarak_minimum ← ∞
    jumlah_titik ← panjang(titik)

    Untuk setiap i dari 0 hingga jumlah_titik - 1:
        Untuk setiap j dari i + 1 hingga jumlah_titik - 1:
            jarak ← JarakEuclidean(titik[i], titik[j])
            jarak_minimum ← minimum(jarak_minimum, jarak)

    Kembalikan jarak_minimum

Fungsi PasanganTerdekatRekursif(titik_urut_x, titik_urut_y):
    jumlah_titik ← panjang(titik_urut_x)

    Jika jumlah_titik ≤ 3:
        Kembalikan PasanganTerdekatBruteForce(titik_urut_x)

    tengah ← jumlah_titik / 2
    kiri_x ← titik_urut_x[0 : tengah]
    kanan_x ← titik_urut_x[tengah : jumlah_titik]

    titik_tengah ← titik_urut_x[tengah].x

    kiri_y ← titik dari titik_urut_y yang titik.x ≤ titik_tengah
    kanan_y ← titik dari titik_urut_y yang titik.x > titik_tengah

    jarak_kiri ← PasanganTerdekatRekursif(kiri_x, kiri_y)
    jarak_kanan ← PasanganTerdekatRekursif(kanan_x, kanan_y)
    jarak_minimum ← minimum(jarak_kiri, jarak_kanan)

    strip ← titik dari titik_urut_y yang |titik.x - titik_tengah| < jarak_minimum

    jarak_strip_minimum ← jarak_minimum
    Untuk setiap i dari 0 hingga panjang(strip) - 1:
        Untuk setiap j dari i + 1 hingga minimum(i + 7, panjang(strip)):
            jarak ← JarakEuclidean(strip[i], strip[j])
            jarak_strip_minimum ← minimum(jarak_strip_minimum, jarak)
```

Kembalikan jarak_strip_minimum

Fungsi PasanganTerdekat(titik):
titik_urut_x ← titik diurutkan berdasarkan koordinat x
titik_urut_y ← titik diurutkan berdasarkan koordinat y
Kembalikan PasanganTerdekatRekursif(titik_urut_x, titik_urut_y)

PROGRAM UTAMA:
titik ← [(2, 3), (12, 30), (40, 50), (5, 1), (12, 10), (3, 4)]
Cetak "Jarak pasangan terdekat:", PasanganTerdekat(titik)

C. SOURCE CODE

```
# Nama  : Aisyah Farhanah
# NIM   : 23343003
# Algoritma Closest-pair Problems

 import math

 # Fungsi untuk menghitung jarak Euclidean antara dua titik
 def jarak_euclidean(titik1, titik2):
     return math.sqrt((titik1[0] - titik2[0])**2 + (titik1[1] -
      titik2[1])**2)

 # Algoritma Brute Force
 def pasangan_terdekat_brute_force(titik):
     jarak_minimum = float("inf")
     jumlah_titik = len(titik)

     for i in range(jumlah_titik):
         for j in range(i + 1, jumlah_titik):
             jarak = jarak_euclidean(titik[i], titik[j])
             jarak_minimum = min(jarak_minimum, jarak)

     return jarak_minimum

 # Fungsi utama Divide and Conquer
 def pasangan_terdekat_rekursif(titik_urut_x, titik_urut_y):
     jumlah_titik = len(titik_urut_x)

     # Jika jumlah titik kecil, gunakan brute-force
     if jumlah_titik <= 3:
         return pasangan_terdekat_brute_force(titik_urut_x)

     # Bagi titik menjadi dua bagian
     tengah = jumlah_titik // 2
     kiri_x = titik_urut_x[:tengah]
     kanan_x = titik_urut_x[tengah:]
```

```
    titik_tengah = titik_urut_x[tengah][0]

    kiri_y = list(filter(lambda t: t[0] <= titik_tengah,
     titik_urut_y))
    kanan_y = list(filter(lambda t: t[0] > titik_tengah,
     titik_urut_y))

    # Rekursif ke dua bagian
    jarak_kiri = pasangan_terdekat_rekursif(kiri_x, kiri_y)
    jarak_kanan = pasangan_terdekat_rekursif(kanan_x, kanan_y)
    jarak_minimum = min(jarak_kiri, jarak_kanan)

    # Cari pasangan titik yang melewati garis tengah dalam strip
     selebar 2 * jarak_minimum
    strip = [t for t in titik_urut_y if abs(t[0] - titik_tengah) <
     jarak_minimum]

    # Periksa titik dalam strip
    jarak_strip_minimum = jarak_minimum
    for i in range(len(strip)):
        for j in range(i + 1, min(i + 7, len(strip))):  # Hanya
           periksa hingga 6 titik setelahnya
            jarak = jarak_euclidean(strip[i], strip[j])
            jarak_strip_minimum = min(jarak_strip_minimum, jarak)

    return jarak_strip_minimum

# Fungsi utama untuk menjalankan algoritma
def pasangan_terdekat(titik):
    titik_urut_x = sorted(titik, key=lambda t: t[0])  # Urutkan
     berdasarkan x
    titik_urut_y = sorted(titik, key=lambda t: t[1])  # Urutkan
     berdasarkan y
    return pasangan_terdekat_rekursif(titik_urut_x, titik_urut_y)
titik = [(2, 3), (12, 30), (40, 50), (5, 1), (12, 10), (3, 4)]
print("Jarak pasangan terdekat:", pasangan_terdekat(titik))
```

D. ANALISIS KEBUTUHAN WAKTU

1) Analisis Berdasarkan Operasi/Instruksi

Algoritma ini memiliki beberapa operasi penting, yaitu:

a) Operator Penugasan (=, +=, *=, dll.)

- Variabel untuk menyimpan nilai minimum (jarak_minimum = float("inf"))
- Loop untuk perhitungan jarak Euclidean (for i in range(...))
- Rekursi untuk membagi titik menjadi dua bagian (jarak_kiri = pasangan_terdekat_rekursif(...))
- Memfilter titik dalam strip tengah (strip = [t for t in titik_urut_y if ...])

b) Operator Aritmatika (+, -, *, /, **)

Perhitungan Jarak Euclidean

$d = \sqrt{(x2-x1)2+(y2-y1)2}$

- Menggunakan operasi pengurangan (-), pangkat (**2), dan akar kuadrat (math.sqrt()).
- Pembagian untuk mencari titik tengah (tengah = jumlah_titik // 2)

2) Analisis Berdasarkan Jumlah Operasi Abstrak

Misalkan n adalah jumlah titik.

- Sorting awal (O(n log n))
  - Titik diurutkan berdasarkan x dan y menggunakan Timsort dengan kompleksitas O(n log n).
- Divide and Conquer (O(n log n))
  - Membagi titik menjadi dua bagian dalam setiap rekursi → O(log n) tahap.
  - Untuk setiap tahap, pencarian dalam strip tengah dilakukan dalam O(n).
- Brute Force ($O(n^2)$, hanya untuk $n \leq 3$)
  - Ketika rekursi mencapai basis kasus (≤ 3 titik), perhitungan dilakukan dengan nested loop O(1) karena jumlah titik kecil.

3) Analisis Berdasarkan Best-Case, Worst-Case, dan Average-Case

- Best-Case (Kasus Terbaik)

  Kasus terbaik terjadi ketika titik-titik dalam bidang dua dimensi sudah dikelompokkan secara optimal, sehingga algoritma tidak perlu melakukan banyak perhitungan tambahan. Misalnya, jika pasangan titik terdekat ditemukan dalam tahap awal rekursi dan tidak ada pasangan lebih dekat di strip tengah, maka algoritma dapat langsung menghentikan pencarian lebih lanjut. Dalam kondisi ini, sorting awal tetap berjalan dalam O(n log n), dan rekursi membagi titik menjadi dua bagian tanpa perlu banyak pemeriksaan dalam strip. Meskipun ada beberapa pengurangan jumlah operasi dibandingkan dengan kasus rata-rata, kompleksitas waktu keseluruhan tetap O(n log n) karena struktur Divide and Conquer tetap berjalan penuh.

- Worst-Case (Kasus Terburuk)

  Kasus terburuk terjadi ketika setiap rekursi harus memeriksa hampir semua kemungkinan pasangan titik dalam strip tengah. Hal ini dapat terjadi jika titik-titik tersebar dalam pola yang memaksa algoritma untuk membandingkan banyak pasangan titik dalam strip selebar 2d yang berada di sekitar garis pembagi. Sorting awal tetap berjalan dalam O(n log n), dan rekursi tetap membagi titik menjadi dua bagian dengan kompleksitas O(n log n). Namun, pada setiap tahap, algoritma harus mengevaluasi lebih banyak pasangan titik dalam strip, meningkatkan jumlah operasi yang dilakukan. Meskipun jumlah perhitungan meningkat, kompleksitas asimtotik tetap O(n log n) karena masih mengikuti pola pembagian yang efisien.

- Average-Case (Kasus Rata-Rata)

  Kasus rata-rata terjadi ketika titik-titik tersebar secara acak dalam bidang 2D, tanpa pola khusus yang menyebabkan peningkatan jumlah perhitungan secara signifikan. Algoritma tetap bekerja dengan membagi titik menjadi dua bagian, menyelesaikan submasalah secara rekursif, dan memeriksa strip tengah untuk mencari pasangan titik terdekat yang mungkin melewati garis pembagi. Sorting awal tetap memerlukan O(n log n), dan rekursi berjalan dalam O(n log n) karena pembagian titik terjadi hingga mencapai basis kasus ($\leq$ 3 titik). Dalam pemeriksaan strip tengah, algoritma hanya perlu mengevaluasi hingga 6 titik setelahnya, sehingga jumlah operasi tetap terbatas dalam O(n). Dengan demikian, kompleksitas total tetap O(n log n), yang merupakan kasus umum dalam penggunaan algoritma ini.

E. REFERENSI

Levitin, A. (2012). Introduction to the design & analysis of algorithms (3rd ed.). Pearson Education, Inc

F. LINK GITHUB

https://github.com/ffrhnh/The-Closest-pair-Problems.git