

# **LAPORAN PRAKTIKUM 5**

## **Analysis Algorithm**

**Problem-Problem dengan Pemecahan Masalah Menggunakan Paradigma Divide & Conquer**



**Disusun oleh :**

**Mohammad Dhikri**  
**140810180075**

**PROGRAM STUDI S1 TEKNIK INFORMATIKA**  
**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**  
**UNIVERSITAS PADJADJARAN**  
**2020**

## STUDI KASUS (LANJUTAN)

### Studi Kasus 5: Mencari Pasangan Titik Terdekat (Closest Pair of Points)

#### Identifikasi Problem:

Diberikan array  $n$  poin dalam bidang kartesius, dan problemnya adalah mencari tahu pasangan poin terdekat dalam bidang tersebut dengan merepresentasikannya ke dalam array. Masalah ini muncul di sejumlah aplikasi. Misalnya, dalam kontrol lalu lintas udara, kita mungkin ingin memantau pesawat yang terlalu berdekatan karena ini mungkin menunjukkan kemungkinan tabrakan. Ingat rumus berikut untuk jarak antara dua titik  $p$  dan  $q$ .

$$\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Solusi Solusi umum dari permasalahan tersebut adalah menggunakan algoritma Brute force dengan  $O(n^2)$ , hitung jarak antara setiap pasangan dan kembalikan yang terkecil. Namun, kita dapat menghitung jarak terkecil dalam waktu  $O(n \log n)$  menggunakan strategi Divide and Conquer. Ikuti algoritma berikut:

#### Tugas:

- 1) Buatlah program untuk menyelesaikan problem closest pair of points menggunakan algoritma divide & conquer yang diberikan. Gunakan bahasa C++

Jawab :

```
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <math>
using namespace std;
struct point{
    int x, y;
};
int compareX(const void* a, const void* b){
    point *p1 = (point *)a, *p2 = (point *)b;
    return (p1->x - p2->x);
}
int compareY(const void* a, const void* b){
    point *p1 = (point *)a, *p2 = (point *)b;
    return (p1->y - p2->y);
}
float dist(point p1, point p2){
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y))
;
}
float small_dist(point P[], int n){
    float min = FLT_MAX;
    for (int i = 0; i < n; ++i){
        for (int j = i + 1; j < n; ++j){
```

```

        if (dist(P[i], P[j]) < min)
            min = dist(P[i], P[j]);
    }
}
return min;
}

float stripClosest(point strip[], int size, float d){
    float min = d;
    for (int i = 0; i < size; ++i){
        for (int j = i + 1; j < size && (strip[j].y - strip[i].y) < min; ++j){
            if (dist(strip[i], strip[j]) < min)
                min = dist(strip[i], strip[j]);
        }
    }
    return min;
}

float closestUtil(point Px[], point Py[], int n){
    if (n <= 3)
        return small_dist(Px, n);
    int mid = n / 2;
    point midPoint = Px[mid];
    point Pyl[mid + 1];
    point Pyr[n - mid - 1];
    int li = 0, ri = 0;
    for (int i = 0; i < n; i++){
        if (Py[i].x <= midPoint.x)
            Pyl[li++] = Py[i];
        else
            Pyr[ri++] = Py[i];
    }
    float dl = closestUtil(Px, Pyl, mid);
    float dr = closestUtil(Px + mid, Pyr, n - mid);
    float d = min(dl, dr);
    point strip[n];
    int j = 0;
    for (int i = 0; i < n; i++){
        if (abs(Py[i].x - midPoint.x) < d)
            strip[j] = Py[i], j++;
    }
    return min(d, stripClosest(strip, j, d));
}

float closest(point P[], int n){
    point Px[n];
    point Py[n];
    for (int i = 0; i < n; i++){
        Px[i] = P[i];
        Py[i] = P[i];
    }
}

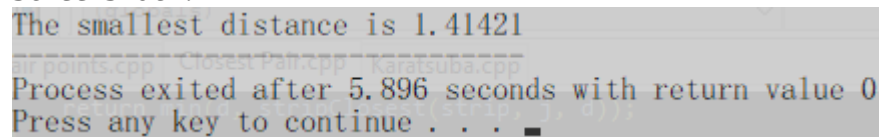
```

```

    qsort(Px, n, sizeof(point), compareX);
    qsort(Py, n, sizeof(point), compareY);
    return closestUtil(Px, Py, n);
}
int main(){
    point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};
    int n = sizeof(P) / sizeof(P[0]);
    cout << "Jarak point terdekat adalah : " << closest(P, n);
    return 0;
}

```

ScreenShot :



```

The smallest distance is 1.41421
Process exited after 5.896 seconds with return value 0
Press any key to continue . . .

```

- 2) Tentukan rekurensi dari algoritma tersebut, dan selesaikan rekurensinya menggunakan **metode recursion tree** untuk membuktikan bahwa algoritma tersebut memiliki Big-O ( $n \lg n$ )

Jawab :

### Kompleksitas Waktu

Biarkan kompleksitas waktu dari algoritma di atas menjadi  $T(n)$ . Mari kita asumsikan bahwa kita menggunakan algoritma pengurutan  $O(n \lg n)$ . Algoritma di atas membagi semua titik dalam dua set dan secara rekursif memanggil dua set. Setelah membelah, ia menemukan strip dalam waktu  $O(n)$ , mengurutkan strip dalam waktu  $O(n \lg n)$  dan akhirnya menemukan titik terdekat dalam strip dalam waktu  $O(n)$ . Jadi  $T(n)$  dapat dinyatakan sebagai berikut

$$T(n) = 2T(n/2) + O(n) + O(n \lg n) + O(n)$$

$$T(n) = 2T(n/2) + O(n \lg n)$$

$$T(n) = T(n \times \lg n \times \lg n)$$

Keterangan :

- Kompleksitas waktu dapat ditingkatkan menjadi  $O(n \lg n)$  dengan mengoptimalkan langkah 5 dari algoritma di atas.
- Kode menemukan jarak terkecil. Dapat dengan mudah dimodifikasi untuk menemukan titik dengan jarak terkecil.
- Kode ini menggunakan pengurutan cepat yang bisa  $O(n^2)$  dalam kasus terburuk. Untuk memiliki batas atas sebagai  $O(n (\lg n)^2)$ , algoritma pengurutan  $O(n \lg n)$  seperti pengurutan gabungan atau pengurutan tumpukan dapat digunakan

## Studi Kasus 6: Algoritma Karatsuba untuk Perkalian Cepat Identifikasi Problem:

Diberikan dua string biner yang mewakili nilai dua bilangan bulat, cari produk (hasil kali) dari dua string. Misalnya, jika string bit pertama adalah "1100" dan string bit kedua adalah "1010", output harus 120. Supaya lebih sederhana, panjang dua string sama dan menjadi  $n$ .

Solusi: Salah satu solusinya adalah dengan naïve approach yang pernah kita pelajari di sekolah. Satu per satu ambil semua bit nomor kedua dan kalikan dengan semua bit nomor pertama. Akhirnya tambahkan semua perkalian. Algoritma ini membutuhkan waktu  $O(n^2)$ .

Algoritma Karatsuba Solusi lain adalah dengan menggunakan Algoritma Karatsuba yang berparadigma Divide dan Conquer, kita dapat melipatgandakan dua bilangan bulat dalam kompleksitas waktu yang lebih sedikit. Kami membagi angka yang diberikan dalam dua bagian. Biarkan angka yang diberikan menjadi X dan Y.

- Untuk kesederhanaan, kita asumsikan bahwa n adalah genap:

$$\begin{aligned} X &= X_l \cdot 2^{n/2} + X_r & [X_l \text{ and } X_r \text{ contain leftmost and rightmost } n/2 \text{ bits of } X] \\ Y &= Y_l \cdot 2^{n/2} + Y_r & [Y_l \text{ and } Y_r \text{ contain leftmost and rightmost } n/2 \text{ bits of } Y] \end{aligned}$$

- Produk XY dapat ditulis sebagai berikut:

$$\begin{aligned} XY &= (X_l \cdot 2^{n/2} + X_r) (Y_l \cdot 2^{n/2} + Y_r) \\ &= 2^n X_l Y_l + 2^{n/2} (X_l Y_r + X_r Y_l) + X_r Y_r \end{aligned}$$

- Jika kita melihat rumus di atas, ada empat perkalian ukuran  $n/2$ , jadi pada dasarnya kita membagi masalah ukuran n menjadi empat sub-masalah ukuran  $n/2$ . Tetapi itu tidak membantu karena solusi pengulangan  $T(n) = 4T(n/2) + O(n)$  adalah  $O(n^2)$ . Bagian rumit dari algoritma ini adalah mengubah dua istilah tengah ke bentuk lain sehingga hanya satu perkalian tambahan yang cukup. Berikut ini adalah tricky expression untuk dua middle terms tersebut.

$$X_l Y_r + X_r Y_l = (X_l + X_r) (Y_l + Y_r) - X_l Y_l - X_r Y_r$$

- Jadi nilai akhir XY menjadi:

$$XY = 2^n X_l Y_l + 2^{n/2} * [(X_l + X_r) (Y_l + Y_r) - X_l Y_l - X_r Y_r] + X_r Y_r$$

- Dengan trik di atas, perulangan menjadi  $T(n) = 3T(n/2) + O(n)$  dan solusi dari perulangan ini adalah  $O(n^{1.59})$

Bagaimana jika panjang string input berbeda dan tidak genap? Untuk menangani kasus panjang yang berbeda, kita menambahkan 0 di awal. Untuk menangani panjang ganjil, kita menempatkan bit floor ( $n/2$ ) di setengah kiri dan ceil ( $n/2$ ) bit di setengah kanan. Jadi ekspresi untuk XY berubah menjadi berikut.

$$XY = 2^{2\text{ceil}(n/2)} X_l Y_l + 2^{\text{ceil}(n/2)} * [(X_l + X_r) (Y_l + Y_r) - X_l Y_l - X_r Y_r] + X_r Y_r$$

### Tugas:

- 1) Buatlah program untuk menyelesaikan problem fast multiplication menggunakan algoritma divide & conquer yang diberikan (Algoritma Karatsuba). Gunakan bahasa C++

Jawab :

```
/*
Nama : Mohammad Dhikri
NPM : 140810180075
Kelas : A
```

Tanggal : 30 - Maret -2020

```
*/
#include <iostream>
#include <math.h>
using namespace std;

int getLength(long value) {
    int counter = 0;
    while (value != 0) {
        counter++;
        value /= 10;
    }
    return counter;
}

long karatsuba(long x, long y) {
    int xLength = getLength(x);
    int yLength = getLength(y);
    int N = (int)(fmax(xLength, yLength));
    if (N < 10){
        return x * y;
    }
    N = (N/2) + (N%2);
    long perkalian = 10*N;

    long b = x/perkalian;
    long a = x - (b * perkalian);
    long d = y / perkalian;
    long c = y - (d * N);

    long z0 = karatsuba(a,c);
    long z1 = karatsuba(a + b, c + d);
    long z2 = karatsuba(b, d);
    return z0 + ((z1 - z0 - z2) * perkalian) + (z2 * (long)(10*(2 * N)));
}

int main() {
    long a,b;
    cout<<"Angka Pertama : "; cin>>a;
    cout<<"Angka Kedua : "; cin>>b;
    cout << karatsuba(a,b) << endl;
    return 0;
}
```

Screenshot :

```
Angka Pertama : 240
Angka Kedua : 23
5520
```

- 2) Rekurensi dari algoritma tersebut adalah  $T(n) = 3T(n/2) + O(n)$ , dan selesaikan rekurensinya menggunakan **metode substitusi** untuk membuktikan bahwa algoritma tersebut memiliki Big-O ( $n \lg n$ )

Jawab :

Batas untuk rekurensi :  $T(n) = 3T(n/2) + O(n)$

Untuk  $n > 2$  dan  $T(3) \leq c$

- ➔ Solusi tebakannya adalah  $T(n) = O(n \lg n)$
- ➔ Buktikan bahwa  $T(n) \leq cn \lg n$  untuk  $n \geq 2$ , dengan a pilihan konstanta positif yang sesuai c
- ➔ Perulangan berlaku untuk  $n=2$ , karena dalam kasus ini  $cn \lg n = 2c$ , dan  $T(3) \leq c$  sehingga kita memiliki  $T(n) \leq cn \lg n$
- ➔ Sebagai hipotesis induktif, kita mengasumsikan bahwa ikatan itu berlaku untuk  $n/2$ , yaitu  $T([n/2]) \leq c[n/2] \lg [n/2] \rightarrow$  **Substitusikan** ini kedalam rekurensi
  - $T(n) = 3(c[n/2] \lg [n/2]) + cn$   
 $= cn \lg (n/2) + cn$   
 $= cn \lg n - cn \lg 2 + cn$   
 $= cn \lg n$
- ➔ Ini menetapkan batas yang kita inginkan untuk  $T(n)$ , dengan asumsi itu berlaku untuk nilai yang lebih kecil  $m < n$ . Dengan demikian argument induksi dilengkapi.

## Studi Kasus 7: Permasalahan Tata Letak Keramik Lantai (Tiling Problem)

### Identifikasi Problem:

Diberikan papan berukuran  $n \times n$  dimana  $n$  adalah dari bentuk  $2^k$  dimana  $k \geq 1$  (Pada dasarnya  $n$  adalah pangkat dari 2 dengan nilai minimumnya 2). Papan memiliki satu sel yang hilang (ukuran  $1 \times 1$ ). Isi papan menggunakan ubin berbentuk L. Ubin berbentuk L berukuran  $2 \times 2$  persegi dengan satu sel berukuran  $1 \times 1$  hilang.

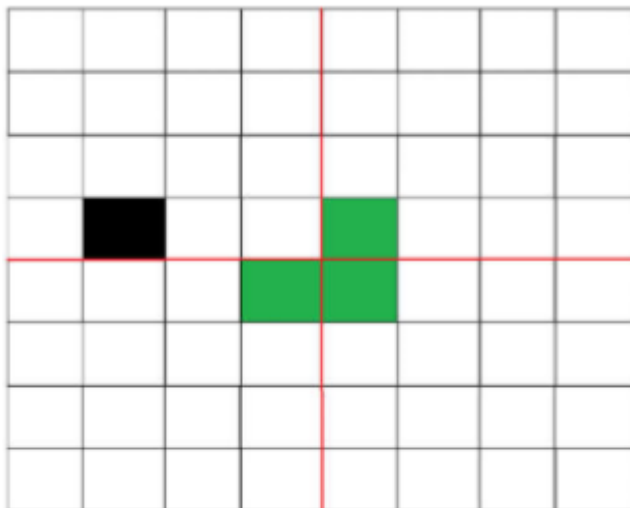


Gambar 2. Ilustrasi tiling problem

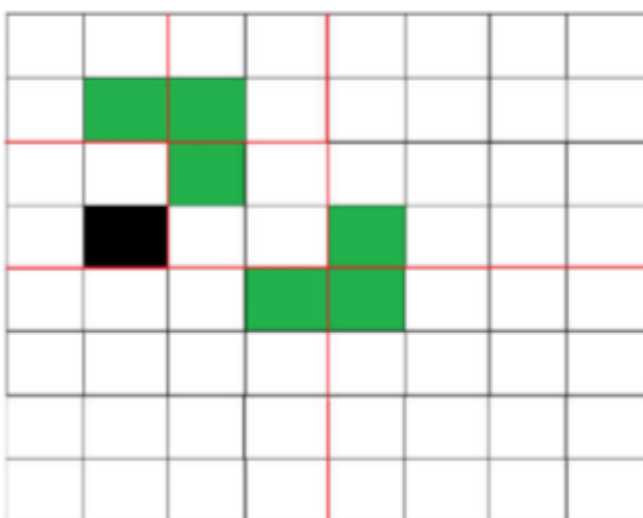
```
// n is size of given square, p is location of missing cell
Tile(int n, Point p)
```

- 1) Base case:  $n = 2$ , A  $2 \times 2$  square with one cell missing is nothing but a tile and can be filled with a single tile.
- 2) Place a L shaped tile at the center such that it does not cover the  $n/2 \times n/2$  subsquare that has a missing square. **Now all four subsquares of size  $n/2 \times n/2$  have a missing cell** (a cell that doesn't need to be filled). See figure 3 below.
- 3) Solve the problem recursively for following four. Let  $p_1$ ,  $p_2$ ,  $p_3$  and  $p_4$  be positions of the 4 missing cells in 4 squares.
  - a) Tile( $n/2$ ,  $p_1$ )
  - b) Tile( $n/2$ ,  $p_2$ )
  - c) Tile( $n/2$ ,  $p_3$ )
  - d) Tile( $n/2$ ,  $p_3$ )

Gambar di bawah ini menunjukkan kerja algoritma di atas.

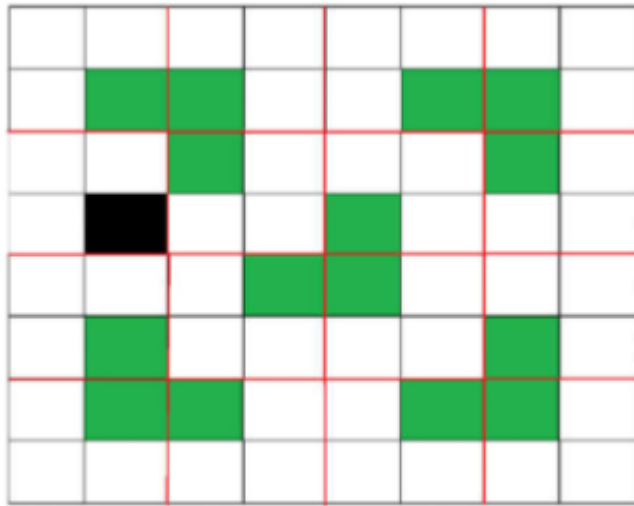


Gambar 3. Ilustrasi setelah tile pertama ditempatkan



Gambar 4 Perulangan untuk subsquare pertama.





Gambar 5. Memperlihatkan langkah pertama di keempat subsquares.

### Tugas:

- 1) Buatlah program untuk menyelesaikan problem tiling menggunakan algoritma divide & conquer yang diberikan. Gunakan bahasa C++

Jawab :

```
#include <bits/stdc++.h>
using namespace std;
int banyakPola(int n, int m)
{
    int jumlah[n + 1];
    jumlah[0] = 0;
    for (int i = 1; i <= n; i++) {
        if (i > m)
            jumlah[i] = jumlah[i - 1] + jumlah[i - m];
        else if (i < m)
            jumlah[i] = 1;
        else
            jumlah[i] = 2;
    }
    return jumlah[n];
}
int main()
{
    int n, m;
    cout<<"Angka Pertama : "; cin>>n;
    cout<<"Angka Kedua : "; cin>>m;
    cout << "Banyak Pola = "
        << banyakPola(n, m);
    return 0;
}
```

Screenshot:

```
Angka Pertama : 7
Angka Kedua : 4
Banyak Pola = 5
```

- 2) Relasi rekurensi untuk algoritma rekursif di atas dapat ditulis seperti di bawah ini. C adalah konstanta.  $T(n) = 4T(n/2) + C$ . Selesaikan rekurensi tersebut dengan **Metode Master**

Jawab :

Kompleksitas Waktu:

Relasi perulangan untuk algoritma rekursif di atas dapat ditulis seperti di bawah ini. C adalah konstanta.

$$T(n) = 4T(n/2) + C$$

$$a = 4, b = 2, f(n) = C$$

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

$$f(n) = n = O(n^{\log_2 4 - \epsilon}) \text{ untuk } \epsilon = 1$$

Kasus 1 terpenuhi :

maka :

$$T(n) = O(n^2)$$