

LAPORAN PRAKTIKUM 5

ANALISIS ALGORITMA

Problem-Problem dengan Pemecahan Masalah Menggunakan Paradigma Divide &
Conquer



Disusun oleh :

Nama : Sharashena Chairani
Kelas : B
NPM : 140810180022

Program Studi S-1 Teknik Informatika
Departemen Ilmu Komputer
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Padjadjaran
2020

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.

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

Tugas:

- 1) Buatlah program untuk menyelesaikan problem closest pair of points menggunakan algoritma divide & conquer yang diberikan. Gunakan bahasa C++
- 2) Tentukan rekurensi dari algoritma tersebut, dan selesaikan rekurensinya menggunakan metode recursion tree untuk membuktikan bahwa algoritma tersebut memiliki Big-O ($n \lg n$)

Jawaban :

1. Program Closest Pair of Points dengan Algoritma Divide & Conquer

/*

Nama : Sharashena Chairani

Kelas : B

NPM : 140810180022

Program : Closest pair of points

*/

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
class Point {
```

```
public:
```

```
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 bruteForce(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 min(float x, float y){
    return (x < y)? x : y;
}

```

```

float stripClosest(Point strip[], int size, float d){
    float min = d; //Inisiasi jarak minimum = d

    qsort(strip, size, sizeof(Point), compareY);

```

```

    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 P[], int n){
    //gunakan brute force, jika n kurang dari atau sama dengan 3
    if (n <= 3)
        return bruteForce(P, n);

```

```

    int mid = n/2;
    Point midPoint = P[mid];

```

```

    float dl = closestUtil(P, mid);
    float dr = closestUtil(P + mid, n - mid);
    float d = min(dl, dr);

```

```

    Point strip[n];

```

```

    int j = 0;
    for (int i = 0; i < n; i++)
        if (abs(P[i].x - midPoint.x) < d)
            strip[j] = P[i], j++;

    return min(d, stripClosest(strip, j, d) );
}

```

```

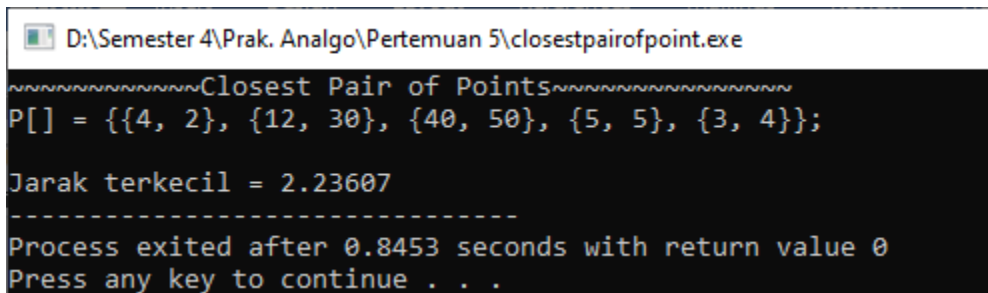
float closest(Point P[], int n){
    qsort(P, n, sizeof(Point), compareX);

    return closestUtil(P, n);
}

int main(){
    cout<<"~~~~~Closest Pair of Points~~~~~" << endl;
    Point P[] = {{4, 2}, {12, 30}, {40, 50}, {5, 5}, {3, 4}};
    int n = sizeof(P) / sizeof(P[0]);

    cout<<"P[] = {{4, 2}, {12, 30}, {40, 50}, {5, 5}, {3, 4}};"<<endl<<endl;
    cout<<"Jarak terkecil = "<<closest(P, n);
}

```



```

D:\Semester 4\Prak. Analgo\Pertemuan 5\closestpairstofpoint.exe
~~~~~Closest Pair of Points~~~~~
P[] = {{4, 2}, {12, 30}, {40, 50}, {5, 5}, {3, 4}};
Jarak terkecil = 2.23607
-----
Process exited after 0.8453 seconds with return value 0
Press any key to continue . . .

```

2. Kompleksitas Waktu

Biarkan kompleksitas waktu dari algoritma di atas menjadi $T(n)$. Mari kita asumsikan bahwa kita menggunakan algoritma pengurutan $O(n \log n)$. 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 $O(n \log n)$ dan akhirnya menemukan titik terdekat dalam strip dalam waktu $O(n)$. Jadi $T(n)$ dapat dinyatakan sebagai berikut

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

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

$$T(n) = T(n * \log n * \log n)$$

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 .

Tugas:

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

Jawaban :

1. Program Algoritma Karatsuba

/*

Nama : Sharashena Chairani

Kelas : B

NPM : 140810180022

Program : Karatsuba Algorithm

*/

```
#include<iostream>
```

```
#include<stdio.h>
```

```
using namespace std;
```

```
int makeEqualLength(string &str1, string &str2){
```

```
    int len1 = str1.size();
```

```
    int len2 = str2.size();
```

```
    if (len1 < len2){
```

```
        for (int i = 0 ; i < len2 - len1 ; i++)
```

```

        str1 = '0' + str1;
        return len2;
    }
    else if (len1 > len2){
        for (int i = 0 ; i < len1 - len2 ; i++)
            str2 = '0' + str2;
    }
    return len1; // If len1 >= len2
}

// The main function that adds two bit sequences and returns the addition
string addBitStrings( string first, string second ){
    string result;

    int length = makeEqualLength(first, second);
    int carry = 0;

    // Add all bits one by one
    for (int i = length-1 ; i >= 0 ; i--){
        int firstBit = first.at(i) - '0';
        int secondBit = second.at(i) - '0';

        int sum = (firstBit ^ secondBit ^ carry)+'0';

        result = (char)sum + result;

        carry = (firstBit&secondBit) | (secondBit&carry) | (firstBit&carry);
    }

    if (carry) result = '1' + result;

```

```

        return result;
    }

int multiplyiSingleBit(string a, string b){
    return (a[0] - '0')*(b[0] - '0');
}

long int multiply(string X, string Y){
    int n = makeEqualLength(X, Y);

    if (n == 0) return 0;
    if (n == 1) return multiplyiSingleBit(X, Y);

    int fh = n/2;
    int sh = (n-fh);

    string Xl = X.substr(0, fh);
    string Xr = X.substr(fh, sh);

    string Yl = Y.substr(0, fh);
    string Yr = Y.substr(fh, sh);

    long int P1 = multiply(Xl, Yl);
    long int P2 = multiply(Xr, Yr);
    long int P3 = multiply(addBitStrings(Xl, Xr), addBitStrings(Yl, Yr));

    return P1*(1<<(2*sh)) + (P3 - P1 - P2)*(1<<sh) + P2;
}

int main(){

```



```
        cout<<"~~~~~Karatsuba Algorithm~~~~~" << endl;
cout<<"String X: 1100, String Y: 1010";
cout<<"\nHasil kali: "<<multiply("1100", "1010");
cout<<endl;

cout<<"\nString X: 11, String Y : 1010";
cout<<"\nHasil kali: "<<multiply("11", "1010");
cout<<endl;

cout<<"\nString X: 110, String Y : 1010";
cout<<"\nHasil kali: "<<multiply("110", "1010");
cout<<endl;

cout<<"\nString X: 11, String Y: 11";
cout<<"\nHasil kali: "<<multiply("11", "11");
cout<<endl;

cout<<"\nString X: 111, String Y : 1011";
cout<<"\nHasil kali: "<<multiply("111", "111");
}
```

D:\Semester 4\Prak. Analgo\Pertemuan 5\karatsuba.exe

```
~~~~~Karatsuba Algorithm~~~~~
String X: 1100, String Y: 1010
Hasil kali: 120

String X: 11, String Y : 1010
Hasil kali: 30

String X: 110, String Y : 1010
Hasil kali: 60

String X: 11, String Y: 11
Hasil kali: 9

String X: 111, String Y : 1011
Hasil kali: 49
-----
Process exited after 0.02607 seconds with return value 0
Press any key to continue . . .
```

2. Rekurensi Algoritma

- Let's try divide and conquer.
 - Divide each number into two halves.
 - $x = x_H r^{n/2} + x_L$
 - $y = y_H r^{n/2} + y_L$
 - Then:
$$xy = (x_H r^{n/2} + x_L) y_H r^{n/2} + y_L$$
$$= x_H y_H r^n + (x_H y_L + x_L y_H) r^{n/2} + x_L y_L$$
 - Runtime?
 - $T(n) = 4 T(n/2) + O(n)$
 - $T(n) = O(n^2)$
- Instead of 4 subproblems, we only need 3 (with the help of clever insight).
- Three subproblems:
 - $a = x_H y_H$
 - $d = x_L y_L$
 - $e = (x_H + x_L) (y_H + y_L) - a - d$
- Then $xy = a r^n + e r^{n/2} + d$
- $T(n) = 3 T(n/2) + O(n)$
- $T(n) = O(n^{\log_2 3}) = O(n^{1.584...})$

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 1×1). Isi papan menggunakan ubin berbentuk L. Ubin berbentuk L berukuran 2×2 persegi dengan satu sel berukuran 1×1 hilang.

Tugas:

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

Jawaban :

1. Program Tiling dengan C++

/*

Nama : Sharashena Chairani

Kelas : B

NPM : 140810180022

Program : Program Tiling

*/

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
// function to count the total number of ways
```

```
int countWays(int n, int m)
```

```
{
```

```
    // table to store values
```

```
    // of subproblems
```

```
    int count[n + 1];
```

```
    count[0] = 0;
```

```

// Fill the table upto value n
for (int i = 1; i <= n; i++) {
    // recurrence relation
    if (i > m)
        count[i] = count[i - 1] + count[i - m];

    // base cases
    else if (i < m)
        count[i] = 1;

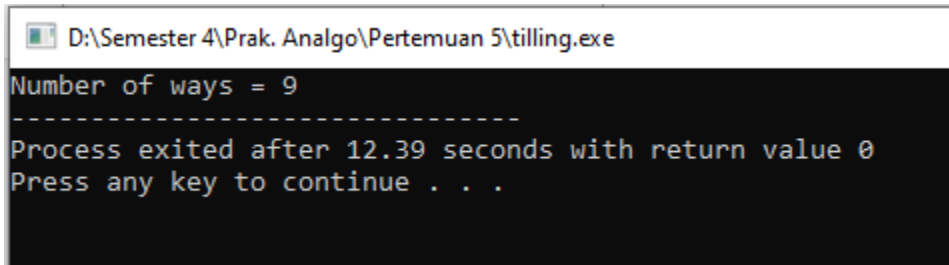
    // i == m
    else
        count[i] = 2;
}

// required number of ways
return count[n];
}

// Driver program to test above
int main()
{
    int n = 7, m = 3;
    cout << "Number of ways = "
        << countWays(n, m);
    return 0;
}

```

Screenshot :



```
D:\Semester 4\Prak. Analgo\Pertemuan 5\tilling.exe
Number of ways = 9
-----
Process exited after 12.39 seconds with return value 0
Press any key to continue . . .
```

2. Relasi rekurensi dengan Metode Master

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

C adalah konstanta.

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

Rekursi di atas dapat diselesaikan dengan menggunakan Metode Master dan kompleksitas waktu adalah $O(n^2)$

Algoritma Divide and Conquer dapat dibuktikan menggunakan Mathematical Induction. Biarkan kuadrat input berukuran $2k \times 2k$ di mana $k \geq 1$.

Kasus Dasar : Kita tahu bahwa masalahnya dapat diselesaikan untuk $k = 1$. Kami memiliki 2×2 persegi dengan satu sel hilang.

Hipotesis Induksi : Biarkan masalah dapat diselesaikan untuk $k-1$.

Untuk membuktikan bahwa masalah dapat diselesaikan untuk k jika dapat diselesaikan untuk $k-1$.

Untuk k , ditempatkan ubin berbentuk L di tengah dan memiliki empat subsquare dengan dimensi $2k-1 \times 2k-1$ seperti yang ditunjukkan pada gambar 2 di atas. Jadi jika dapat menyelesaikan 4 subkuarses, dapat menyelesaikan kuadrat lengkap