LAPORAN PRAKTIKUM

ANALISIS ALGORITMA

Disusun Oleh:

Natasya Rizky Maharani 140810180004

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS PADJADJARAN

2020

Studi Kasus 5: Mencari pasangan titik terdekat (Closest Pair of points)

1. **Program**

```
/*
Nama        : Natasya Rizky Maharani
NPM         : 140810180004
Kelas       : B
Deskripsi   : Program ini menampilkan 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;

    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)
{
    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()
{
```

```
    Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};
    int n = sizeof(P) / sizeof(P[0]);
    cout << "The smallest distance is " << closest(P, n);
    return 0;
}
```

```
The smallest distance is 1.41421




Exit code: 0 (normal program termination)
```

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

   Jawab: Asumsikan bahwa kita menggunakan algoritma pengururan 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)(n), mengurutkan strip dalam waktu O (n lg n) dan akhirnya menemukan titik terdekat dalam strip dalam watku 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 x lg n x lg n)

Catatan :

1. Kompleksitas waktu dapat ditingkatkan menjadi O(n lg n) dengan mengoptimalkan langkah 5 dari algoritma di atas.

2. Kode menemukan jarak terkecil dapat dengan mudah dimodifikasi untuk menemukan titik dengan jarak terkecil.

3. Kode ini menggunakan pengurutan cepat yang bisa O(n2) 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

1. **Program**

```
/*
Nama            : Natasya Rizky Maharani
NPM             : 140810180004
Kelas           : B
Deskripsi    : Program ini menampilkan algoritma karatsuba
*/

#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;
}
string addBitStrings( string first, string second )
{
    string result;
    int length = makeEqualLength(first, second);
    int carry = 0;
    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()
{
    printf ("%ld\n", multiply("1100", "1010"));
    printf ("%ld\n", multiply("110", "1010"));
    printf ("%ld\n", multiply("11", "1010"));
    printf ("%ld\n", multiply("1", "1010"));
    printf ("%ld\n", multiply("0", "1010"));
    printf ("%ld\n", multiply("111", "111"));
    printf ("%ld\n", multiply("11", "11"));
}
```

```
120
60
30
10
0
49
9


Exit code: 0 (normal program termination)
```

2. **Rekurensi dan 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)!**

- 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?
    - $T(n) = 4\,T(n/2) + O(n)$
    - $T(n) = O(n\text{^}2)$
- 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 $xy = a\ r^n + e\ r^{n/2} + d$
- $T(n) = 3\ T(n/2) + O(n)$
- $T(n) = O(n^{\log 3}) = O(n^{1.584...})$

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

1. **Program**

```
/*
Nama            : Natasya Rizky Maharani
NPM             : 140810180004
Kelas           : B
Deskripsi    : Program ini menampilkan Tilling Problem
*/
```

```
#include <bits/stdc++.h>
using namespace std;

int countWays(int n, int m)
{

        int count[n + 1];
        count[0] = 0;
        for (int i = 1; i <= n; i++) {
        if (i > m)
        count[i] = count[i - 1] + count[i - m];
        else if (i < m)
        count[i] = 1;
        else
        count[i] = 2;
        }
        return count[n];
}

int main()
{
        int n = 4, m = 2;
        cout << "Number of ways = "
        << countWays(n, m);
        return 0;
}
```

```
Number of ways = 5

Exit code: 0 (normal program termination)
```

// n adalah ukuran kotak yang diberikan, p adalah lokasi sel yang hilang Tile (int n, Point p)

1) Kasus dasar: n = 2, A 2 x 2 persegi dengan satu sel yang hilang tidak ada apa-apanya tapi ubin dan bisa diisi dengan satu ubin.

2) Tempatkan ubin berbentuk L di tengah sehingga tidak menutupi subsquare n / 2 * n / 2 yang memiliki kuadrat yang hilang. Sekarang keempat subsquare ukuran n / 2 x n / 2 memiliki sel yang hilang (sel yang tidak perlu diisi). Lihat gambar 2 di bawah ini.

3) Memecahkan masalah secara rekursif untuk mengikuti empat. Biarkan p1, p2, p3 dan p4 menjadi posisi dari 4 sel yang hilang dalam 4 kotak.

a) Ubin (n / 2, p1)

b) Ubin (n / 2, p2)

c) Ubin (n / 2, p3)

d) Ubin (n / 2, p3)

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.**

   Kompleksitas Waktu :

   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(n2)

Bagaimana cara kerjanya?

Pengerjaan algoritma divide and conquer dapat dibuktikan menggunakan mathematical induction. Biarkan kuadrat input berukuran 2k x 2k dimana k>=1.

Kasus Dasar : Kita tahu bahwa masalahnya dapat diselesaikan untuk k=1. Kami memiliki 2x2 persegi dengan satu sel hilang.

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

Sekarang perlu dibuktikan 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 x 2k-1 seperti yang ditunjukan pada gambar 2 di atas. Jadi jika dapat menyelesaikan 4 subsquares, dapat menyelesaikan kuadrat lengkap