2021-2022 GÜZ YARIYILI ALGORİTMA ANALİZİ ÖDEV – 1

Divide and Conquer ile Closest Pair Problemi Çözümü

Bu ödevde kartezyen düzleminde verilen n adet nokta arasında birbirine en yakın 2 noktanın Divide and Conquer yaklaşımı ile bulunması istenmektedir. Problemde tüm noktaların birbirinden farklı koordinatlarda yer aldığı kabul edilmelidir.

Umutcan SEVDİ

19011091

Algoritmamı geliştirirken programın mümkün olduğunca optimize, anlaşılır ve açık olmasına dikkat ettim. Sıralama algoritması olarak Quick Sort kullandım. Algoritmanın implementasyonunu kendim yazdım.

Programı mümkün olduğunca modüler olarak yazmaya dikkat ettim. Programım başlangıçta açılacak dosyanın ismini kullanıcıya sormaktadır. Dosyadaki satır sayısına göre **malloc()** fonksiyonu ile yer açıp ardından içine yerleştirme yapmaktadır. Kod içerisinde hiçbir yerde global değişken kullanılmamıştır. Gerektiği durumlarda bütün aktarımlar **pointer**lar üzerinden yapılmıştır. Program tekrar kullanılan işlemleri fonksiyonlara ayırmıştır. Minimumu bulma kısmı ise makrolar üzerinden tanımlanarak yapılmıştır. Kullanıcıya program çalışırken yapılan her fonksiyon çağrısının yapıldığı outputlar ile bildirilmiştir. Noktalar x ve y değerlerine sahip structlar dizisi şeklinde tanımlanmıştır.

Quick Sort içerisinde ve gerçekleşen minimum aramada da Divide and Conquer algoritması kullanılmıştır.

Kodu açıklamalarıyla devam eden sayfalarda ve arşiv dosyası içerisinde görüntüleyebilirsiniz.

FONKSİYONLAR

*getDistance(**point*** from, **point*** to)*

getDistance fonksiyonu iki nokta arasındaki mesafeyi **double** cinsinden döndüren fonksiyondur.

$$f(x,y) = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$$

*swap(**point*** a, **point*** b)*

Verilen iki noktanın adreslerini yer değiştirir.

*quickSort(**point*** array, **int** low, **int** high)*

Quick Sort fonksiyonu değerlerin x eksenine göre sıralanmasında kullanılır. Eğer girilen değişkenler (low ve high) doğru sıradaysa ***partition*** fonksiyonunu kullanarak değerlerin ayarlanmasını sağlar. Ardında ise Divide and Conquer algoritmasıyla ayrılan ***low*** ve ***high*** kısımlarına da aynı işlemi kendini çağırarak gerçekleştirir. Bu işlem low ve high değerler birbirine eşit olana yani en iç noktaya ulaşılana kadar devam eder.

int** partition(**point arrray, **int** low, **int** high)*

Bu fonksiyon her bir quick sort iterasyonlarında çağrılır. Bu fonksiyon içerisine gönderilen dizi üzerinde hareket ederek pivot değerinden daha küçük olan bir değerin pivotun sağındaki değerle yer değiştirmesini sağlar.

*bruteForce(**point*** array, **int** low, **int** high, **double*** result)*

Verilen noktalar arasındaki yakınlığın her bir ihtimali tek tek deneyerek bulunduğu bölmedir. Bu kısımda MIN makrosu kullanılır. İç içe for döngüsü ile noktalar arasındaki mesafeler hesaplanır.

*recursive(**point*** array, **int** low, **int** high, **double** *d)*

Bu fonksiyon belirli bir aralığa gelene kadar her seferinde daha küçük olacak şekilde hem sol hem de sağ taraftan kendini çağırır. Aralıkların yeterince küçülmesi durumunda **bruteForce** fonksiyonunu kullanır.

*printArray(**point*** *array,* ***int*** *low,* ***int*** *high)*

Girilen dizinin beirli bir aralıktaki elemanlarını yazdırır.

GENEL KARMAŞIKLIK

Sıralama sırasında quicksort algoritması($\mathrm{n} * \log n$) kullanıldı.

Algoritma tüm noktalar için sürekli olarak ikiye bölünüp yeterli aralığa ulaşana kadar kendini yinelemektedir. Böldükten sonra dikdörtgen dediğimiz bölmelerin oluşturulma kısmı O(n) karmaşıklığındadır. Bu küçük dikdörtgen alan içerisindeki hesaplama ise O(n) iledir. Bu durumda;

$$T(n) = 2T\left(\frac{N}{2}\right) + O(N) + O(N) + O(N) = 2T\left(\frac{N}{2}\right) + 3O(N) = N * \log N$$

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MIN(a, b) (((a) < (b)) ? (a) : (b))

typedef struct POINT
{
    int x;
    int y;
} point;

double getDistance(point *, point *);
void swap(point *, point *);
void quickSort(point *, int, int);
int partition(point *, int, int);
void bruteForce(point *, int, int, double *);
void recursive(point *, int, int, double *);
void printArray(point *, int, int);

int main()
{
    int i = 0, j = 0;
    // Name of the file that will be scanned
    char filename[20];
    // Length of the point array
    int length = 1;
    // A value that represents the closest distance between pairs
    double min = 99999;
    // At the end of the program, this value displays the shortest distance between closests points
    int shortestIndex = 0;
    // At the end of sorting, this index displays the index of the closest points
    printf("Enter name of the file you would like to scan: ");
    scanf("%s", filename);

    FILE *fp = fopen(filename, "r");
    // Opens a file and assigns it to a file pointer

    if (fp == NULL)
    {
        perror("\nError while opening the file.\n");
        exit(1);
    }

    // Reads the entire file to find the length of the array
    char ch;
    while ((ch = fgetc(fp)) != EOF)
        if (ch == '\n')
            length++;

    // Rewinds the filepointer, moves back to the top
    rewind(fp);

    // Allocates a point array based on the number of lines
    point *space = (point *)malloc(length * sizeof(point));
```

```
    //Reads the entire file, assigns x and y values to an array of struct
    while (fscanf(fp, "%d %d", &space[i].x, &space[i].y) != EOF)
        i++;
    fclose(fp);

    for (i = 0; i < length; i++)
        printf("\n-[%d] (%d,%d)", i, space[i].x, space[i].y);

    // Starts sorting
    quickSort(space, 0, length);

    //Recursively controls the array using Divide & Conquer
    recursive(space, 0, length - 1, &min);

    // After sorting the values and retrieving the median we can find the pairs distance to the median
    i = length / 2, j = length / 2;
    while (space[i].x - space[length / 2].x < min && space[length / 2].x - space[j].x < min)
    {
        printf("\n%d -> %d", i, j);

        if (getDistance(&space[i + 1], &space[i]) < min)
        {
            printf(" true1 %lf", min);
            min = getDistance(&space[i + 1], &space[i]);
        }
        if (getDistance(&space[j], &space[j - 1]) < min)
        {
            printf(" true2 %lf", min);

            min = getDistance(&space[j], &space[j - 1]);
        }
        i++;
        j--;
    }
    printf("\nDistance between closest pairs are -> %lf", min);
    free(space);
}

// A function that returns the distance between two points
double getDistance(point *from, point *to)
{
    return sqrt(pow((to->y - from->y), 2) + pow((to->x - from->x), 2));
}

void swap(point *a, point *b)
{
    point tmp = *a;
    *a = *b;
    *b = tmp;
}

// Recursive quick sort function
void quickSort(point *array, int low, int high)
```

```
{
    printf("\nquicksort :");
    printArray(array, low, high);
    int pi;
    if (low < high)
    {
        pi = partition(array, low, high);
        quickSort(array, low, pi - 1);
        quickSort(array, pi + 1, high);
    }
}

// A function that finds the median value of given array
int partition(point *array, int low, int high)
{

    printf("\npartition :");
    printArray(array, low, high);
    int pivot = array[high].x;
    int i, j = (low - 1);
    for (i = low; i < high; i++)
    {
        if (array[i].x < pivot)
        {
            j++;
            swap(&array[i], &array[j]);
        }
    }
    swap(&array[j + 1], &array[high]);
    printf("%d -> %d", pivot, j + 1);
    return (j + 1);
}

// When the array size is small enough finds the shortest distance
void bruteForce(point *array, int low, int high, double *result)
{
    int i, j;
    double min = getDistance(&array[high], &array[low]);
    for (i = low; i < high; i++)
    {
        for (j = i + 1; j <= high; j++)
            min = MIN(getDistance(&array[j], &array[i]), min);
    }
    if (min < *result)
        *result = min;
}

// Recursively divides the array until it's size is enough to perform Brute Force
void recursive(point *array, int low, int high, double *d)
{
    int med;
    if ((high - low) <= 2 && (high - low) >= 1)
        bruteForce(array, low, high, d);
    else
```

```
    {
        med = (high + low) / 2;
        recursive(array, low, med, d);
        recursive(array, med, high, d);
    }
}

// A function that prints all values of the given array to screen
void printArray(point *array, int low, int high)
{
    int i;
    for (i = low; i < high; i++)
        printf("%2d(%2d,%2d)  ", i, array[i].x, array[i].y);
}
```