YILDIZ TEKNİK ÜNİVERSİTESİ

ELEKTRİK-ELEKTRONİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

2019-2020 GÜZ YARIYILI

BLM3021 ALGORİTMA ANALİZİ DERSİ

GRUP-2

1.ÖDEV RAPORU

Hazırlayan: Mehmet Hayri Çakır – 16011023

Dersin Yürütücüsü: Dr. Öğr. Üyesi M. Amaç GÜVENSAN

İstanbul, Ekim 2019

İçindekiler

1. Yöntem

Problem: 2 boyutlu bir uzayda (x-y uzayı) n sayıdaki noktalardan birbirine en yakın olan 2 tanesinin efektif bir şekilde bulunması.

Bu problemin çözümü için divide-and-conquer yöntemini kullandım. Önce x'lerine göre sıraladığım noktaları rekürsif olarak sol ve sağ şeklinde bölerek sol ve sağ dizilerin minimum mesafesini bulduktan sonra x koordinatlarına göre tam ortadan bir çizgiyle ayırarak, o çizgiye minimum mesafe kadar yakın olan soldan ve sağdan birer nokta seçerek onların arasındaki mesafeleri hesapladım.

2. Uygulama

Örnek-1

```
File name: Input.txt

>Printing given array..
x        y
----------------------------------------
2        3
8        10
4        6
5        1
3        7
7        12

>Printing sorted array..
x        y
----------------------------------------
2        3
3        7
4        6
5        1
7        12
8        10

>Printing closest points..
Point1   Point2  Distance
----------------------------------------
4,6      3,7     1.414214

Press any key to continue . . .
```

Örnek-2

```
File name: Input2.txt

>Printing given array..
x         y
-----------------------------------------
1         150
2         11
3         22
4         33
5         44
6         55
7         66
8         77
9         88
10        -150

>Printing sorted array..
x         y
-----------------------------------------
1         150
2         11
3         22
4         33
5         44
6         55
7         66
8         77
9         88
10        -150

>Printing closest points..
Point1    Point2   Distance
-----------------------------------------
6,55      7,66     11.045361

Press any key to continue . . .
```

3. Kod

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#define COLNUMBER 2
#define TRUE 1
#define FALSE 0

#define RED   "\x1B[31;1m" //kirmizi yazi
#define GRN   "\x1B[32;1m" //yesil
```

```c
#define YEL   "\x1B[33;1m" //sari
#define BLU   "\x1B[34;1m" //mavi
#define MAG   "\x1B[35;1m" //magenta
#define CYN   "\x1B[36;1m" //cyan
#define WHT   "\x1B[37;1m" //beyaz
#define GRY "\x1B[30;1m"   //gri

typedef struct
{
	int x;
	int y;
}Point;

typedef struct
{
	Point p1;
	Point p2;
	double d;
}ClosestPair;

Point* GetPoints(int);
void QuickSort(Point*, int, int, char);
void PrintArray(Point*, int);
double bruteForceClosestPair(Point*, ClosestPair*, int);
double minOfTwoDouble(double, double);
double distanceBetweenTwoPoints(Point, Point, ClosestPair*);
void findMinInSubarrays(Point *, ClosestPair*, ClosestPair*, ClosestPair*, int);

int main()
{
	FILE* fp = NULL;	//dosya pointerı
	Point* points;		//noktalar dizisi
	char line[32];		//her satırı alacağım dizi
	char fileName[32];	//dosya adı
	char *token;  //strtok fonksiyonunda kullanılacak
	ClosestPair * closestPair = (ClosestPair *) calloc (1, sizeof(ClosestPair));
		//en yakın nokta için struct değişkeni
	ClosestPair * closestPairLeft = (ClosestPair *) calloc (1, sizeof(ClosestPair));
	//sol
	ClosestPair * closestPairRight = (ClosestPair *) calloc (1,
sizeof(ClosestPair));// ve sağ en yakın noktalar için değişkenler
	int n = 0;	//satır sayısını tutacak
	int i; //döngü değişkeni

	system("COLOR F5");
	(*closestPair).d = INT_MAX;
	(*closestPair).p1.x = 0;
	(*closestPair).p1.y = 0;
	(*closestPair).p2.x = 0;
	(*closestPair).p2.y = 0;
	(*closestPairLeft).d = INT_MAX;
	(*closestPairRight).d = INT_MAX;
	printf("File name: ");
	scanf("%s", fileName);

	if((fp = fopen(fileName, "r")) == 0)
	{
		exit(-2);
	}
	while(fgets(line, 32, fp))
	{
		++n;
```

```
	}
	points = (Point*) calloc(n,sizeof(Point));
	fseek(fp,0,SEEK_SET);
	for (i = 0; i < n; ++i)
	{
		fgets(line, 32, fp);
		token = strtok(line," \n");
		points[i].x = atoi(token);
		token = strtok(NULL," \n");
		points[i].y = atoi(token);
	}
	printf(GRY"\n>Printing given array..\nx \t y\n--------------------------------
------\n"RED);
	PrintArray(points, n);

	//points = GetPoints(n);
	//createPointsArray(points,n);
	QuickSort(points, 0, n-1, 'x');
	findMinInSubarrays(points, closestPair, closestPairLeft, closestPairRight, n);

	printf(GRY"\n>Printing sorted array..\nx \t y\n--------------------------------
-------\n"MAG);
	PrintArray(points, n);

	printf(GRY"\n>Printing closest points..\nPoint1 \t Point2\t Distance\n----------
------------------------------\n"BLU);
	printf("%d,%d \t %d,%d \t %f\n\n"GRY, (*closestPair).p1.x, (*closestPair).p1.y,
(*closestPair).p2.x, (*closestPair).p2.y, (*closestPair).d);
	system("PAUSE");
	return 0;
}

//orta çizgiye yakın noktaların minimum mesafesini bulur.
double findMinInMidLine(Point* pointsAroundMidLine, int j, double minOfLeftRightD,
ClosestPair *closestPair)
{
    double min = minOfLeftRightD;
	int i;
	int k;
	QuickSort(pointsAroundMidLine, 0, j-1, 'y');
    for (i = 0; i < j; ++i)
	{
		for (k = i+1; k < j && (pointsAroundMidLine[k].y -
pointsAroundMidLine[i].y) < min; ++k)
		{
			if
(distanceBetweenTwoPoints(pointsAroundMidLine[i],pointsAroundMidLine[k], closestPair)
< min)
			{
				min = distanceBetweenTwoPoints(pointsAroundMidLine[i],
pointsAroundMidLine[k], closestPair);
			}
		}
	}
    return min;
}


//sol ve sağ dizilerde minimum mesafeyi bulur.
void findMinInSubarrays(Point * points, ClosestPair* closestPair,ClosestPair*
closestPairLeft,ClosestPair* closestPairRight, int n)
{
```

```
	Point* pointsAroundMidLine;
	double minOfLeftRightD;
	int i;
	int j;
	if((n >= 2) && (n <= 3))
	{
		bruteForceClosestPair(points, closestPair, n);
		return;
	}
	findMinInSubarrays(points, closestPairLeft, closestPair, closestPairRight, n/2);
	findMinInSubarrays(points + n/2, closestPairRight, closestPair, closestPairLeft,
n-n/2);
	minOfLeftRightD = minOfTwoDouble((*closestPair).d, (*closestPairRight).d);

	pointsAroundMidLine = (Point*) calloc (n, sizeof(Point));
	j = 0;
	for (i = 0; i < n; ++i)
	{
		if (abs(points[i].x - points[n/2].x) < minOfLeftRightD)
		{
	     pointsAroundMidLine[j] = points[i];
			j++;
		}
	}

	findMinInMidLine(pointsAroundMidLine, j, minOfLeftRightD, closestPair);
}


//iki doubledan küçük olanını döner.
double minOfTwoDouble(double a, double b)
{
	return (a > b) ? b : a;
}

//iki nokta arası mesafeyi bulup, closestpair.d sinden küçükse güncelliyor.
double distanceBetweenTwoPoints(Point p1, Point p2, ClosestPair* closestPair)
{
	double dist = sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y -
p2.y));
	if(dist < (*closestPair).d)
	{
		(*closestPair).p1 = p1;
		(*closestPair).d = dist;
		(*closestPair).p2 = p2;
	}
	return dist;
}


//brute force karşılaştırma fonksiyonu
double bruteForceClosestPair(Point* points, ClosestPair* closestPair, int n)
{
	int i;//döngü değişkeni
	int j;//döngü değişkeni
	for (i = 0; i < n; ++i)
	{
		for (j = i + 1; j < n; ++j)
		{
			if(distanceBetweenTwoPoints(points[i], points[j], closestPair) <
(*closestPair).d)
			{
```

```
                        (*closestPair).p1 = points[i];
                        (*closestPair).p2 = points[j];
                        (*closestPair).d = distanceBetweenTwoPoints(points[i],
points[j], closestPair);
                    }
                }
        }
        return (*closestPair).d;
}

//diziyi yazdırmak için
void PrintArray(Point* points, int n)
{
        int i;
        for(i = 0; i < n; ++i)
        {
                printf("%d \t %d\n", points[i].x,points[i].y);
        }
}

//yer değişme işlemini yapacak
void Swap(int* a, int* b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}


//quicksort için bölme(divide) işlemi
int Partition (Point* arr, int lo, int hi, char whichAxis)
{
        int j;
    int pivot = whichAxis=='x' ? arr[hi].x : arr[hi].y;
       int valToComparePivot;
    int i = (lo - 1);

    for (j = lo; j <= hi- 1; j++)
    {
              valToComparePivot = whichAxis=='x' ? arr[j].x : arr[j].y;
        if (valToComparePivot < pivot)
        {
            i++;
            Swap(&arr[i].x, &arr[j].x);
            Swap(&arr[i].y, &arr[j].y);
        }
    }
    Swap(&arr[i + 1].x, &arr[hi].x);
    Swap(&arr[i + 1].y, &arr[hi].y);
    return (i + 1);
}

void QuickSort(Point* arr, int lo, int hi, char whichAxis)
{
        int pi;//pivot
    if (lo < hi)
    {
        pi = Partition(arr, lo, hi, whichAxis);

        QuickSort(arr, lo, pi - 1, whichAxis);
        QuickSort(arr, pi + 1, hi, whichAxis);
    } }
```