DIVIDE AND CONQUER APPROACH TO CLOSEST PAIR PROBLEM

15.11.2021

ALGORİTMA ANALİZİ – BLM3021

1. ÖDEV RAPORU

18011047 – BERKAY KOÇ

YÖNTEM

Closest pair problem, uzaydaki n sayıdaki nokta arasından en yakın iki noktayı bulmayı hedefleyen problemdir. Çözüm için akla ilk gelen çözüm tüm ikilileri birbirleriyle karşılaştırarak en kısa mesafeyi bulmaktır. Fakat bu çok fazla işlem yapmayı gerektiren bir yöntemdir ve n arttıkça problem çok daha büyüyecektir. Bu sebeple önerilen yöntem divide and conquer yöntemidir. Bu yönteme göre noktalar öncelikle x değerlerine göre sıralanmalı, sonra noktalar kümesi baştan sona bakıldığında tam ortada kalan medyan değeri ile ikiye bölünmeli, sağ ve solda kalan sayılar kümesi 3'ün altına inene kadar bu işlem tekrar edilmeli, bu noktadan sonra noktaların arasındaki mesafe hesaplanarak en yakın ikili güncellenmelidir. Fakat bu aynı kümenin sağ ve sol yarısındaki iki elemanın birbirine en yakın elemanlar olabileceğini göz ardı ettiğinden sağ ve sol kümeler medyanla ayrılıp kontrol edildikten sonra sol ve sağ taraftaki ikililer de kontrol edilmelidir. Fakat yine her ikiliyi kontrol etmeye gerek yoktur. Medyan değerinin o ana kadar bulunmuş en küçük değer kadar sağında ve solundaki değerleri birbirleriyle karşılaştırmak yeterli olacaktır. Bu yöntem sayesinde daha az ikili birbiriyle karşılaştırılabilmektedir. Rekürsif fonksiyonumuz medyan tarafından bölünen dizinin elemanları 3'den büyük olduğu sürece ikiye bölünecek, 3'den küçük oldukları anda brute force metot ile hesaplanarak sağdaki ve soldaki aralık bulunacaktır. Sonrasında ise sağ ve sol parçalar bir kontrol ile karşılaştırılacaktır.

KARMAŞIKLIK

Fonksiyon içerisinde elemanları sıralama işlemini bubble sort kullanarak yapıyoruz. Bubble sort'un worst case ve best case karmaşıklıkları şöyledir:

Worst case: $O(n^2)$

Best case: $\Omega(n)$

Rekürsif olarak ikiye bölerek ilerlediğimiz için C(n/2), bunu her iki taraf için de yaptığımız için 2C(n/2) olmalıdır. Bunun yanında yaptığımız dikdörtgenlerin içerisindeki değerleri hesaplama işlemi(Compare pairs) ise n karmaşıklığa sahiptir.

C(n) = 2C(n/2) + n

Master teoremi kullanarak karmaşıklığı O(nlogn) olarak buluruz.

Best case durumunu hesaplamak için compare pairs fonksiyonunun yani dikdörtgenlerin içerisindeki ikilerin hesaplandığı durumun devre dışı kalması gerekir. Bu durumda da fonksiyon;

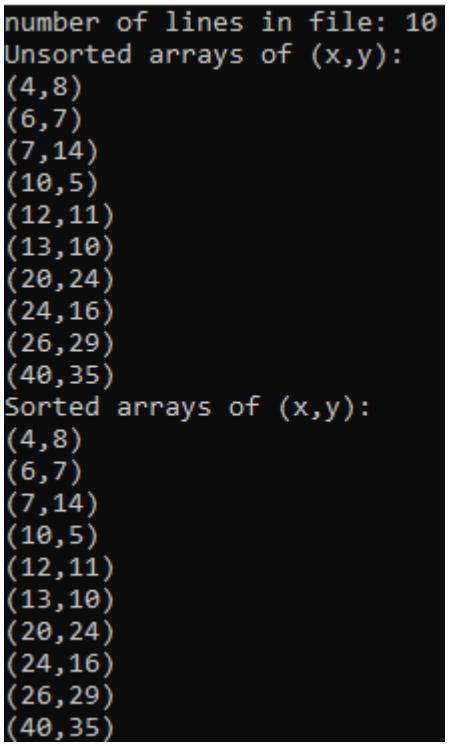
C(n) = 2C(n/2) + 1 halini alır.

Bu durumda master teoreme göre karmaşıklık: O(n) olacaktır.

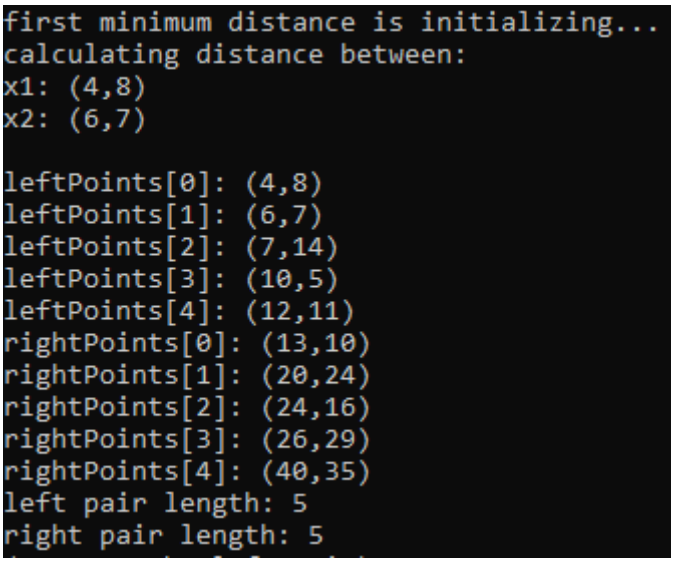
Programın best case durumu O(n)

Programın worst case durumu O(n^2) olacaktır.

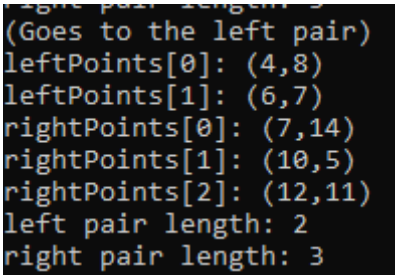
UYGULAMA



Öncelikle sample.txt'deki ikililerin sayısı, sonrasında elemanların sırasız halleri, sonrasında sıralı halleri ekrana gelir.



Sonra başta dizinin en yakın ikilisi seçilecek ilk iki nokta ekrana gelir. Sonra dizi sol ve sağ parçalara bölünür. Bu parçalar tek tek ekrana yazılır.



Sol parçaya gidilir ve sol parça eleman sayısı 3'ten büyük olduğundan dizi ikiye bölünür.

```
comparing (4, 8), (6, 7)
calculating distance between:
x1: (4,8)
x2: (6,7)

distance between (4, 8) and (6, 7) = 2.236068
```

Bir kez daha sol parçaya gidilir ve sol parçadaki eleman sayısı 2 olduğundan ikili arasındaki uzaklık hesaplanır.

```
comparing (7, 14),(10, 5),(12, 11)
calculating distance between:
x1: (7,14)
x2: (10,5)

distance between (7, 14) and (10, 5) = 9.486833
calculating distance between:
x1: (7,14)
x2: (12,11)

distance between (7, 14) and (12, 11) = 5.830952
calculating distance between:
x1: (10,5)
x2: (12,11)

distance between (10, 5) and (12, 11) = 6.324555
calculating distance between:
x1: (6,7)
x2: (7,14)
```

Sol parçadaki işlemler tamamlandığından sağ parçadaki hesaplamalar yapılır.

Burada 3 eleman olduğundan aralarındaki uzaklık teker teker hesaplanır.

```
(Goes to the right pair)
leftPoints[0]: (13,10)
leftPoints[1]: (20,24)
rightPoints[0]: (24,16)
rightPoints[1]: (26,29)
rightPoints[2]: (40,35)
left pair length: 2
right pair length: 3
```

Sample.txt'den gelen dizideki sol kısım tamamen hesaplanmıştır. Artık sağ kısım ikiye bölünerek hesaplanacaktır. 2. Resimdeki rightPoints dizisinin burada iki parçaya bölündüğünü görüyoruz.

```
(Goes to the left pair)


comparing (13, 10), (20, 24)
calculating distance between:
x1: (13,10)
x2: (20,24)

distance between (13, 10) and (20, 24) = 15.652476
```

Yukarıdaki leftPoints dizisi iki elemanlı olduğundan aralarındaki mesafe hesaplanır.

```
(Goes to the right pair)
comparing (24, 16),(26, 29),(40, 35)
calculating distance between:
x1: (24,16)
x2: (26,29)

distance between (24, 16) and (26, 29) = 13.152946
calculating distance between:
x1: (24,16)
x2: (40,35)

distance between (24, 16) and (40, 35) = 24.839485
calculating distance between:
x1: (26,29)
x2: (40,35)

distance between (26, 29) and (40, 35) = 15.231546
calculating distance between:
x1: (12,11)
x2: (13,10)
```

Daha sonra kalan son üçlü hesaplanır. Bu üçlü arasında da en küçük değerden küçük değer elde edilmediğinden minVal değişkeni güncellenmez.

```
calculating distance between:
x1: (12,11)
x2: (13,10)

comparing (12, 11), (13, 10)
calculating distance between:
x1: (12,11)
x2: (13,10)

minimum distance is updated!
Minimum distance is between pairs (12, 11) and (13, 10). Minimum distance is: 1.414214
```

Son olarak medyan değerinin sağında ve solundaki d uzaklıktaki parçalar birbiriyle karşılaştırılır. Bu parçalardan (12, 11) ve (13, 10) birbirine minVal değerinden daha yakın olduğundan minVal değeri güncellenir. Rekürsif adımlar bittiğinden ekrana bulunan en yakın ikili yazdırılır ve program sonlanır.

KOD:

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>


struct point {

   int x, y;

};


struct point point1;

struct point point2;

float minVal;


float findMedian(struct point *points, int n);

float findDistance(struct point p1, struct point p2);

void insertionSort(struct point *points, int n);

void bruteForce(struct point *points, int n);

void splitAndCalculate(struct point *points, int n);

void comparePairs(struct point *leftPoints, struct point *rightPoints, int leftLength, int rightLength,
float median);


int main(){

        FILE *inputFile; //değerlerin okunacağı dosya

        int n=0;                    //noktalardan oluşacak olan struct dizisinin boyutu

        int temp;                           //struct dizisinin boyutunu bulmak için gerekli ilk gezintide
satırları saklamak için kullanılacak temp değer

        int i=0;                    //sıralı listeyi ekrana yazdıracak for döngüsü

        struct point *points;      //point struct yapılarından oluşacak struct dizisi



        if ((inputFile = fopen ("sample.txt", "r+")) == NULL){
```

```
    printf("can't open the file");
    return 0;
  }

      while(!feof(inputFile)){
      fscanf(inputFile,"%d %d\n", &temp, &temp);
      n++;

      }
      printf("number of lines in file: %d\n", n);

      points = (struct point *)malloc(sizeof(struct point) * n);

      fclose(inputFile);                              //dosya pointerının sıfırlanması için kapatılıyor, tekrar açılacak.

      if ((inputFile = fopen ("sample.txt", "r+")) == NULL){
    printf("can't open the file");
    return 0;
  }

      for(i=0; i<n; i++){

      fscanf(inputFile,"%d %d\n", &points[i].x, &points[i].y);
      }
      fclose(inputFile);
      printf("Unsorted arrays of (x,y):\n");
      for(i=0; i<n; i++){

            printf("(%d,%d)\n", points[i].x, points[i].y);
      }
```

```
   insertionSort(points, n);


   printf("Sorted arrays of (x,y):\n");
         for(i=0; i<n; i++){

               printf("(%d,%d)\n", points[i].x, points[i].y);
         }
         printf("first minimum distance is initializing...\n");
         minVal = findDistance(points[0], points[1]);       //en küçük değeri initialize etmek için 0 ile 1. noktalar arasındaki fark hesaplanıyor.
         point1 = points[0];
         point2 = points[1];
         splitAndCalculate(points, n);
         printf("Minimum distance is between pairs (%d, %d) and (%d, %d). Minimum distance is: %f\n",point1.x, point1.y, point2.x, point2.y, minVal);


         return 0;
}


void insertionSort(struct point *points, int n){ //noktalar listesini x değerlerine göre sıralayacak olan fonksiyondur. Struct dizisini ve dizinin boyutunu alır, dönüş değeri yoktur çünkü dizi pointer olarak pass edilmiştir.
   int i; //diziyi baştan sonra gezecek olan indis
   int j; //dizinin geride kalan kısmını sıralayacak olan indis
   struct point key; //dizideki sıralanmış ve sıralanmamış elemanları ayıracak olan eleman


   for (i=1; i<n; i++){
      j = i-1;
      key = points[i];
      while (j >= 0 && points[j].x > key.x){
         points[j + 1] = points[j];
         j = j - 1;
      }
```

```
        points[j + 1] = key;
    }
}

void selectionSort(struct point *points, int n)
{
    int i; //diziyi baştan sonra gezecek olan indis
    int j; //dizinin geride kalan kısmını sıralayacak olan indis
    int key; //dizideki sıralanmış ve sıralanmamış elemanları ayıracak olan eleman
            struct point temp; //swap işlemi için kullanılacak temp değeri

    for (i = 0; i < n-1; i++){
        key = points[i].x;
        for (j = i+1; j < n; j++)
          if (points[j].x < points[key].x);
            key = j;
        temp = points[i];
        points[i] = points[key];
        points[key] = temp;
    }
}

float findDistance(struct point p1, struct point p2){ // iki nokta arasındaki euclidean distance'ı bulacak olan fonksiyon. İki noktayı parametre olarak alır farklarını float değer olarak geri döndürür.
            printf("calculating distance between:\nx1: (%d,%d)\nx2: (%d,%d)\n\n", p1.x, p1.y, p2.x, p2.y);
            return sqrt((p1.x  - p2.x)*(p1.x  - p2.x) + (p1.y - p2.y)*(p1.y - p2.y));
}

float findMedian(struct point *points, int n){ //verilen listeyi ortadan ikiye bölmek için gerekecek medyan değerini bulur. Diziyi ve dizinin uzunluğunu parametre olarak alır. Geriye medyan değerini float olarak döndürür
            if(n%2 == 1){
```

```
		return points[n/2].x;
	}
	else{
		float fl = (n-1)/2;
		float cl = (n/2);
		return (float)(points[(int)fl].x + points[(int)cl].x)/2;
	}
}
```

void splitAndCalculate(struct point *points, int n){ // diziyi ikiye böle böle rekürsif olarak yarılayacak, değer 3'den küçükse brute force metodunu çağıracak, değilse tekrar ikiye bölecek olan fonksiyon. Noktalar dizisini ve dizinin uzunluğunu alır. Dönüş değeri yoktur.

```
	int i;	//Bölünecek olan sol ve sağ dizilere elemanları yerleştirmek için gerekli olan indis.
	int leftLength=0;		//sol dizinin büyüklüğü
	int rightLength=0;		//sağ dizinin büyüklüğü
	float median;			//parametre olarak verilen dizinin medyan değeri.
	struct point *leftPoints;//soldaki bölünmüş dizinin elemanlarını tutacak olan struct dizisi
	struct point *rightPoints;		//sağdaki bölünmüş dizinin elemanlarını tutacak olan struct dizisi

	if(n%2 == 1){ //dizi tek sayıda veya çift sayıda elemandan oluşmasına göre iki farklı yöntemde oluşturulabilir.
		leftPoints = (struct point *) malloc(sizeof(struct point) * (n/2));
		leftLength = n/2;
		rightPoints = (struct point *) malloc(sizeof(struct point) * (n+1)/2);
		rightLength = (n+1)/2;
	}
	else{
		leftPoints = (struct point *) malloc(sizeof(struct point) * (n/2));
		leftLength = n/2;
		rightPoints = (struct point *) malloc(sizeof(struct point) * (n/2));
		rightLength = (n+1)/2;
```

```
	}
	if(n<=3){
		bruteForce(points, n);
	}
	else{
		median = findMedian(points, n);
		for(i=0; i<leftLength; i++){
			leftPoints[i] = points[i];
			printf("leftPoints[%d]: (%d,%d)\n",i, leftPoints[i].x,leftPoints[i].y);
		}
		for(i=0; i<rightLength; i++){
			rightPoints[i] = points[i+leftLength];
			printf("rightPoints[%d]: (%d,%d)\n",i, rightPoints[i].x,rightPoints[i].y);
		}
		printf("left pair length: %d\n", leftLength);
		printf("right pair length: %d\n", rightLength);
		printf("(Goes to the left pair)\n");
		splitAndCalculate(leftPoints, leftLength);
		printf("(Goes to the right pair)\n");
		splitAndCalculate(rightPoints, rightLength);
		comparePairs(leftPoints, rightPoints, leftLength, rightLength, median);
	}
}
```

void bruteForce(struct point *points, int n){ //2 veya 3 elemanlı dizideki elemanların aralarındaki farkları brute force olarak hesaplayacak fonksiyon. noktalar dizisini ve dizinin büyüklüğünü parametre olarak alır. Dönüş değeri yoktur, global değişkenleri günceller

	float dist1; //dizi 3 elemanlıysa 0. ve 1. elemanlar arasındaki farkı, iki elemanlıysa iki eleman arasındaki farkın değerini tutacak olan değişken

	float dist2; //dizi 3 elemanlıysa 0. ve 2. elemanlar arasındaki farkı tutacak olan değişken

	float dist3; //dizi 3 elemanlıysa 1. ve 2. elemanlar arasındaki farkı tutacak olan değişken

struct point temp1; //hesaplanan uzaklık değerinin en küçük uzaklıktan küçük olup olmamasına göre noktalar değişecektir. Bu süre içerisinde 1. noktayı tutacak olan temp değeri

struct point temp2; //hesaplanan uzaklık değerinin en küçük uzaklıktan küçük olup olmamasına göre noktalar değişecektir. Bu süre içerisinde 2. noktayı tutacak olan temp değeri

```
if(n == 3){
        printf("comparing (%d, %d),(%d, %d),(%d, %d)\n", points[0].x, points[0].y, points[1].x, points[1].y, points[2].x, points[2].y);
        dist1 = findDistance(points[0], points[1]);
        printf("distance between (%d, %d) and (%d, %d) = %f\n", points[0].x, points[0].y, points[1].x, points[1].y, dist1);
        dist2 = findDistance(points[0], points[2]);
        printf("distance between (%d, %d) and (%d, %d) = %f\n", points[0].x, points[0].y, points[2].x, points[2].y, dist2);
        dist3 = findDistance(points[1], points[2]);
        printf("distance between (%d, %d) and (%d, %d) = %f\n", points[1].x, points[1].y, points[2].x, points[2].y, dist3);
        if(dist2 < dist1){
                if(dist2 < dist3){
                        temp1 = points[0];
                        temp2 = points[2];
                        dist1 = dist2;
                }
                else{
                        temp1 = points[1];
                        temp2 = points[2];
                        dist1 = dist3;
                }
        }
        else if(dist3 < dist1){
                temp1 = points[1];
                temp2 = points[2];
                dist1 = dist3;
```

```
		}
		else{
			temp1 = points[0];
			temp2 = points[1];
		}
	}
	else{
		printf("\n\ncomparing (%d, %d), (%d, %d)\n", points[0].x, points[0].y,  points[1].x, points[1].y);
		temp1 = points[0];
		temp2 = points[1];
		dist1 = findDistance(points[0], points[1]);
		printf("distance between (%d, %d) and (%d, %d) = %f\n", points[0].x, points[0].y, points[1].x, points[1].y, dist1);
	}
	if(dist1 < minVal){ //bulunan en küçük mesafe daha önce bulunan mesafeden küçükse minVal global değişkeni , point1 ve point2 değişkenleri güncellenir.
		point1 = temp1;
		point2 = temp2;
		minVal = dist1;
		printf("minimum distance is updated!\n");
	}
}

void comparePairs(struct point *leftPoints, struct point *rightPoints, int leftLength, int rightLength, float median){ //medyanın sağ ve solundaki en fazla minVal uzaklıktaki değerleri birbirleriyle karşılaştıracak, eğer minVal'den küçük değer varsa minVal, point1, point2 değerlerini güncelleyecek olan fonksiyon.

// Parametre olarak sol dizideki struct değerlerini, sağ dizideki struct değerlerini, sol dizinin uzunluğunu, sağ dizinin uzunluğunu ve dizinin medyan değerini alır. Dönüş değeri yoktur. global değişkenleri günceller.

	int i = leftLength-1; //soldaki dizinin elemanlarını mesafe hesabı için tek tek gezecek olan indis
```

```
	int j = 0; //sağdaki dizinin elemanlarını mesafe hesabı için tek tek gezecek olan indis
	while(leftPoints[i].x > (int)(median - minVal)){
		while(rightPoints[j].x < (int)(median + minVal)){
			if(findDistance(leftPoints[i], rightPoints[j]) < minVal){
				printf("comparing (%d, %d), (%d, %d)\n", leftPoints[i].x, leftPoints[i].y, rightPoints[j].x, rightPoints[j].y);
				minVal = findDistance(leftPoints[i], rightPoints[j]);
				point1 = leftPoints[i];
				point2 = rightPoints[j];
				printf("minimum distance is updated!\n");
			}
			j++;
		}
		i--;
	}
}
```