DİNAMİK PROGRAMLAMA KULLANARAK EN KÂRLI REKLAM DİZİSİNİ BULMAK

13.12.2021

ALGORİTMA ANALİZİ – BLM3021

2. ÖDEV RAPORU

18011047 – BERKAY KOÇ

YÖNTEM:

Reklamlar arasından en kârlı diziyi seçmek için öncelikle bir struct yapısı oluşturdum. Struct yapısının içerisinde input değerleri olan startTime, duration ve value değerleri, bunun dışında startTime ve duration toplamını tutması için bir de finish değeri vardır. Dosyadan okunan değerler öncelikle okunduğu sıra ile struct dizisi içerisine yerleştirilir. Sonrasında QuickSort kullanılarak finish değerlerine göre sıralı hale getirilir. Devamında sıralı haldeki elemanlar tek tek gezilerek seçilme durumları kontrol edilir. O elemana kadar oluşan en iyi kâr değerinin saklanması için girilen eleman sayısı kadar bir integer dizisi açılır. Eğer önceki reklama kadar hesaplanmış value değeri o anki eleman ve kendisinden önce kendisiyle çakışmayan son elemanın toplam kâr değerinden daha büyük ise önceki eleman tekrar seçilir. Tam tersi durumda ise yeni kâr değeri o anki eleman ve kendisinden önce kendisiyle çakışmayan son elemanın toplam kâr değeri olacaktır. Bu işlem sonucunda en yüksek kâr bulunur. Sonrasında liste sondan başa gezilerek son değişiklik olan kâr değerinden geriye çakışmayan elemanlar bulunarak ekrana yazdırılır. Bu işlem sonucunda sıralanmış listenin ekranda yazdırılan sıralarındaki reklamlar en kârlı reklamlar olacaktır.

REKÜRANS BAĞINTISI:

$$\boldsymbol{Profit(n) = max\{Profit(n-1), ads(n) + Profit(i)\}}$$

$$\boldsymbol{where, i\ is\ the\ last\ unconflicted\ ad\ with\ nth\ ad}$$

ZAMAN VE YER KARMAŞIKLIĞI:

Quick Sort Yer ve Zaman Karmaşıklıkları:

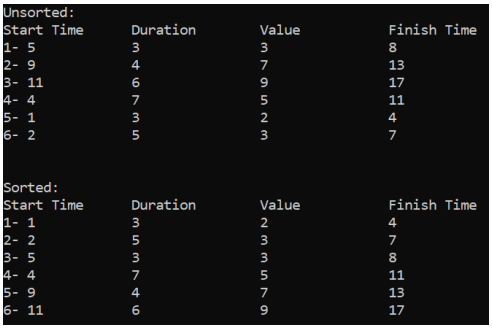
Zaman Karmaşıklığı, Master Theorem kullanılarak 2T(n/2) + 1 bağıntısından O(nlog(n)) olacaktır.

Yer Karmaşıklığı, ekstra bir bellek alanı kullanılmadığından O(n) olacaktır.

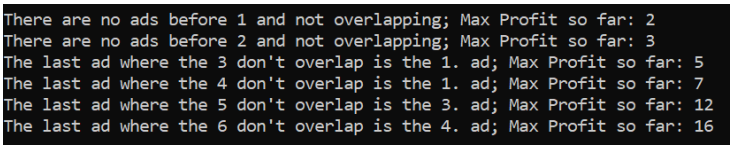
Fonksiyonun geneline baktığımızda ise Profit ve ads dizileri n boyutundadır. Bunun dışında bellek alanı kullanılmamaktadır. Dolayısıyla yer karmaşıklığı O(n) olacaktır.

Sıralı hale getirilmiş dizinin baştan sona gezilerek Profit dizisinin doldurulduğu kısımdaki for ve while döngülerine bakmamız yeterli olacaktır. Bu kısım karmaşıklığı oluşturan kısımdır. Elemanları belirleme aşamasında sondan başa yalnızca bir geçiş yapılmaktadır. Bu döngülere bakıldığında for döngüsünün her ihtimalde n kadar döneceğini, while döngüsünün ise en iyi durum olan her seferinde bir önceki reklam ile çakışmaması durumunda 1 kez döneceğini, en kötü durumda ise her seferinde logn kadar döneceğini görürüz. Bunları değerlendirdiğimizde en iyi durumdaki karmaşıklığımız quick sorttaki karmaşıklık nedeniyle O(n logn), en kötü durumdaki karmaşıklığımız için O(n logn) olacaktır.

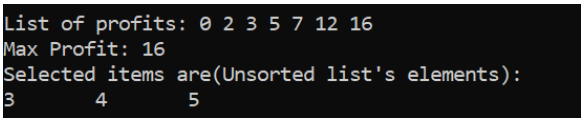
UYGULAMA:



Verilen reklamların önce sıralanmamış hali sonrasında ise sıralanmış hali ekrana yazdırılır.



Sıralı hali verilmiş reklamlar teker teker gezilirken o reklama kadarki çakışmalar ve o ana kadarki kâr ile ilgili bilgi verilir. Bu bilgiler sıralı listeye göredir.



Sıralı olarak kârlar yazdırılır, en yüksek alınabilecek kâr yazdırılır, sonrasında ise bu kâr sıralı haldeki reklamlardan seçilmesi gerekenler ekrana yazdırılır.

KOD:

```
#include <stdio.h>

#include <stdlib.h>


struct ad{

        int startTime; // reklamın başlangıç zamanı

        int duration; // reklamın süresi

        int value; // reklamın değeri, ücreti

        int finish; // reklamın bitiş zamanı, duration + startTime olarak hesaplanır

        int index; // dizinin sırasız halindeki indisini tutacak indis elemanı

};


void quickSort(struct ad *ads,int first,int last); //reklamların sırasız listesini, başlangıç ve bitiş indislerini alarak listeyi sıralar.


int main(){

        int i; //indisleme için kullanılır, kod içerisinde farklı amaçlarla kullanılmıştır.
```

```
        int j; //indisleme için kullanılır, kod içerisinde farklı amaçlarla kullanılmıştır.
        int n; //reklamların dizi uzunluğunu ifade eder.
        int tempProfit; //karşılarştırmaların yapılabilmesi için geçici bir profit değerlerini tutacak değişken.
        struct ad *ads; //reklam listesini içerecek olan struct dizisi
        int *profit; //kâr değerlerini içerisinde tutacak olan reklam sayısı uzunluklu kâr dizisi
        FILE *fp; //dosyadan değer okumak için kullanılacak file pointer.

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

  i = 1;

        ads = (struct ad*)malloc(sizeof(struct ad) * 1);

        ads[0].startTime = 0;
  ads[0].duration = 0;
  ads[0].value = 0;

        while(!feof(fp)){  // dosyanın sonuna kadar realloc kullanarak elemanlar alınır, yerleştirilir.
                ads = (struct ad*) realloc(ads, sizeof(struct ad)*(i+1));
                fscanf(fp,"%d %d %d\n", &ads[i].startTime, &ads[i].duration, &ads[i].value);
                ads[i].finish = ads[i].startTime + ads[i].duration;
                ads[i].index = i;
                i++;
        }

        n=i-1;
        profit = (int *)malloc(sizeof(int) * i);
```

```
	printf("Unsorted: \n");
	printf("Start Time\tDuration\tValue\t\tFinish Time\n");
	for(i=1; i<n+1; i++){
		printf("%d- %d\t\t%d\t\t%d\t\t%d\n", i, ads[i].startTime, ads[i].duration, ads[i].value, ads[i].finish);
	}

	quickSort(ads, 0, n);

	printf("\n\n");
	printf("Sorted: \n");
	printf("Start Time\tDuration\tValue\t\tFinish Time\n");
	for(i=1; i<n+1; i++){
		printf("%d- %d\t\t%d\t\t%d\t\t%d\n", i, ads[i].startTime, ads[i].duration, ads[i].value, ads[i].finish);
	}

	profit[0] = 0;
	profit[1] = ads[1].value;
	tempProfit = 0;

	printf("\n");
	for(i=1; i<n+1; i++){ //sıralı hale getirilen dizi baştan sona gezilecektir.
		j = i-1;
		while(ads[j].finish > ads[i].startTime && j>0) // çakışma durumu bu döngü ve kontrol içerisinde gerçekleştirilecektir
			j--;
		if(j != 0)
			printf("The last ad where the %d don't overlap is the %d. ad", i, j);
		else
			printf("There are no ads before %d and not overlapping", i);
		tempProfit = ads[i].value + profit[j];
```

```
            if(tempProfit > profit[i-1])
                profit[i] = tempProfit;
            else
                profit[i] = profit[i-1];
            printf("; Max Profit so far: %d\n", profit[i]);
        }
        printf("\nList of profits: ");
        for(i=0; i<n+1; i++){
            printf("%d ", profit[i]);
        }

        printf("\nMax Profit: %d\n", profit[n]);

        i=n;
        printf("Selected items are(Unsorted list's elements): \n");
        while(i>0){ //dizi sondan başa olarak gezilerek indis değerleri tespit edilecek.
            if(profit[i-1] == profit[i]){ //değişim olmadıkça geri gidilecek
                i--;
            }
            else{ // değişim varsa geri gidilerek son çakışmayan eleman bulunur
                j = i;
                while(j>0 && ads[i].startTime < ads[j].finish){     // çakışmayan elemana kadar
geri gidilir
          j--;
        }
                printf("%d\t", ads[i].index);
        i=j;
            }
        }
        free(profit);
        free(ads);
```

```
	return 0;
}

void quickSort(struct ad *ads, int first, int last){ //reklamların sırasız listesini, başlangıç ve bitiş indislerini alarak listeyi finish değerine göre sıralar.
	int i; //indisleme için kullanılacaktır.
	int j; //indisleme için kullanılacaktır.
	int pivot; //seçilecek pivot değerinin(dizinin ilk elemanı) tutacak olan değişken.
	struct ad temp; //swap işleminde kullanılacak olan geçici değişken.

	if(first < last){
    pivot = first;
    i = first;
    j = last;

	while(i<j){
	while(ads[i].finish<=ads[pivot].finish && i<last)
	i++;
	while(ads[j].finish > ads[pivot].finish)
	j--;
	if(i<j){
	temp = ads[i];
	ads[i] = ads[j];
	ads[j] = temp;
	}
    }

    temp = ads[pivot];
    ads[pivot] = ads[j];
    ads[j] = temp;

    quickSort(ads,first,j-1);
```

```
        quickSort(ads,j+1,last);


    }
}
```