ALGORİTMA ANALIZI

2. Ödev Raporu

En Yüksek Reklam Kazancını Dinamik Programlama ile Hesaplama

ELİF YAGMUR DURAN

18011071

TABLE OF CONTENTS

PROGRAMIN MANTIĞI

Program hazırlık ve çözüm olmak üzere iki aşamada çalışır.

Hazırlık aşamasında struct içerisinde tutulan reklamlar tablosu initialize edilir. Reklamların yerleri, sorttan sonra karışıklık oluşmaması için harflerle adlandırılmıştır. İlk adımda bitiş saatleri hesaplanır. Daha sonra tablonun struct yapısına özel olarak modifiye edilmiş bir merge sort fonksiyonu kullanılarak bitiş saatlerine göre tablo sort edilir.

Çözüm aşamasında profit tablosu toplam kazanç değerini tutmak için kullanılır. İlk başta 0. Ve 1. Değerler, dinamik programlamanın mantığı sebebiyle, 0 ve ilk reklamın değerleri ile doldurulur. 2. Değerden başlanarak hesaplamalar yapılır. Profit tablosunun gözlerini doldurma için yapılan işlemlerin olduğu her adımda o reklam için kendinden önceki reklamlar ile çakışma olup olmadığı karşılaştırılır. Çakışma olmayan ilk reklama rastlandığında durulur. Daha sonra bu adım içerisinde çakışma olmayan son reklama bu reklamın eklenip eklenemeyeceği belirlenir. Sonuca göre ya bu reklamın eklendiği profit kullanılır, ya da bundan önceki adıma ait profit kullanılır.

Rekürans bağıntısı için öncelikle ilk iki göz ayarlanır:

$$profit[0] = 0;$$

$$profit[1] = table[0].value;$$

Bağıntı:

$$profit[i] = \max\left((profit[k] + value[i]) , profit[i-1] \right)$$

Burada k, i. adım hesaplanırken sıradaki reklam ile çakışmayan son reklamdır. Pratikte yukardaki formül işlenir, fakat gerçekte profit tablosu ve value tablosu indisleri arasında, profit tablosunun ilk gözünün 0 olması gerekliliğinden dolayı bir fark olacağı için kodda rekürans bağıntısı daha farklı şekilde yazılmaktadır.

$$j = i - 1;\ k = i - 2;$$

$$profit[i] = max((profit[k + 1] + table[j].value), profit[i - 1]);$$

Verilen örnekte programın cevabı e, d ve c reklamları olarak göstermesi beklenmektedir.

KULLANILAN FONKSİYONLAR

MERGESORT

Programda kullanılan struct türüne göre özelleştirilmiştir, endtime özelliğine göre sort yapar.

void merge(struct ad table[], int l, int m, int r) { int i, j, k;	Birleştirme fonksiyonu. İşleyeceği tabloyu ve sağ sol ile orta değerlerini alır.

int n1 = m - l + 1;
int n2 = r - m;
struct ad L[n1], R[n2];
for (i = 0; i < n1; i++)
L[i] = table[l + i];
for (j = 0; j < n2; j++)
R[j] = table[m + 1 + j];
	İlk iş olarak değerleri yedekleyecek bir sağ ve sol tablo, yani struct dizisi oluşturur.
	bu tabloların büyüklükleri, sol için, o adımdaki ilk eleman (yani l) den ortadaki elemana kadar olan kısımdır.
	Sağ için ise orta elemandan sonraki eleman ve son elemana kadar olan kısımdır.
	n1 ve n2 ile büyüklükler hesaplanır ve elemanlar yedeklenir.

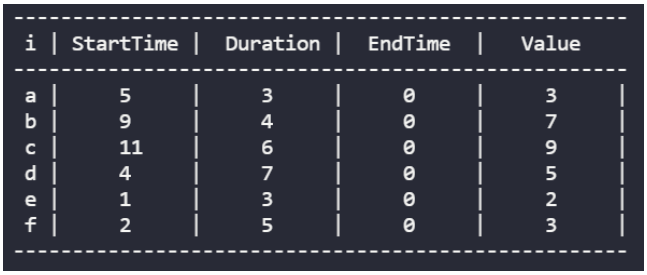
i = 0; j = 0; k = l;
while (i < n1 && j < n2) {
if (L[i].endtime <= R[j].endtime) {
table[k] = L[i];
i++;
} else {
table[k] = R[j];
j++;
}
k++;
}
	i ve j sağ ve sol dizileri gezmeye yarar, k elemanı ise üzerinde çalıştığımız struct dizisindeki yeri tutar.
	k'nın gösterdiği elemana yazmak için sol diziden i'nin gösterdiği ve sağ diziden j'nin gösterdiği karşılaştırılır. Hangisi daha küçükse o ana array'e yazılır ve indisi artırılır. Karşılaştırmadan sonra k artırılır.

while (i < n1) {
table[k] = L[i];
i++;
k++;
}
while (j < n2) {
table[k] = R[j];
j++;
k++;
}
}
	Bu iki adım sağ ya da sol diziden biri bittiğinde öbüründe kalan elemanları sırayla kalan yerlere doldurur.

void mergeSort(struct ad table[], int l, int r)
{
if (l < r) {
int m = l + (r - l) / 2;
mergeSort(table, l, m);
mergeSort(table, m + 1, r);
merge(table, l, m, r);
}
}
	Standart mergesort fonksiyonu.
	Right left i geçmediği sürece ikiye bölerek devam eder. Middle right ve left in ortasındaki sayıdır.
	Önce sol taraflar çağırılır, daha sonra sağ taraflar. Bu ikisi bittikten sonra iki taraf yukarıda açıklandığı gibi merge edilir.

PRİNT

Reklamlar tablosunu gözlemleyebilmek için yazılmıştır.

void print(int n, struct ad table[]) {
int i, j;
printf("\n----------");
printf("\n i | StartTime | Duration | EndTime | Value\n----------");
for (i = 0; i < n; i++) {
printf("\n %c |", table[i].index);
	İşlem sonucunda şu görüntü amaçlanmaktadır:
	 

```
        if (table[i].start < 10) {
            printf("     %d     |", table[i].start);
        } else {
            printf("     %d    |", table[i].start);
        }
        printf("     %d     |", table[i].duration);
        if (table[i].endtime < 10) {
            printf("     %d     |",
table[i].endtime);
        } else {
            printf("     %d    |",
table[i].endtime);
        }
        printf("     %d     |", table[i].value);
    }
    printf("\n----------------------------------------
------------\n");
}
```

Bunun için 10'dan büyük değer olduğu bilinen sütunlarda kaymayı engellemek için basamak sayısı kontrolü yapılmıştır.
EndTime doldurulup sort yapıldıktan sonra şu görüntü elde edilir:

```
sorted table:
-------------------------------------------------
 i | StartTime |  Duration |  EndTime  |   Value
-------------------------------------------------
 e |     1     |     3     |     4     |     2     |
 f |     2     |     5     |     7     |     3     |
 a |     5     |     3     |     8     |     3     |
 d |     4     |     7     |     11    |     5     |
 b |     9     |     4     |     13    |     7     |
 c |     11    |     6     |     17    |     9     |
-------------------------------------------------
```

KULLANILAN DEĞİŞKENLER

struct ad {
  char index;
  int start;
  int duration;
  int endtime;
  int value;
};
	Reklamların bilgilerini tutan struct. İndex kısmı reklamlar sort edildikten sonra karışmasın diye harfe çevrilmiştir. Main içerisinde bu tipten bir table[] struct array'i yaratılır.

struct prft {
  int revenue;
  char *ads;
};
	Profit tablosunu tutan struct. Revenue o adımdaki toplam kazancı, ads seçilen reklam harflerinin string ini temsil eder.
	Main içerisinde bu tipten bir profit[] struct array'i yaratılır.

int i, j, k; int n = 6;	Main içi değişkenleri. i, j ve k indisler içindir. n boyutu temsil eder.

char temp[] = "-";	temp değişkeni string fonksiyonlarında yardımcı olmak içindir.
struct prft profit[7]; struct ad table[6] = { {'a', 5, 3, 0, 3}, {'b', 9, 4, 0, 7}, {'c', 11, 6, 0, 9}, {'d', 4, 7, 0, 5}, {'e', 1, 3, 0, 2}, {'f', 2, 5, 0, 3}};	Struct'ların initialize edilmesi

MAİN

HAZIRLIK AŞAMASI

printf("advertisements table:");
print(n, table);
printf("\nindex names have been changed to characters in order to avoid confusion.");
	Tablonun ilk hali gösterilir, açıklama yapılır.

printf("\ncalculating the ending hours...\n");
for (i = 0; i < n; i++) {
table[i].endtime = table[i].start + table[i].duration;
}
	Reklamların bitiş saatleri hesaplanır.

printf("sorting...\n");
mergeSort(table, 0, n - 1);
printf("\nsorted table:");
print(n, table);
	Tablo mergesort fonksiyonuna sokulur. Kullanıcıya tekrar gösterildikten sonra işlemler için hazırdır.

ÇÖZÜM AŞAMASI

for (i = 0; i < 7; i++) {
profit[i].ads = strdup(temp);
}
	Profit struct array'inde sıfırıncı ve birinci göz özel durum olduğu için önden doldurulmalıdır. Ayrıca her stringin ilk gözüne karışıklığı önlemek ve initialization yapmış omak için strdup ile "-" koyulur.

profit[0].revenue = 0;
profit[1].revenue = table[0].value;
temp[0] = table[0].index;
strcat(profit[1].ads, temp);
	İlk göz için toplam revenue 0.
	İkinci göz için ilk reklama eşit.
	İlk gözde harfler yok. "-" koyma işlemi buna önlem olarak yapıldı.
	İkinci gözde harf ilk reklamın harfi.

for (i = 2; i < 7 + 1; i++) {
j = i - 1;
k = i - 2;
	Döngüye i değişkeni profit[2] den başlamalıdır.
	k ve j değişkenleri table üzerinde gezecektir. Bu durumda j, i ile aynı reklamı, k ise j den bir önceki reklamı gösterir.

while ((k >= 0) && (table[k].endtime > table[j].start)) {
k--;
}
	k geriye doğru giderken j ile çakışma olmayan ilk reklam aranır. Bu durumda geride olan (k'nın gösterdiği) reklamın bitiş saati, önde olan (j'nin gösterdiği) reklamın başlangıç saatinden küçük olmalıdır. Böyle bir reklam bulunana kadar k azaltılır. Eğer k -1 değerine ulaşırsa çakışmayan reklam yoktur. Bu durumda profit[0] da reklam olmaması işimize yarar

//max{(profit[k + 1].revenue + table[j].value), profit[i - 1].revenue}	Table[k] nın gösterdiği reklam, profit[k+1] in gösterdiği reklamdır. Bu durumda ya k nın gösterdiği yerde bulunan en son çakışmayan reklam ve onun toplam revenue su üzerine j deki reklam eklenecektir, ya da i. adımdan önceki adımdaki reklam kombinasyonu kullanılacaktır. İki kere if anlaşılma kolaylığı için yazılmıştır.

if ((profit[k + 1].revenue + table[j].value) > profit[i - 1].revenue) {
profit[i].revenue = profit[k + 1].revenue + table[j].value;
strncat(profit[i].ads, &table[j].index, 1);
}
	j. reklamı eklemeyi seçersek;
	i. adımdaki toplam revenueyu, knın gösterdiği reklamın toplam kazancına j yi ekleyerek buluruz.
	Ayrıca strncat fonksiyonu ile k+1 de şimdiye kadar toplanmış olan reklam harflerine j reklamının harfini ekleriz.

if ((profit[k + 1].revenue + table[j].value) < profit[i - 1].revenue) {
profit[i].revenue = profit[i - 1].revenue;
strncpy(profit[i].ads, &profit[i-1].ads, 1);
}
}
	j. reklamı eklemeyi seçmezsek;
	i. adımdaki toplam revenue, i-1 adımdaki revenuedur.
	Ayrıca strncpy fonksiyonu i-1. adımdaki harfleri tamamen buraya kopyalarız.

SONUÇ

printf("\nbiggest possible revenue is:%d\nads to be used are:%s", profit[6].revenue, profit[6].ads); printf("\ngoodbye.");	Gösterilecek cevaplar 16 ve e,d,c dir.