



Algoritma Analizi-2

Madenci Problemi

Versiyon 1.0

Prepared by Onur Demir

Grup 2-Ödev 2

onur.demir2@std.yildiz.edu.tr

04.11.2022

Video Linki : <https://youtu.be/8o3OHh-ld4M>

İçerik Tablosu

İçerik Tablosu.....	ii
Revizyon Tarihi.....	ii
1. Giriş.....	1
1.1 Problem.....	1
1.2 Problem Tanımı	1
2. Yöntem ve Analiz	1-4
2.1 Brute Force Yaklaşımı	1
2.2 Brute Force Sözde Kodu.....	2
2.3 Brute Force Matematiksel Analizi	2
2.4 Divide and Conquer Yaklaşımı	3
2.5 Divide and Conquer Sözüde Kod Analizi	3-4
3. Uygulama ve Sonuç.....	4
3.1 Test Caseler ve Ekran Çıktıları	4-7

Revizyon Tarihi

İsim	Tarih	Değişim Sebebi	Versiyon
Onur Demir	04.11.2022	Max değeri ve aralığı extend_array metodu ile bulan.	1.0
Onur Demir	05.11.2022	Test Caselere göre düzeltmeler yapıldı ve extend_array metoduna gereksinim kaldırıldı.	1.1

Video Linki : <https://youtu.be/8o3OHh-ld4M>

1. Giriş

1.1 Problem

Bir madenci değerli taşlar ararken kazancını en yüksekte tutabilmek için kazı işlemini yapacağı bölgeyi özel olarak seçmektedir. Maden bölgesinin bloklardan oluştuğu bilinmektedir. Problemi basitleştirmek adına blokların 1 boyutlu düzlemde yan yana yer aldığı kabul edilmektedir. Her blok için yapılacak kazının sağlayacağı kazanç (pozitif veya negative) a_i olarak bilindiğine göre madencinin en büyük kazancı sağlayacağı kesintisiz bloklar bütünü ve kazanç miktarını

- 1.Brute force yaklaşımı ile
- 2.Divide and Conquer yaklaşımı ile bulan algoritmayı tasarlayıp C dilinde kodunu yazınız.
- 3.Her iki yaklaşım için sözde kod yazıp algoritmanızın karmaşıklığını matematiksel analiz ile ifade ediniz

Blok	0	1	2	3	4	5	6	7	8	9	10
Kazanç	8	-30	36	2	-6	52	8	-1	-11	10	4

1.2 Problem Tanımı

Bir dizi içerisindeki maksimum değer toplamı alt diziyi sormaktadır.Bunun için üç tür yaklaşım uygulanabilir. İlk olarak maksimum alt dizi exhaustive search ile bulunabilir. Bir dizinin tüm alt dizilerine bakmayı sıra ile yapabiliriz. Her index için kendisinden sonraki aralıkları arayabiliriz. Böylelikle Brute-force yaklaşımı oluşur. Divide and Conquer yaklaşımı için prefix ve suffix toplamlarını önceden alıp. En büyük değeri divide and conquer yaklaşımı ile arayabiliriz.

2. Yöntem

2.1 Brute-force Yaklaşımı

Brute-force yaklaşımı için her bir bloktaki değerlerin diğer bloklara kadar olan toplamı incelenebilir ve maksimum değer alınıp diğer bloğa geçilebilir. Aşağıda Problem kısmında verilen dizi için çözümü incelersek;

i

8	-30	36	2	-6	52	8	-1	-11	10	4
---	-----	----	---	----	----	---	----	-----	----	---

j
max:8 g_max:8

i

8	-30	36	2	-6	52	8	-1	-11	10	4
---	-----	----	---	----	----	---	----	-----	----	---

j
max:-22 g_max:8

i

8	-30	36	2	-6	52	8	-1	-11	10	4
---	-----	----	---	----	----	---	----	-----	----	---

j
max:14 g_max:14 Text

i

8	-30	36	2	-6	52	8	-1	-11	10	4
---	-----	----	---	----	----	---	----	-----	----	---

j
max:72 g_max:72

i sağa hareket eder

i

8	-30	36	2	-6	52	8	-1	-11	10	4
---	-----	----	---	----	----	---	----	-----	----	---

j
max:-30 g_max:72

i

8	-30	36	2	-6	52	8	-1	-11	10	4
---	-----	----	---	----	----	---	----	-----	----	---

j
max:6 g_max:72

i

8	-30	36	2	-6	52	8	-1	-11	10	4
---	-----	----	---	----	----	---	----	-----	----	---

j
max:94 g_max:94 en büyük değeri bulur.

i

8	-30	36	2	-6	52	8	-1	-11	10	4
---	-----	----	---	----	----	---	----	-----	----	---

j
max:4 g_max:94 bitirme durumu ile biter.

her bir iterasyonda en büyük local maksimum değeri bulunur ve global maksimum ile karşılaştırılır.

2.2 Brute-force Söзде Kodu

```

Algorithm BruteForceMaxIntervalSearch(A[0....n-1])
  tmpMax ← rightIdx ← max ← 0
  for i ← 0 to n-1 do
    for j ← i to n-1 do
      tmpMax += A[j]
      if tmpMax > max:
        leftIdx ← i
        rightIdx ← j
        max ← tmpMax
  return leftIdx, rightIdx, max

```

2.3 Brute Force Matematiksel Analizi

Algorithm BruteForceMaxIntervalSearch(A[0....n-1])	cost
tmpMax ← rightIdx ← max ← 0	3
for i ← 0 to n-1 do	(n+1)
for j ← i to n-1 do	n*(n+1)
tmpMax += A[j]	2*n*n
if tmpMax > max:	n*n
leftIdx ← i	n*n
rightIdx ← j	n*n
max ← tmpMax	n*n
return leftIdx, rightIdx, max	7n ² + 2n + 3

Algoritma 2.1 de belirtildiği gibi erken durma durumu olmadan gerçekleştirileceği yani tüm for loop ları boyunca en iyi ve n kötü durumda da dönmek zorunda kalacağından için $\theta(n^2)$ complexity e sahip olacaktır.

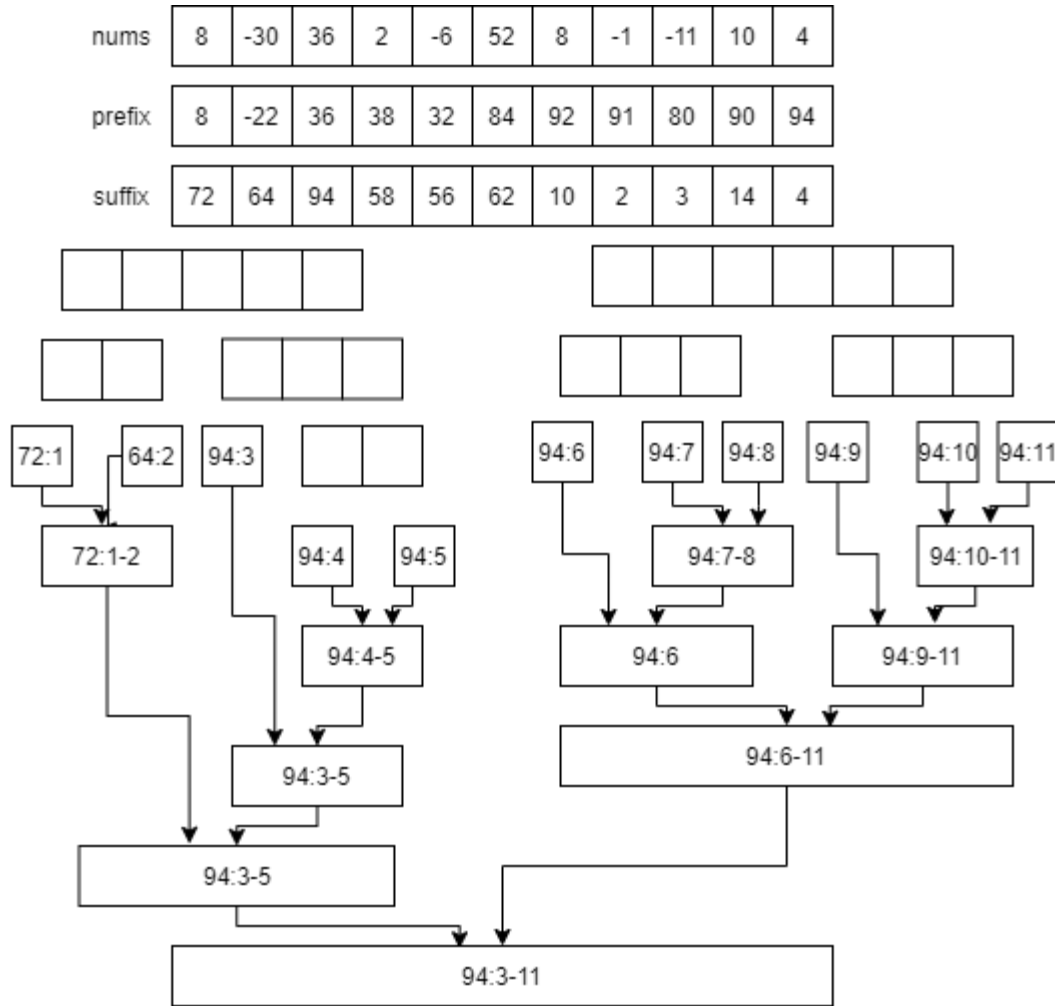
2.4 Divide and Conquer Yaklaşımı

Divide and Conquer yaklaşımı için prefix ve suffix toplamları kullanılmıştır ve prefix ve suffix içerisinde divide and conquer ile arama yapılmıştır.

Bu yaklaşımın ilgili örnek veri için çözümü aşağıdaki gibidir.

Sadece Prefix ya da sadece suffix kullanarak şu şekilde yapılır.

Diğer durumda prefix, suffix toplamla şu şekilde yapılır.



En büyük alan bulunur daha sonra 3.sayısının sağ ve solu prefix ve suffix kullanılarak bulunur.

2.5 Divide and Conquer Sözde Kod Analizi

Algorithm ExtendTuple(Tuple, Prefix, Suffix, arrSize):

“Tuple : structure consist of left_idx, right_idx and area of index”

while(Prefix[tuple.leftIdx] > 0 AND ~tuple.leftIdx)

 Tuple.leftIdx-=1

while(Suffix[tuple.leftIdx] > 0 AND tuple.rightIdx < arrSize)

 Tuple.rightIdx+=1

Tuple.leftIdx+=1; Tuple.rightIdx-=1;

Algorithm PreComputePrefixSuffix(Prefix, Suffix, Array):

 Prefix = Suffix = Array

 for(i = 1 ; i < numSize ; i++)

 Prefix[i] += max(0, Prefix[i-1])

 for(i = numSize ; ~i ; i--)

 Suffix[i] += max(0, Suffix[i+1])

 return Prefix, Suffix

// Aşağıdaki pseudo kod sadece max değerini bulmak için doğru çalışır. Test Caseler kullanılarak düzeltilmiştir.

Algorithm DivideAndConquerMaxInterval($A[0 \dots n-1]$, L, R)

mid $\leftarrow (L+R)/2$

if $L = R$:

maxTuple.leftIdx = L

maxTuple.rightIdx=R

maxTuple.area = $A[L] // A[mid] // pre[mid] + suff[mid+1]$

return maxTuple

leftTuple \leftarrow DivideAndConquerMaxInterval(A , L, mid)

rightTuple \leftarrow DivideAndConquerMaxInterval(A , mid+1, R)

state \leftarrow findMaxWhich(leftTuple.area, rightTuple.area, $pre[mid] + suff[mid+1]$)

if state == LEFT:

return leftTuple

else if state == RIGHT:

return rightTuple

else

maxTuple.area $\leftarrow pre[mid]+suff[mid+1]$

if leftTuple.area > rightTuple.area:

maxTuple.leftIdx = leftTuple.leftIdx

maxTuple.rightIdx = leftTuple.rightIdx

return maxTuple

else:

maxTuple.leftIdx \leftarrow rightTuple.leftIdx

maxTuple.rightIdx \leftarrow rightTuple.rightIdx

return MaxTuple

İlk iki fonksiyon ExtendTuple, PreComputePrefixSuffix karmaşıklığı n mertebesinde.

Extend Tuple en iyi durumda $\Omega(1)$ da sonuçlanabilir.(iki negative sayı sağ ve solunda ise)

En kötü durumda $O(n)$ olmaktadır(tüm while döngüsünde gezer.).

PreComputePrefixSuffix karmaşıklığı $\Theta(n)$ şeklindedir.(tek for içerisinde döngü ve erken bitme durumu yoktur.)

Divide and Conquer algoritması için Master theorem ile aşağıdaki rekürans bağıntısı çözülmelidir. 2 alt problem vardır ve birleşme sırasında iki alt problem çözülmektedir ve çözülürken $O(1)$ süre harcanmaktadır(Bu temel işlem maksimum tuple bulmak için karşılaştırmadır.).

$$C(n) = 2C(n/2) + 1$$

$$a=2$$

$$b=2$$

$$c=0$$

$$a > b^d$$

$\Theta(N)$ olarak bulunur bu yüzden toplam algoritma karmaşıklığı $\Theta(N)$ denilebilir.

Aşağıda algoritmanın uç durumlarının test edilmiş halinin pseudocode u bulunmaktadır.

Algorithm findMaxDividedAreaOptim($A[0 \dots n-1]$, int L, int R) {

$mid = (L + R) / 2$;

 if($L == R$)

$max_tuple.left_idx \leftarrow L$;

$max_tuple.right_idx \leftarrow R$;

 if($mid == n-1$) :

$max_tuple \rightarrow area \leftarrow pre[mid]$;

 else if($mid == 0$) :

$max_tuple \rightarrow area \leftarrow suf[mid]$;

 else :

 if ($pre[mid-1] + suf[mid] > pre[mid] + suf[mid+1]$)

$max_tuple \rightarrow area \leftarrow pre[mid-1] + suf[mid]$;

 else:

$max_tuple \rightarrow area \leftarrow pre[mid] + suf[mid+1]$;

 if ($max_tuple \rightarrow area < 0 \parallel A[mid] > max_tuple \rightarrow area$) :

$max_tuple \rightarrow area \leftarrow A[mid]$;

 return max_tuple ;

$left_tuple \leftarrow findMaxDividedAreaOptim(A, L, mid, size)$;

$right_tuple \leftarrow findMaxDividedAreaOptim(A, mid+1, R, size)$;

 if($left_tuple \rightarrow area > right_tuple \rightarrow area$) :

$max_tuple.left_idx \leftarrow left_tuple.left_idx$;

$max_tuple.right_idx \leftarrow left_tuple.right_idx$;

$max_tuple.area = left_tuple.area$;

 free($left_tuple$);

 return max_tuple ;

 else if ($left_tuple \rightarrow area < right_tuple \rightarrow area$):

$max_tuple \rightarrow left_idx \leftarrow right_tuple \rightarrow left_idx$;

$max_tuple \rightarrow right_idx \leftarrow right_tuple \rightarrow right_idx$;

$max_tuple \rightarrow area \leftarrow right_tuple \rightarrow area$;

 free($right_tuple$);

 return max_tuple ;

 else :

 if($A[mid] > pre[mid] + suf[mid+1]$):

$max_tuple \rightarrow left_idx \leftarrow left_tuple \rightarrow left_idx$;

$max_tuple \rightarrow right_idx \leftarrow right_tuple \rightarrow right_idx$;

$max_tuple \rightarrow area \leftarrow A[mid]$;

 else :

 if ($pre[mid] + suf[mid+1] < right_tuple \rightarrow area$) :

$max_tuple \rightarrow left_idx \leftarrow left_tuple \rightarrow left_idx$;

$max_tuple \rightarrow right_idx \leftarrow left_tuple \rightarrow right_idx$;

$max_tuple \rightarrow area \leftarrow right_tuple \rightarrow area$;

 else :

$max_tuple \rightarrow left_idx \leftarrow left_tuple \rightarrow left_idx$;

$max_tuple \rightarrow right_idx \leftarrow right_tuple \rightarrow right_idx$;

$max_tuple \rightarrow area \leftarrow pre[mid] + suf[mid+1]$;

 return max_tuple ;

3. Uygulama

3.1 Test Caseler ve Ekran Çıktıları

Test durumu için rastgele 20 test oluşturacak kod yazılmıştır. Bu testler brute force kodu her zaman doğru çalışacağı varsayılarak divide and conquer bazlı algoritmanın doğruluğu test edilmiştir.

Test durumları için.

```

filiz@DESKTOP-POT7LBL ➤ cpp_algo ➤ .\18011078.exe
TEST CASE 0 :
24:-11:19:-10:-12:
TEST CASE 1 :
12:-3:-11:5:-22:19:23:-6:11:15:-15:-12:9:-4:-3:0:
TEST CASE 2 :
14:21:-19:15:22:7:-23:20:11:20:3:-5:-7:15:-4:13:
TEST CASE 3 :
1:-1:-21:2:9:-7:13:3:19:-25:11:24:-9:2:-12:4:13:9:
TEST CASE 4 :
-7:-10:3:-9:-18:-8:22:-18:-14:-21:11:10:8:5:-16:8:23:-22:9:22:
TEST CASE 5 :
20:5:8:4:17:5:0:-3:-23:-12:20:2:
TEST CASE 6 :
5:20:
TEST CASE 7 :
11:15:-11:
TEST CASE 8 :
18:-5:2:12:3:-3:-21:-20:-3:21:5:18:16:10:
TEST CASE 9 :
-25:24:-15:20:-2:1:-12:-23:8:-11:3:-1:13:-10:-24:-5:
TEST CASE 10 :
13:-2:21:13:
TEST CASE 11 :
22:-9:8:23:-25:-16:-24:4:
TEST CASE 12 :
-13:-24:-6:23:22:-10:-7:
TEST CASE 13 :
-22:-10:-14:-14:-10:5:-11:0:-18:-11:15:1:12:-8:-3:
TEST CASE 14 :
-11:11:19:22:6:-3:19:6:-2:-20:-10:7:5:4:
TEST CASE 15 :
-3:-13:24:8:12:-5:13:-2:-10:-17:18:
TEST CASE 16 :
18:-16:-21:
TEST CASE 17 :
16:-5:-8:-19:-15:-10:7:1:4:-14:-14:22:18:8:-13:-10:8:0:
TEST CASE 18 :
23:-6:22:-9:1:19:9:-18:-14:-4:-8:12:-11:13:-19:-6:
TEST CASE 19 :
23:
TEST CASE 20 :
11:-8:1:-13:9:-22:-21:
TEST CASE 0 bf solution : 0-2:32
TEST CASE 0 dc solution : 0-2:32
TEST CASE 1 bf solution : 5-9:62
TEST CASE 1 dc solution : 5-9:62
TEST CASE 2 bf solution : 0-15:103
TEST CASE 2 dc solution : 0-15:103
TEST CASE 3 bf solution : 3-17:56

```

Sonuçlar şu şekildedir;

```
TEST CASE 20 :
1:-8:1:-13:9:-22:-21:
TEST CASE 0 bf solution : 0-2:32
TEST CASE 0 dc solution : 0-2:32
TEST CASE 1 bf solution : 5-9:62
TEST CASE 1 dc solution : 5-9:62
TEST CASE 2 bf solution : 0-15:103
TEST CASE 2 dc solution : 0-15:103
TEST CASE 3 bf solution : 3-17:56
TEST CASE 3 dc solution : 3-17:56
TEST CASE 4 bf solution : 10-19:58
TEST CASE 4 dc solution : 10-19:58
TEST CASE 5 bf solution : 0-5:59
TEST CASE 5 dc solution : 0-6:59
TEST CASE 6 bf solution : 0-1:25
TEST CASE 6 dc solution : 0-1:25
TEST CASE 7 bf solution : 0-1:26
TEST CASE 7 dc solution : 0-1:26
TEST CASE 8 bf solution : 9-13:70
TEST CASE 8 dc solution : 9-13:70
TEST CASE 9 bf solution : 1-3:29
TEST CASE 9 dc solution : 1-3:29
TEST CASE 10 bf solution : 0-3:45
TEST CASE 10 dc solution : 0-3:45
TEST CASE 11 bf solution : 0-3:44
TEST CASE 11 dc solution : 0-3:44
TEST CASE 12 bf solution : 3-4:45
TEST CASE 12 dc solution : 3-4:45
TEST CASE 13 bf solution : 10-12:28
TEST CASE 13 dc solution : 10-12:28
TEST CASE 14 bf solution : 1-7:80
TEST CASE 14 dc solution : 1-7:80
TEST CASE 15 bf solution : 2-6:52
TEST CASE 15 dc solution : 2-6:52
TEST CASE 16 bf solution : 0-0:18
TEST CASE 16 dc solution : 0-0:18
TEST CASE 17 bf solution : 11-13:48
TEST CASE 17 dc solution : 11-13:48
TEST CASE 18 bf solution : 0-6:59
TEST CASE 18 dc solution : 0-6:59
TEST CASE 19 bf solution : 0-0:23
TEST CASE 19 dc solution : 0-0:23
TEST CASE 20 bf solution : 0-0:11
TEST CASE 20 dc solution : 0-0:11
INDIVIDUAL TEST CASE 21 bf solution : 7-8:23
INDIVIDUAL TEST CASE 21 dc solution : 7-8:23
filiz@DESKTOP-POT7LBL > ■ cpp_algo
```

(%d-%d:%d)(2-10:94), sırasıyla sol index sağ index ve kazınması sonucunda maksimum elde edilecek kazanç şeklindedir.

Video Linki: <https://youtu.be/8o3OHh-ld4M>