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-Id4M

İçerik Tablosu

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 bütününü 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

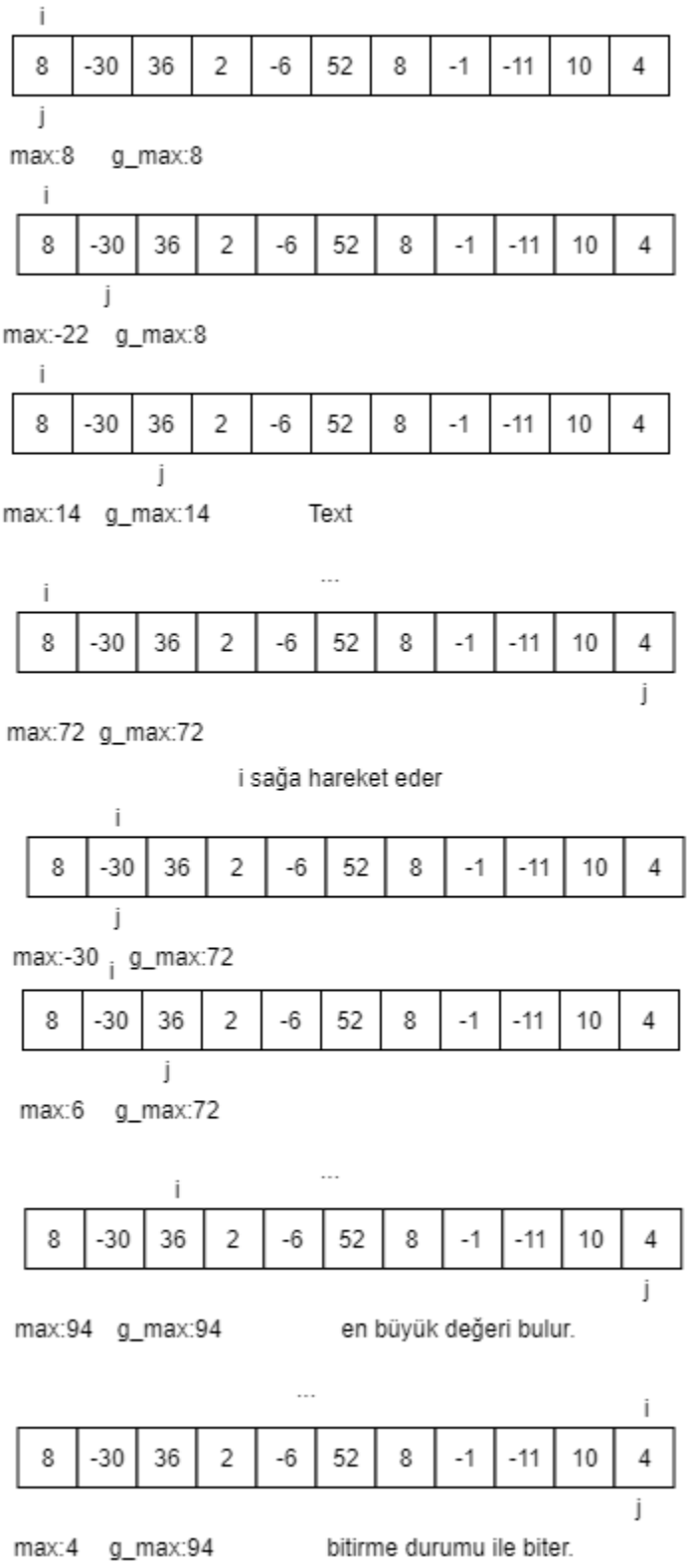
1.2 Problem Tanımı

Bir dizi içerisindeki maksimum değer toplamlı 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 toplamları incelenebilir ve maksimum değer alınıp diğer bloğa geçilebilir.
Aşağıda Problem kısmında verilen dizi için çözümü incelersek;



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

2.2 Brute-force Sözde 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	$7n^2 + 2n + 3$

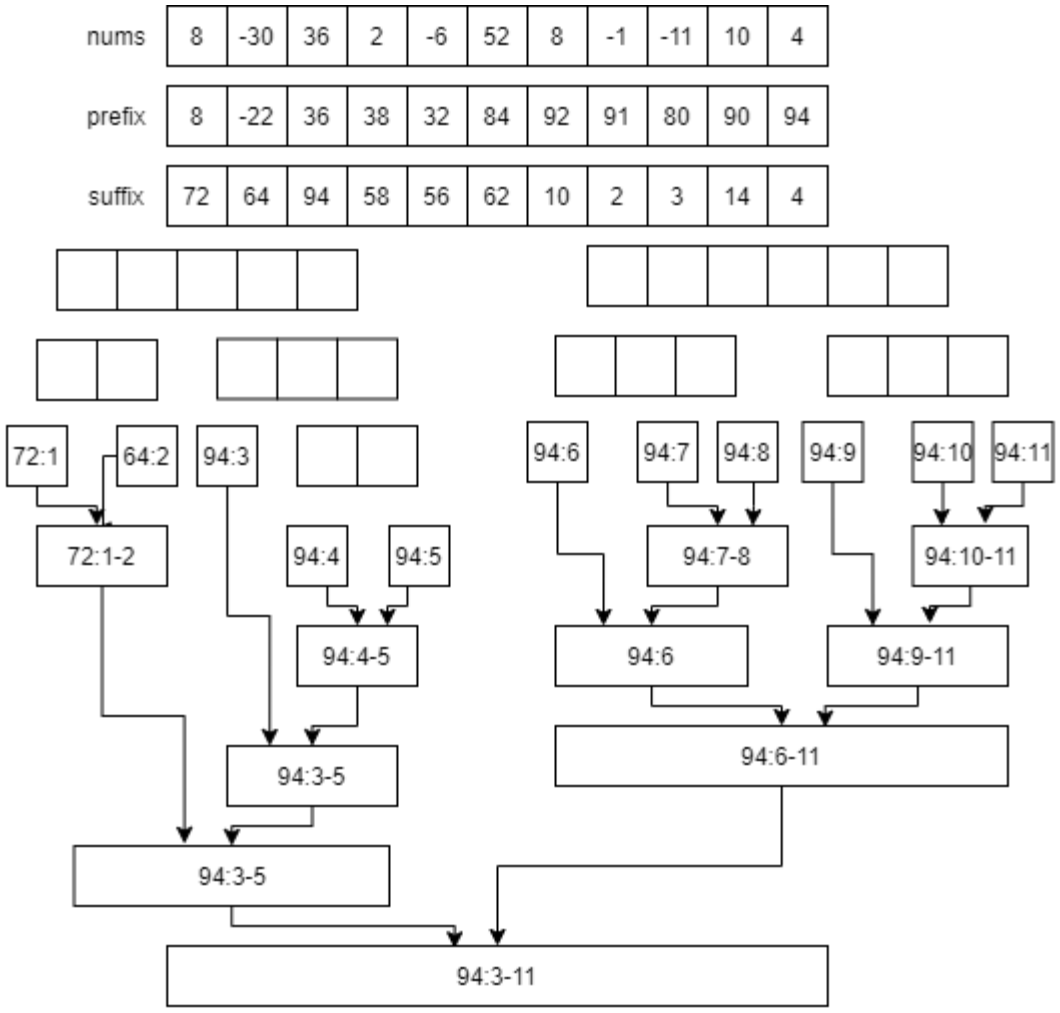
Algoritma 2.1 de belirtildiği gibi erken durma durumu olmadan gerçekleneceği yani tüm for loop ları boyunca en iyi vee n kötü durumda da dönmek zorunda kalacağından için $\theta(n^2)$ complexity e sahip olacaktır.

.

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 sağı 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 = numsSize ; ~i ; i--)
    Suffix[i] += max(0, Suffix[i+1])
  return Prefix, Suffix
```

```
// Aşağıdaki pseudo kod sadece max değerini bumak için doğru çalışır. Test Caseler kullanılarak düzeltilmiştir.
Algorithm DivideAndConquerMaxInterval(A[0…n-1], L, R)
  mid ← (L+R)/2
  if L == R:
    maxTuple.leftIdx = L
    maxTuple.rightIdx=R
    maxTuple.area = A[L] // A[mid] // pre[mid] + suf[mid+1]
    return maxTuple

 leftTuple ← DivideAndConquerMaxInterval(A, L, mid)
 rightTuple ← DivideAndConquerMaxInterval(A, mid+1, R)

 state ← findMaxWhich( leftTuple.area, rightTuple.area, pre[mid] + suf[mid+1]

if state == LEFT:
  return leftTuple
else if state == RIGHT:
  return rightTuple
else
  maxTuple.area ←pre[mid]+suff[mid+1]
  if leftTuple.area > rightTuple.area:
    maxTuple.leftIdx = leftTuple.leftIdx
    maxTuple.rightIdx = leftTuple.rightIdx
    return maxTuple
  else:
    maxTuple.leftIdx ←rightTuple.leftIdx
    maxTuple.rightIdx ← rightTuple.rightIdx
    return MaxTuple
```

İlk iki fonksiyon ExtendTuple, PreComputePrefixSuffix karmaşıklığı n mertebesindedir.

Extend Tuple en iyi durumda $\Omega(1)$ da sonuçlanabilir.(iki negative sayı sağı 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 aşaıdaki rekurans 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.

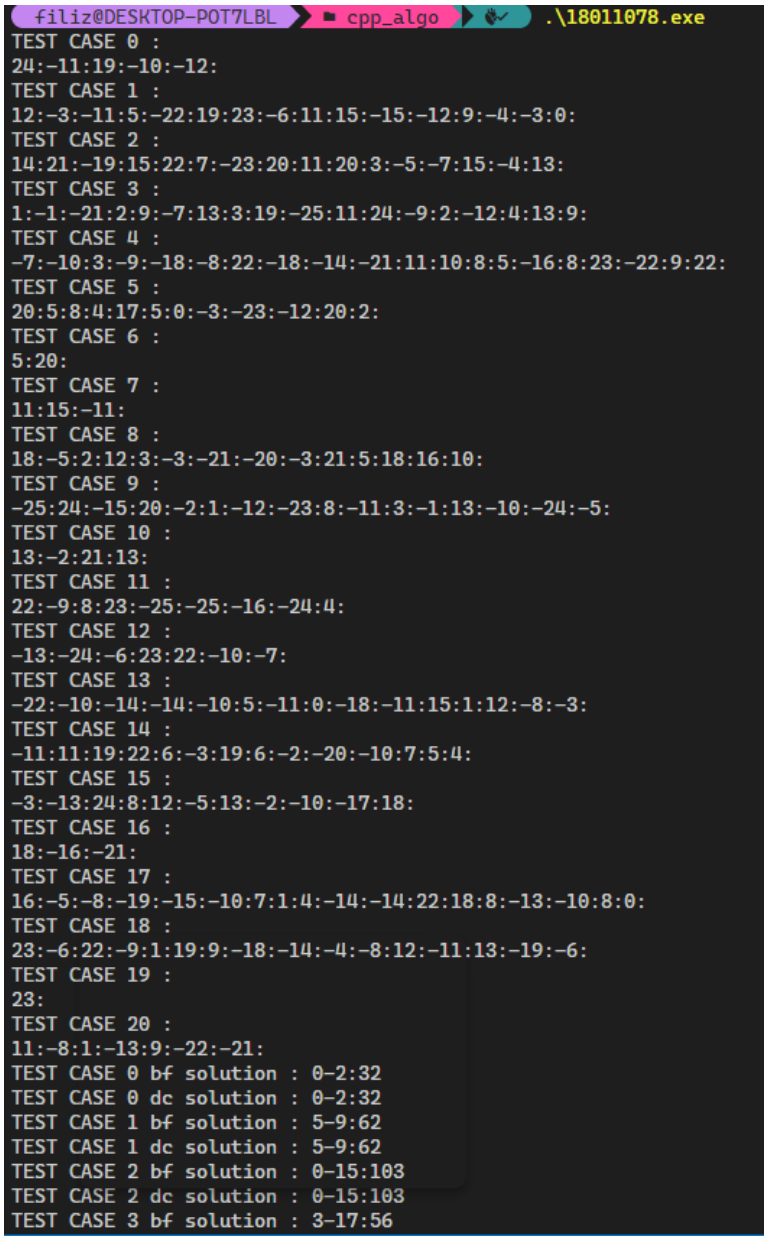
```
Algorithm findMaxDividedAreaOptim(A[0…n-1], int L, int R) {
    mid = (L + R) / 2;
    if(L == R)
        max_tuple.left_idx ← L;
        max_tuple.right_idx ← R;
        if(mid == n-1) :
            max_tuple->area ← pre[mid];
        else if(mid==0) :
            max_tuple->area ← suf[mid];
        else :
            if (pre[mid-1] + suf[mid] > pre[mid] + suf[mid+1]  )
                max_tuple->area ← pre[mid-1] + suf[mid];
            else:
                max_tuple->area ← pre[mid] + suf[mid+1];
        if (max_tuple->area < 0 || A[mid] > max_tuple->area) :
            max_tuple->area ← A[mid];
        return max_tuple;

    left_tuple ← findMaxDividedAreaOptim(A, L, mid, size);
    right_tuple ← findMaxDividedAreaOptim(A, mid+1, R, size);

    if(left_tuple->area > right_tuple->area) :
        max_tuple.left_idx ← left_tuple.left_idx;
        max_tuple.right_idx = left_tuple.right_idx;
        max_tuple.area =  left_tuple.area;
        free(left_tuple);
        return max_tuple;
    else if (left_tuple->area < right_tuple->area):
        max_tuple->left_idx ← right_tuple->left_idx;
        max_tuple->right_idx ← right_tuple->right_idx;
        max_tuple->area ←  right_tuple->area;
        free(right_tuple);
        return max_tuple;
    else :
        if(A[mid] > pre[mid] + suf[mid+1]):
            max_tuple->left_idx←left_tuple->left_idx;
            max_tuple->right_idx←right_tuple->right_idx;
            max_tuple->area ← A[mid];
        else :
            if( pre[mid] + suf[mid+1] < right_tuple->area) :
                max_tuple->left_idx←left_tuple->left_idx;
                max_tuple->right_idx←left_tuple->right_idx;
                max_tuple->area ←  right_tuple->area;
            else :
                max_tuple->left_idx←left_tuple->left_idx;
                max_tuple->right_idx←right_tuple->right_idx;
                max_tuple->area ←  pre[mid] + suf[mid+1];



        return max_tuple;
```

3. Uygulama

3.1 Test Caseler ve Ekran Cıktıları

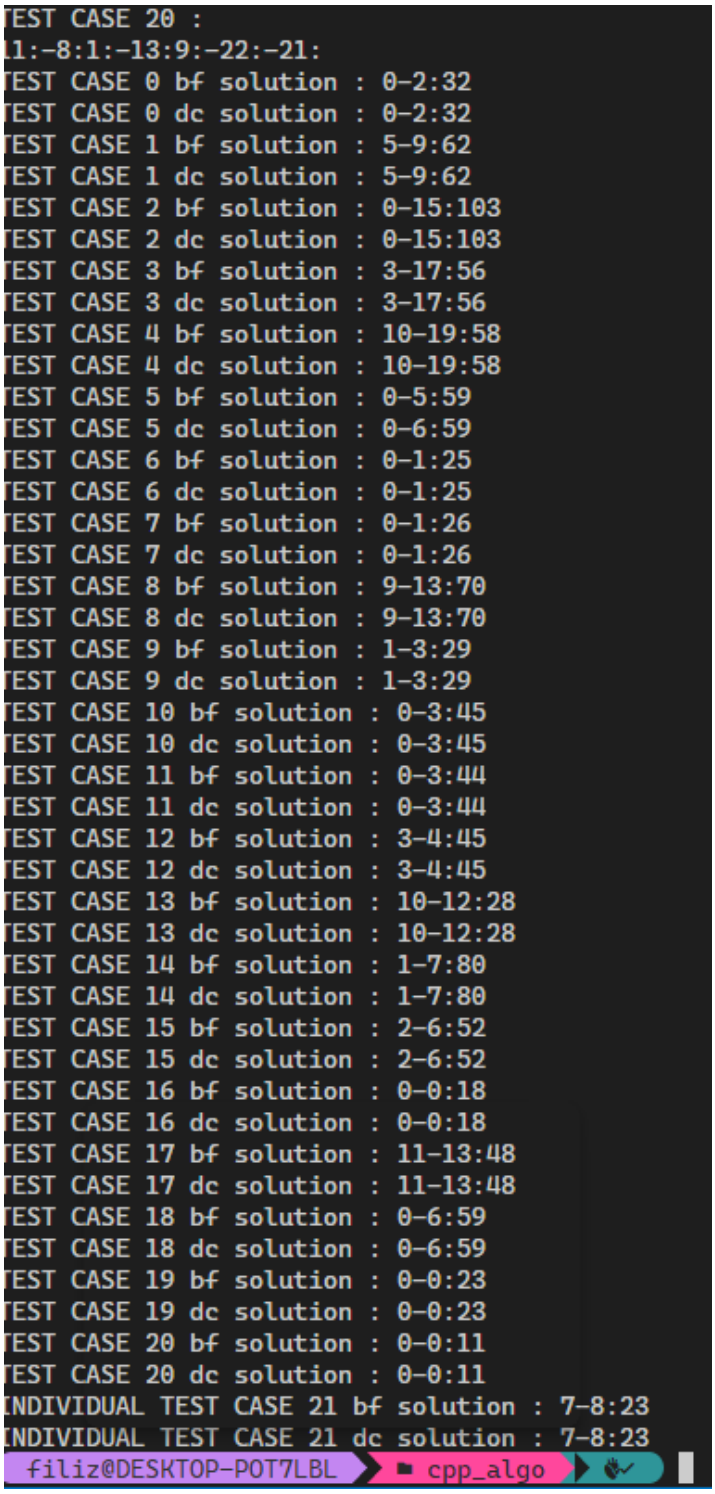
Test durumu için rastgele 20 test oluşturacak kod yazılmıştır. Bu testler brute force kodu her zaman doğru çalışacağı varsayılarak divide and conquer bazlı algoritmanın doğruluğu test edilmiştir.

Test durumları için.



,

Sonuçlar şu şekildedir;



(%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