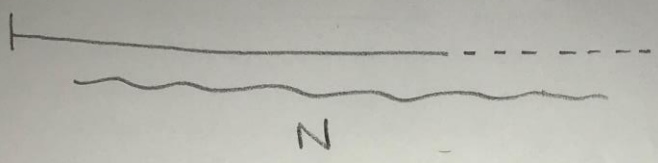
ALGORİTMALAR FİNAL ÖDEVİ KISIM-2 BERK BARIŞ KARA - 18253007

GRUP OLARAK YAPILMAMIŞTIR

Rod Cutting Problem (Çubuk kesme Problemi)

Problemin tanımı;



∝N uzunluğunda sonsuza giden bir çubuğumuz var.

∝Bu çubuğu parçalara bölerek maksimum kârı elde edecek şekil satmak istiyoruz.

∝ Yani maksimum kâr edecek kombinasyonu arıyoruz.

Uzunluk	1	2	3	4	5	6	7	8
Fiyat	1	5	8	9	10	17	17	20

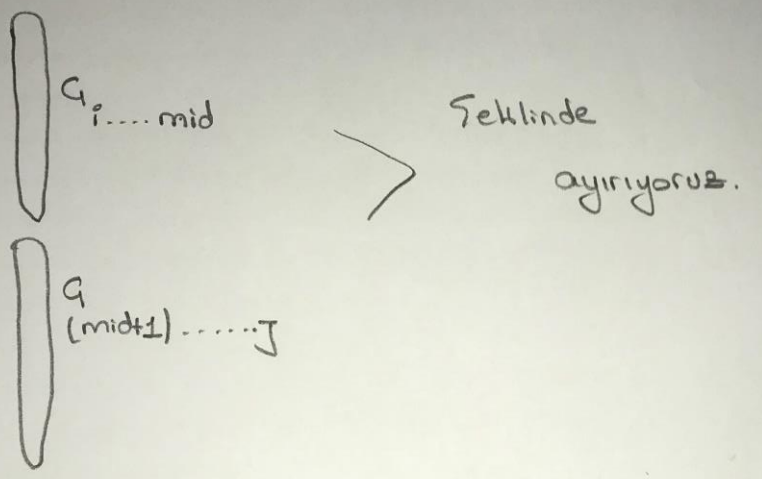
→Uzunlukların fiyat tablosu.

Daha iyi anlamak için problemi 4 uzunluğudaki çubuğun maksimum satış fiyatını inceleyelim:

Çözüm: ilk adım;

İpi veya çubuğumuzu dik olacak şekilde modellersek;

Bir orta nokta seçiyorum "mid" isminde ve çubuğu tam ortadan ikiye ayırıyorum.



Bu şekilde yaptığımızda şimdi aynı problemden iki tane oluyor. Bu aynı şekilde devam edebilir o yüzden daha mutlak kesin bir çözüme ihtiyacımız var. Yani sonsuz tane problemimizi dinamik programlama mantığı ile daha küçük problemler haline getirip büyük çözüme gitmeye çalışalım.

Öncelikle çubuğumuzun uzunluğu 'N' idi. bu çubuğun her i cmlik parçasının 'pi' gibi fiyatı olsun. i uzunluğundaki bir çubuğun kendinden küçük parçalarla oluşturulabilen max fiyatına da 'ri' diyelim. Fiyat tablomuz ilk sayfada vardı. Şimdi maksimum kârı hesaplayalım.

İlk sayfada yaptığımız 4 paçanın maksimum oluşunu matematiksel olarak yazalım;

$9 = p_4 \neq r_4$

$9 = p_1 + p_3 \neq r_4$

$10 = p_2 + p_2 = r_4$ (?) → max

$4 = p_1 + p_1 + p_1 + p_1 \neq r_4$

$7 = p_1 + p_1 + p_2 \neq r_4$

Bu durumda görüldüğü üzere en pahalı satış iki parçaya bölünerek yapılmaktadır.

$r_4 = p_2 + p_2 = 10$

Çubuğumuzu parçalarsak;

$p = 1, 2, \ldots, n-1, n$

Uzunluk olarak 7 'yi ele alırsak $7 = 2+2+3$ şeklinde 3 parçamız olur. i değerimiz en küçük 1 'e eşit, en büyük n'e eşit olabildiği için, parça sayısını $1 < k < n$ şeklinde gösterebiliriz.

Tüm parçaları toplarsak, $n = i_1 + i_2 + \ldots + i_k$ buluruz. buna bağlı olarak da bir çubuğun en yüksek fiyatını şu şekilde ifade edebiliriz $r_n = \max(p_1 + r_{n-1}, p_2 + r_{n-2} \cdots, p_{n-1} + r_1, p_n + 0)$ R_n'i oluşturan her bir parça da kendini en pahalı yapacak parçalardan oluşmaktadır. Yani;

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

diyebiliriz.

Sözde Kod

```
Cubuk (p,n)
if n == 0
    return 0
q = -∞ (en küçük değer)
for i = 1 to n
    q = max(q, p[i] + Cubuk(p, n-i))
return q
```

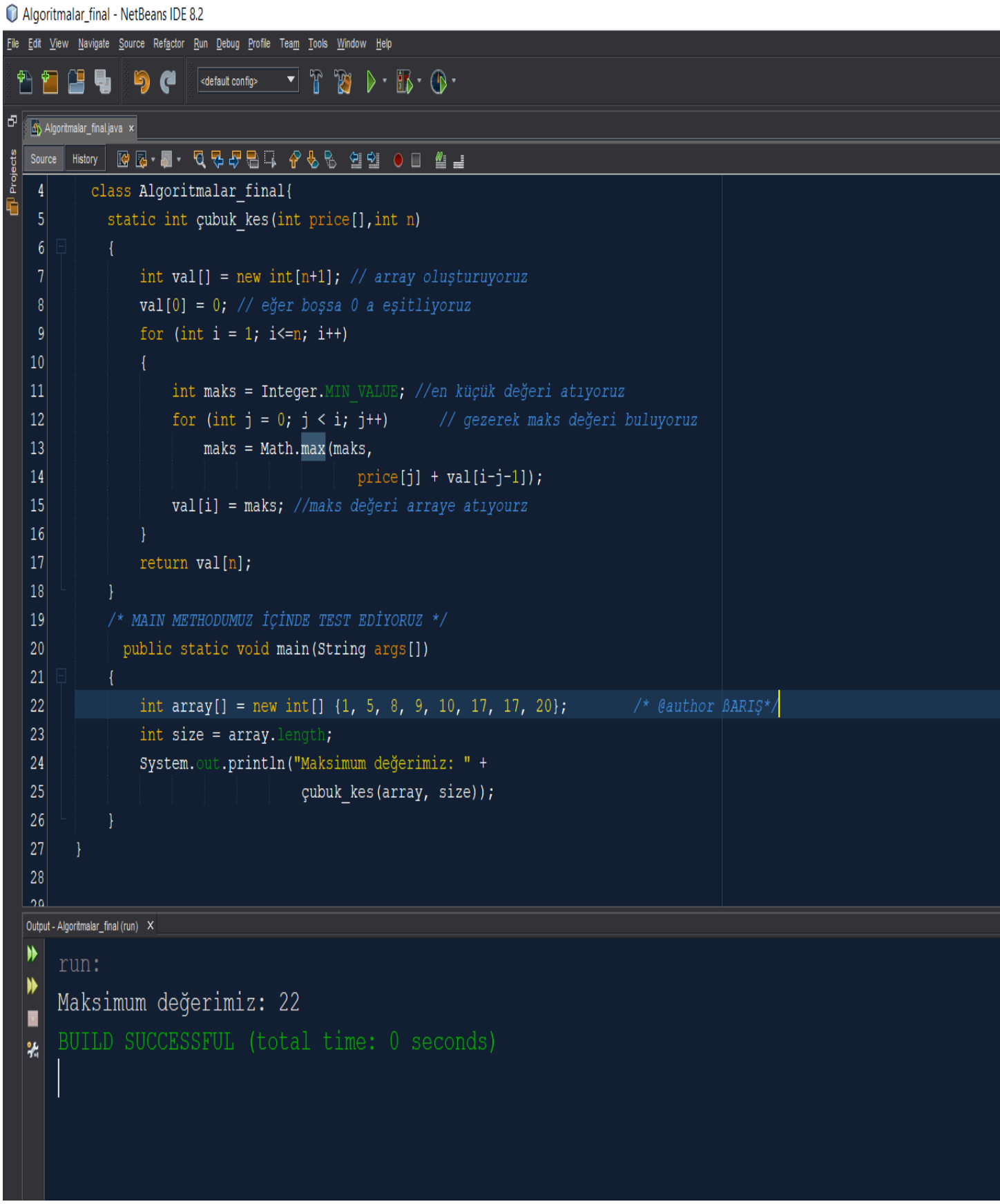
Recursive olarak çözersek 'n == 0' durmasını sağlayan koşulumuzdur. q'ya ilk başta en küçük değer atanır. For döngüsü ile de en büyük değer son olarak q değişkenine kaydedilir. Fakat bu çözüm çok uzundur ($O(2^n)$) şimdi dinamik programlama ile çözümümüzü kısaltalım.

DİĞER SAYFADA DEVAMI ⟹

```
Cubuk(p,n)
let r[0....n] be new Array
r[0] = 0
for j = 1 to n
    q = -∞ (en küçük value)
    for i = 1 to j
        q = max(q, p[i] + r[j-i])
    r[j] = q
return r[n]
```

İlk başta alt problemlerimizin çözümlerini tutabilmek için bir dizi oluşturduk. İkinci olarak hiç parça olmazsa r[0]=0 'a eşitledik. For döngülerimizin içinde de bütün alt problemler çözülür ve j değerimiz 1 'den n'e kadar büyür bu sayede küçükten büyük probleme doğru bir çözüm elde etmiş oluruz. Forların en içinde ise maksimum value q'ya atanır. En sonda ise arrayimiz güncellenir ve değeri döndürülür.

Dinamik programlama ile çözdüğümüz için asimptotik çalışma zamanına sahiptir. Sözde kodumuzda da görüldüğü üzere iç içe for döngümüzden dolayı $O(n^2)$ çalışma zamanına sahiptir. Diyebiliriz.


Algoritmalar_final - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Algoritmalar_final.java
Source History
Projects
class Algoritmalar_final{
static int çubuk_kes(int price[],int n)
{
int val[] = new int[n+1]; // array oluşturuyoruz
val[0] = 0; // eğer boşsa 0 a eşitliyoruz
for (int i = 1; i<=n; i++)
{
int maks = Integer.MIN_VALUE; //en küçük değeri atıyoruz
for (int j = 0; j < i; j++) // gezerek maks değeri buluyoruz
maks = Math.max(maks,
price[j] + val[i-j-1]);
val[i] = maks; //maks değeri arraye atıyourz
}
return val[n];
}
/* MAIN METHODUMUZ İÇİNDE TEST EDİYORUZ */
public static void main(String args[])
{
int array[] = new int[] {1, 5, 8, 9, 10, 17, 17, 20}; /* @author ßARIŞ*/
int size = array.length;
System.out.println("Maksimum değerimiz: " +
çubuk_kes(array, size));
}
}
Output - Algoritmalar_final (run)
run:
Maksimum değerimiz: 22
BUILD SUCCESSFUL (total time: 0 seconds)

ÖZÇIKARIM (DİNAMİK PROGRAMLAMA):

-BİR PROBLEMİN KÜÇÜK PARÇALARINI ÇÖZÜP BÜYÜK ÇÖZÜME GİDERKEN KULLANMAKTIR.

-ARRAYLER BUARADA KÜÇÜK ÇÖZÜMLERİMİZİ DEĞERLENDİRİP KAYDETMEMİZİ SAĞLAR.

-RECURSİVE YAPIDA AYRICA DİNAMİK PROGRAMLAMA MANTIĞINDA KULLANILIR ÇÜNKÜ ESASINDA BİR PROBLEM ALT PROBLEMLERDEN OLUŞTUĞU İÇİN AYNI ŞARTLARI O ALT PROBLEMLERİNDE ÇÖZÜMÜ DE AYNI OLACAĞINDAN RECURSİVE BİR ÇÖZÜM KULLANILMASI GAYET MANTIKLIDIR.

NOT:

ÖDEV KESİNLİKLE GRUP OLARAK DEĞİL KENDİM TARAFINDAN YAPILMIŞTIR.

PDF'DE İSTENİLEN PROBLEMİN TANIMI,SÖZDE KODU,ZAMAN KARMAŞIKLIĞI VE ÖRNEK ÇÖZÜMÜ BULUNMAKTADIR.

ÖRNEK ÇÖZÜM İSE PROBLEMİN JAVA KODU YAZILARAK ÇÖZÜLMÜŞTÜR DİNAMİK PROGRAMLAMA MANTIĞI DOĞRULTUSUNDA.

// ALGORİTMALAR DERSİ FİNAL ÖDEVİ

// BERK BARIŞ KARA -18253007