**TUGAS DESAIN & ANALISIS ALGORITMA**

***Dynamic Programming*: *Rod Cutting Problem***



**Disusun oleh:**

**Adrian Maulana Muhammad** **(06111540000099)**

**Departemen Matematika**

**Fakultas Sains dan Analitika Data**

**Institut Teknologi Sepuluh Nopember**

**Surabaya**

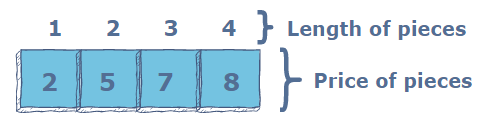
**2020**

1. Diberikan suatu *rod* yangberukuran *n*, dan suatu *array* yang mempunyai elemen berupa harga-harga dari potongan *rod* tersebut yang kurang dari *n*. Tentukan *maximum profit* yang bisa diambil dengan metode *cutting* untuk penjualan potongan-potongan tersebut.

**Penyelesaian:**

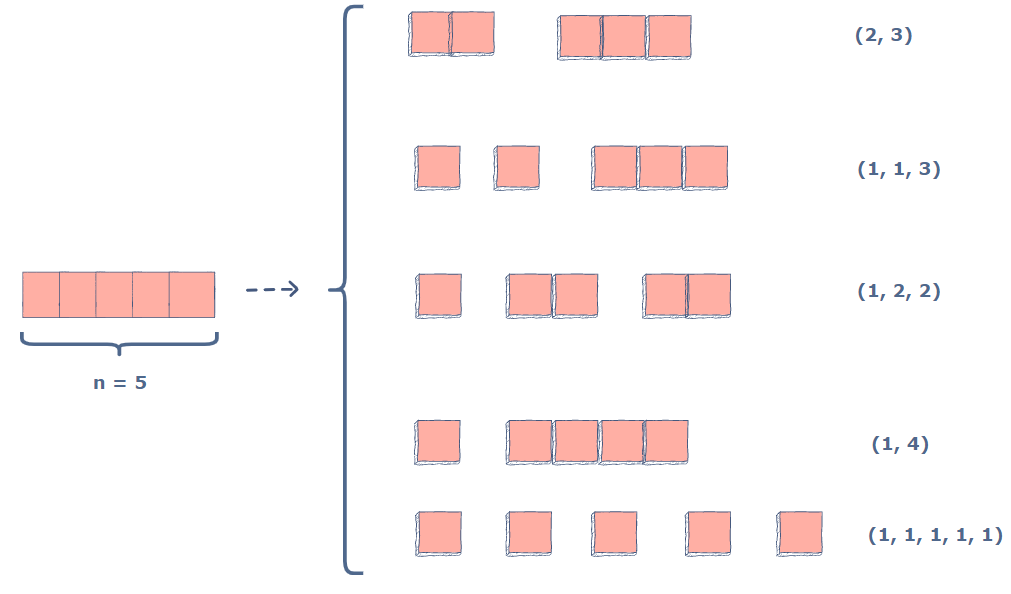
1. **Initialization**

* Anggap kita mempunyai suatu *rod* berukuran 5, dan suatu *array* yang berisi *length* (1, 2, 3, dan 4) dan *price* (2, 5, 7, dan 8) dari potongan-potongan tersebut.



1. **Naive Solution**

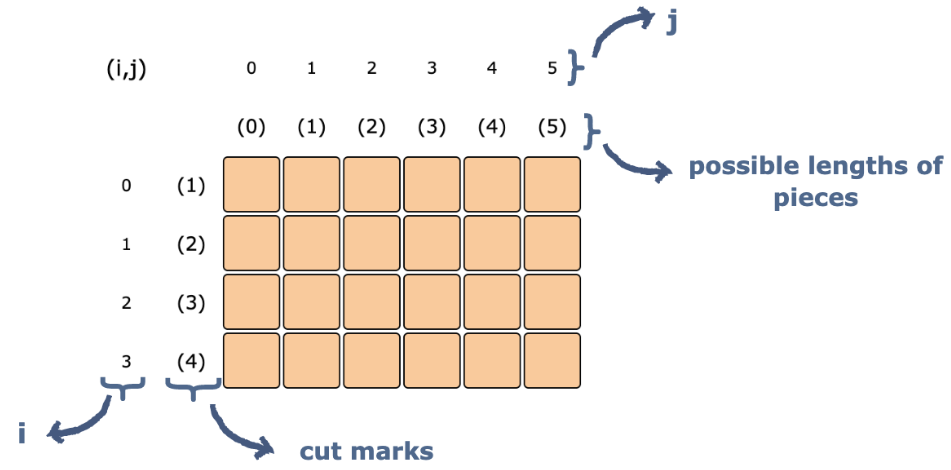
* Terdapat beberapa cara untuk memotong *rod* tersebut ke bentuk *sub-rods*, setiap cara menghasilkan *profit* yang berbeda-beda juga.



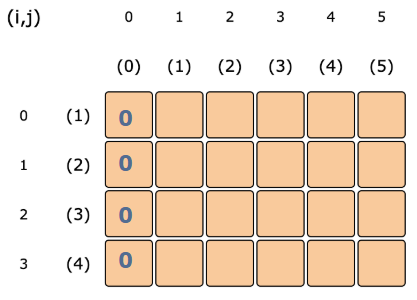
* Dari ilustrasi di atas, jawabannya adalah 12 (dengan menjual *sub-rods* pada *length* (1, 2, 2) dan memberikan *profit* → 2 + 5 + 5 = 12).
* *Naive solution* akan menghasilkan semua konfigurasi dari semua potongan untuk mendapatkan konfigurasi mana yang menghasilkan *price* tertinggi. Namun, metode ini mempunyai *time complexity* berupa eksponential. Oleh karena itu, kita harus mencari algoritma lain yang lebih baik.

1. **Dynamic Programming**

* Karena *dynamic programming* adalah suatu *optimization-over-plain recursion*, kita dapat menggunakannya untuk optimasi algoritma di atas yang mempunyai waktu eksponensial.
* Ide dasarnya adalah untuk menyimpan hasil dari *subproblems* sehingga kita tidak perlu menghitung ulang lagi saat hasil tersebut dibutuhkan.
* Diketahui dari sebelumnya:
  + Rentang dari *length* yang memungkinkan, dari setiap *sub-rod*, adalah dari 0 sampai 5. Suatu *sub-rod* dengan *length* yang termasuk rentang ini dapat dibentuk (*possible lengths of pieces*).
  + Kita mempunyai 4 harga yang berbeda dari masing-masing *length* yang berbeda pada *sub-rods* (*cut marks* → tempat dimana potongan dapat dilakukan untuk menghasilkan *sub-rods*)*.*

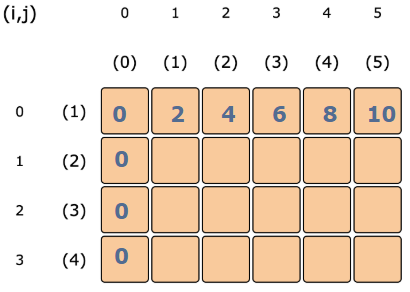


* Ilustrasi *grid* di atas merepresentasikan *sub-problems*, dengan:
  + *i* merupakan indeks dari baris.
  + *j* merupakan indeks dari kolom.
* *Cell* pada indeks [*i*, *j*] merepresentasikan jumlah *maximum profit* yang didapatkan dari menjual *rod* denganukuran awal *j* dan pilihan potongan dari *array* 0 sampai *i*.
* Pertama, akan kita isi *array* pada tabel. Jika *rod* mempunyai ukuran 0, maka *maximum profit* yang didapatkan juga 0.



* Selanjutnya, kita akan isi pada setiap barisnya:
  + Pada baris pertama, kita hanya mempunyai *price* 2 pada *length* 1 di *array*.
  + Pada indeks [0, 1], memotong pada 1 akan memberikan profit2. Tidak memotong juga akan memberikan profit 2.
  + Pada indeks [0, 2], memotong pada 1 akan memberikan profit 4 (2 *rods* dengan *length* 1, masing-masing mempunyai *price* 2). Tidak memotong juga akan memberikan profit 4 (bukan 5, karena *price* dari 2 belum ada di *array*).
  + Lakukan tahap ini pada seluruh baris pertama.

Maka, *grid* menjadi seperti berikut:

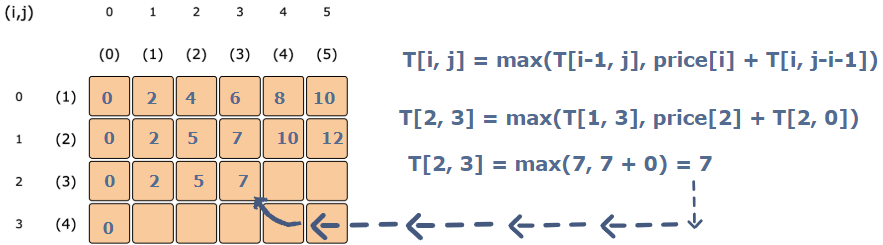


* Selanjutnya, kita akan memutuskan, pada setiap langkah, bahwa apakah penambahan *cut mark* (*i*+1) akan meningkatkan profit pada setiap *rod* ukuran *j*, atau tidak. Kita menentukan dengan mengambil *maximum* dari dua hal:
  + Profit maksimum dari *length* sebelum *cut mark* dilakukan (nilai pada indeks [*i*-1, *j*]). Kasus ini adalah dimana kita tidak memasukan penambahan *cut mark* yang baru.
  + *price*[*i*]+ profit maksimum pada *length j*-*i*-1 untuk nilai sementara dari *array of prices* (nilai dari indeks [*i, j*-*i*-1]) sudah dihitung sebelumnya. Kasus ini adalah dimana kita menggunakan *cut mark* yang baru untuk mempengaruhi profit.
* Oleh karena itu, formula untuk mengisi *grid* adalah:

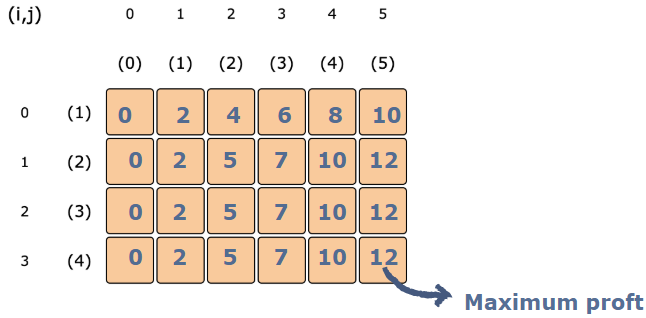


dimana *T* adalah *grid* dan *price* adalah *array* dari harga.

* Saat *j* ≤ *i*, maka *j*-*i*-1 akan bernilai negatif. Oleh karena itu, bisa kita gunakan nilai pada indeks [*i*-1, *j*].
* Proses pengisian *grid* adalah sebagai berikut:



* Setelah *grid* sudah terisi semua, maka didapatkan solusi pada permasalahan mencari profit maksimum dari *rod* dengan ukuran *n* dan *array of prices* dari 0 sampai *n*-1 adalah pada *cell* di bawah-kanan.



1. **Implementasi**

class rod\_cutting{

static int cut\_rod(int price[], int n){

// Declaring a 2D array, T

int T[][] = new int[n-1][n+1];

// Initializing the array to all zeros

for(int i=0; i < n-1; i++){

for(int j=0; j < n+1; j++ ){

T[i][j] = 0;

}}

for(int i=0; i < n-1; i++){

for(int j=0; j < n+1; j++ ){

// First column => 0 length of rod => 0 profit

if(j == 0){

continue;

}

// First row => T[i-1][j] doesn't exist so just pick the // second value

else if(i == 0) {

T[i][j] = price[i] + T[i][j-i-1];

}

// where j <= i => T[i][j-i-1] doesn't exist so just pick // the first value

else if(j-i-1 < 0){

T[i][j] = T[i-1][j];

}

// using the whole expression

else{

T[i][j] = Math.max(T[i-1][j], (price[i] + T[i][j-i-1]));

}}}

return T[n-2][n];

}

public static void main(String args[]){

int price[] = new int[] {2,5,7,8};

int n = 5;

System.out.println("Maximum profit is " + cut\_rod(price, n));

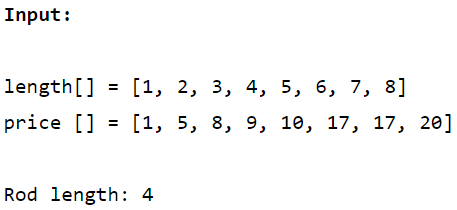
}

}

1. **Output**

* Maximum profit is 12
* Algoritma mempunyai *time complexity O*().

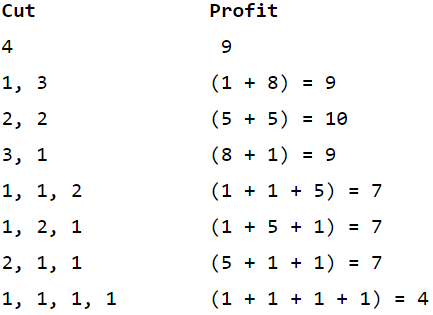
1. Diberikan *input* sebagai berikut:



Tentukan cara yang optimal untuk memotong *rod* dengan *length* 4 agar mendapatkan profit yang maksimal.

**Penyelesaian:**

* Potong *rod* dengan *length* 4 untuk mendapatkan profit yang maksimal.



* Ide dasarnya sederhana, dengan diberikan *array* bernama *price*[], dimana *rod* dengan *length i* mempunyai nilai *price*[*i-*1]. Satu per-satu, kita buat partisi pada *rod* dengan *length n* mejadi *length i* dan *length n*-*i.* Kita lakukan rekursif pada *rod* dengan *length n-i,* tetapi tidak perlu lagi *divide* untuk *rod* dengan *length i*. Lalu, kita ambil semua nilai yang maksimum. Hubungan rekursif bisa dilihat pada relasi sebagai berikut:



* Implementasi denganprogram:

class RodCutting{

public static int rodCut(int[] price, int n){

if (n == 0) {return 0;}

int maxValue = Integer.MIN\_VALUE;

for (int i = 1; i <= n; i++){

int cost = price[i - 1] + rodCut(price, n - i);

if (cost > maxValue) {

maxValue = cost;

}}

return maxValue;}

public static void main(String[] args){

// length[] = { 1, 2, 3, 4, 5, 6, 7, 8 };

int price [] = { 1, 5, 8, 9, 10, 17, 17, 20 };

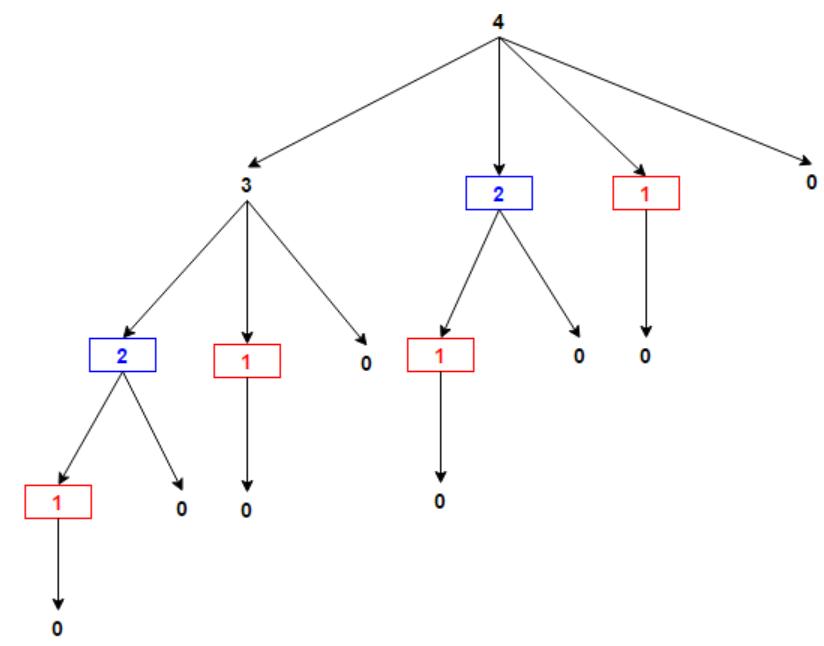
// rod length

int n = 4;

System.out.println("Profit is " + rodCut(price, n));

}}

* Output:
  + 
  + *Time complexity* dari algoritma di atas adalah *O*().
  + *Auxiliary space* dari program adalah *O*(1).
* Bisa kita lihat bahwa permasalahan dapat dibuat menjadi permasalahan yang lebih kecil (*smalller subproblems*) dan ternyata masih bisa dibuat menjadi permasalahan yang lebih kecil lagi. Jadi permasalahan mempunyai *optimal substructure.* Bisa dilihat *recursion tree* untuk *rod* dengan *length* 4:



* Bisa dilihat bahwa, *subproblems* (mempunyai warna yang sama) dihitung berulang kali. Jadi permasalahan juga merupakan *overlapping problems.*
* Permasalahan yang mempunyai *optimal substructure* dan *overlapping problems* dapat diselesaikan dengan *dynamic programming.*
* Kita akan menyelesaikan masalah ini dengan pendekatan *bottom*-*up.* Dengan kata lain, kita menyelesaikan *smaller subproblems* terlebih dahulu, lalu menyelesaikan *larger subproblems.*
* Pendekatan *bottom*-*up* menghitung T[*i*], yang menyimpan profit maksimum didapat dari *rod* dengan *length i* untuk setiap 1 ≤ *i ≤ n.* Kita menggunakan nilai dari *smaller i* yang telah dihitung.
* Implementasi dengan program:

class Util{

public static int rodCut(int[] price, int n){

// T[i] stores maximum profit achieved from rod

int[] T = new int[n + 1];

// divide the rod into two rods

for (int i = 1; i <= n; i++){

for (int j = 1; j <= i; j++){

T[i] = Integer.max(T[i], price[j - 1] + T[i - j]);

}}

// T[n] stores maximum profit achieved from rod

return T[n];}

// main function

public static void main(String[] args){

int[] price = { 1, 5, 8, 9, 10, 17, 17, 20 };

int n = 4; // rod length

System.out.print("Profit is " + rodCut(price, n));

}}

* Output:
  + 
  + *Time complexity* dari algoritma di atas adalah *O*().
  + *Auxiliary space* dari program adalah *O*(*n*).