Rod cutting problem   Bottom-up approach.

Idea: It solves the subproblems from 0 to n iteratively and store them in a table / dictionary / hash table.

Bottom-up-rod-cut (P, n)

let $r[0,\dots,n]$ be a new array

cut $s[0\dots,n]$ " " new array.

$r[0] = 0$

for $j = 1$ to n

$q = -\infty$

$$q = \max_{1 \le i \le j} \{ q, \; p[i] + r[j-i] \}$$

$s[j] = i$ // ← this is the i which gives the maximum q

$r[j] = q$

return $r[n]$

Running time: $O(n^2)$

two for loops

r

0					
0	1	2	3	4	5

i	1	2	3	4
p_i	1	5	8	9

$n = 4$

r

0	1	5	8	10
0	1	2	3	4

$p[1] + r[0] = 1$

$j=2$ for $i=2$ max $\begin{cases} p[1] + r[1] = 2 \\ p[2] + r[0] = 5 \Leftarrow \end{cases}$

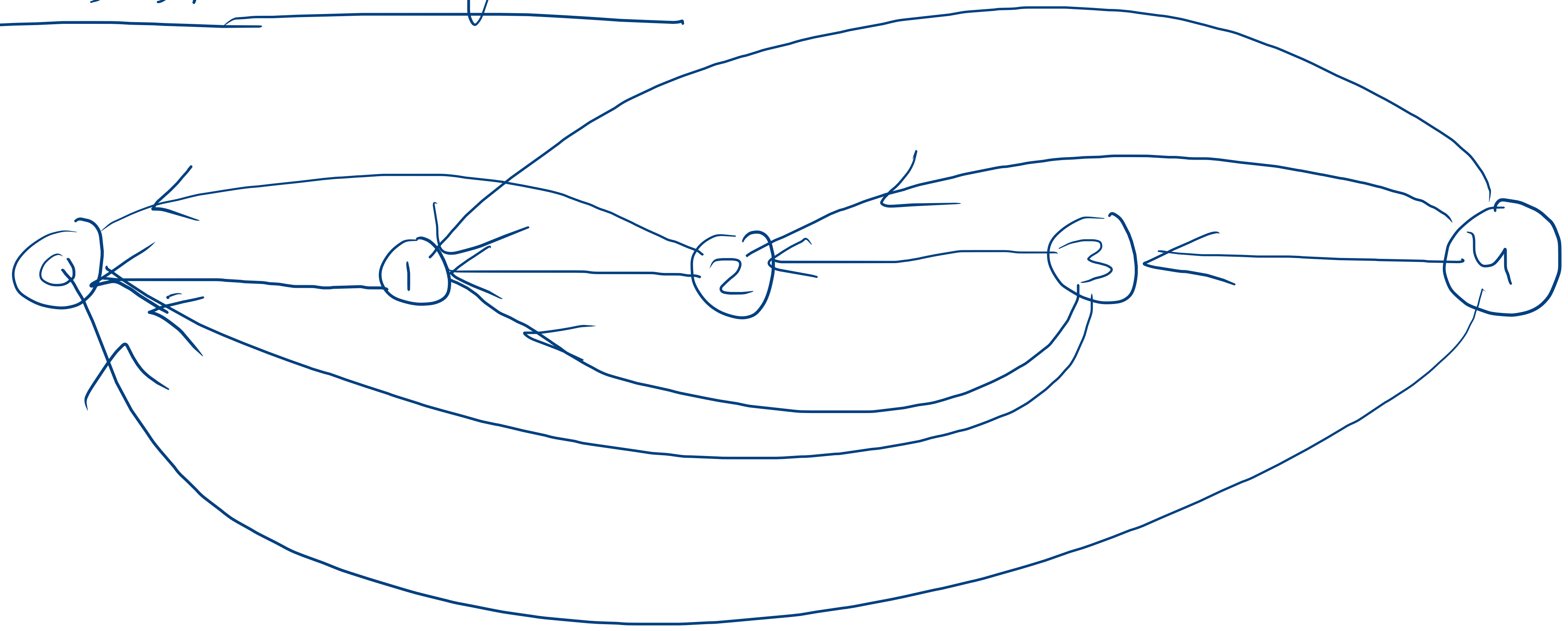
s

0	1	2	3	2
0	1	2	3	4

$j = 3$ for $i = 3$ $p[3] + r[0] = 8$

$j = 4$ for $i = 2$

The subproblem graph



Construction of a solution

```
Print_rod_cut(P, n)
(r, s) = Bottom-up-rod-cut(P, n)
while n > 0
        Print s[n]
        n = n - s[n]
```

For $n = 4$,

Print gives the solution as 2, 2

General outline of a DP problem

Step 1 :- Structure

characterize the structure of an optimum solution by showing that it can be decomposed into optimum subproblems.

Step 2 Recursive

Recursively define the value of an optimum solution by expressing it in terms of optimum solution for smaller subproblems.

Step 3: Optimum value computation

Two approaches

i) Top-down with memoization

ii) Bottom-up with tabulation.

Step4: optimum solution computation

Step4 only requires when one ask for finding an optimum solution.

Sometime additional information is maintained during steps 1 – 3 to easily construct an optimum solution.

Fibonacci number

1, 2, 3, 5, 8, 13, 21, ...

Step 1 If optimally compute previous 2 terms then you can optimally compute that number by summing the previous two numbers.

Step 2.

$$F_n = \begin{cases} 1, & \text{if } n = 1 \\ 2 & \text{if } n = 2 \\ F_{n-1} + F_{n-2} & \text{if } n \geq 3 \end{cases}$$

```
fibonacci (n)
Let F be an array.
if n = 1
        return 1
if n = 2
        return 2
for i = 3 to n
        f[n] = f[n-1] + f[n-2]
return F[n]
```

```
fib (n)
 if n = 1
     return 1
  if n = 2
       return 2
  for i = 3 to n
       return fib(n-1) + fib(n-2)
```