Structuri de date și algoritmi

Tehnica Dynamic Programming

Lect. Dr. Ing. Cristian Chilipirea – cristian.chilipirea@mta.ro

Șirul Fibonacci

$$\boldsymbol{f(n) = f(n-1) + f(n-2)}$$

Șirul Fibonacci

$$\boldsymbol{f(n) = f(n-1) + f(n-2)}$$

$$f(0) = 0, f(1) = 1$$

Secvența: 0 1 1 2 3 5 8 13 21 34 ...

Fibonacci Recursiv (Divide et Impera)

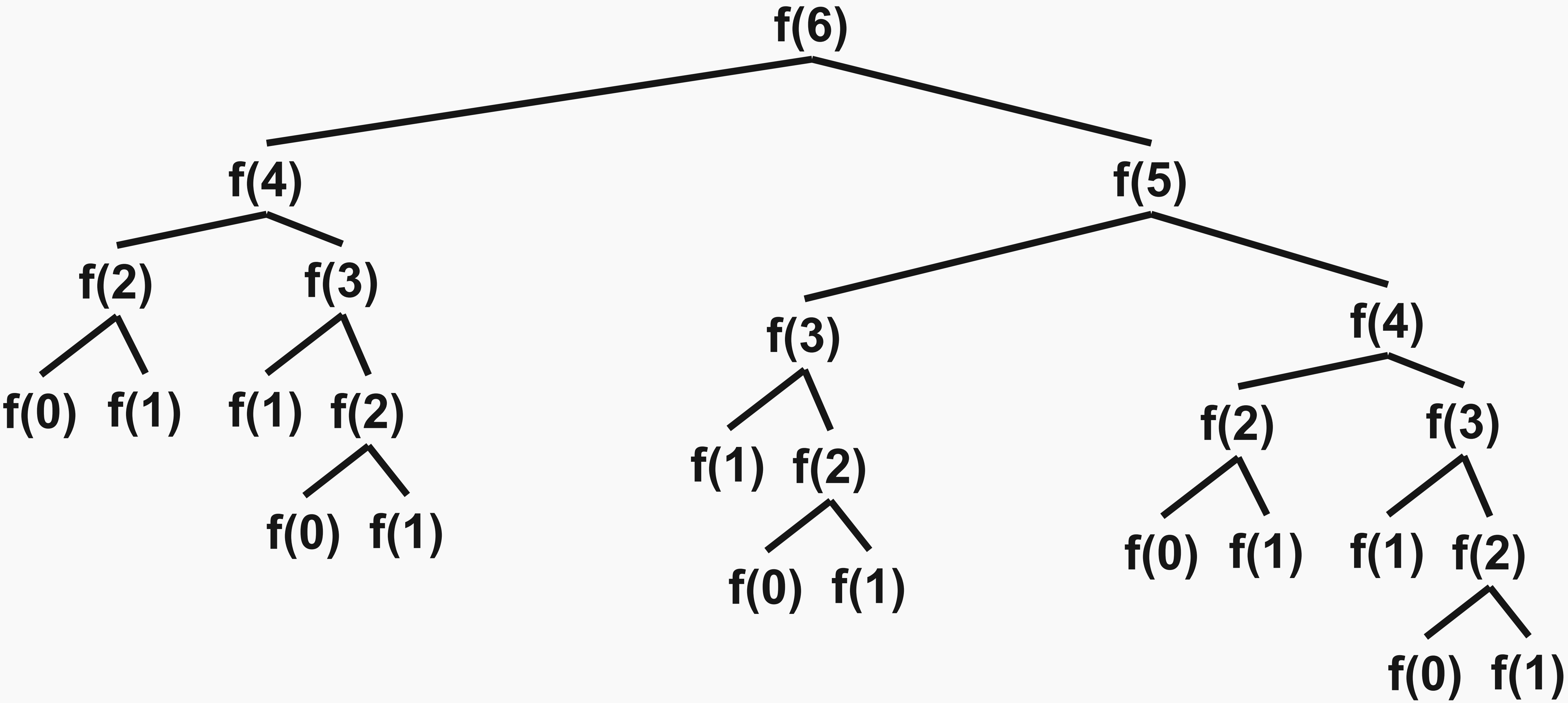
```
int fibonacci(int n)
{
    if (n <= 1)
        return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

Complexitate?

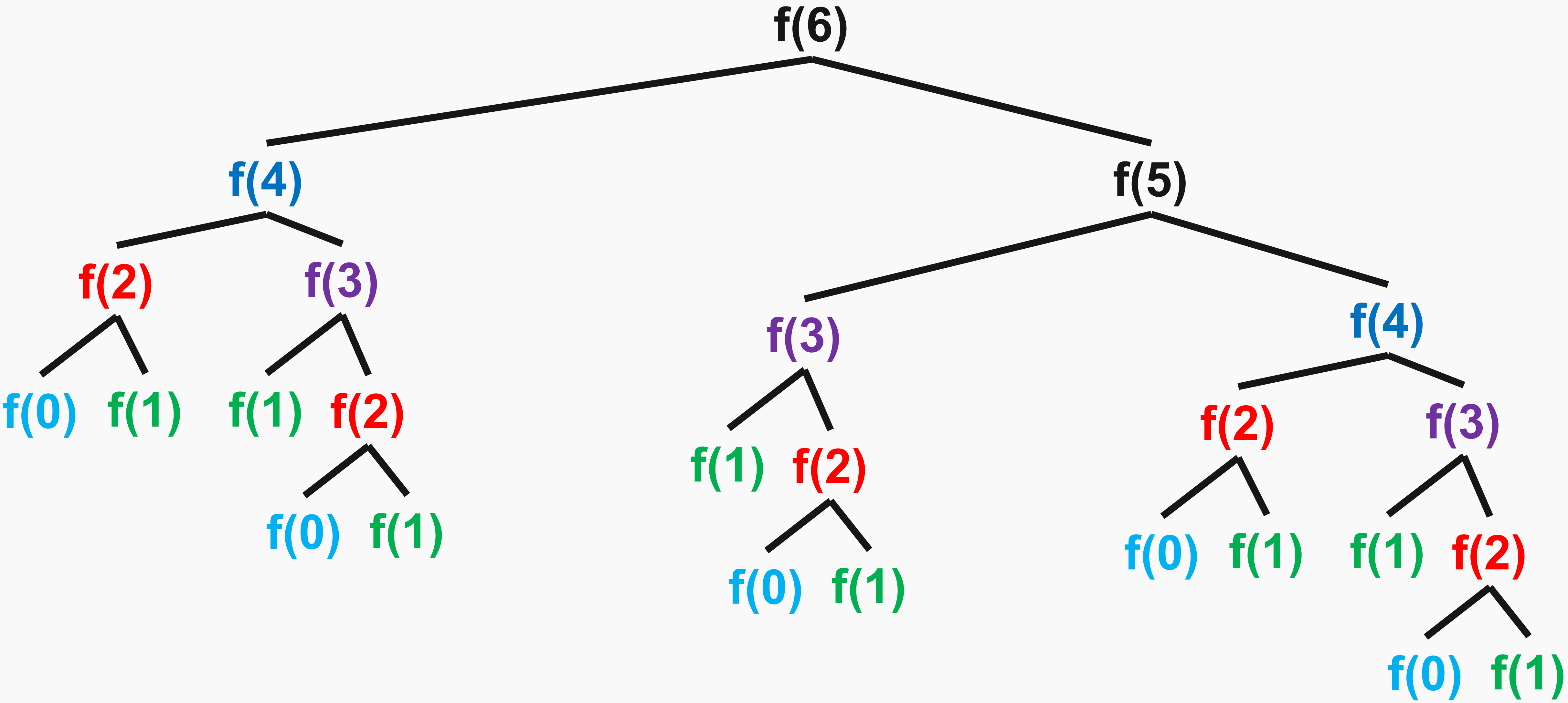
Complexitate?

$$F(n) = \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}}, \varphi = \frac{1+\sqrt{5}}{2} \rightarrow \boldsymbol{\theta}(\boldsymbol{\varphi}^{\boldsymbol{n}})$$

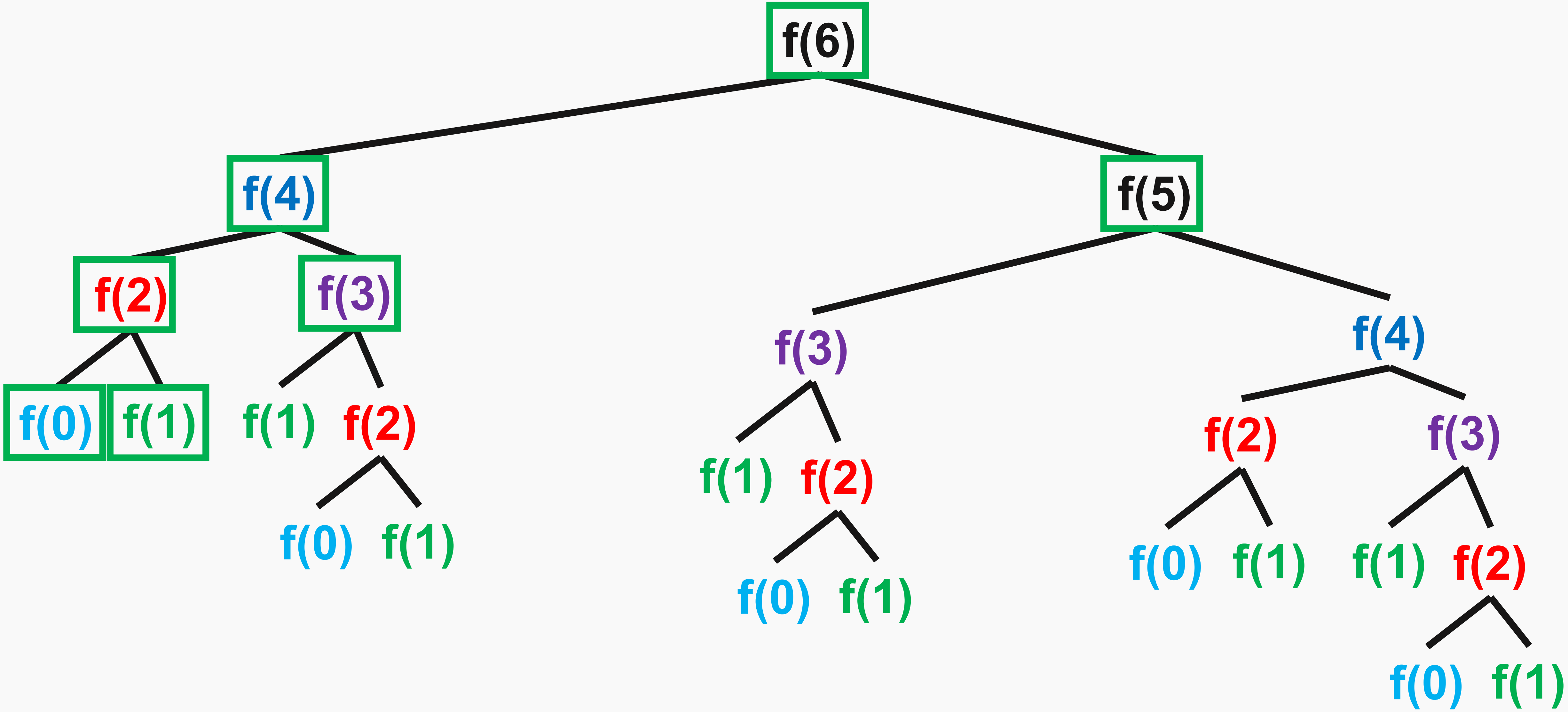
Apeluri


f(6)
f(4)
f(5)
f(2)
f(3)
f(4)
f(3)
f(0)
f(1)
f(1)
f(2)
f(2)
f(3)
f(1)
f(2)
f(0)
f(1)
f(0)
f(1)
f(1)
f(2)
f(0)
f(1)
f(0)
f(1)

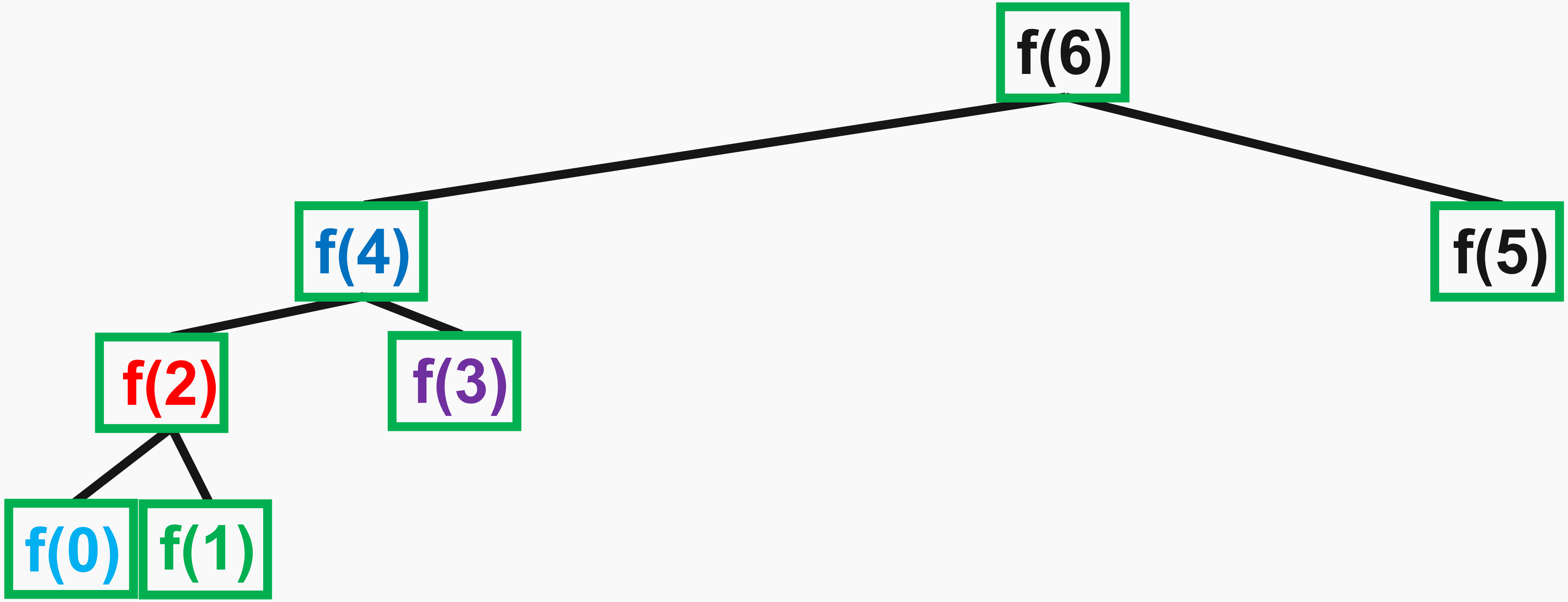
Apeluri


f(6)
f(4)
f(5)
f(2)
f(3)
f(3)
f(4)
f(0) f(1) f(1) f(2)
f(1) f(2)
f(2) f(3)
f(0) f(1)
f(0) f(1)
f(0) f(1) f(1) f(2)
f(0) f(1)

Apeluri


f(6)
f(4)
f(5)
f(2)
f(3)
f(3)
f(4)
f(0)
f(1)
f(1)
f(2)
f(1)
f(2)
f(2)
f(3)
f(0)
f(1)
f(0)
f(1)
f(0)
f(1)
f(1)
f(2)
f(0)
f(1)

Apeluri



Implementare Bottom-up (Programare Dinamică)

```
int F[1000];
int fibonacci(int n)
{
    F[0] = 0;
    F[1] = 1;
    for (int i = 2; i <= n; i++)
        F[i] = F[i - 1] + F[i - 2];
    return F[n];
}
```

Complexitate?

Complexitate?

$$\theta(n)$$

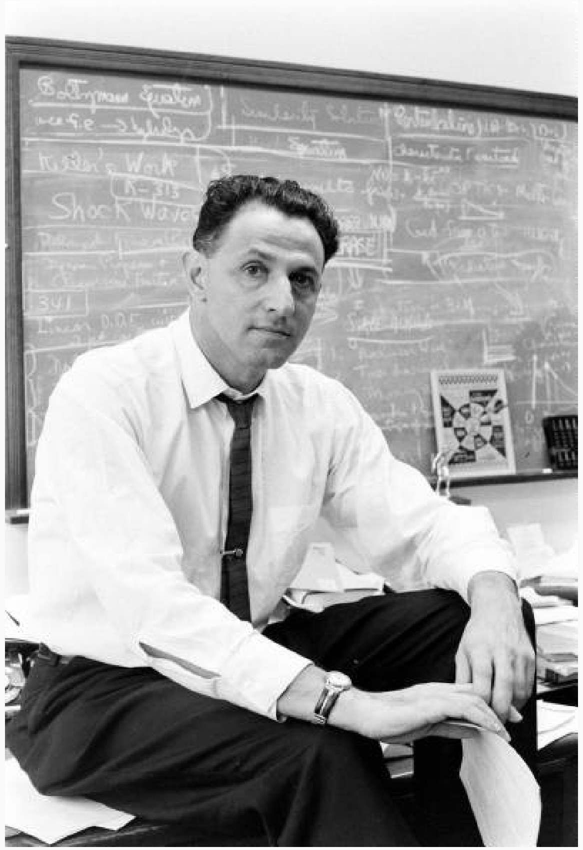
Programare dinamică

1. Identificarea subproblemelor.

2. Determinarea unei **relaţii de recurenţă** pentru descompunere.

3. Alegerea unor structuri care să reţină soluţiile subproblemelor.

4. Rezolvarea recurenţei în mod bottom-up (în ordinea crescătoare a dimensiunilor subproblemelor)



Richard Bellman

Programare dinamică

- Pentru probleme de optimizare (găsire maxim/minim)
- Probleme ce pot fi rupte în subprobleme
- Subproblemele trebuie să fie **independente**.
  - O subproblemă nu poate afecta o alta.
- Subproblemele trebuie să fie overlapping
  - Să apară aceeași subproblemă de mai multe ori.
  - Altfel putem folosi doar Divide et Impera.

Memoization

- Programarea dinamică poate funcționa și top-down, păstrând recursivitatea, dar se va adăuga un mecanism de păstrare și refolosire a soluțiilor parțiale.

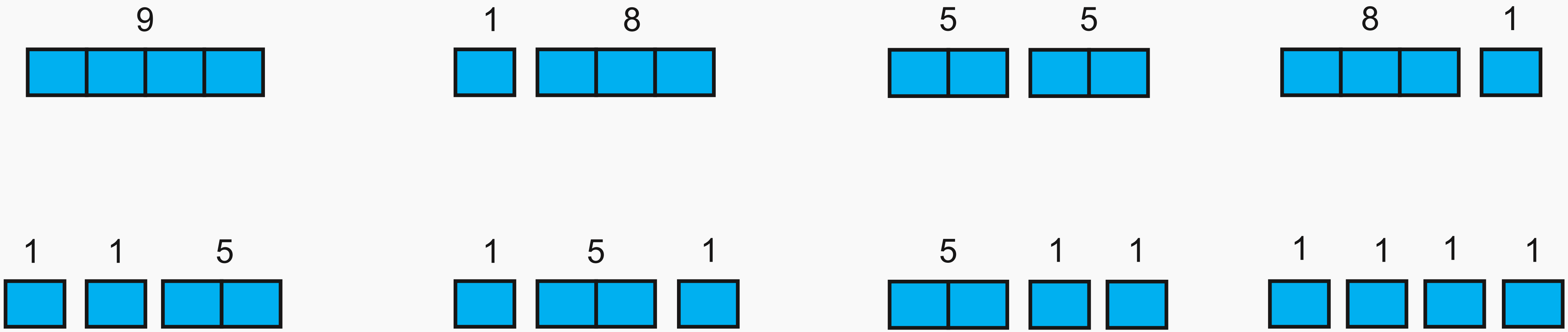
Rod cutting

- Se dă o tijă de o mărime N.
- Tija trebuie tăiată în una sau mai multe bucăți pentru a **maximiza** profitul obținut pentru bucăți.
- Bucățile au preț diferit în funcție de mărimea tijei.

Lungime	1	2	3	4	5	6	7	8	9	10
Preț	1	5	8	9	10	17	17	20	24	30

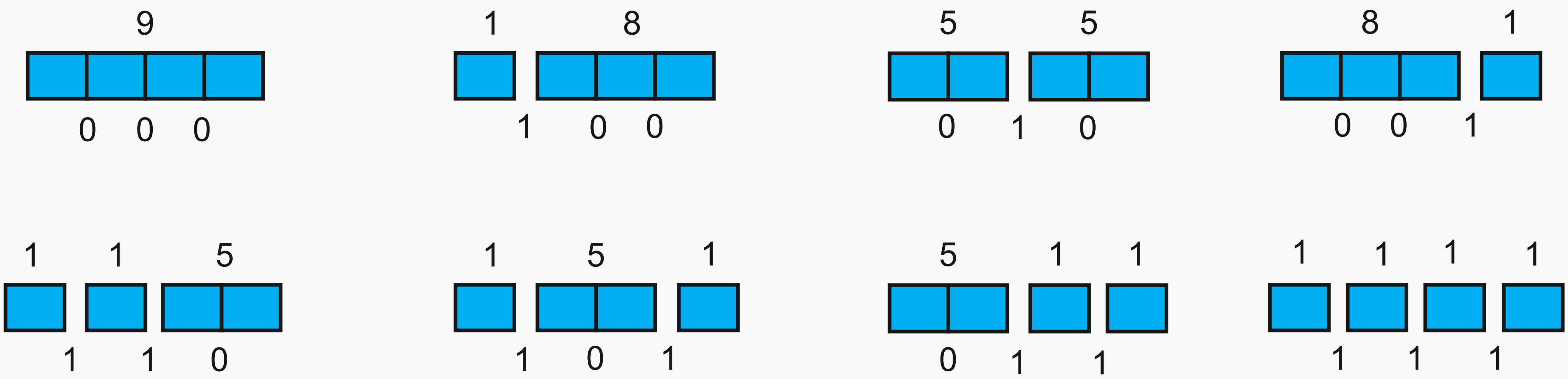
Rod cutting

Lungime	1	2	3	4	5	6	7	8	9	10
Preț	1	5	8	9	10	17	17	20	24	30



Rod cutting

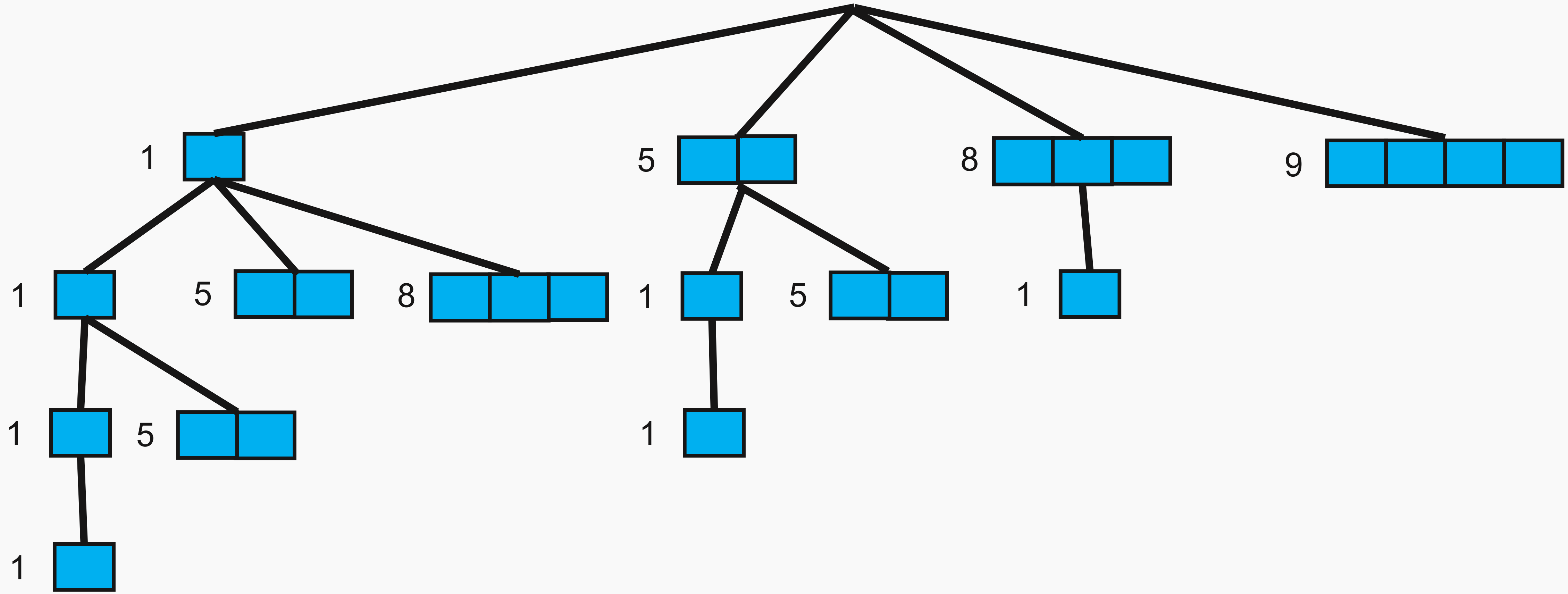
Lungime	1	2	3	4	5	6	7	8	9	10
Preț	1	5	8	9	10	17	17	20	24	30



Implementare recursivă - Rod cutting

```
int p[11] = {0, 1, 5, 8, 9, 10, 17, 17, 20, 24, 30};

int cutrod(int n)
{
    if (n == 0)
        return 0;
    int q = -1;
    for (int i = 1; i <= n; i++)
        q = fmax(q, p[i] + cutrod(n - i));
    return q;
}
```

Rod cutting



Implementare Rod cutting – programare dinamică

```
int p[11] = {0, 1, 5, 8, 9, 10, 17, 17, 20, 24, 30};
int r[11] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};

int memoizedcutrod(int n)
{
    if (r[n] >= 0)
        return r[n];
    int q;
    if (n == 0)
        q = 0;
    else {
        q = -1;
        for (int i = 1; i <= n; i++)
            q = fmax(q, p[i] + memoizedcutrod(n - i));
    }
    r[n] = q;
    return q;
}
```

Implementare Rod cutting – programare dinamică

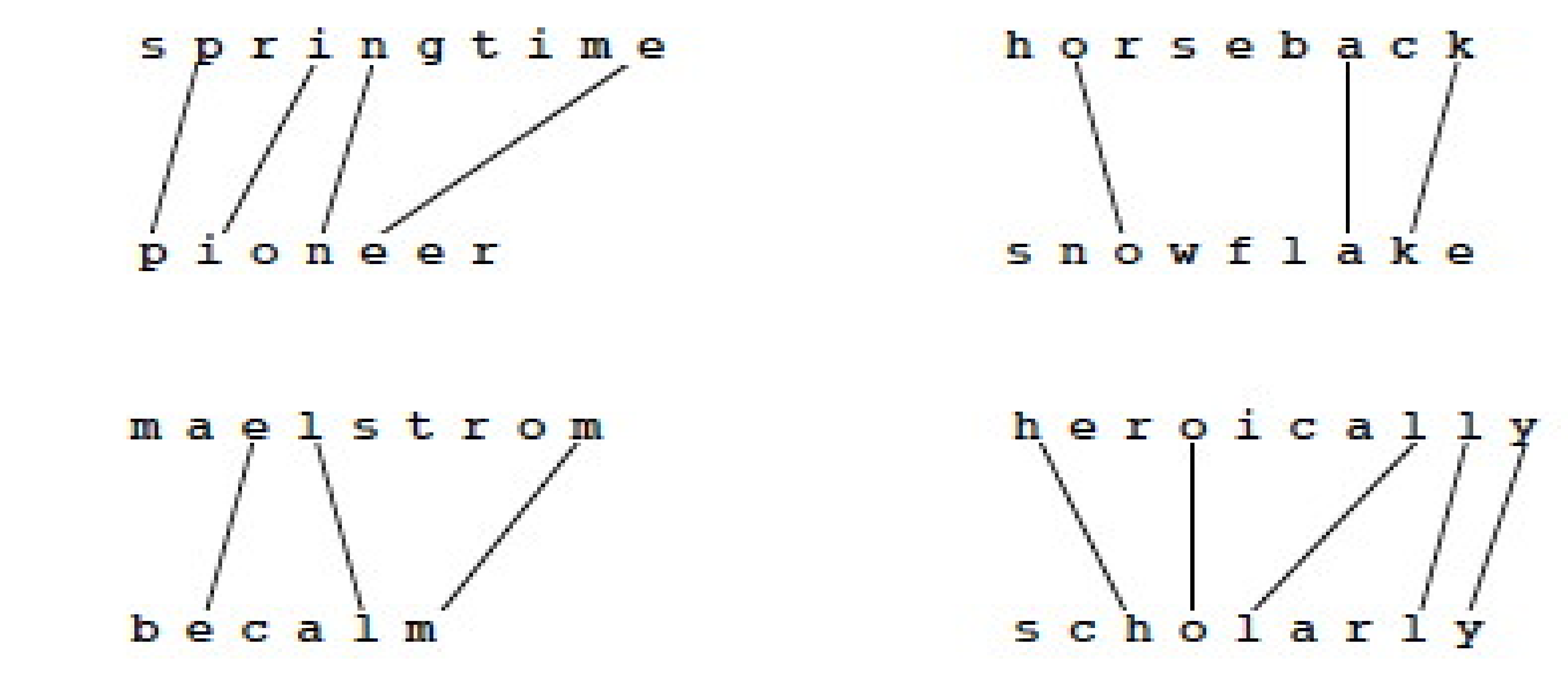
```
int p[11] = {0, 1, 5, 8, 9, 10, 17, 17, 20, 24, 30};


int dpcutrod(int n)
{
    int r[11];
    r[0] = 0;
    for (int j = 1; j <= n; j++) {
        int q = -1;
        for (int i = 1; i <= j; i++)
            q = fmax(q, p[i] + r[j - i]);
        r[j] = q;
    }
    return r[n];
}
```

Graphs

- Drum minim
- Drum maxim

Largest Common Subsequence

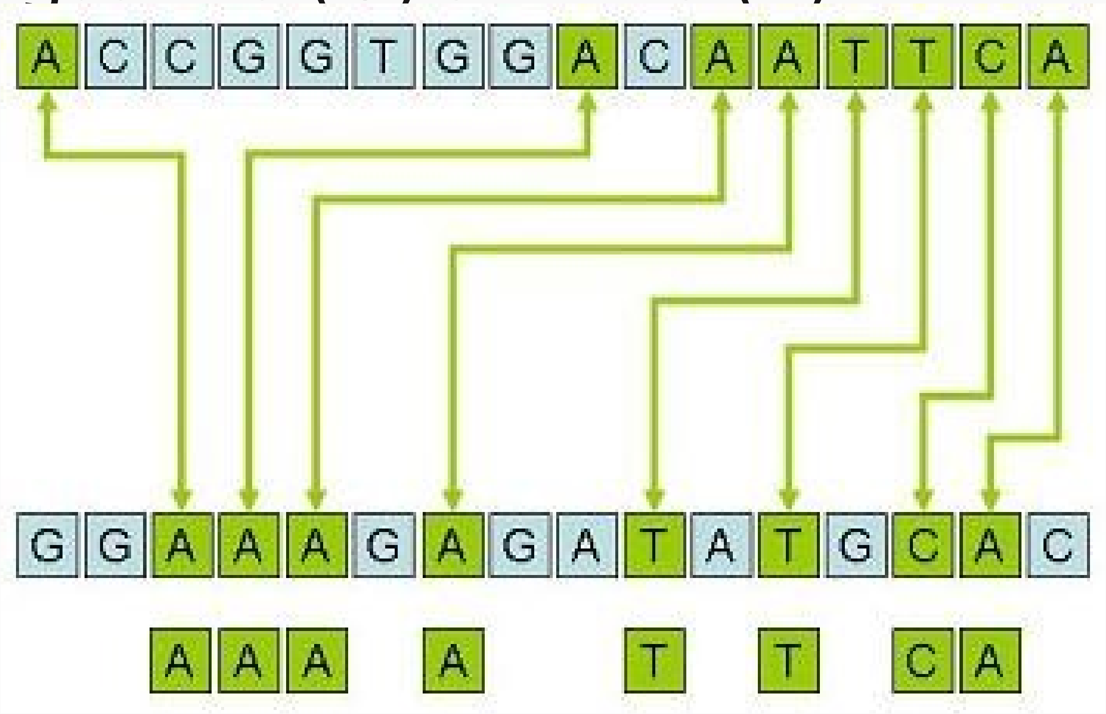
O secvenţă dintr-un şir de caractere : o mulţime de caractere (nu neapărat consecutive) aflate în ordine (de la stânga la dreapta) în şirul de caractere dat.

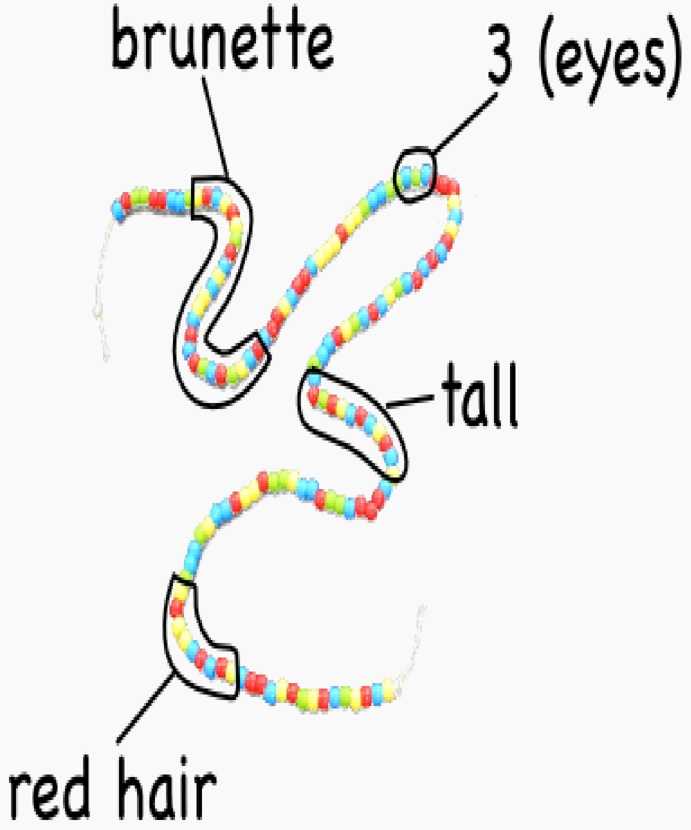


Compararea secvențelor ADN

O secvenţă ADN este compusă din patru tipuri diferite de molecule organice : adenină (A), citozină (C),

guanină (G) si timină (T).





Abordare Naivă : $O(2^n nm)$

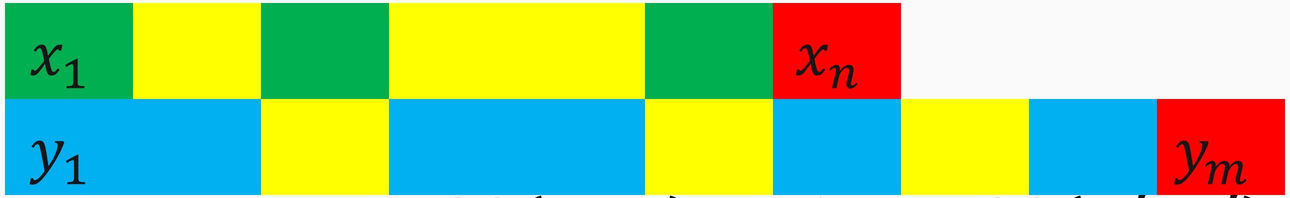
X şi Y şiruri de caractere de lungimi n şi m ($n \leq m$)

1) Generarea tuturor secvenţelor din X (2^n)

2) Verificarea prezenţei unei secvenţe în şirul Y.

```
bool verify (char* s, int l, char* Y, int m){
    int last=-1,i,j;
    for(i=0;i<l;i++){
        for(j=last+1;j<m;j++){
            if(s[i]==Y[j])    {last=j; break;}
        }
        if(j==m) return false; // caracterul nu a fost gasit
    }
    return true;
}
```

Relația de recurență



$$x_n = y_m \rightarrow LCS(X, Y) = 1 + LCS(X', Y')$$

$$x_n \neq y_m \rightarrow \mathrm{LCS}(\mathrm{X}, \mathrm{Y}) = max \begin{cases} \mathrm{LCS}(\mathrm{X}', \mathrm{Y}) \\ \mathrm{LCS}(\mathrm{X}, \mathrm{Y}') \end{cases}$$

$$\boldsymbol{LCS(i, j) = LCS\{X(1 \ldots i), Y(1 \ldots j)\}}$$

$$LCS(m, n) = \begin{cases} 1 + LCS(m-1, n-1) \; daca \; x_n = y_m \\ max(LCS(m, n-1), LCS(m-1, n)) \end{cases}$$

Relatia de Recurenţă : Tabel LCS

Calculul valorilor din tabel: pe linii, de sus în jos

	X		x_1	x_2	…	…	…	x_{n-1}	x_n
Y	$LCS(i,j)$	0	1	2	…	…	…	$n-1$	n
	0	0	0	0	0	0	0	0	0
y_1	1	0							
y_2	2	0							
…	…	0							
…	…	0							
y_{m-1}	$m-1$	0							
y_m	m	0							

Exemplu : X=ABCDA şi Y=ACBDEA

	X		A	B	C	D	A
Y	$LCS(i,j)$	0	1	2	3	4	5
	0	0	0	0	0	0	0
A	1	0	1	1	1	1	1
C	2	0	1	1	2	2	2
B	3	0	1	2	2	2	2
D	4	0	1	2	2	3	3
E	5	0	1	2	2	3	3
A	6	0	1	2	2	3	4

Memorarea Deciziilor Luate

➢Trei tipuri de mutări posibile la calculul unui element : diagonală (cazul 1), stânga sau sus (cazul 2).

➢Determinarea secvenţei prin parcurgerea mutărilor

		A	B	C	D	A
	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0	1	1	2	2	2
B	0	1	2	2	2	2
D	0	1	2	2	3	3
E	0	1	2	2	3	3
A	0	1	2	2	3	4

LCS - "ACDA"

Egalitate Caz 2 ⇒ Sus : A C D A

	X		A	B	C	D	A
Y	$LCS(i,j)$	0	1	2	3	4	5
	0	0	0	0	0	0	0
A	1	0	1	1	1	1	1
C	2	0	1	1	2	2	2
B	3	0	1	2	2	2	2
D	4	0	1	2	2	3	3
E	5	0	1	2	2	3	3
A	6	0	1	2	2	3	4

Egalitate Caz 2 ⇒ Stanga : A B D A

	X		A	B	C	D	A
Y	$LCS(i,j)$	0	1	2	3	4	5
	0	0	0	0	0	0	0
A	1	0	1	1	1	1	1
C	2	0	1	1	2	2	2
B	3	0	1	2	2	2	2
D	4	0	1	2	2	3	3
E	5	0	1	2	2	3	3
A	6	0	1	2	2	3	4

Matrix-chain multiplication

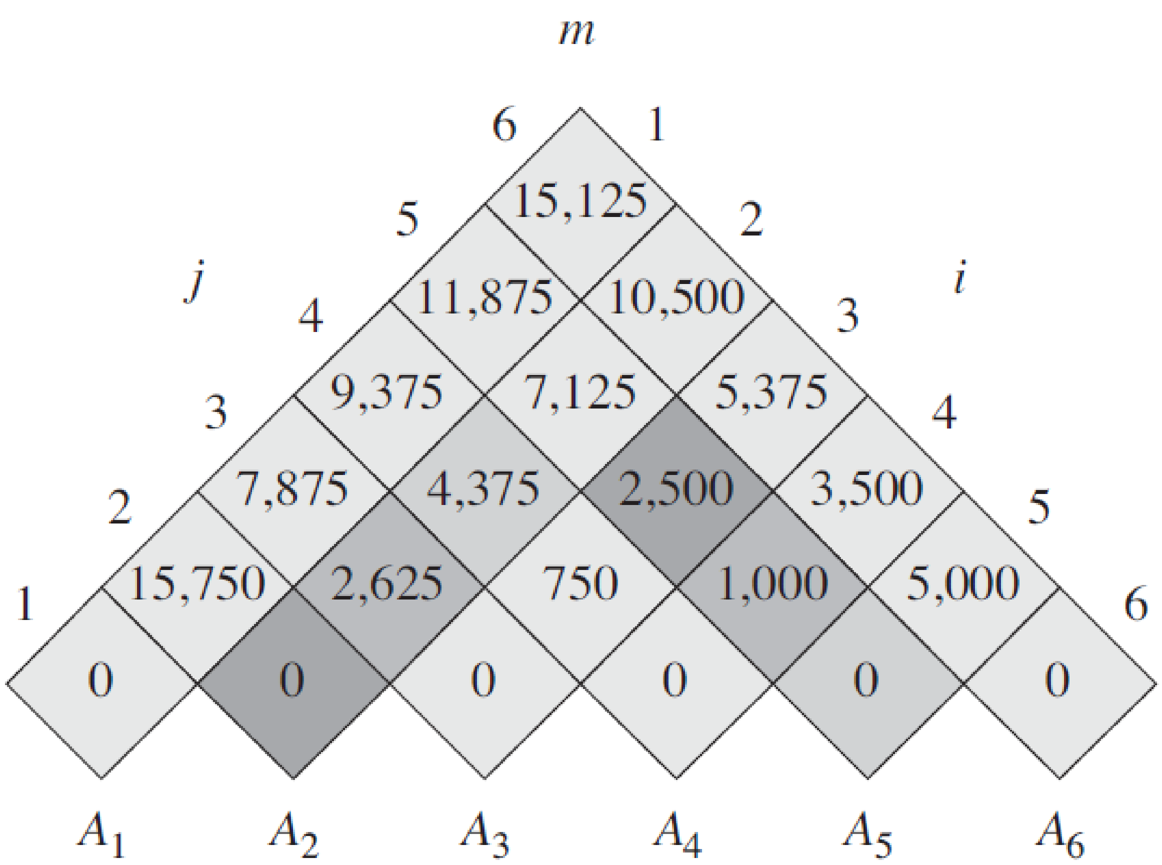
- $(A_1(A_2(A_3A_4)))$
- $(A_1((A_2A_3)A_4))$
- $((A_1A_2)(A_3A_4))$
- $((A_1(A_2A_3))A_4)$
- $(((A_1A_2)A_3)A_4)$

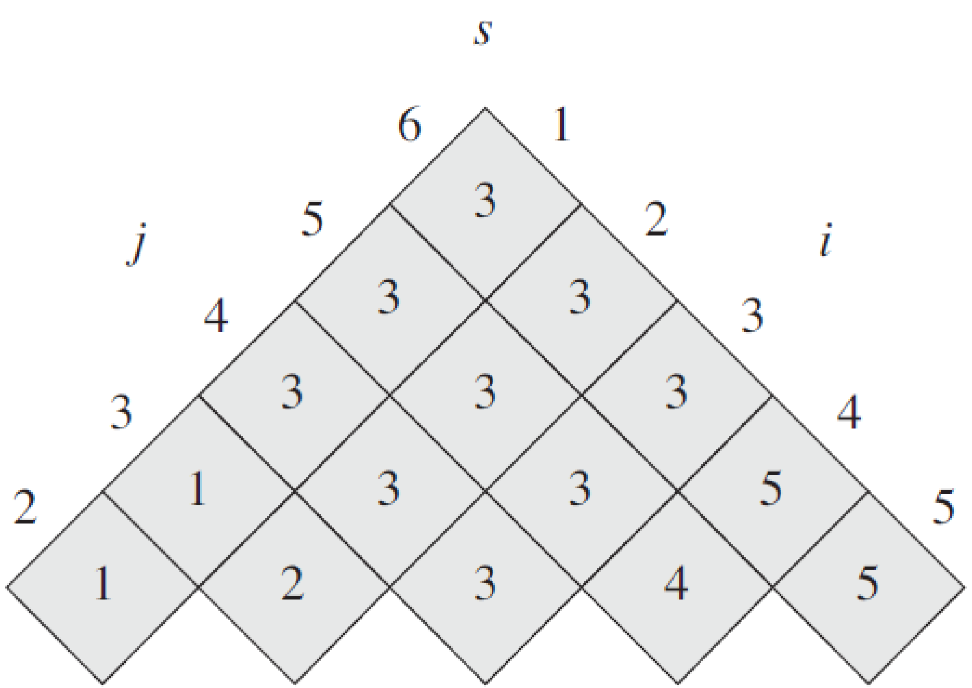
matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

Implementare Matrix-Chain multiplication

```
void matrixChainOrder()
{
    for (int i = 1; i <= n; i++)
        m[i][i] = 0;
    for (int l = 2; l <= n; l++)
        for (int i = 1; i <= n - l + 1; i++) {
            int j = i + l - 1;
            m[i][j] = 10000000;
            for (int k = i; k <= j - 1; k++) {
                int q = m[i][k] + m[k + 1][j] +
 p[i - 1] * p[k] * p[j];
                if (q < m[i][j]) {
                    m[i][j] = q;
                    s[i][j] = k;
                }
            }
        }
}
```

Matrix-chain Multiplication results





matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25