Lezione 11

Sommario della Lezione

Introduzione alla tecnica Programmazione Dinamica

Ricordiamo i passi fondamentali degli algoritmi basati sulla tecnica Divide-et-Impera per risolvere un dato problema algoritmico:

Ricordiamo i passi fondamentali degli algoritmi basati sulla tecnica Divide-et-Impera per risolvere un dato problema algoritmico:

1. Risolvi il problema direttamente se esso è di dimensione sufficientemente piccola,

Ricordiamo i passi fondamentali degli algoritmi basati sulla tecnica Divide-et-Impera per risolvere un dato problema algoritmico:

1. Risolvi il problema direttamente se esso è di dimensione sufficientemente piccola, altrimenti
2. Dividi il problema in sottoproblemi di dimensione inferiore

Ricordiamo i passi fondamentali degli algoritmi basati sulla tecnica Divide-et-Impera per risolvere un dato problema algoritmico:

1. Risolvi il problema direttamente se esso è di dimensione sufficientemente piccola, altrimenti
2. Dividi il problema in sottoproblemi di dimensione inferiore
3. Risolvi (ricorsivamente) i sottoproblemi di dimensione inferiore

Ricordiamo i passi fondamentali degli algoritmi basati sulla tecnica Divide-et-Impera per risolvere un dato problema algoritmico:

1. Risolvi il problema direttamente se esso è di dimensione sufficientemente piccola, altrimenti
2. Dividi il problema in sottoproblemi di dimensione inferiore
3. Risolvi (ricorsivamente) i sottoproblemi di dimensione inferiore
4. Combina le soluzioni dei sottoproblemi in una soluzione per il problema originale

Per i problemi algoritmici visti finora, i sottoproblemi che si ottenevano dalla applicazione del passo 2. dello schema erano **diversi**,

Per i problemi algoritmici visti finora, i sottoproblemi che si ottenevano dalla applicazione del passo 2. dello schema erano **diversi**,

pertanto ciascuno di essi veniva **individualmente** risolto nella relativa chiamata ricorsiva.

Per i problemi algoritmici visti finora, i sottoproblemi che si ottenevano dalla applicazione del passo 2. dello schema erano **diversi**,

pertanto ciascuno di essi veniva **individualmente** risolto nella relativa chiamata ricorsiva.

In molte situazioni, i sottoproblemi ottenuti al passo 2. potrebbero essere **simili**, o addirittura **uguali**.

Per i problemi algoritmici visti finora, i sottoproblemi che si ottenevano dalla applicazione del passo 2. dello schema erano **diversi**,

pertanto ciascuno di essi veniva **individualmente** risolto nella relativa chiamata ricorsiva.

In molte situazioni, i sottoproblemi ottenuti al passo 2. potrebbero essere **simili**, o addirittura **uguali**.

In tal caso, l'algoritmo basato su Divide-et-Impera risolverebbe lo **stesso** problema più volte, svolgendo lavoro inutile.

In situazioni siffatte (ovvero quando i sottoproblemi di un dato problema algoritmico tendono ad essere “simili”, o addirittura “uguali”), è utile impiegare la tecnica della Programmazione Dinamica.

In situazioni siffatte (ovvero quando i sottoproblemi di un dato problema algoritmico tendono ad essere "simili", o addirittura "uguali"), è utile impiegare la tecnica della Programmazione Dinamica.

Questa tecnica è simile alla tecnica Divide-et-Impera, con in più l'accortezza di risolvere ogni sottoproblema **una volta soltanto**.

In situazioni siffatte (ovvero quando i sottoproblemi di un dato problema algoritmico tendono ad essere "simili", o addirittura "uguali"), è utile impiegare la tecnica della Programmazione Dinamica.

Questa tecnica è simile alla tecnica Divide-et-Impera, con in più l'accortezza di risolvere ogni sottoproblema **una volta soltanto**.

Gli algoritmi basati su Programmazione Dinamica risultano quindi essere più efficienti nel caso in cui i sottoproblemi di un dato problema tendono a ripetersi.

Esempio

La sequenza $F_0, F_1, F_2, F_3, \ldots$ dei numeri di Fibonacci è definita dall'equazione di ricorrenza:

$$F_0 = F_1 = 1, \quad \text{e per ogni} \quad n \geq 2 \quad F_n = F_{n-1} + F_{n-2}.$$

Esempio

La sequenza $F_0, F_1, F_2, F_3, \ldots$ dei numeri di Fibonacci è definita dall'equazione di ricorrenza:

$$F_0 = F_1 = 1, \quad \text{e per ogni} \quad n \geq 2 \quad F_n = F_{n-1} + F_{n-2}.$$

Ad esempio, abbiamo che i primi termini della sequenza sono

$$1, 1,$$

Esempio

La sequenza $F_0, F_1, F_2, F_3, \ldots$ dei numeri di Fibonacci è definita dall'equazione di ricorrenza:

$$F_0 = F_1 = 1, \quad \text{e per ogni} \quad n \geq 2 \quad F_n = F_{n-1} + F_{n-2}.$$

Ad esempio, abbiamo che i primi termini della sequenza sono

$$1, 1, 2,$$

Esempio

La sequenza $F_0, F_1, F_2, F_3, \ldots$ dei numeri di Fibonacci è definita dall'equazione di ricorrenza:

$$F_0 = F_1 = 1, \quad \text{e per ogni} \quad n \geq 2 \quad F_n = F_{n-1} + F_{n-2}.$$

Ad esempio, abbiamo che i primi termini della sequenza sono

$$1, 1, 2, 3,$$

Esempio

La sequenza $F_0, F_1, F_2, F_3, \ldots$ dei numeri di Fibonacci è definita dall'equazione di ricorrenza:

$$F_0 = F_1 = 1, \quad \text{e per ogni} \quad n \geq 2 \quad F_n = F_{n-1} + F_{n-2}.$$

Ad esempio, abbiamo che i primi termini della sequenza sono

$$1, 1, 2, 3, 5,$$

Esempio

La sequenza $F_0, F_1, F_2, F_3, \ldots$ dei numeri di Fibonacci è definita dall'equazione di ricorrenza:

$$F_0 = F_1 = 1, \quad \text{e per ogni} \quad n \geq 2 \quad F_n = F_{n-1} + F_{n-2}.$$

Ad esempio, abbiamo che i primi termini della sequenza sono

$$1, 1, 2, 3, 5, 8,$$

Esempio

La sequenza $F_0, F_1, F_2, F_3, \ldots$ dei numeri di Fibonacci è definita dall'equazione di ricorrenza:

$$F_0 = F_1 = 1, \quad \text{e per ogni} \quad n \geq 2 \quad F_n = F_{n-1} + F_{n-2}.$$

Ad esempio, abbiamo che i primi termini della sequenza sono

$$1, 1, 2, 3, 5, 8, 13,$$

Esempio

La sequenza $F_0, F_1, F_2, F_3, \ldots$ dei numeri di Fibonacci è definita dall'equazione di ricorrenza:

$$F_0 = F_1 = 1, \quad \text{e per ogni} \quad n \geq 2 \quad F_n = F_{n-1} + F_{n-2}.$$

Ad esempio, abbiamo che i primi termini della sequenza sono

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

Esempio

La sequenza $F_0, F_1, F_2, F_3, \ldots$ dei numeri di Fibonacci è definita dall'equazione di ricorrenza:

$$F_0 = F_1 = 1, \quad \text{e per ogni} \quad n \geq 2 \quad F_n = F_{n-1} + F_{n-2}.$$

Ad esempio, abbiamo che i primi termini della sequenza sono

$$1, 1, 2, 3, 5, 8, 13, 21, 34,$$

Esempio

La sequenza $F_0, F_1, F_2, F_3, \ldots$ dei numeri di Fibonacci è definita dall'equazione di ricorrenza:

$$F_0 = F_1 = 1, \quad \text{e per ogni} \quad n \geq 2 \quad F_n = F_{n-1} + F_{n-2}.$$

Ad esempio, abbiamo che i primi termini della sequenza sono

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55,$$

Esempio

La sequenza $F_0, F_1, F_2, F_3, \ldots$ dei numeri di Fibonacci è definita dall'equazione di ricorrenza:

$$F_0 = F_1 = 1, \quad \text{e per ogni} \quad n \geq 2 \quad F_n = F_{n-1} + F_{n-2}.$$

Ad esempio, abbiamo che i primi termini della sequenza sono

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,$$

Esempio

La sequenza $F_0, F_1, F_2, F_3, \ldots$ dei numeri di Fibonacci è definita dall'equazione di ricorrenza:

$$F_0 = F_1 = 1, \quad \text{e per ogni} \quad n \geq 2 \quad F_n = F_{n-1} + F_{n-2}.$$

Ad esempio, abbiamo che i primi termini della sequenza sono

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,$$

Esempio

La sequenza $F_0, F_1, F_2, F_3, \ldots$ dei numeri di Fibonacci è definita dall'equazione di ricorrenza:

$$F_0 = F_1 = 1, \quad \text{e per ogni} \quad n \geq 2 \quad F_n = F_{n-1} + F_{n-2}.$$

Ad esempio, abbiamo che i primi termini della sequenza sono

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, \ldots$$

Esempio

La sequenza $F_0, F_1, F_2, F_3, \ldots$ dei numeri di Fibonacci è definita dall'equazione di ricorrenza:

$$F_0 = F_1 = 1, \quad \text{e per ogni} \quad n \geq 2 \quad F_n = F_{n-1} + F_{n-2}.$$

Ad esempio, abbiamo che i primi termini della sequenza sono

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, \ldots$$

I numeri di Fibonacci crescono rapidamente, si può provare infatti che $F_n \approx 2^{.694n}$.

Calcolo dei Numeri di Fibonacci via D&I

```
Fib(n)                                                    (F_n = F_{n-1} + F_{n-2})
1.   IF ((n==0)||(n==1)) {
2.     RETURN 1
```

Calcolo dei Numeri di Fibonacci via D&I

```
Fib(n)                                                    (F_n = F_{n-1} + F_{n-2})
1.    IF ((n==0)||(n==1)) {
2.       RETURN 1
3.       } ELSE {
4.       RETURN Fib(n-1)+Fib(n-2)
         }
```

Calcolo dei Numeri di Fibonacci via D&I

```
Fib(n)                                                    (F_n = F_{n-1} + F_{n-2})
1.    IF ((n==0)||(n==1)) {
2.       RETURN 1
3.       } ELSE {
4.       RETURN Fib(n-1)+Fib(n-2)
         }
```

Sia $T(n)$ la complessità di `Fib`(n).

Calcolo dei Numeri di Fibonacci via D&I

```
Fib(n)                                                    (F_n = F_{n-1} + F_{n-2})
1.   IF ((n==0)||(n==1)) {
2.     RETURN 1
3.     } ELSE {
4.     RETURN Fib(n-1)+Fib(n-2)
       }
```

Sia $T(n)$ la complessità di `Fib`(n). Abbiamo che

$$T(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ \end{cases}$$

Calcolo dei Numeri di Fibonacci via D&I

```
Fib(n)                                                    (F_n = F_{n-1} + F_{n-2})
1.   IF ((n==0)||(n==1)) {
2.     RETURN 1
3.     } ELSE {
4.     RETURN Fib(n-1)+Fib(n-2)
       }
```

Sia $T(n)$ la complessità di `Fib`(n). Abbiamo che

$$T(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ T(n-1) + T(n-2) + 1 & \text{altrimenti} \end{cases}$$

Calcolo dei Numeri di Fibonacci via D&I

```
Fib(n)                                                    (F_n = F_{n-1} + F_{n-2})
1.   IF ((n==0)||(n==1)) {
2.     RETURN 1
3.     } ELSE {
4.     RETURN Fib(n-1)+Fib(n-2)
       }
```

Sia $T(n)$ la complessità di `Fib`(n). Abbiamo che

$$T(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ T(n-1) + T(n-2) + 1 & \text{altrimenti} \end{cases}$$

Poniamo $T'(n) = T(n) + 1$

Calcolo dei Numeri di Fibonacci via D&I

```
Fib(n)                                                    (F_n = F_{n-1} + F_{n-2})
1.    IF ((n==0)||(n==1)) {
2.       RETURN 1
3.       } ELSE {
4.       RETURN Fib(n-1)+Fib(n-2)
         }
```

Sia $T(n)$ la complessità di `Fib`(n). Abbiamo che

$$T(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ T(n-1) + T(n-2) + 1 & \text{altrimenti} \end{cases}$$

Poniamo $T'(n) = T(n) + 1$ e otteniamo che $T'(0) = T'(1) = 2$

Calcolo dei Numeri di Fibonacci via D&I

```
Fib(n)                                                    (F_n = F_{n-1} + F_{n-2})
1.    IF ((n==0)||(n==1)) {
2.       RETURN 1
3.       } ELSE {
4.       RETURN Fib(n-1)+Fib(n-2)
         }
```

Sia $T(n)$ la complessità di `Fib`(n). Abbiamo che

$$T(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ T(n-1) + T(n-2) + 1 & \text{altrimenti} \end{cases}$$

Poniamo $T'(n) = T(n) + 1$ e otteniamo che $T'(0) = T'(1) = 2$ mentre $\forall\ n \geq 2$

$$T'(n) = T(n) + 1$$

Calcolo dei Numeri di Fibonacci via D&I

```
Fib(n)                                                    (F_n = F_{n-1} + F_{n-2})
1.    IF ((n==0)||(n==1)) {
2.       RETURN 1
3.       } ELSE {
4.       RETURN Fib(n-1)+Fib(n-2)
         }
```

Sia $T(n)$ la complessità di `Fib`(n). Abbiamo che

$$T(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ T(n-1) + T(n-2) + 1 & \text{altrimenti} \end{cases}$$

Poniamo $T'(n) = T(n) + 1$ e otteniamo che $T'(0) = T'(1) = 2$ mentre $\forall\ n \geq 2$

$$T'(n) = T(n) + 1 = T(n-1) + T(n-2) + 1 + 1$$

Calcolo dei Numeri di Fibonacci via D&I

```
Fib(n)                                                    (F_n = F_{n-1} + F_{n-2})
1.    IF ((n==0)||(n==1)) {
2.       RETURN 1
3.       } ELSE {
4.       RETURN Fib(n-1)+Fib(n-2)
         }
```

Sia $T(n)$ la complessità di `Fib`(n). Abbiamo che

$$T(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ T(n-1) + T(n-2) + 1 & \text{altrimenti} \end{cases}$$

Poniamo $T'(n) = T(n) + 1$ e otteniamo che $T'(0) = T'(1) = 2$ mentre $\forall\ n \geq 2$

$$\begin{aligned} T'(n) &= T(n) + 1 = T(n-1) + T(n-2) + 1 + 1 \\ &= T'(n-1) + T'(n-2) \end{aligned}$$

Calcolo dei Numeri di Fibonacci via D&I

```
Fib(n)                                                    (F_n = F_{n-1} + F_{n-2})
1.    IF ((n==0)||(n==1)) {
2.       RETURN 1
3.       } ELSE {
4.       RETURN Fib(n-1)+Fib(n-2)
         }
```

Sia $T(n)$ la complessità di `Fib`(n). Abbiamo che

$$T(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ T(n-1) + T(n-2) + 1 & \text{altrimenti} \end{cases}$$

Poniamo $T'(n) = T(n) + 1$ e otteniamo che $T'(0) = T'(1) = 2$ mentre $\forall\ n \geq 2$

$$\begin{aligned} T'(n) &= T(n) + 1 = T(n-1) + T(n-2) + 1 + 1 \\ &= T'(n-1) + T'(n-2) \end{aligned}$$

una semplice induzione ci permette di provare che

$$T'(n) = 2F_n, \quad \text{ovvero} \quad T(n) = 2F_n - 1 \approx 2 \times 2^{.694n}$$

Calcolo dei Numeri di Fibonacci via D&I

```
Fib(n)                                                    (F_n = F_{n-1} + F_{n-2})
1.    IF ((n==0)||(n==1)) {
2.       RETURN 1
3.       } ELSE {
4.       RETURN Fib(n-1)+Fib(n-2)
         }
```

Sia $T(n)$ la complessità di `Fib`(n). Abbiamo che

$$T(n) = \begin{cases} 1 & \text{se } n \leq 1 \\ T(n-1) + T(n-2) + 1 & \text{altrimenti} \end{cases}$$

Poniamo $T'(n) = T(n) + 1$ e otteniamo che $T'(0) = T'(1) = 2$ mentre $\forall\ n \geq 2$

$$\begin{aligned} T'(n) &= T(n) + 1 = T(n-1) + T(n-2) + 1 + 1 \\ &= T'(n-1) + T'(n-2) \end{aligned}$$

una semplice induzione ci permette di provare che

$$T'(n) = 2F_n, \quad \text{ovvero} \quad T(n) = 2F_n - 1 \approx 2 \times 2^{.694n} \quad \text{troppo!}$$

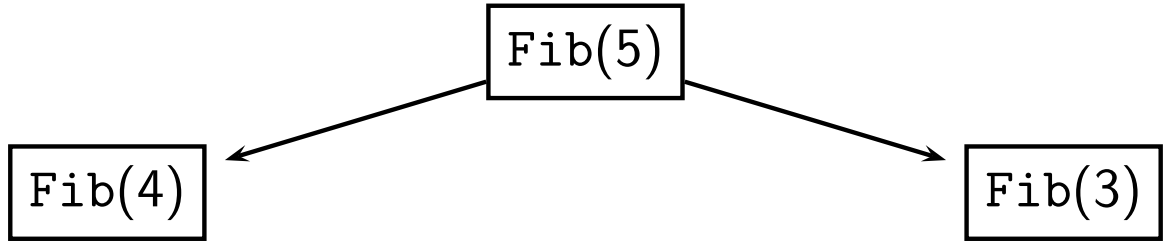
Dove sta il problema?

Dove sta il problema?

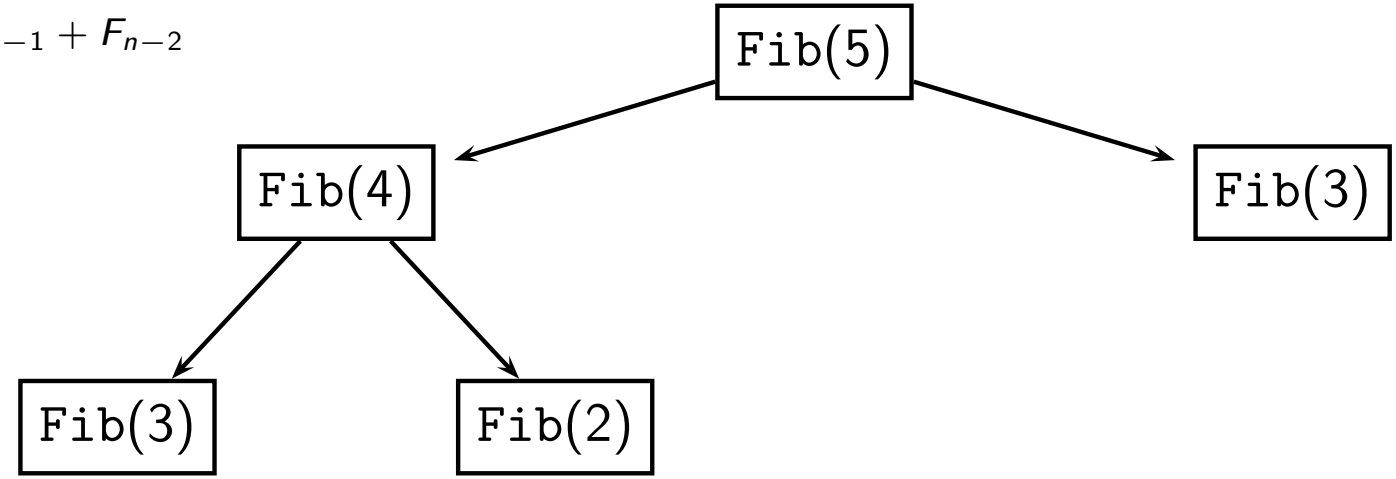
Il problema sta nel fatto che l'algoritmo `Fib`(n) viene chiamato sullo stesso input molte volte, e ciò è chiaramente inutile.

$$F_n = F_{n-1} + F_{n-2}$$

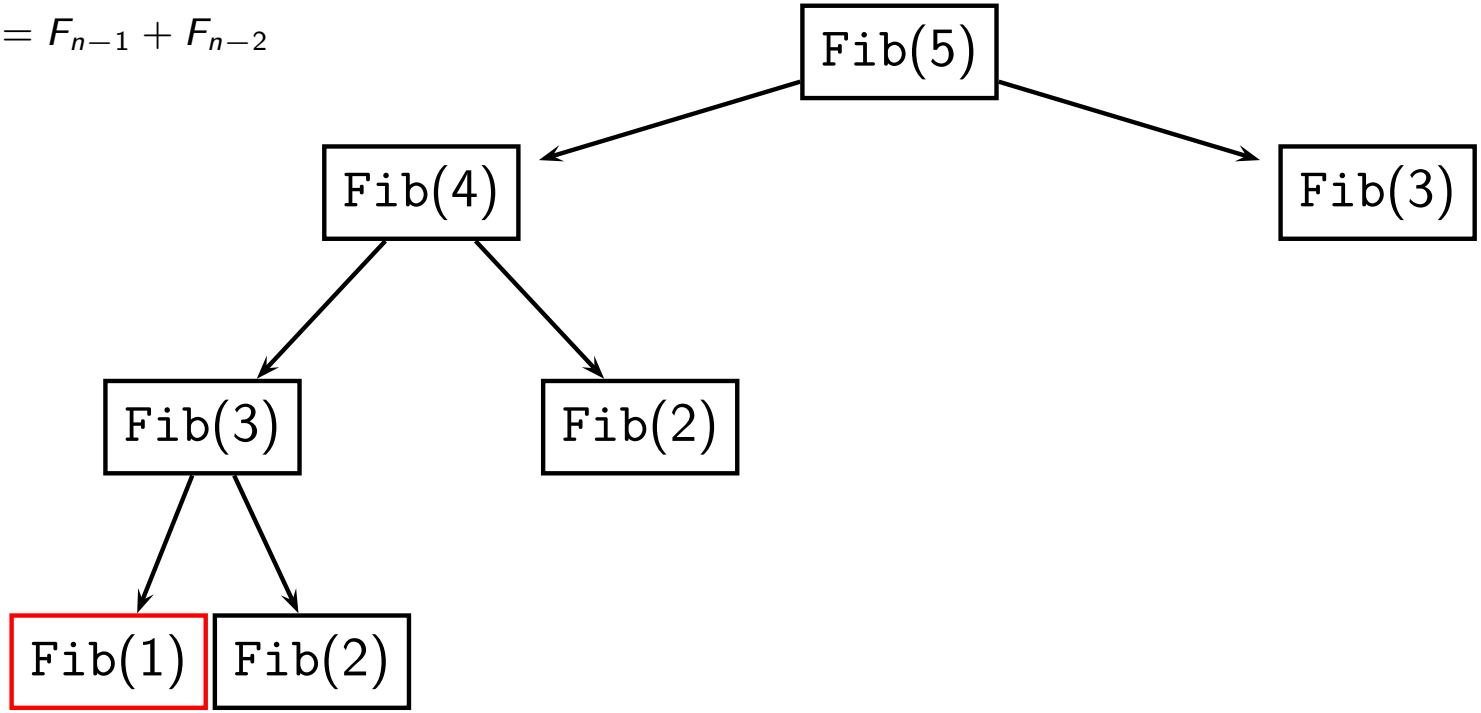
$F_n = F_{n-1} + F_{n-2}$



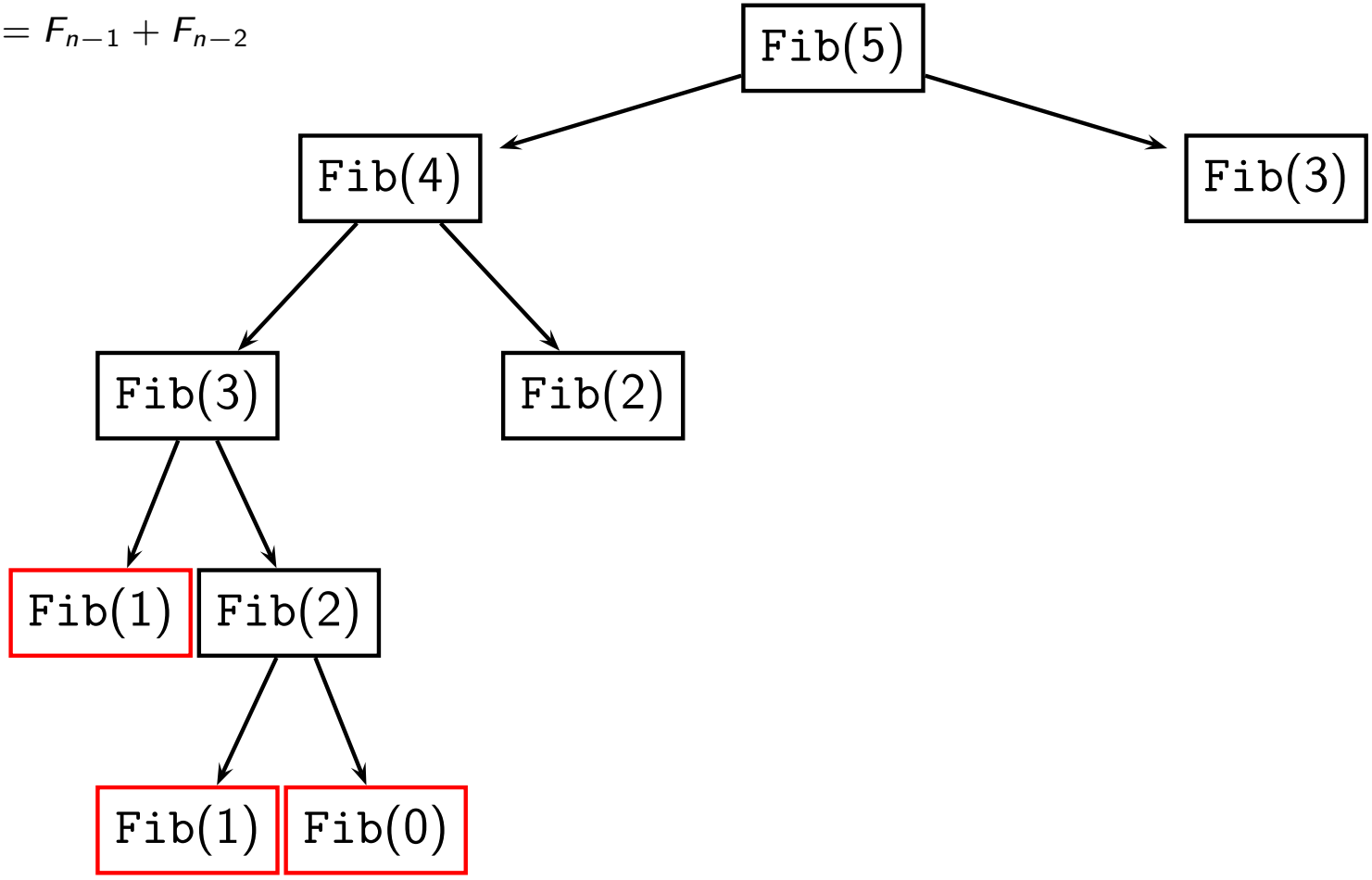
$F_n = F_{n-1} + F_{n-2}$



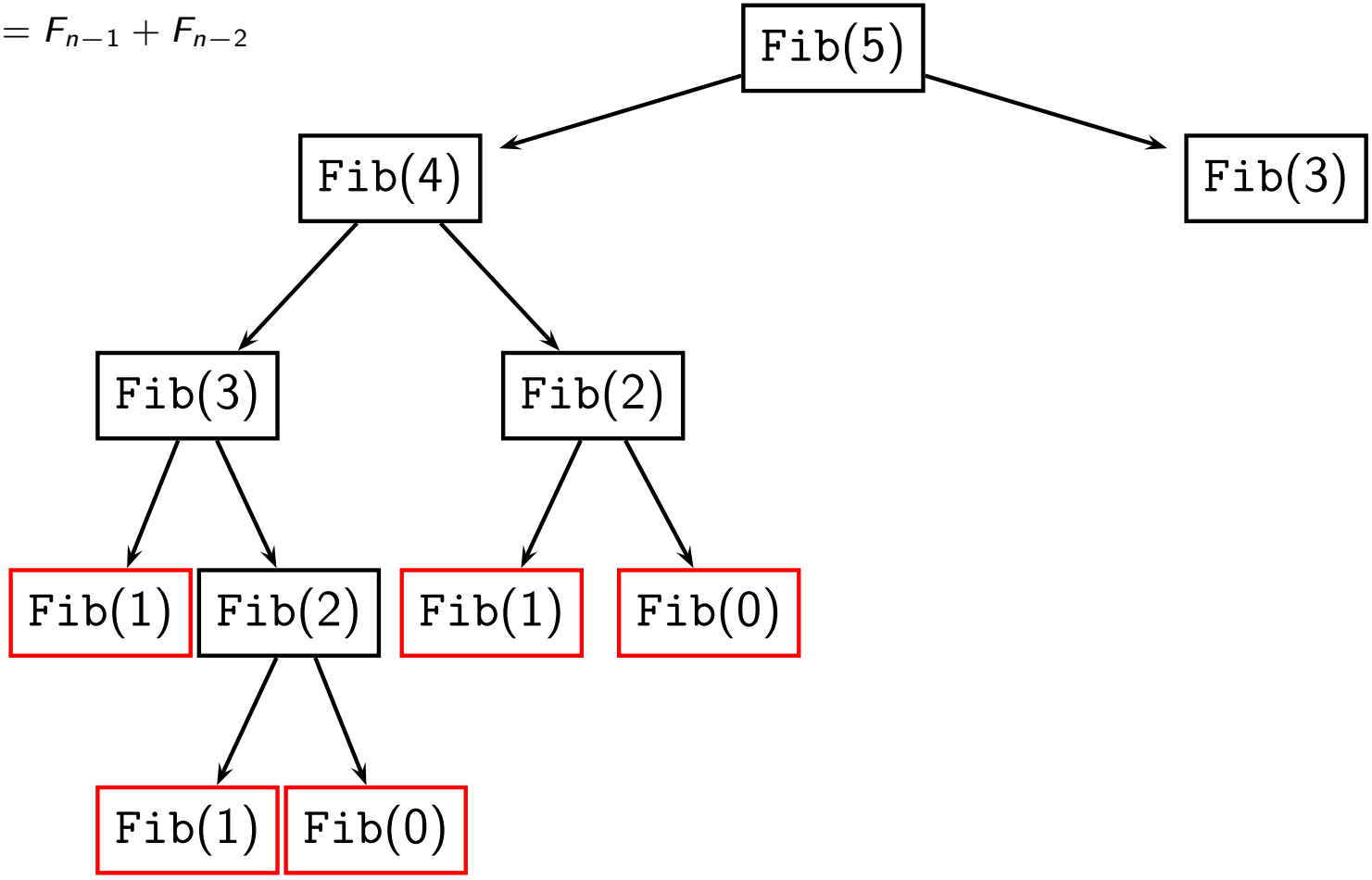
$F_n = F_{n-1} + F_{n-2}$



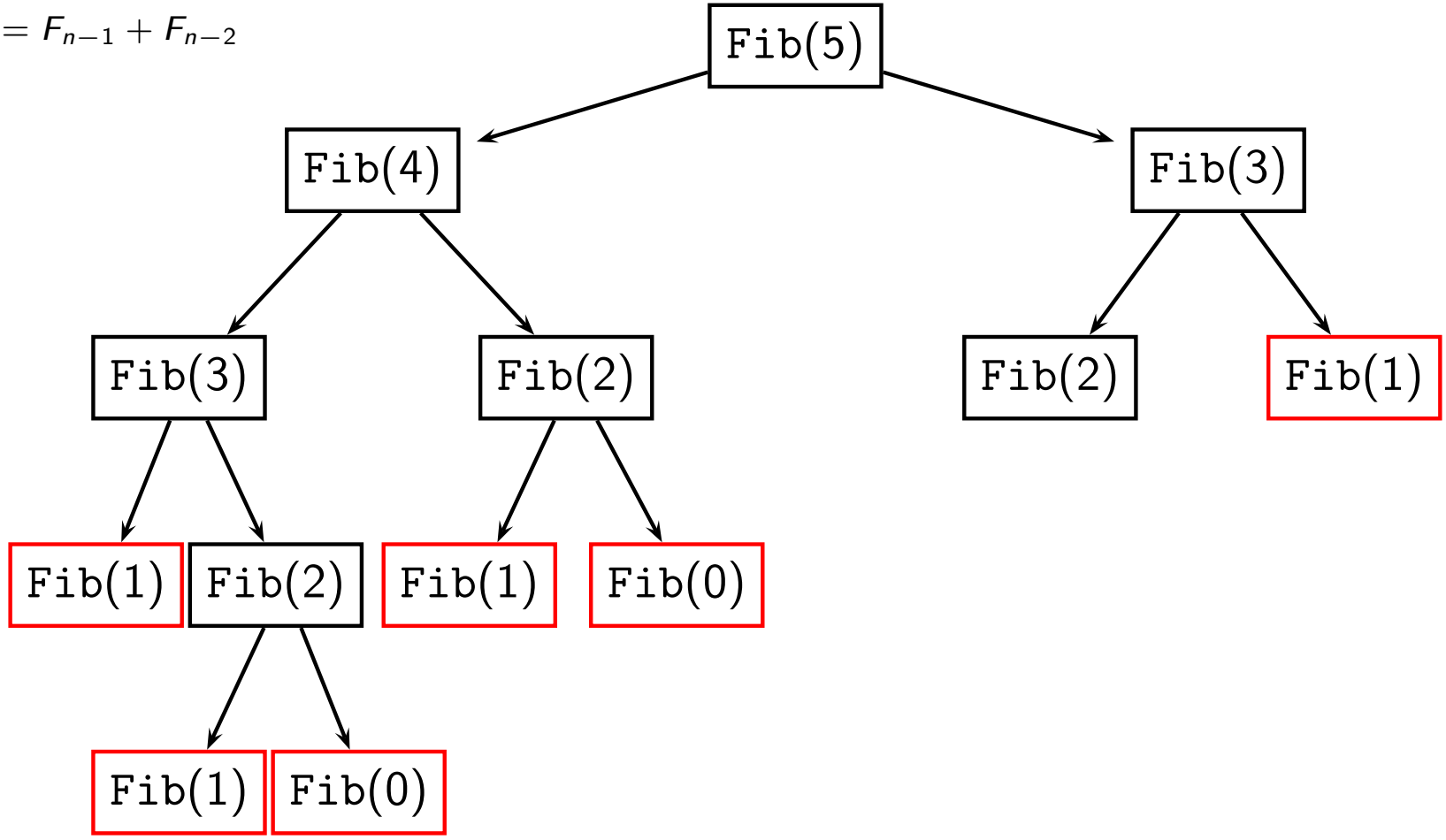
$F_n = F_{n-1} + F_{n-2}$



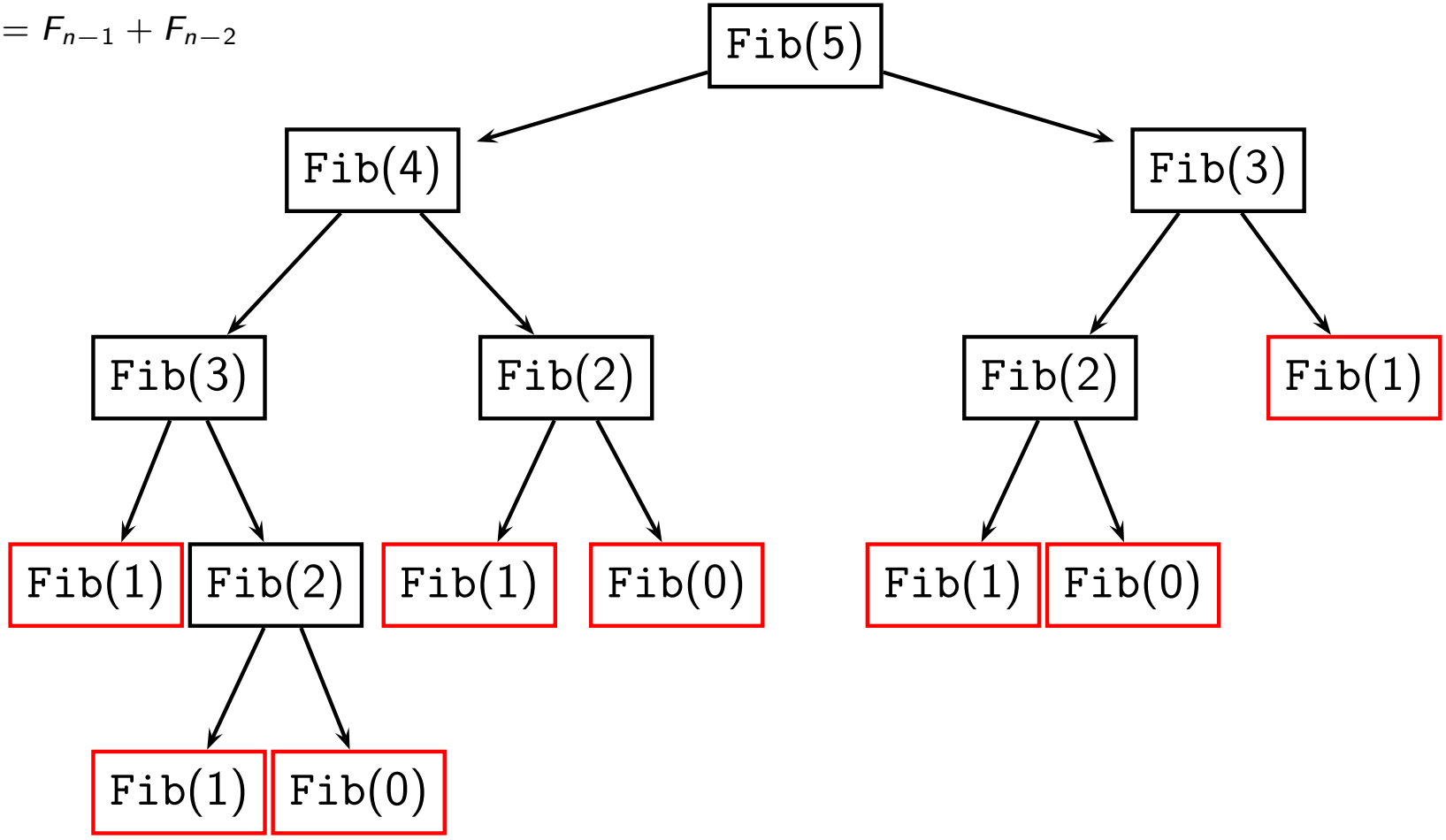
$F_n = F_{n-1} + F_{n-2}$



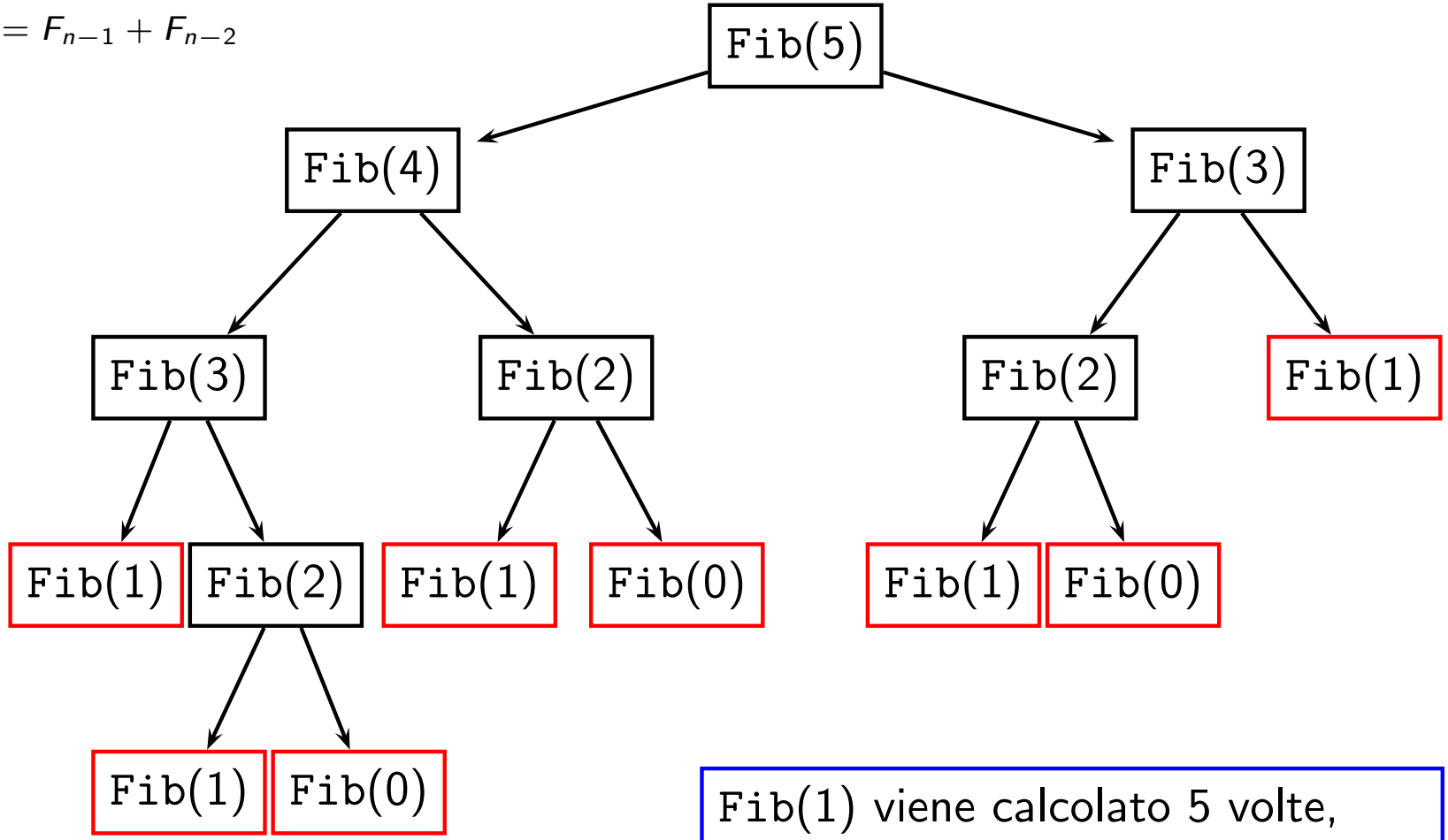
$F_n = F_{n-1} + F_{n-2}$


Fib(5)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(2)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

$F_n = F_{n-1} + F_{n-2}$


Fib(5)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(2)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

$F_n = F_{n-1} + F_{n-2}$



`Fib(1)` viene calcolato 5 volte, `Fib(0)` e `Fib(2)` 3 volte, `Fib(3)` 2 volte

Memorizziamo allora in un vettore i valori $\texttt{Fib}(i), i < n$ quando li si calcola per **prima volta**,

Memorizziamo allora in un vettore i valori $\texttt{Fib}(i), i < n$ quando li si calcola per **prima volta**, cosicchè in future chiamate ricorsive a $\texttt{Fib}(i)$ non ci sarà piú bisogno di calcolarli,

Memorizziamo allora in un vettore i valori $\texttt{Fib}(i), i < n$ quando li si calcola per **prima volta**, cosicchè in future chiamate ricorsive a $\texttt{Fib}(i)$ non ci sarà piú bisogno di calcolarli, ma basterà leggerli dal vettore.

Memorizziamo allora in un vettore i valori $\texttt{Fib}(i), i < n$ quando li si calcola per **prima volta**, cosicchè in future chiamate ricorsive a $\texttt{Fib}(i)$ non ci sarà piú bisogno di calcolarli, ma basterà leggerli dal vettore.

Risparmiamo tempo di calcolo alle spese di un piccolo aumento di occupazione di memoria.

Memorizziamo allora in un vettore i valori $\texttt{Fib}(i), i < n$ quando li si calcola per **prima volta**, cosicchè in future chiamate ricorsive a $\texttt{Fib}(i)$ non ci sarà piú bisogno di calcolarli, ma basterà leggerli dal vettore.

Risparmiamo tempo di calcolo alle spese di un piccolo aumento di occupazione di memoria.

Il seguente algoritmo usa questa osservazione.

Memorizziamo allora in un vettore i valori $\texttt{Fib}(i), i < n$ quando li si calcola per **prima volta**, cosicchè in future chiamate ricorsive a $\texttt{Fib}(i)$ non ci sarà piú bisogno di calcolarli, ma basterà leggerli dal vettore.

Risparmiamo tempo di calcolo alle spese di un piccolo aumento di occupazione di memoria.

Il seguente algoritmo usa questa osservazione. I valori intermedi per il calcolo del numero di Fibonacci F_n vengono memorizzati in un array $a = a[0]a[1]\ldots a[n]$.

```
MemFib(n)
1. IF ((n==0)||(n==1)) {
2.     RETURN 1
```

Memorizziamo allora in un vettore i valori $\texttt{Fib}(i), i < n$ quando li si calcola per **prima volta**, cosicchè in future chiamate ricorsive a $\texttt{Fib}(i)$ non ci sarà piú bisogno di calcolarli, ma basterà leggerli dal vettore.

Risparmiamo tempo di calcolo alle spese di un piccolo aumento di occupazione di memoria.

Il seguente algoritmo usa questa osservazione. I valori intermedi per il calcolo del numero di Fibonacci F_n vengono memorizzati in un array $a = a[0]a[1]\ldots a[n]$.

```
MemFib(n)
1. IF ((n==0)||(n==1)) {
2.     RETURN 1
3.     } ELSE {
4.     IF(a[n] non è definito)
```

Memorizziamo allora in un vettore i valori $\texttt{Fib}(i), i < n$ quando li si calcola per **prima volta**, cosicchè in future chiamate ricorsive a $\texttt{Fib}(i)$ non ci sarà piú bisogno di calcolarli, ma basterà leggerli dal vettore.

Risparmiamo tempo di calcolo alle spese di un piccolo aumento di occupazione di memoria.

Il seguente algoritmo usa questa osservazione. I valori intermedi per il calcolo del numero di Fibonacci F_n vengono memorizzati in un array $a = a[0]a[1]\ldots a[n]$.

```
MemFib(n)
1. IF ((n==0)||(n==1)) {
2.      RETURN 1
3.      } ELSE {
4.      IF(a[n] non è definito)
5.          a[n]= MemFib(n-1)+ MemFib(n-2)
6.       }
```

Memorizziamo allora in un vettore i valori $\texttt{Fib}(i), i < n$ quando li si calcola per **prima volta**, cosicchè in future chiamate ricorsive a $\texttt{Fib}(i)$ non ci sarà piú bisogno di calcolarli, ma basterà leggerli dal vettore.

Risparmiamo tempo di calcolo alle spese di un piccolo aumento di occupazione di memoria.

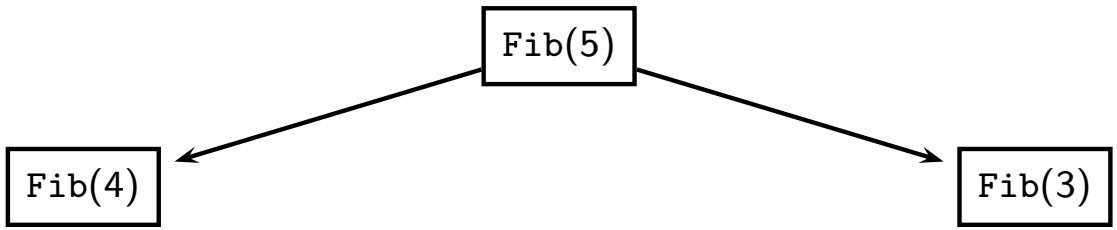
Il seguente algoritmo usa questa osservazione. I valori intermedi per il calcolo del numero di Fibonacci F_n vengono memorizzati in un array $a = a[0]a[1]\ldots a[n]$.

```
MemFib(n)
1. IF ((n==0)||(n==1)) {
2.      RETURN 1
3.      } ELSE {
4.      IF(a[n] non è definito)
5.          a[n]= MemFib(n-1)+ MemFib(n-2)
6.       }
7. RETURN a[n]
```

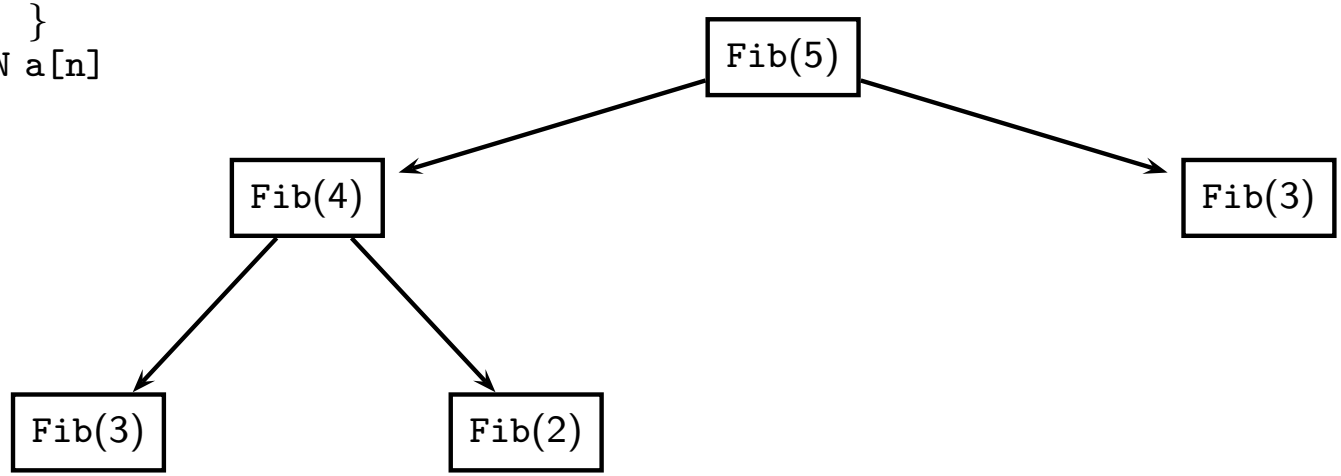
```
MemFib(n)
1. IF ((n==0)||(n==1)) {
2.     RETURN 1
3.     } ELSE {
4.     IF(a[n] non è definito)
5.         a[n]= MemFib(n-1)+ MemFib(n-2)
6.       }
7. RETURN a[n]
```

Fib(5)

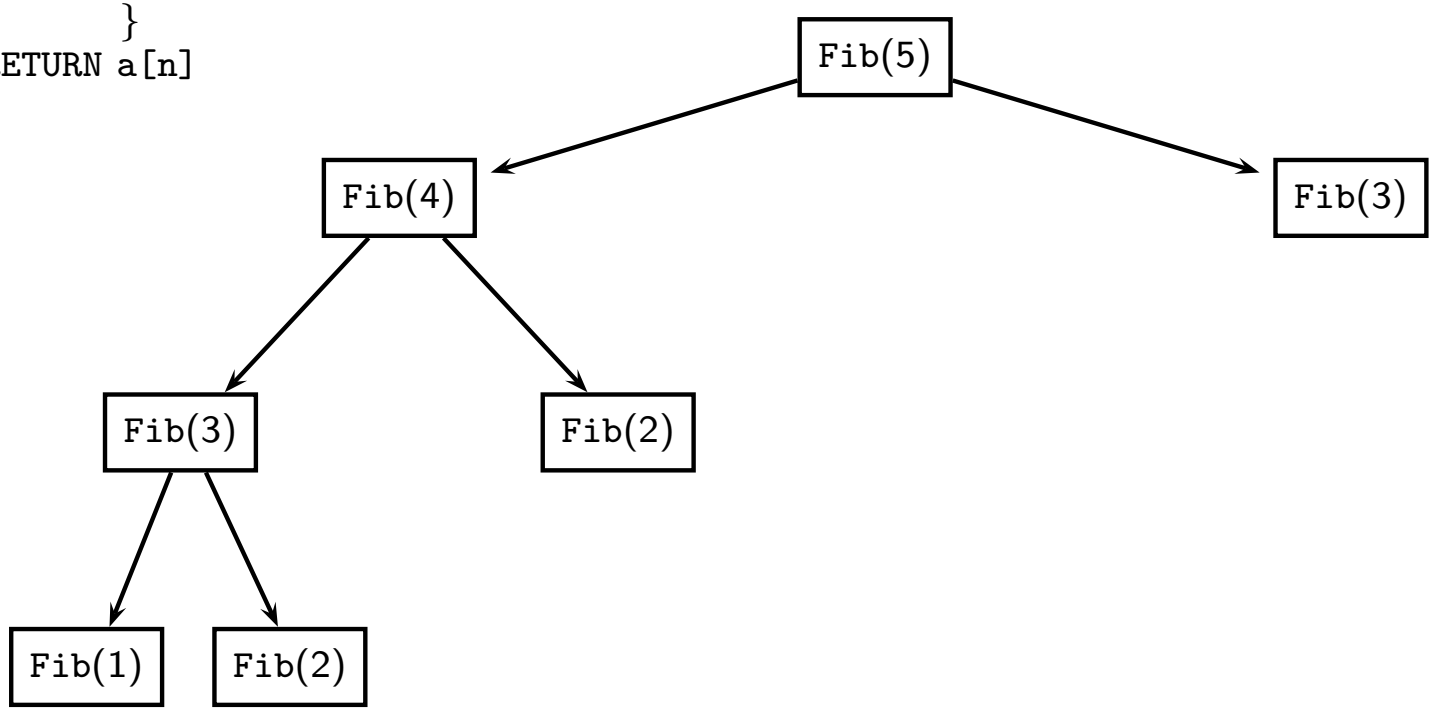
```
MemFib(n)
1. IF ((n==0)||(n==1)) {
2.     RETURN 1
3.     } ELSE {
4.     IF(a[n] non è definito)
5.         a[n]= MemFib(n-1)+ MemFib(n-2)
6.       }
7. RETURN a[n]
```



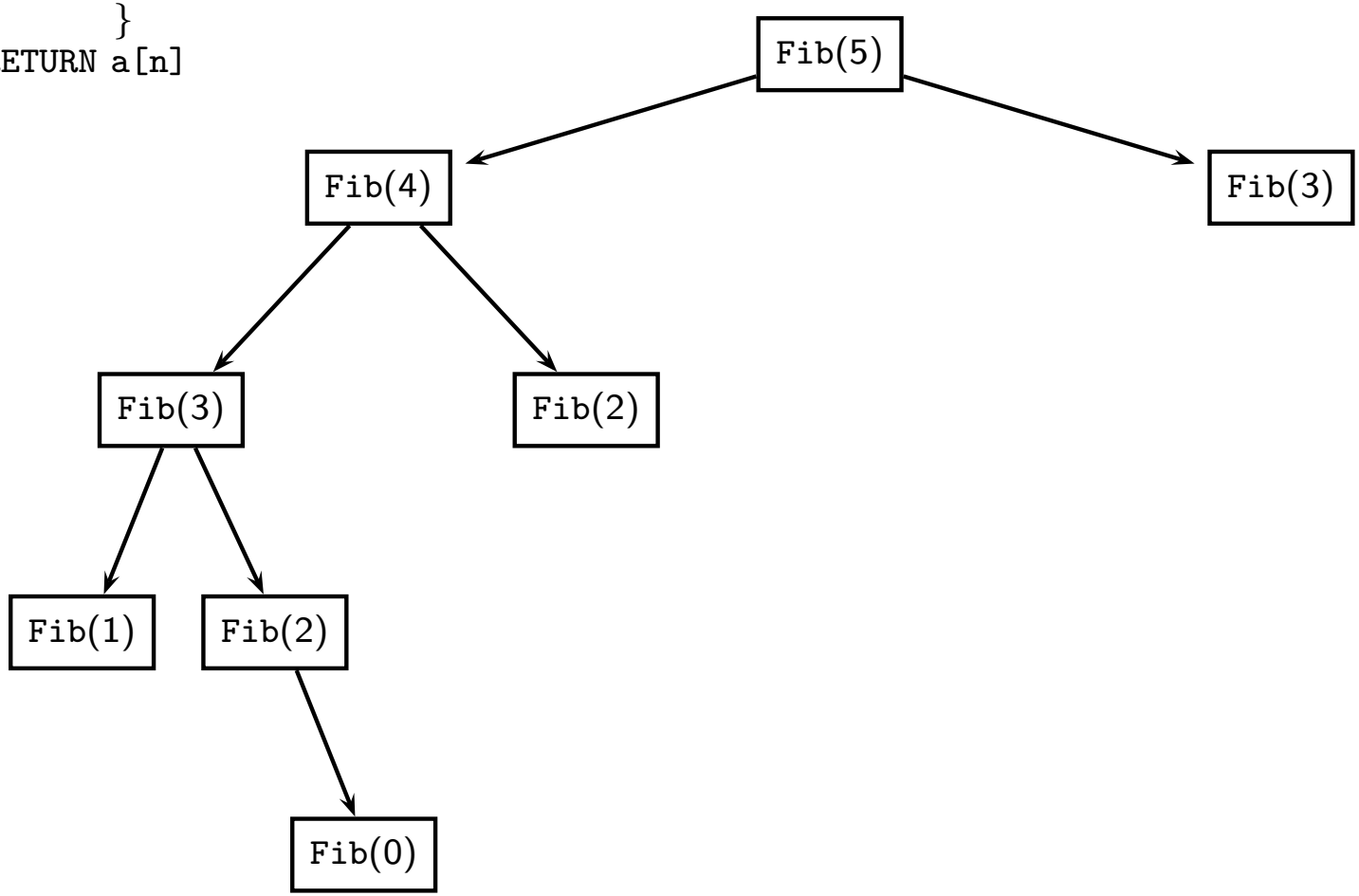
```
MemFib(n)
1. IF ((n==0)||(n==1)) {
2.      RETURN 1
3.      } ELSE {
4.      IF(a[n] non è definito)
5.          a[n]= MemFib(n-1)+ MemFib(n-2)
6.         }
7. RETURN a[n]
```



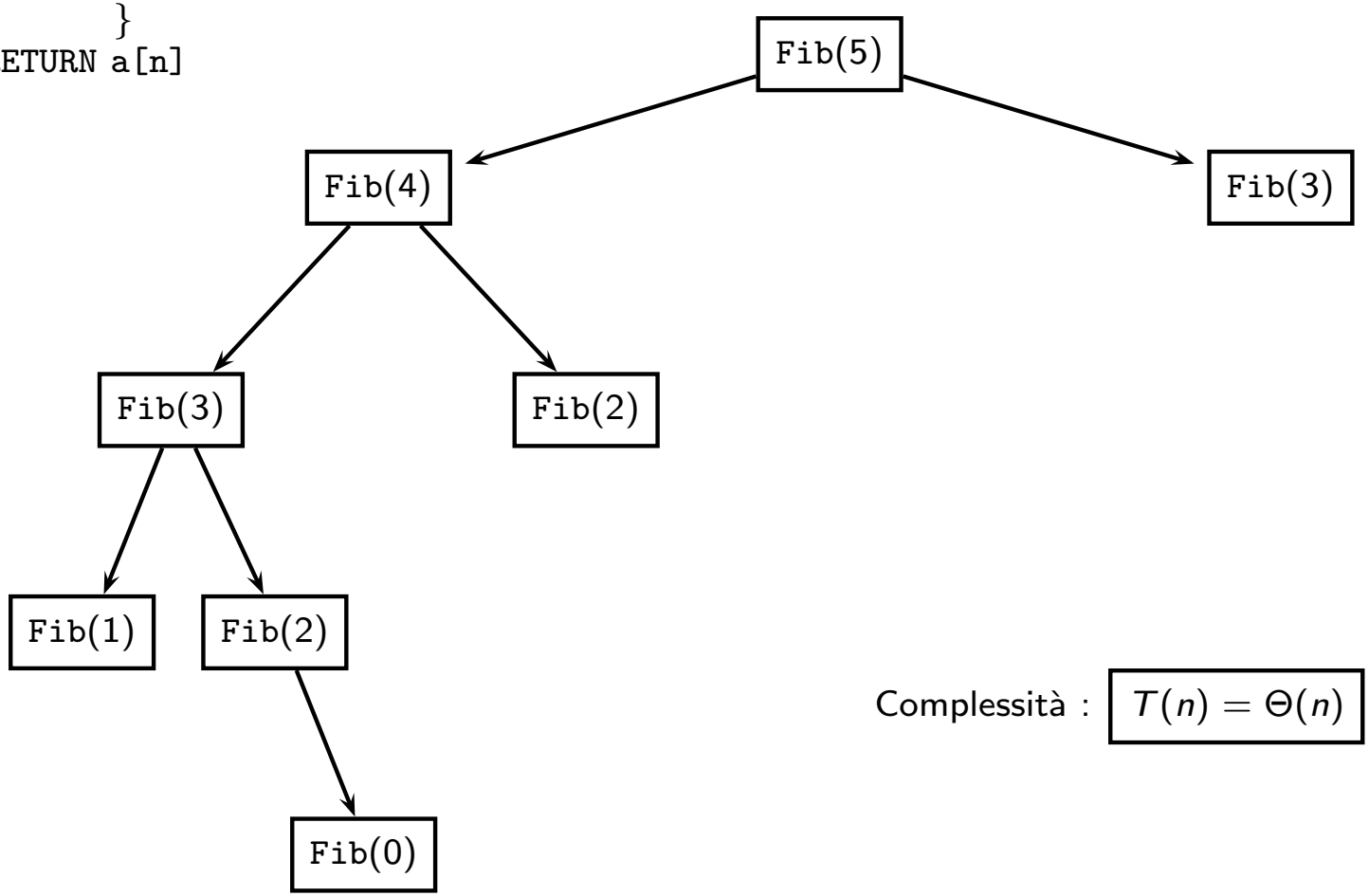
```
MemFib(n)
1. IF ((n==0)||(n==1)) {
2.     RETURN 1
3.     } ELSE {
4.     IF(a[n] non è definito)
5.         a[n]= MemFib(n-1)+ MemFib(n-2)
6.       }
7. RETURN a[n]
```



```
MemFib(n)
1. IF ((n==0)||(n==1)) {
2.     RETURN 1
3.     } ELSE {
4.     IF(a[n] non è definito)
5.         a[n]= MemFib(n-1)+ MemFib(n-2)
6.       }
7. RETURN a[n]
```



```
MemFib(n)
1. IF ((n==0)||(n==1)) {
2.     RETURN 1
3.     } ELSE {
4.     IF(a[n] non è definito)
5.         a[n]= MemFib(n-1)+ MemFib(n-2)
6.       }
7. RETURN a[n]
```



Complessità : $T(n) = \Theta(n)$

Se sviluppiamo le chiamate ricorsive di `MemFib`(5), ci rendiamo conto che le componento dell'array $a = [0] \ldots a[5]$ vengono calcolate e assegnate nell'ordine $a[2]$,

Se sviluppiamo le chiamate ricorsive di `MemFib`(5), ci rendiamo conto che le componento dell'array $a = [0] \ldots a[5]$ vengono calcolate e assegnate nell'ordine $a[2], a[3],$

Se sviluppiamo le chiamate ricorsive di `MemFib`(5), ci rendiamo conto che le componento dell'array $a = [0] \ldots a[5]$ vengono calcolate e assegnate nell'ordine $a[2], a[3], a[4],$

Se sviluppiamo le chiamate ricorsive di `MemFib`(5), ci rendiamo conto che le componento dell'array $a = [0] \ldots a[5]$ vengono calcolate e assegnate nell'ordine $a[2], a[3], a[4], a[5]$.

Se sviluppiamo le chiamate ricorsive di `MemFib`(5), ci rendiamo conto che le componento dell'array $a = [0] \dots a[5]$ vengono calcolate e assegnate nell'ordine $a[2], a[3], a[4], a[5]$. Possiamo quindi anche usare un algoritmo iterativo.

```
IterFib(n)
```

Se sviluppiamo le chiamate ricorsive di `MemFib`(5), ci rendiamo conto che le componento dell'array $a = [0] \ldots a[5]$ vengono calcolate e assegnate nell'ordine $a[2], a[3], a[4], a[5]$. Possiamo quindi anche usare un algoritmo iterativo.

```
IterFib(n)
1. a[0]= 1
```

Se sviluppiamo le chiamate ricorsive di `MemFib`(5), ci rendiamo conto che le componento dell'array $a = [0] \ldots a[5]$ vengono calcolate e assegnate nell'ordine $a[2], a[3], a[4], a[5]$. Possiamo quindi anche usare un algoritmo iterativo.

```
IterFib(n)
1. a[0]= 1
2. a[1]=1
```

Se sviluppiamo le chiamate ricorsive di `MemFib`(5), ci rendiamo conto che le componento dell'array $a = [0] \ldots a[5]$ vengono calcolate e assegnate nell'ordine $a[2], a[3], a[4], a[5]$. Possiamo quindi anche usare un algoritmo iterativo.

```
IterFib(n)
1. a[0]= 1
2. a[1]=1
3. FOR(i=2, i<n+1, i=i+1) {
```

Se sviluppiamo le chiamate ricorsive di `MemFib`(5), ci rendiamo conto che le componento dell'array $a = [0] \dots a[5]$ vengono calcolate e assegnate nell'ordine $a[2], a[3], a[4], a[5]$. Possiamo quindi anche usare un algoritmo iterativo.

```
IterFib(n)
1. a[0]= 1
2. a[1]=1
3. FOR(i=2, i<n+1, i=i+1) {
4.   a[i]= a[i-1]+a[i-2]
```

Se sviluppiamo le chiamate ricorsive di `MemFib`(5), ci rendiamo conto che le componento dell'array $a = [0] \dots a[5]$ vengono calcolate e assegnate nell'ordine $a[2], a[3], a[4], a[5]$. Possiamo quindi anche usare un algoritmo iterativo.

```
IterFib(n)
1. a[0]= 1
2. a[1]=1
3. FOR(i=2, i<n+1, i=i+1) {
4.   a[i]= a[i-1]+a[i-2]
5.    }
6. RETURN a[n]
```

L'algoritmo `IterFib(n)` richiede tempo $O(n)$ e memoria $O(n)$ per calcolare l'n-esimo numero di Fibonacci F_n,

L'algoritmo `IterFib(n)` richiede tempo $O(n)$ e memoria $O(n)$ per calcolare l'n-esimo numero di Fibonacci F_n, un miglioramento esponenziale rispetto al nostro primo algoritmo `Fib(n)`.

L'algoritmo `IterFib(n)` richiede tempo $O(n)$ e memoria $O(n)$ per calcolare l'n-esimo numero di Fibonacci F_n, un miglioramento esponenziale rispetto al nostro primo algoritmo `Fib(n)`.

Un ulteriore miglioramento lo si può ottenere sulla memoria usata.

L'algoritmo `IterFib(n)` richiede tempo $O(n)$ e memoria $O(n)$ per calcolare l'n-esimo numero di Fibonacci F_n, un miglioramento esponenziale rispetto al nostro primo algoritmo `Fib(n)`.

Un ulteriore miglioramento lo si può ottenere sulla memoria usata. Infatti, delle locazioni dell'array a ce ne occorrono solo le ultime due calcolate, e non tutte le n.

Pausa di riflessione...

1. Siamo partiti da un algoritmo ricorsivo (e non efficiente), ottenuto applicando la tecnica Divide-et-Impera

Pausa di riflessione...

1. Siamo partiti da un algoritmo ricorsivo (e non efficiente), ottenuto applicando la tecnica Divide-et-Impera
2. Abbiamo scoperto che i motivi dell'inefficienza dell'algoritmo risiedevano nel fatto che l'algoritmo risolveva più volte lo stesso sottoproblema,

Pausa di riflessione...

1. Siamo partiti da un algoritmo ricorsivo (e non efficiente), ottenuto applicando la tecnica Divide-et-Impera
2. Abbiamo scoperto che i motivi dell'inefficienza dell'algoritmo risiedevano nel fatto che l'algoritmo risolveva più volte lo stesso sottoproblema,
3. Abbiamo aggiunto una tabella all'algoritmo,

Pausa di riflessione...

1. Siamo partiti da un algoritmo ricorsivo (e non efficiente), ottenuto applicando la tecnica Divide-et-Impera
2. Abbiamo scoperto che i motivi dell'inefficienza dell'algoritmo risiedevano nel fatto che l'algoritmo risolveva più volte lo stesso sottoproblema,
3. Abbiamo aggiunto una tabella all'algoritmo, indicizzata dai possibili valori input (= numero di sottoproblemi) alle chiamate ricorsive.

Pausa di riflessione...

1. Siamo partiti da un algoritmo ricorsivo (e non efficiente), ottenuto applicando la tecnica Divide-et-Impera
2. Abbiamo scoperto che i motivi dell'inefficienza dell'algoritmo risiedevano nel fatto che l'algoritmo risolveva più volte lo stesso sottoproblema,
3. Abbiamo aggiunto una tabella all'algoritmo, indicizzata dai possibili valori input (= numero di sottoproblemi) alle chiamate ricorsive.
4. Abbiamo aggiunto, prima di ogni chiamata ricorsiva dell'algoritmo su di un particolare sottoproblema,

Pausa di riflessione...

1. Siamo partiti da un algoritmo ricorsivo (e non efficiente), ottenuto applicando la tecnica Divide-et-Impera
2. Abbiamo scoperto che i motivi dell'inefficienza dell'algoritmo risiedevano nel fatto che l'algoritmo risolveva più volte lo stesso sottoproblema,
3. Abbiamo aggiunto una tabella all'algoritmo, indicizzata dai possibili valori input (= numero di sottoproblemi) alle chiamate ricorsive.
4. Abbiamo aggiunto, prima di ogni chiamata ricorsiva dell'algoritmo su di un particolare sottoproblema, un *controllo* sulla tabella per verificare se la soluzione a quel sottoproblema era stata già calcolata in precedenza.

Pausa di riflessione...

1. Siamo partiti da un algoritmo ricorsivo (e non efficiente), ottenuto applicando la tecnica Divide-et-Impera
2. Abbiamo scoperto che i motivi dell'inefficienza dell'algoritmo risiedevano nel fatto che l'algoritmo risolveva più volte lo stesso sottoproblema,
3. Abbiamo aggiunto una tabella all'algoritmo, indicizzata dai possibili valori input (= numero di sottoproblemi) alle chiamate ricorsive.
4. Abbiamo aggiunto, prima di ogni chiamata ricorsiva dell'algoritmo su di un particolare sottoproblema, un *controllo* sulla tabella per verificare se la soluzione a quel sottoproblema era stata già calcolata in precedenza.
5. Nel caso affermativo, ritorniamo semplicemente la soluzione al sottoproblema

Pausa di riflessione...

1. Siamo partiti da un algoritmo ricorsivo (e non efficiente), ottenuto applicando la tecnica Divide-et-Impera
2. Abbiamo scoperto che i motivi dell'inefficienza dell'algoritmo risiedevano nel fatto che l'algoritmo risolveva più volte lo stesso sottoproblema,
3. Abbiamo aggiunto una tabella all'algoritmo, indicizzata dai possibili valori input (= numero di sottoproblemi) alle chiamate ricorsive.
4. Abbiamo aggiunto, prima di ogni chiamata ricorsiva dell'algoritmo su di un particolare sottoproblema, un *controllo* sulla tabella per verificare se la soluzione a quel sottoproblema era stata già calcolata in precedenza.
5. Nel caso affermativo, ritorniamo semplicemente la soluzione al sottoproblema (**senza ricalcolarla**).

Pausa di riflessione...

1. Siamo partiti da un algoritmo ricorsivo (e non efficiente), ottenuto applicando la tecnica Divide-et-Impera
2. Abbiamo scoperto che i motivi dell'inefficienza dell'algoritmo risiedevano nel fatto che l'algoritmo risolveva più volte lo stesso sottoproblema,
3. Abbiamo aggiunto una tabella all'algoritmo, indicizzata dai possibili valori input (= numero di sottoproblemi) alle chiamate ricorsive.
4. Abbiamo aggiunto, prima di ogni chiamata ricorsiva dell'algoritmo su di un particolare sottoproblema, un *controllo* sulla tabella per verificare se la soluzione a quel sottoproblema era stata già calcolata in precedenza.
5. Nel caso affermativo, ritorniamo semplicemente la soluzione al sottoproblema (**senza ricalcolarla**). Se invece la soluzione al sottoproblema oggetto della chiamata ricorsiva non è stata precedentemente calcolata,

Pausa di riflessione...

1. Siamo partiti da un algoritmo ricorsivo (e non efficiente), ottenuto applicando la tecnica Divide-et-Impera
2. Abbiamo scoperto che i motivi dell'inefficienza dell'algoritmo risiedevano nel fatto che l'algoritmo risolveva più volte lo stesso sottoproblema,
3. Abbiamo aggiunto una tabella all'algoritmo, indicizzata dai possibili valori input (= numero di sottoproblemi) alle chiamate ricorsive.
4. Abbiamo aggiunto, prima di ogni chiamata ricorsiva dell'algoritmo su di un particolare sottoproblema, un *controllo* sulla tabella per verificare se la soluzione a quel sottoproblema era stata già calcolata in precedenza.
5. Nel caso affermativo, ritorniamo semplicemente la soluzione al sottoproblema (**senza ricalcolarla**). Se invece la soluzione al sottoproblema oggetto della chiamata ricorsiva non è stata precedentemente calcolata, chiamiamo ricorsivamente l'algoritmo per risolvere il (nuovo) sottoproblema.

Questa tecnica prende il nome di **Memoization** ha validità abbastanza generale, e ci permette di ottenere algoritmi efficienti per molti problemi algoritmici.

Altro esempio

Altro esempio

Denotiamo con

$$\binom{n}{r}$$

il numero di modi con cui possiamo scegliere r oggetti da un insieme di n elementi.

Altro esempio

Denotiamo con

$$\binom{n}{r}$$

il numero di modi con cui possiamo scegliere r oggetti da un insieme di n elementi.
Vale che

$$\binom{n}{r} = \begin{cases} 1 & \text{se } r = 0 \text{ oppure } r = n, \end{cases}$$

Altro esempio

Denotiamo con

$$\binom{n}{r}$$

il numero di modi con cui possiamo scegliere r oggetti da un insieme di n elementi.
Vale che

$$\binom{n}{r} = \begin{cases} 1 & \text{se } r = 0 \text{ oppure } r = n, \\ \binom{n-1}{r-1} + \binom{n-1}{r} & \text{altrimenti .} \end{cases}$$

Applichiamo la tecnica Divide-et-Impera per calcolare $\binom{n}{r}$

Applichiamo la tecnica Divide-et-Impera per calcolare $\binom{n}{r}$

```
Choose(n,r)

1. IF ((r==0)||(n==r)) {
2.     RETURN 1
```

$$\binom{n}{r} = \begin{cases} 1 & \text{se } r = 0 \text{ oppure } r = n, \\ \binom{n-1}{r-1} + \binom{n-1}{r} & \text{altrimenti .} \end{cases}$$

Applichiamo la tecnica Divide-et-Impera per calcolare $\binom{n}{r}$

```
Choose(n,r)

1. IF ((r==0)||(n==r)) {
2.     RETURN 1
3.     } ELSE {
4.     RETURN(Choose(n-1,r-1)+Choose(n-1,r))
       }
```

$$\binom{n}{r} = \begin{cases} 1 & \text{se } r = 0 \text{ oppure } r = n, \\ \binom{n-1}{r-1} + \binom{n-1}{r} & \text{altrimenti .} \end{cases}$$

Applichiamo la tecnica Divide-et-Impera per calcolare $\binom{n}{r}$

```
Choose(n,r)

1. IF ((r==0)||(n==r)) {
2.     RETURN 1
3.     } ELSE {
4.     RETURN(Choose(n-1,r-1)+Choose(n-1,r))
       }
```

$$\binom{n}{r} = \begin{cases} 1 & \text{se } r = 0 \text{ oppure } r = n, \\ \binom{n-1}{r-1} + \binom{n-1}{r} & \text{altrimenti .} \end{cases}$$

Ai fini dell'analisi dell'algoritmo `Choose`(n, r), denotiamo con $T(n, r)$ il numero di operazioni effettuate dall'algoritmo `Choose`(n, r),

Applichiamo la tecnica Divide-et-Impera per calcolare $\binom{n}{r}$

```
Choose(n,r)

1. IF ((r==0)||(n==r)) {
2.     RETURN 1
3.     } ELSE {
4.     RETURN(Choose(n-1,r-1)+Choose(n-1,r))
       }
```

$$\binom{n}{r} = \begin{cases} 1 & \text{se } r = 0 \text{ oppure } r = n, \\ \binom{n-1}{r-1} + \binom{n-1}{r} & \text{altrimenti .} \end{cases}$$

Ai fini dell'analisi dell'algoritmo `Choose`(n, r), denotiamo con $T(n, r)$ il numero di operazioni effettuate dall'algoritmo `Choose`(n, r), e sia $T(n) = \max_r T(n, r)$.

Applichiamo la tecnica Divide-et-Impera per calcolare $\binom{n}{r}$

```
Choose(n,r)

1. IF ((r==0)||(n==r)) {
2.     RETURN 1
3.     } ELSE {
4.     RETURN(Choose(n-1,r-1)+Choose(n-1,r))
       }
```

$$\binom{n}{r} = \begin{cases} 1 & \text{se } r = 0 \text{ oppure } r = n, \\ \binom{n-1}{r-1} + \binom{n-1}{r} & \text{altrimenti .} \end{cases}$$

Ai fini dell'analisi dell'algoritmo Choose(n, r), denotiamo con $T(n, r)$ il numero di operazioni effettuate dall'algoritmo Choose(n, r), e sia $T(n) = \max_r T(n, r)$. Abbiamo che

$$T(n) = \begin{cases} c & \text{se } n = 1, \\ \end{cases}$$

Applichiamo la tecnica Divide-et-Impera per calcolare $\binom{n}{r}$

```
Choose(n,r)

1. IF ((r==0)||(n==r)) {
2.     RETURN 1
3.     } ELSE {
4.     RETURN(Choose(n-1,r-1)+Choose(n-1,r))
       }
```

$$\binom{n}{r} = \begin{cases} 1 & \text{se } r = 0 \text{ oppure } r = n, \\ \binom{n-1}{r-1} + \binom{n-1}{r} & \text{altrimenti .} \end{cases}$$

Ai fini dell'analisi dell'algoritmo `Choose`(n, r), denotiamo con $T(n, r)$ il numero di operazioni effettuate dall'algoritmo `Choose`(n, r), e sia $T(n) = \max_r T(n, r)$. Abbiamo che

$$T(n) = \begin{cases} c & \text{se } n = 1, \\ 2T(n-1) + d & \text{se } n > 1 \end{cases}$$

Risolviamo l' equazione di ricorrenza

$$T(n) = 2T(n-1) + d$$

Risolviamo l' equazione di ricorrenza $T(n) = 2T(n-1) + d$

$$T(n) = 2T(n-1) + d$$

Risolviamo l' equazione di ricorrenza

$T(n) = 2T(n-1) + d$

$$\begin{aligned} T(n) &= 2T(n-1) + d \\ &= 2(2T(n-2) + d) + d \end{aligned}$$

Risolviamo l' equazione di ricorrenza

$T(n) = 2T(n-1) + d$

$$
\begin{aligned}
T(n) &= 2T(n-1) + d \\
&= 2(2T(n-2) + d) + d \\
&= 4T(n-2) + 2d + d
\end{aligned}
$$

Risolviamo l' equazione di ricorrenza

$T(n) = 2T(n-1) + d$

$$
\begin{aligned}
T(n) &= 2T(n-1) + d \\
&= 2(2T(n-2) + d) + d \\
&= 4T(n-2) + 2d + d \quad = 2^2 T(n-2) + 2d + d
\end{aligned}
$$

Risolviamo l' equazione di ricorrenza

$T(n) = 2T(n-1) + d$

$$
\begin{aligned}
T(n) &= 2T(n-1) + d \\
&= 2(2T(n-2) + d) + d \\
&= 4T(n-2) + 2d + d \quad = 2^2 T(n-2) + 2d + d \\
&\cdots \\
&= 2^i T(n-i) + d \sum_{j=0}^{i-1} 2^j
\end{aligned}
$$

Risolviamo l' equazione di ricorrenza

$T(n) = 2T(n-1) + d$

$$
\begin{aligned}
T(n) &= 2T(n-1) + d \\
&= 2(2T(n-2) + d) + d \\
&= 4T(n-2) + 2d + d \quad = 2^2 T(n-2) + 2d + d \\
&\dots \\
&= 2^i T(n-i) + d \sum_{j=0}^{i-1} 2^j \\
&= 2^{n-1} T(1) + d \sum_{j=0}^{n-2} 2^j
\end{aligned}
$$

Risolviamo l' equazione di ricorrenza

$T(n) = 2T(n-1) + d$

$$
\begin{aligned}
T(n) &= 2T(n-1) + d \\
&= 2(2T(n-2) + d) + d \\
&= 4T(n-2) + 2d + d \quad = 2^2 T(n-2) + 2d + d \\
&\dots \\
&= 2^i T(n-i) + d \sum_{j=0}^{i-1} 2^j \\
&= 2^{n-1} T(1) + d \sum_{j=0}^{n-2} 2^j \\
&= c2^{n-1} + d(2^{n-1} - 1)
\end{aligned}
$$

Risolviamo l' equazione di ricorrenza

$T(n) = 2T(n-1) + d$

$$
\begin{aligned}
T(n) &= 2T(n-1) + d \\
&= 2(2T(n-2) + d) + d \\
&= 4T(n-2) + 2d + d \quad = 2^2 T(n-2) + 2d + d \\
&\dots \\
&= 2^i T(n-i) + d \sum_{j=0}^{i-1} 2^j \\
&= 2^{n-1} T(1) + d \sum_{j=0}^{n-2} 2^j \\
&= c2^{n-1} + d(2^{n-1} - 1) = (c+d)2^{n-1} - d.
\end{aligned}
$$

Risolviamo l' equazione di ricorrenza

$T(n) = 2T(n-1) + d$

$$
\begin{aligned}
T(n) &= 2T(n-1) + d \\
&= 2(2T(n-2) + d) + d \\
&= 4T(n-2) + 2d + d \ = 2^2 T(n-2) + 2d + d \\
&\dots \\
&= 2^i T(n-i) + d \sum_{j=0}^{i-1} 2^j \\
&= 2^{n-1} T(1) + d \sum_{j=0}^{n-2} 2^j \\
&= c2^{n-1} + d(2^{n-1} - 1) = (c+d)2^{n-1} - d.
\end{aligned}
$$

Quindi $T(n) = \Theta(2^n)$, e non stà bene.

Risolviamo l' equazione di ricorrenza

$T(n) = 2T(n-1) + d$

$$
\begin{aligned}
T(n) &= 2T(n-1) + d \\
&= 2(2T(n-2) + d) + d \\
&= 4T(n-2) + 2d + d \ = 2^2 T(n-2) + 2d + d \\
&\dots \\
&= 2^i T(n-i) + d \sum_{j=0}^{i-1} 2^j \\
&= 2^{n-1} T(1) + d \sum_{j=0}^{n-2} 2^j \\
&= c2^{n-1} + d(2^{n-1} - 1) = (c+d)2^{n-1} - d.
\end{aligned}
$$

Quindi $T(n) = \Theta(2^n)$, e non stà bene. Perchè l'algoritmo è così inefficiente?

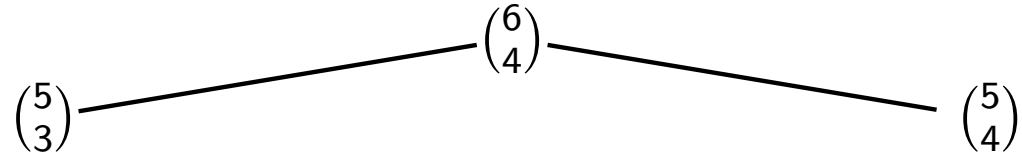
Risolviamo l' equazione di ricorrenza

$T(n) = 2T(n-1) + d$

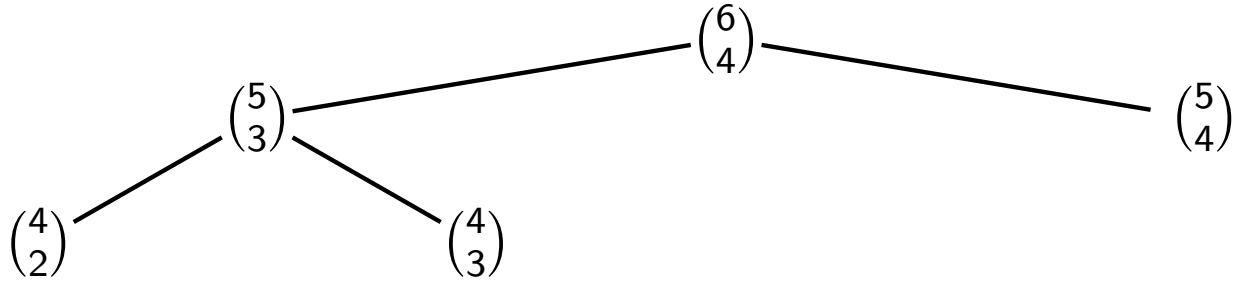
$$
\begin{aligned}
T(n) &= 2T(n-1) + d \\
&= 2(2T(n-2) + d) + d \\
&= 4T(n-2) + 2d + d \ = 2^2 T(n-2) + 2d + d \\
&\dots \\
&= 2^i T(n-i) + d \sum_{j=0}^{i-1} 2^j \\
&= 2^{n-1} T(1) + d \sum_{j=0}^{n-2} 2^j \\
&= c2^{n-1} + d(2^{n-1} - 1) = (c+d)2^{n-1} - d.
\end{aligned}
$$

Quindi $T(n) = \Theta(2^n)$, e non stà bene. Perchè l'algoritmo è così inefficiente? Perchè risolve gli stessi sottoproblemi più volte.

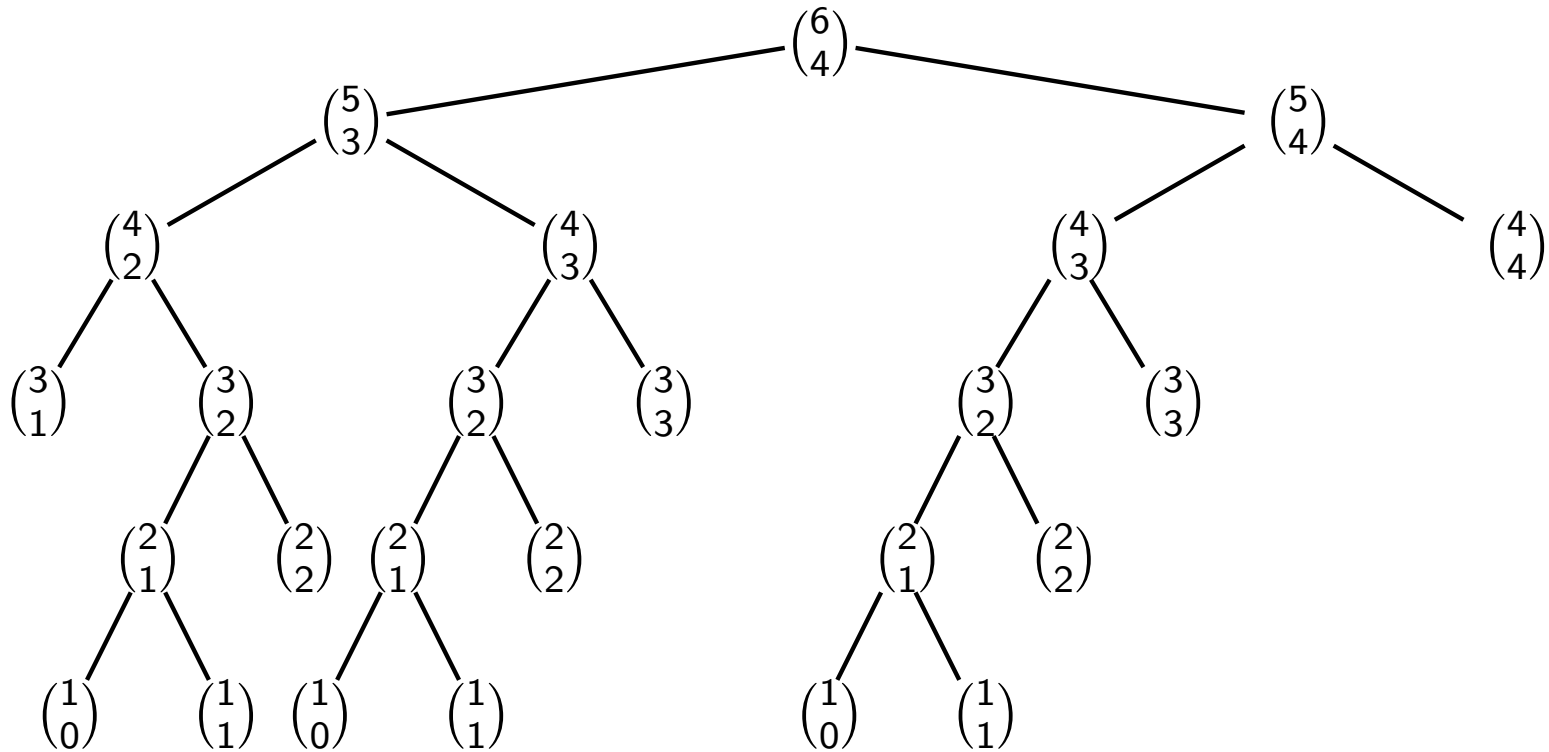
Diamo infatti un'occhiata all'albero delle chiamate ricorsive di $\texttt{Choose}(6, 4)$).



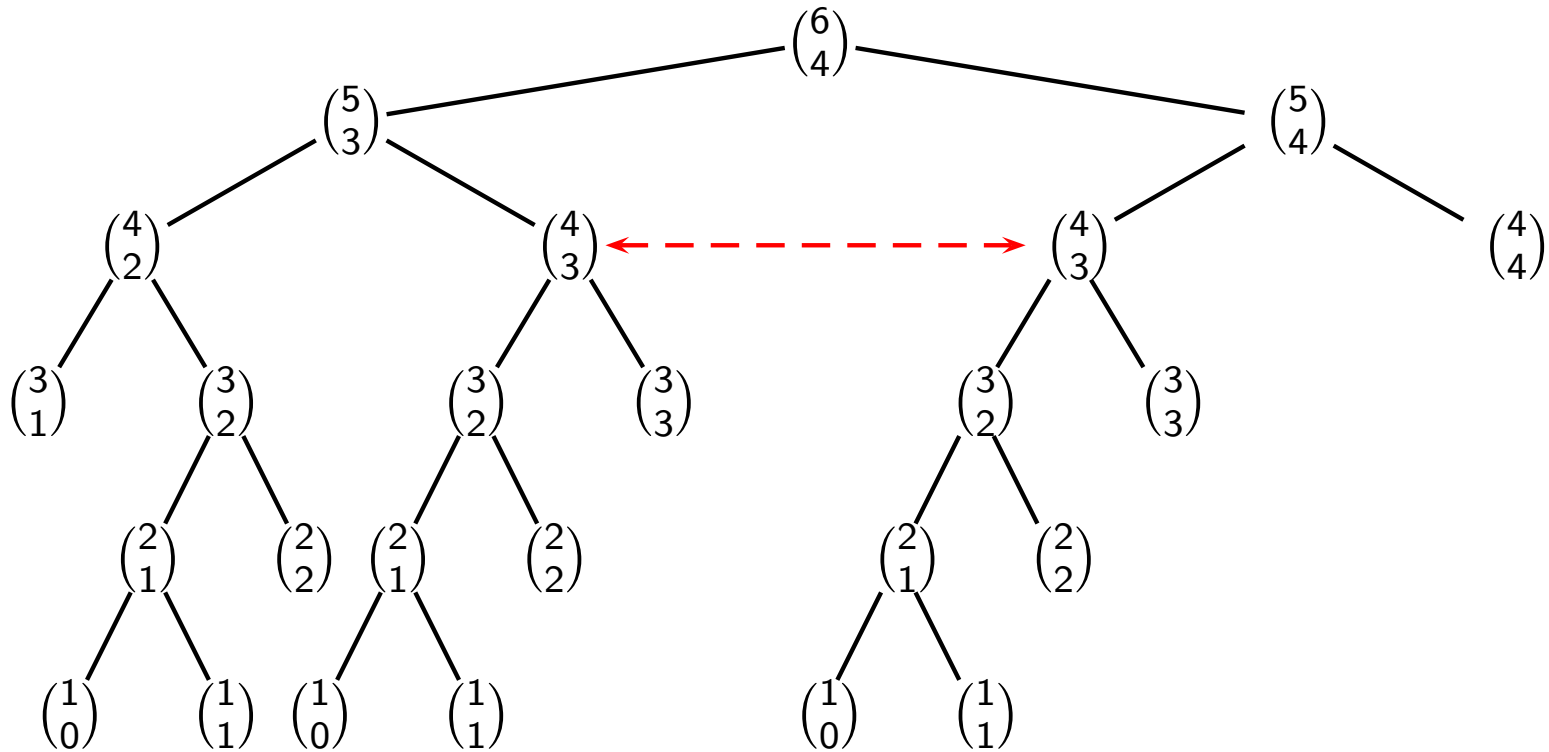
Diamo infatti un'occhiata all'albero delle chiamate ricorsive di $\texttt{Choose}(6, 4)$).



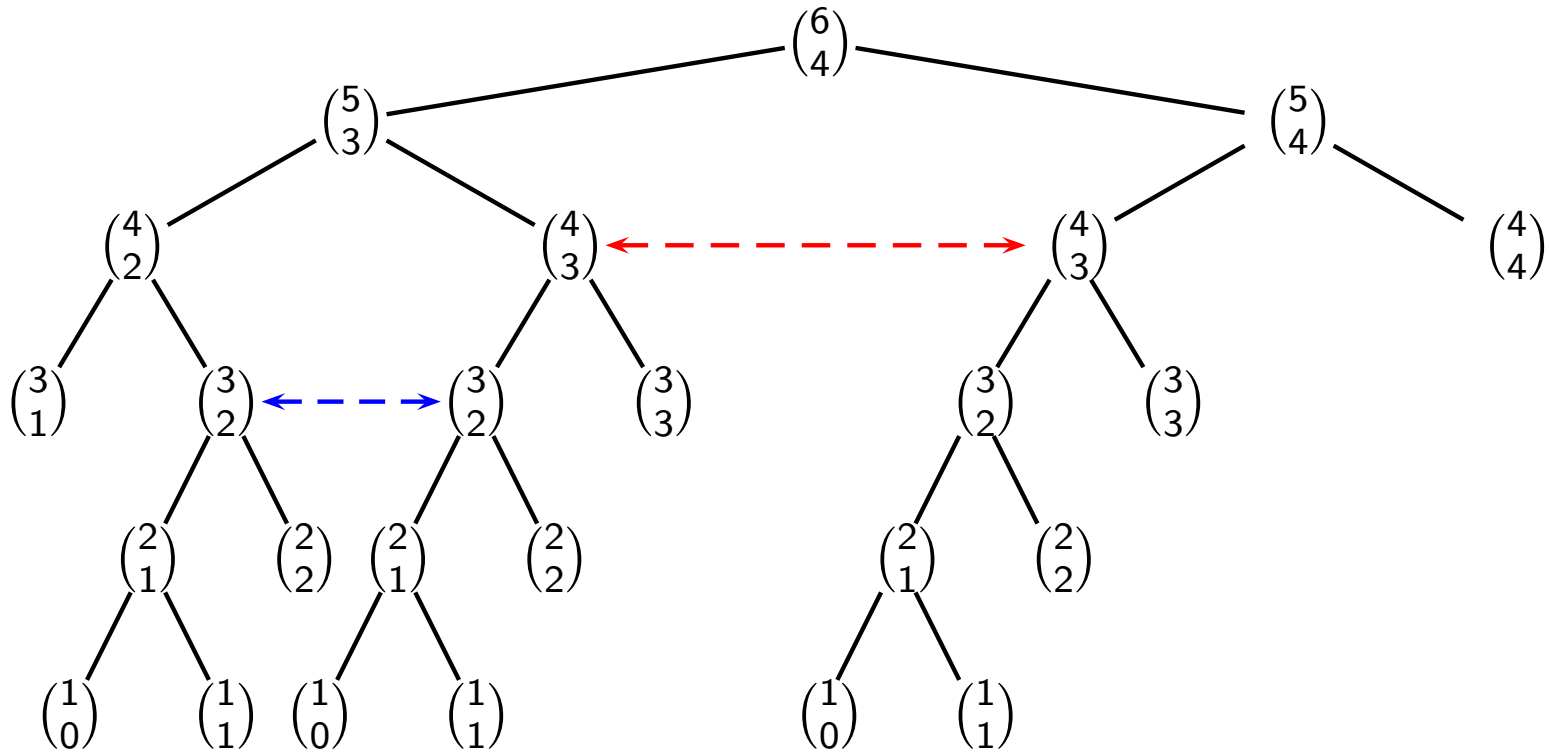
Diamo infatti un'occhiata all'albero delle chiamate ricorsive di $\texttt{Choose}(6, 4)$).



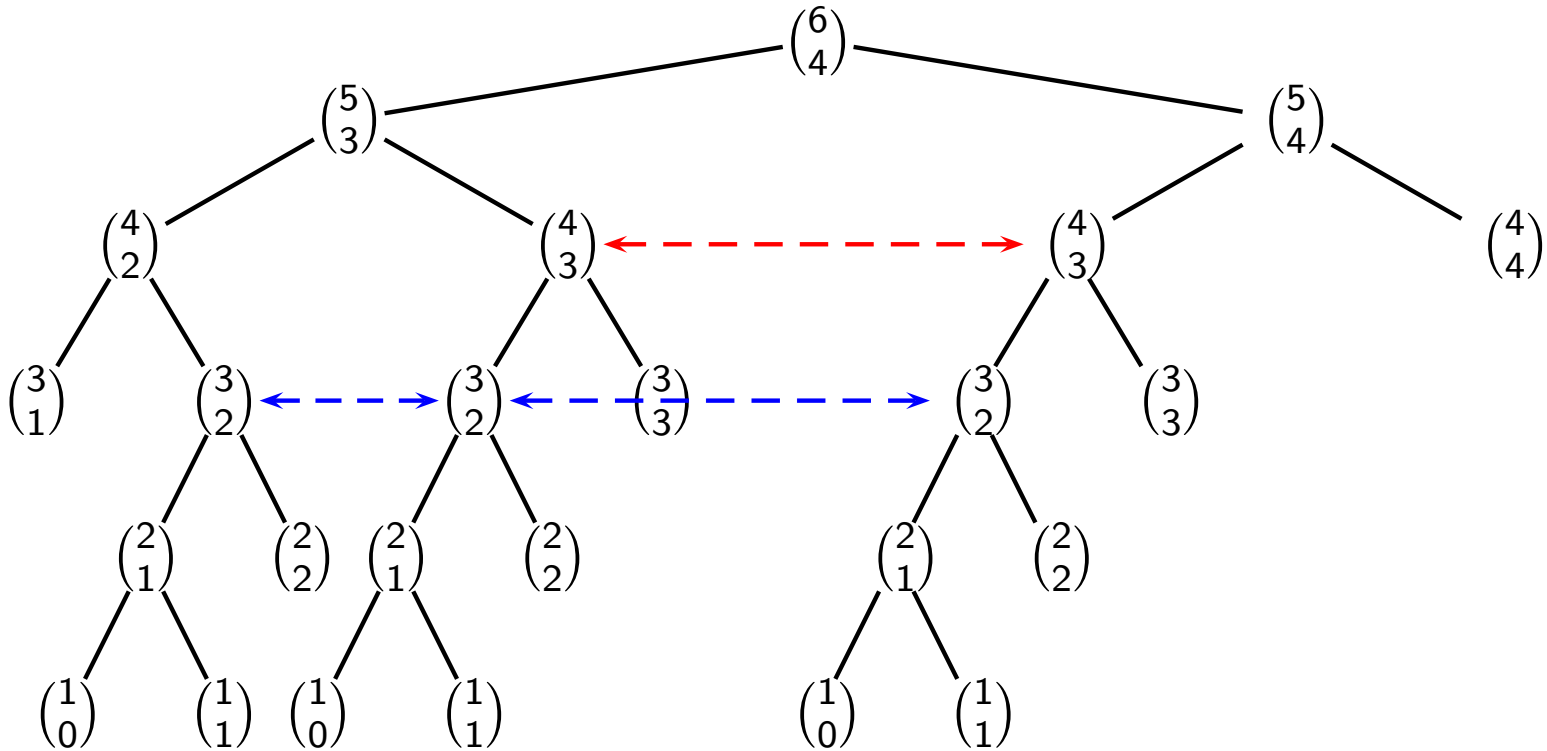
Diamo infatti un'occhiata all'albero delle chiamate ricorsive di `Choose`$(6, 4)$).



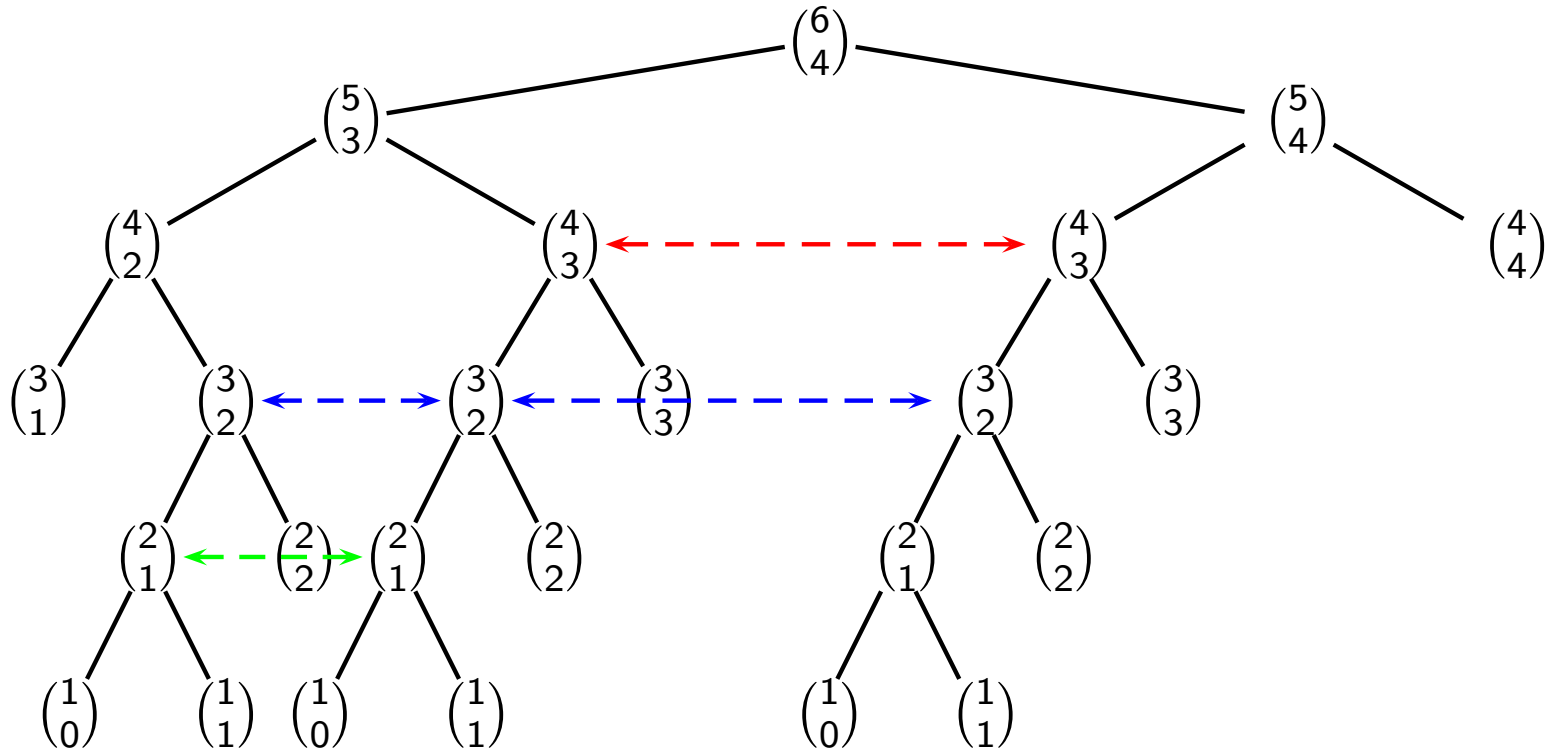
Diamo infatti un'occhiata all'albero delle chiamate ricorsive di `Choose`$(6, 4)$).



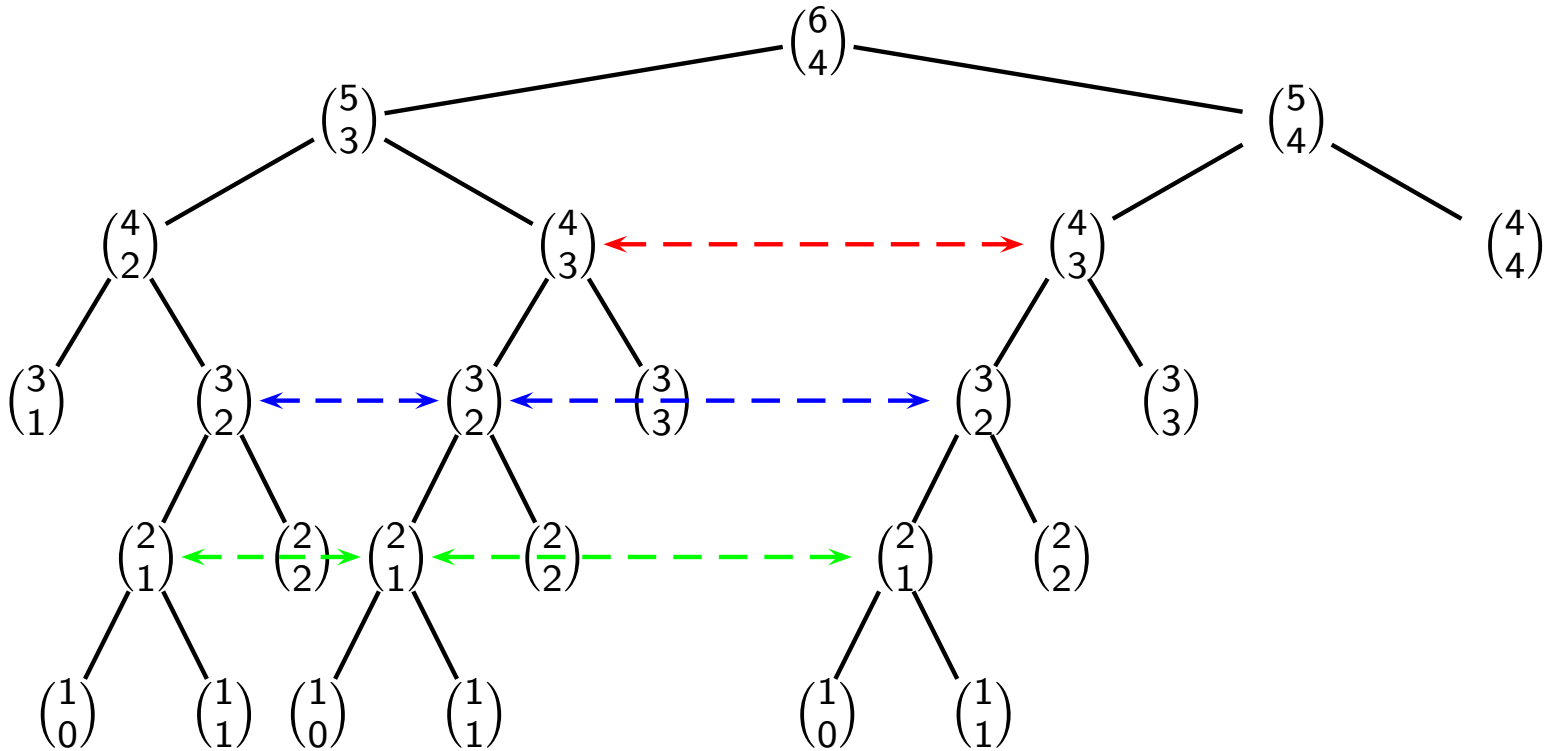
Diamo infatti un'occhiata all'albero delle chiamate ricorsive di $\texttt{Choose}(6, 4)$).



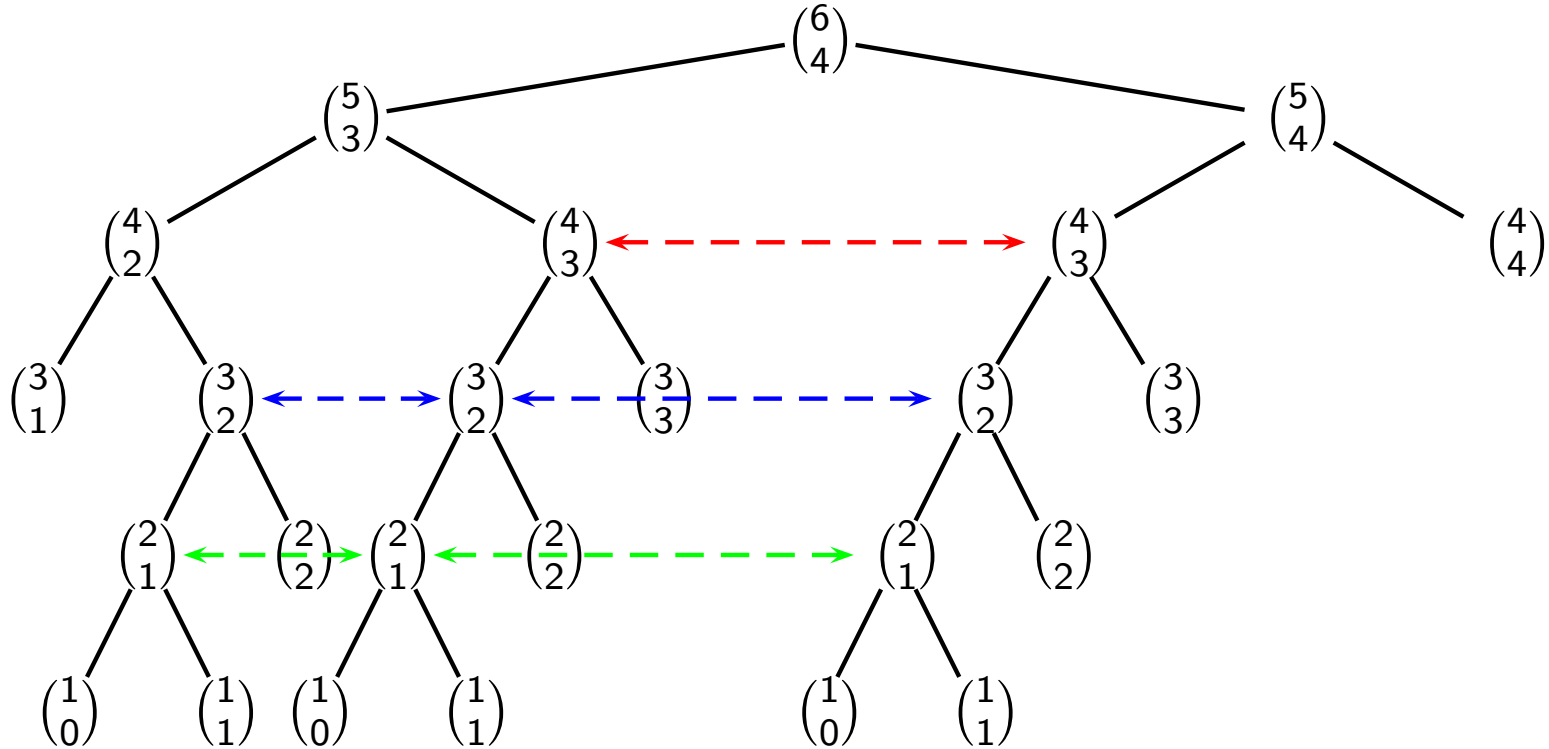
Diamo infatti un'occhiata all'albero delle chiamate ricorsive di `Choose`$(6, 4)$).



Diamo infatti un'occhiata all'albero delle chiamate ricorsive di `Choose`$(6, 4)$).



Diamo infatti un'occhiata all'albero delle chiamate ricorsive di `Choose`$(6, 4)$).



Di nuovo, **stessi** sottoproblemi vengono risolti **più volte.**

Situazione già vista...

Situazione già vista...

Quindi

- Aggiungiamo allora all'algoritmo una tabella $T[i, j]$ che contenga i valori $\binom{i}{j}$ man mano che li calcoliamo.

Situazione già vista...

Quindi

- Aggiungiamo allora all'algoritmo una tabella $T[i, j]$ che contenga i valori $\binom{i}{j}$ man mano che li calcoliamo.
- Facciamo precedere ad ogni chiamata ricorsiva dell'algoritmo `Choose`(i, j) un controllo per verificare se numero $\binom{i}{j}$ è stato precedentemente computato;

Situazione già vista...

Quindi

- Aggiungiamo allora all'algoritmo una tabella $T[i,j]$ che contenga i valori $\binom{i}{j}$ man mano che li calcoliamo.
- Facciamo precedere ad ogni chiamata ricorsiva dell'algoritmo `Choose`(i,j) un controllo per verificare se numero $\binom{i}{j}$ è stato precedentemente computato;
- **Se** questo è il caso, ci limiteremo a leggerlo dalla tabella in $T[i,j]$,

Situazione già vista...

Quindi

- Aggiungiamo allora all'algoritmo una tabella $T[i,j]$ che contenga i valori $\binom{i}{j}$ man mano che li calcoliamo.
- Facciamo precedere ad ogni chiamata ricorsiva dell'algoritmo `Choose`(i,j) un controllo per verificare se numero $\binom{i}{j}$ è stato precedentemente computato;
- **Se** questo è il caso, ci limiteremo a leggerlo dalla tabella in $T[i,j]$, **altrimenti** sviluppiamo la ricorsione per calcolarlo (ciò accadrà *una sola volta*), e lo memorizzeremo in $T[i,j]$.

L'algoritmo

```
MemChoose(n,r)
1. IF((r==0)||(n==r)) {
2.   RETURN 1
```

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$

L'algoritmo

```
MemChoose(n,r)
1. IF((r==0)||(n==r)) {
2.    RETURN 1
3.    } ELSE {
4.      IF(T[n,r] non è definito) {
```

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$

L'algoritmo

```
MemChoose(n,r)
1. IF((r==0)||(n==r)) {
2.    RETURN 1
3.    } ELSE {
4.      IF(T[n,r] non è definito) {
5.          T[n,r]=MemChoose(n-1,r-1)
                    + MemChoose(n-1,r)
6.        }
7.    }
```

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$

L'algoritmo

```
MemChoose(n,r)
1. IF((r==0)||(n==r)) {
2.    RETURN 1
3.    } ELSE {
4.      IF(T[n,r] non è definito) {
5.          T[n,r]=MemChoose(n-1,r-1)
                    + MemChoose(n-1,r)
6.        }
7.    }
8. RETURN T[n,r]
```

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$

L'algoritmo

```
MemChoose(n,r)
1. IF((r==0)||(n==r)) {
2.    RETURN 1
3.    } ELSE {
4.      IF(T[n,r] non è definito) {
5.          T[n,r]=MemChoose(n-1,r-1)
                    + MemChoose(n-1,r)
6.        }
7.    }
8. RETURN T[n,r]
```

$\binom{n-1}{r-1} + \binom{n-1}{r}$

Complessità : $\Theta(nr) = O(n^2)$

Vediamo l'algoritmo iterativo. Calcoliamo i valori $\binom{i}{j}$, per $i = 0, \ldots n$ e $j = 0, \ldots r$, e li mettiamo nella tabella $T[i, j]$

Vediamo l'algoritmo iterativo. Calcoliamo i valori $\binom{i}{j}$, per $i = 0, \ldots n$ e $j = 0, \ldots r$, e li mettiamo nella tabella $T[i, j]$ (noi siamo interessati al valore $T[n, r]$).

Vediamo l'algoritmo iterativo. Calcoliamo i valori $\binom{i}{j}$, per $i = 0, \ldots n$ e $j = 0, \ldots r$, e li mettiamo nella tabella $T[i, j]$ (noi siamo interessati al valore $T[n, r]$).

Ricordiamo che $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$ e $\binom{i}{i} = 1 = \binom{i}{0}$.

Vediamo l'algoritmo iterativo. Calcoliamo i valori $\binom{i}{j}$, per $i = 0, \ldots n$ e $j = 0, \ldots r$, e li mettiamo nella tabella $T[i, j]$ (noi siamo interessati al valore $T[n, r]$).

Ricordiamo che $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$ e $\binom{i}{i} = 1 = \binom{i}{0}$.

```
IterChoose(n,r)
1. FOR(i=0,i<n+1, i=i+1) {
2.    T[i,0]= 1
       }
```

$\binom{n-1}{r-1} + \binom{n-1}{r}$

Vediamo l'algoritmo iterativo. Calcoliamo i valori $\binom{i}{j}$, per $i = 0, \ldots n$ e $j = 0, \ldots r$, e li mettiamo nella tabella $T[i, j]$ (noi siamo interessati al valore $T[n, r]$).

Ricordiamo che $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$ e $\binom{i}{i} = 1 = \binom{i}{0}$.

```
IterChoose(n,r)
1. FOR(i=0,i<n+1, i=i+1) {
2.    T[i,0]= 1
       }
3. FOR(i=0,i<r+1,i=i+1) {
4.    T[i,i]=1
       }
```

$\binom{n-1}{r-1} + \binom{n-1}{r}$

Vediamo l'algoritmo iterativo. Calcoliamo i valori $\binom{i}{j}$, per $i = 0, \ldots n$ e $j = 0, \ldots r$, e li mettiamo nella tabella $T[i, j]$ (noi siamo interessati al valore $T[n, r]$).

Ricordiamo che $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$ e $\binom{i}{i} = 1 = \binom{i}{0}$.

```
IterChoose(n,r)
1. FOR(i=0,i<n+1, i=i+1) {
2.    T[i,0]= 1
      }
3. FOR(i=0,i<r+1,i=i+1) {
4.    T[i,i]=1
      }
5. FOR(j=1,j<r+1,j=j+1) {
```

$\binom{n-1}{r-1} + \binom{n-1}{r}$

Vediamo l'algoritmo iterativo. Calcoliamo i valori $\binom{i}{j}$, per $i = 0, \ldots n$ e $j = 0, \ldots r$, e li mettiamo nella tabella $T[i, j]$ (noi siamo interessati al valore $T[n, r]$).

Ricordiamo che $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$ e $\binom{i}{i} = 1 = \binom{i}{0}$.

```
IterChoose(n,r)
1. FOR(i=0,i<n+1, i=i+1) {
2.    T[i,0]= 1
      }
3. FOR(i=0,i<r+1,i=i+1) {
4.    T[i,i]=1
      }
5. FOR(j=1,j<r+1,j=j+1) {
6.    FOR(i=j+1,i<n+1,i=i+1) {
```

$\binom{n-1}{r-1} + \binom{n-1}{r}$

Vediamo l'algoritmo iterativo. Calcoliamo i valori $\binom{i}{j}$, per $i = 0, \ldots n$ e $j = 0, \ldots r$, e li mettiamo nella tabella $T[i, j]$ (noi siamo interessati al valore $T[n, r]$).

Ricordiamo che $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$ e $\binom{i}{i} = 1 = \binom{i}{0}$.

```
IterChoose(n,r)
1. FOR(i=0,i<n+1, i=i+1) {
2.    T[i,0]= 1
      }
3. FOR(i=0,i<r+1,i=i+1) {
4.    T[i,i]=1
      }
5. FOR(j=1,j<r+1,j=j+1) {
6.    FOR(i=j+1,i<n+1,i=i+1) {
7.      T[i,j]=T[i-1,j-1]+T[i-1,j]
        }
      }
```

$\binom{n-1}{r-1} + \binom{n-1}{r}$

Vediamo l'algoritmo iterativo. Calcoliamo i valori $\binom{i}{j}$, per $i = 0, \ldots n$ e $j = 0, \ldots r$, e li mettiamo nella tabella $T[i, j]$ (noi siamo interessati al valore $T[n, r]$).

Ricordiamo che $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$ e $\binom{i}{i} = 1 = \binom{i}{0}$.

```
IterChoose(n,r)                                   (n-1 choose r-1) + (n-1 choose r)
1. FOR(i=0,i<n+1, i=i+1) {
2.    T[i,0]= 1
      }
3. FOR(i=0,i<r+1,i=i+1) {
4.    T[i,i]=1
      }
5. FOR(j=1,j<r+1,j=j+1) {
6.    FOR(i=j+1,i<n+1,i=i+1) {
7.      T[i,j]=T[i-1,j-1]+T[i-1,j]
        }
      }
RETURN T[n,r]
```

Vediamo l'algoritmo iterativo. Calcoliamo i valori $\binom{i}{j}$, per $i = 0, \ldots n$ e $j = 0, \ldots r$, e li mettiamo nella tabella $T[i, j]$ (noi siamo interessati al valore $T[n, r]$).

Ricordiamo che $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$ e $\binom{i}{i} = 1 = \binom{i}{0}$.

```
IterChoose(n,r)                                   (n-1 choose r-1) + (n-1 choose r)
1. FOR(i=0,i<n+1, i=i+1) {
2.    T[i,0]= 1
      }
3. FOR(i=0,i<r+1,i=i+1) {
4.    T[i,i]=1
      }
5. FOR(j=1,j<r+1,j=j+1) {
6.    FOR(i=j+1,i<n+1,i=i+1) {
7.      T[i,j]=T[i-1,j-1]+T[i-1,j]
        }
      }
RETURN T[n,r]
```

Al momento dell' assegnazione `T[i,j] = T[i-1,j-1] + T[i-1,j]` occorre che `T[i-1,j-1]` e + `T[i-1,j]` siano stati già calcolati

Vediamo l'algoritmo iterativo. Calcoliamo i valori $\binom{i}{j}$, per $i = 0, \ldots n$ e $j = 0, \ldots r$, e li mettiamo nella tabella $T[i, j]$ (noi siamo interessati al valore $T[n, r]$).

Ricordiamo che $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$ e $\binom{i}{i} = 1 = \binom{i}{0}$.

```
IterChoose(n,r)                                   (n-1 choose r-1) + (n-1 choose r)
1. FOR(i=0,i<n+1, i=i+1) {
2.    T[i,0]= 1
      }
3. FOR(i=0,i<r+1,i=i+1) {
4.    T[i,i]=1
      }
5. FOR(j=1,j<r+1,j=j+1) {
6.    FOR(i=j+1,i<n+1,i=i+1) {
7.      T[i,j]=T[i-1,j-1]+T[i-1,j]
        }
      }
RETURN T[n,r]
```

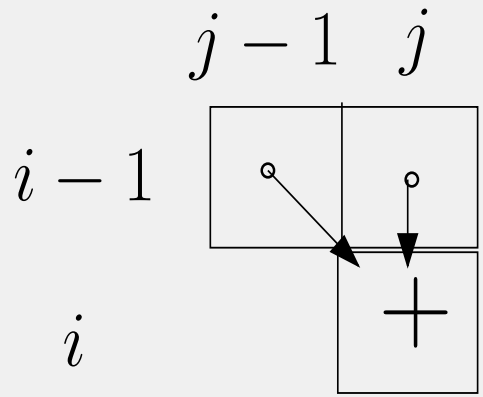
Al momento dell' assegnazione `T[i,j] = T[i-1,j-1] + T[i-1,j]` occorre che `T[i-1,j-1]` e + `T[i-1,j]` siano stati già calcolati (e infatti l'algoritmo correttamente procede in questo modo).

Vediamo l'algoritmo iterativo. Calcoliamo i valori $\binom{i}{j}$, per $i = 0, \ldots n$ e $j = 0, \ldots r$, e li mettiamo nella tabella $T[i, j]$ (noi siamo interessati al valore $T[n, r]$).

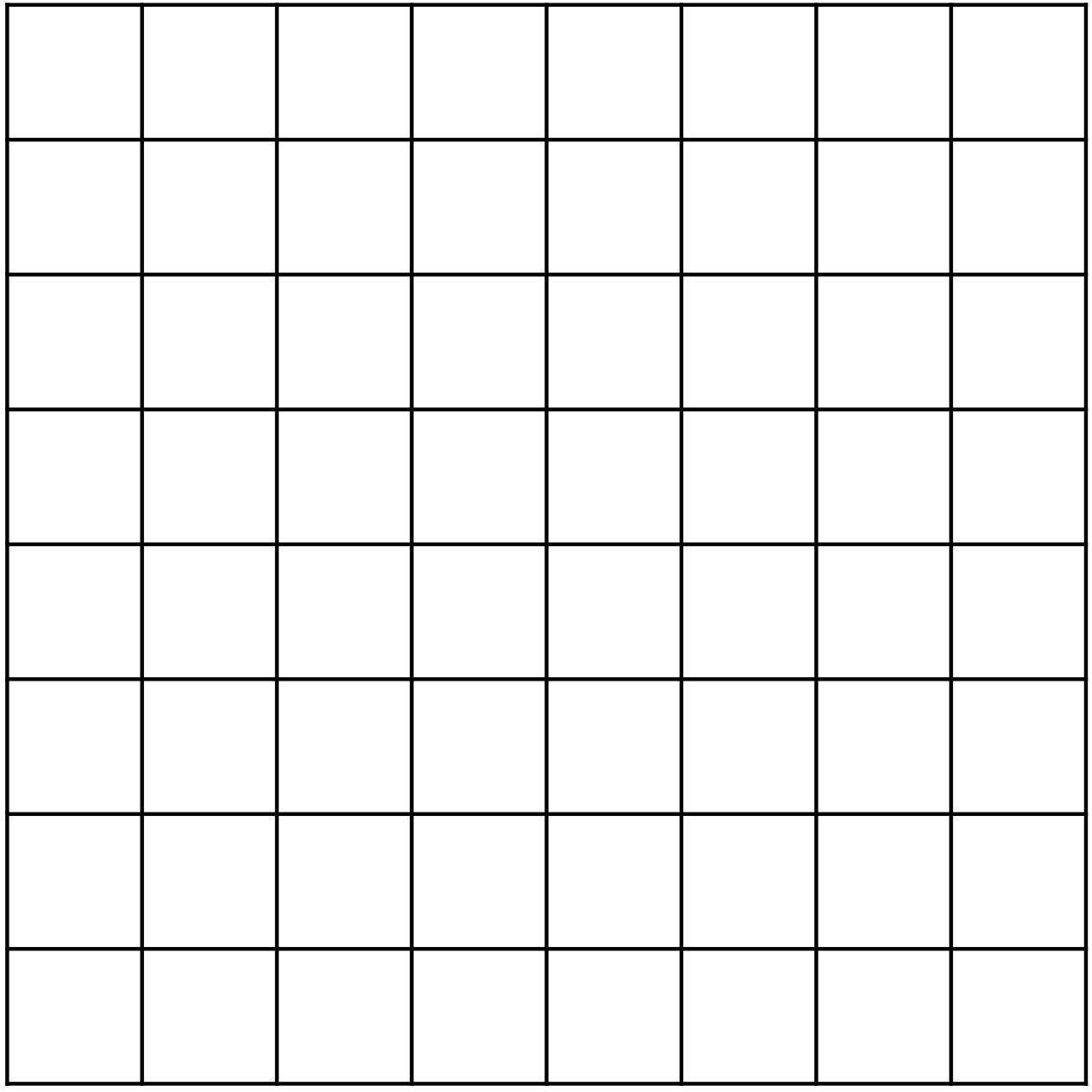
Ricordiamo che $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$ e $\binom{i}{i} = 1 = \binom{i}{0}$.

```
IterChoose(n,r)
1. FOR(i=0,i<n+1, i=i+1) {
2.    T[i,0]= 1
      }
3. FOR(i=0,i<r+1,i=i+1) {
4.    T[i,i]=1
      }
5. FOR(j=1,j<r+1,j=j+1) {
6.    FOR(i=j+1,i<n+1,i=i+1) {
7.      T[i,j]=T[i-1,j-1]+T[i-1,j]
        }
      }
RETURN T[n,r]
```

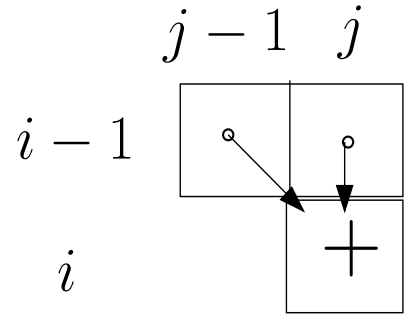
$\binom{n-1}{r-1} + \binom{n-1}{r}$



Al momento dell' assegnazione `T[i,j] = T[i-1,j-1] + T[i-1,j]` occorre che `T[i-1,j-1]` e + `T[i-1,j]` siano stati già calcolati (e infatti l'algoritmo correttamente procede in questo modo).

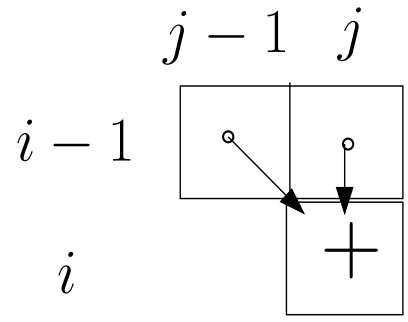


$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



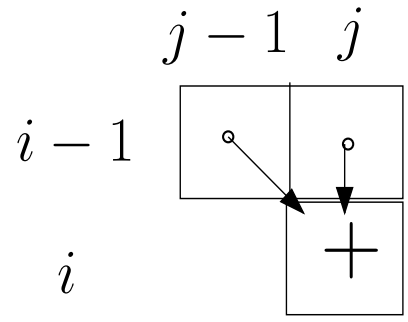
1							
1							
1							
1							
1							
1							
1							
1							

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



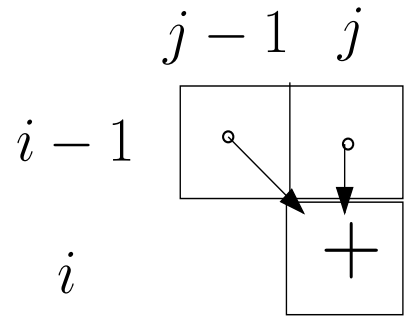
1							
1	1						
1		1					
1			1				
1				1			
1					1		
1						1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



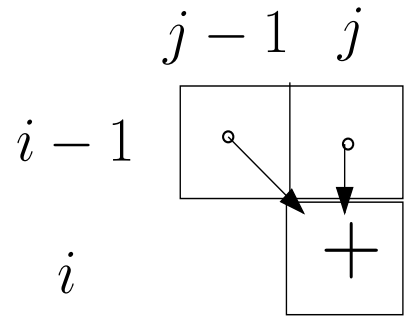
1							
1	1						
1	2	1					
1			1				
1				1			
1					1		
1						1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



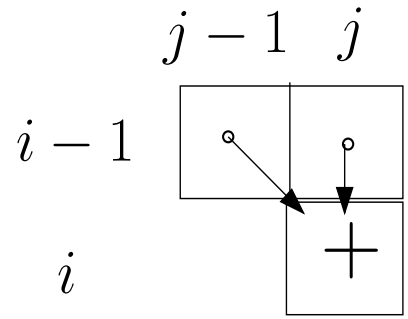
1							
1	1						
1	2	1					
1	3		1				
1				1			
1					1		
1						1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



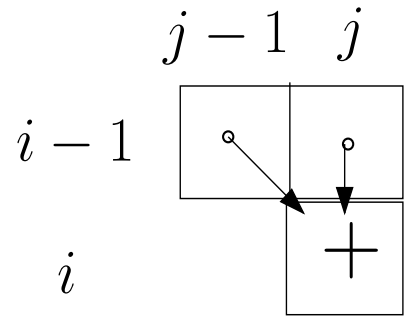
1							
1	1						
1	2	1					
1	3	3	1				
1				1			
1					1		
1						1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



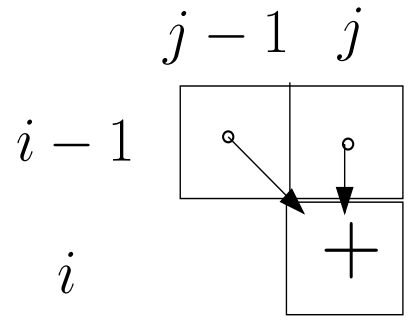
1							
1	1						
1	2	1					
1	3	3	1				
1	4			1			
1					1		
1						1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



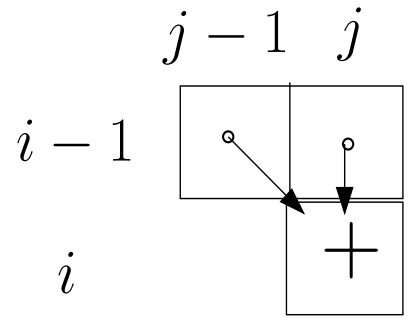
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6		1			
1					1		
1						1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



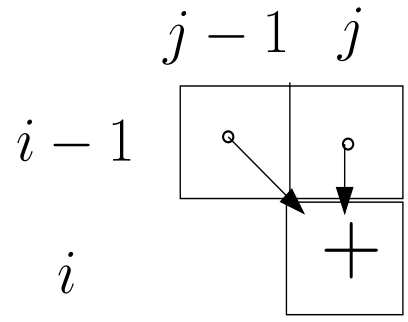
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1					1		
1						1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



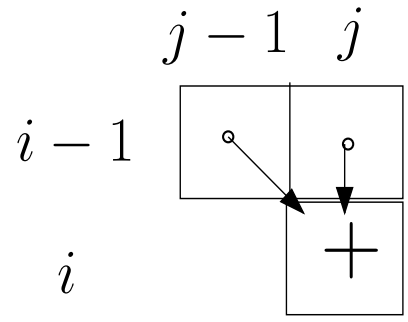
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5				1		
1						1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



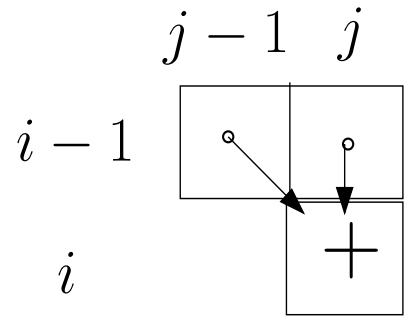
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10			1		
1						1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



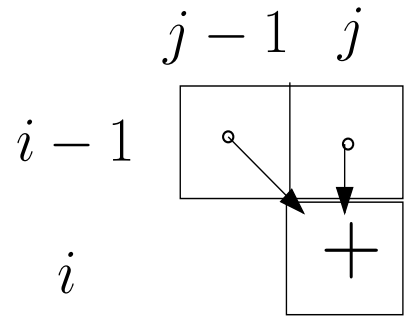
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10		1		
1						1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



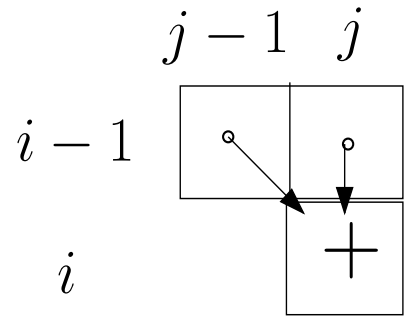
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1						1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



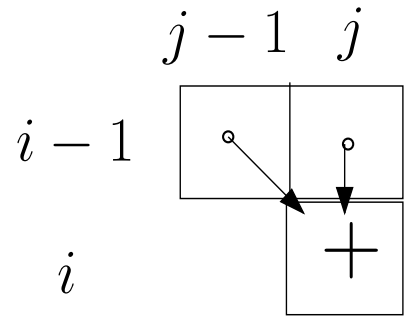
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1	6					1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



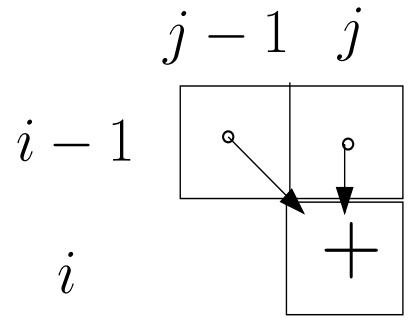
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1	6	15				1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



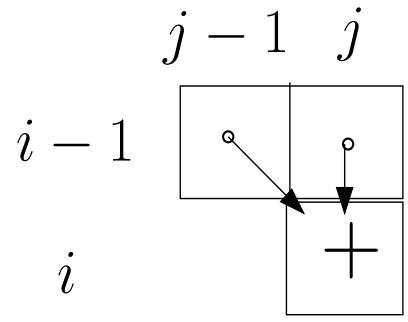
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1	6	15	20			1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



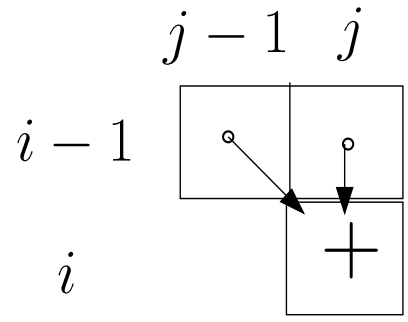
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1	6	15	20	15		1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



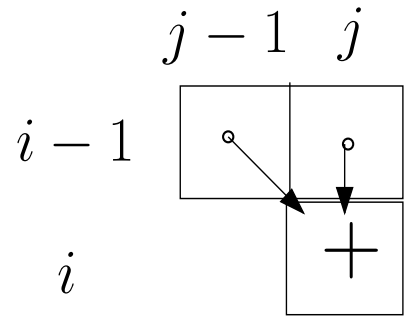
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1	6	15	20	15	6	1	
1							1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



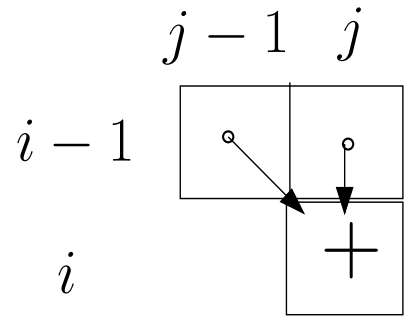
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1	6	15	20	15	6	1	
1	7						1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



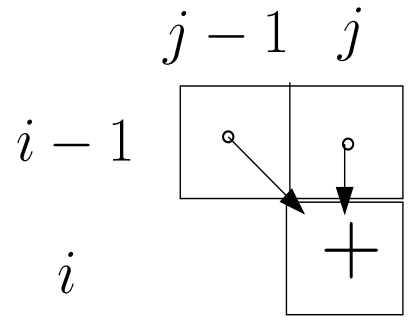
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1	6	15	20	15	6	1	
1	7	21					1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



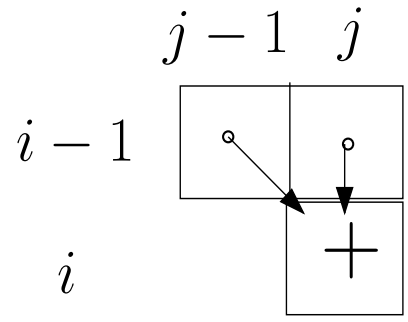
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1	6	15	20	15	6	1	
1	7	21	35				1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



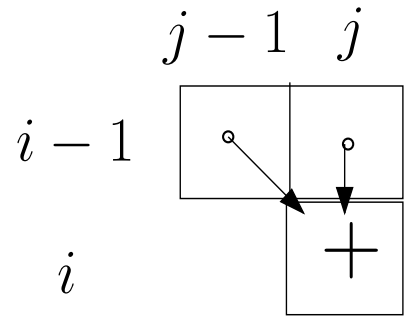
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1	6	15	20	15	6	1	
1	7	21	35	35			1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



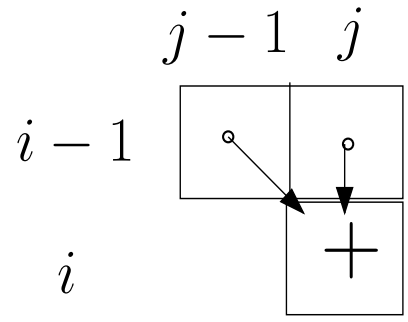
1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1	6	15	20	15	6	1	
1	7	21	35	35	21		1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1	6	15	20	15	6	1	
1	7	21	35	35	21	7	1

$$\binom{n-1}{r-1} + \binom{n-1}{r}$$



Analizziamo l'algoritmo `IterChoose(n,r)`.

Analizziamo l'algoritmo `IterChoose(n,r)`.

La tabella `T[0...n,0...r]` ha $n \cdot r \leq n^2$ entrate,

Analizziamo l'algoritmo `IterChoose(n,r)`.

La tabella `T[0...n,0...r]` ha $n \cdot r \leq n^2$ entrate, ciascuna viene calcolata con una addizione dalle precedenti due (ciò richiede tempo $O(1)$).

Analizziamo l'algoritmo `IterChoose(n,r)`.

La tabella `T[0...n,0...r]` ha $n \cdot r \leq n^2$ entrate, ciascuna viene calcolata con una addizione dalle precedenti due (ciò richiede tempo $O(1)$).

Quindi l'algoritmo ha complessità $O(n^2)$ (ricordiamo che il primo algoritmo ricorsivo aveva complessità $\Theta(2^n)$).

Analizziamo l'algoritmo `IterChoose(n,r)`.

La tabella `T[0...n,0...r]` ha $n \cdot r \leq n^2$ entrate, ciascuna viene calcolata con una addizione dalle precedenti due (ciò richiede tempo $O(1)$).

Quindi l'algoritmo ha complessità $O(n^2)$ (ricordiamo che il primo algoritmo ricorsivo aveva complessità $\Theta(2^n)$).

L'algoritmo usa $O(n^2)$ spazio per memorizzare la tabella,

Analizziamo l'algoritmo `IterChoose(n,r)`.

La tabella `T[0...n,0...r]` ha $n \cdot r \leq n^2$ entrate, ciascuna viene calcolata con una addizione dalle precedenti due (ciò richiede tempo $O(1)$).

Quindi l'algoritmo ha complessità $O(n^2)$ (ricordiamo che il primo algoritmo ricorsivo aveva complessità $\Theta(2^n)$).

L'algoritmo usa $O(n^2)$ spazio per memorizzare la tabella, ma poichè `T[i,j]=T[i-1,j-1]+T[i-1,j]`, ad ogni iterazione dell'algoritmo basta solo memorizzare la colonna $j-1$ e j.

Analizziamo l'algoritmo `IterChoose(n,r)`.

La tabella `T[0...n,0...r]` ha $n \cdot r \leq n^2$ entrate, ciascuna viene calcolata con una addizione dalle precedenti due (ciò richiede tempo $O(1)$).

Quindi l'algoritmo ha complessità $O(n^2)$ (ricordiamo che il primo algoritmo ricorsivo aveva complessità $\Theta(2^n)$).

L'algoritmo usa $O(n^2)$ spazio per memorizzare la tabella, ma poichè `T[i,j]=T[i-1,j-1]+T[i-1,j]`, ad ogni iterazione dell'algoritmo basta solo memorizzare la colonna $j-1$ e j. Quindi l'algoritmo necessita solo di $O(n)$ locazioni di memoria.

Morale della lezione

Morale della lezione

- Quando, nella risoluzione di un problema, Divide et Impera genera molti sottoproblemi identici, la ricorsione porta ad algoritmi inefficienti.

Morale della lezione

- Quando, nella risoluzione di un problema, Divide et Impera genera molti sottoproblemi identici, la ricorsione porta ad algoritmi inefficienti.
- Conviene allora memorizzare la soluzione ai sottoproblemi in una tabella, e leggerli all'occorrenza.

Morale della lezione

- Quando, nella risoluzione di un problema, Divide et Impera genera molti sottoproblemi identici, la ricorsione porta ad algoritmi inefficienti.
- Conviene allora memorizzare la soluzione ai sottoproblemi in una tabella, e leggerli all'occorrenza.
- Questa è l'essenza della Programmazione Dinamica