Terzo allenamento

Olimpiadi Italiane di Informatica - Selezione territoriale

Luca Chiodini
luca@chiodini.org - l.chiodini@campus.unimib.it

16 marzo 2017

Programma

1. Lettura e analisi di un problema
2. Soluzione naïve
3. Soluzione naïve migliorata
4. Spiegazione teorica
5. Soluzione ottima

Lettura e analisi di un problema

“Sommelier” (territoriali 2014)

“Sommelier” (territoriali 2014)

Paolo, per festeggiare il suo quarantesimo compleanno, si è iscritto a un corso per sommelier, dove impara a distinguere ed apprezzare le diverse tipologie di vini.

Si è accorto però che, nonostante prenda solo un assaggio di ogni tipo di vino, per lui vale la regola fondamentale delle bevande alcoliche: quando le bevi, mai scendere di gradazione. Infatti, se per esempio Paolo assaggia un vino da 9 gradi e poi uno da 7, il giorno dopo si sveglierà con un grosso mal di testa indipendentemente dalle quantità.

“Sommelier” (territoriali 2014)

Per fortuna, in ogni serata del corso è disponibile l’elenco dei vini che verranno portati uno dopo l’altro, e di ogni vino viene riportata la gradazione alcolica. Non è ammesso mettere da parte un vino per berlo in seguito: ogni volta che gli viene passato un vino Paolo può decidere se assaggiarlo o meno, versandone un poco nel suo Tastevin.

Inoltre, dal momento che dopo aver assaggiato un vino Paolo deve pulire accuratamente il suo Tastevin con un panno, questa operazione in pratica gli impedisce di assaggiare due vini consecutivi. Paolo desidera assaggiare il maggior numero di vini possibile.

“Sommelier” (territoriali 2014)

Assunzioni

- $2 \leq N \leq 99$.
- I vini hanno una gradazione alcolica compresa tra 1 e 99

input.txt	output.txt
9 11 13 10 16 12 12 13 11 13	4

Soluzione naïve

“Sommelier” - Soluzione naïve

In questo genere di problemi, un approccio che funziona *sempre* è il cosiddetto metodo “forza bruta” (detto informalmente: “le provo tutte”). Come si può applicare in questo problema?

11	13	10	16	12	12	13	11	13

“Sommelier” - Soluzione naïve

In questo genere di problemi, un approccio che funziona *sempre* è il cosiddetto metodo “forza bruta” (detto informalmente: “le provo tutte”). Come si può applicare in questo problema?

11	13	10	16	12	12	13	11	13

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1	1

. . .

1	1	1	1	1	1	1	1	1

“Sommelier” - Soluzione naïve

Costruiamo un vettore V dove $V_i = 1$ sse Paolo beve l’i-esimo vino.

“Sommelier” - Soluzione naïve

Costruiamo un vettore V dove $V_i = 1$ sse Paolo beve l’i-esimo vino.

Chiaramente non tutte le configurazioni sono *valide*. La seguente è una configurazione valida?

11	13	10	16	12	12	13	11	13

1	0	0	0	0	0	0	0	0

“Sommelier” - Soluzione naïve

Costruiamo un vettore V dove $V_i = 1$ sse Paolo beve l’i-esimo vino.

Chiaramente non tutte le configurazioni sono *valide*. La seguente è una configurazione valida?

11	13	10	16	12	12	13	11	13

1	0	0	0	0	1	0	1	0

“Sommelier” - Soluzione naïve

Costruiamo un vettore V dove $V_i = 1$ sse Paolo beve l’i-esimo vino.

Chiaramente non tutte le configurazioni sono *valide*. La seguente è una configurazione valida?

11	13	10	16	12	12	13	11	13

0	0	0	0	0	1	1	0	0

“Sommelier” - Soluzione naïve

Soluzione

La soluzione consiste quindi nel cercare tra tutte le possibili configurazioni *valide* quella che permette a Paolo di assaggiare il *maggior numero* di vini.

“Sommelier” - Soluzione naïve

Soluzione

La soluzione consiste quindi nel cercare tra tutte le possibili configurazioni *valide* quella che permette a Paolo di assaggiare il *maggior numero* di vini.

Complessità computazionale

Qual è la complessità computazionale della soluzione?

“Sommelier” - Soluzione naïve

Soluzione

La soluzione consiste quindi nel cercare tra tutte le possibili configurazioni *valide* quella che permette a Paolo di assaggiare il *maggior numero* di vini.

Complessità computazionale

Qual è la complessità computazionale della soluzione?

- Intuitivamente: abbiamo due possibilità per la prima posizione del vettore. Quindi, fissata la prima posizione, ne abbiamo altre due per la seconda posizione (e così via). In tutto quindi

$$\underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{N \text{ volte}}$$

“Sommelier” - Soluzione naïve

Soluzione

La soluzione consiste quindi nel cercare tra tutte le possibili configurazioni *valide* quella che permette a Paolo di assaggiare il *maggior numero* di vini.

Complessità computazionale

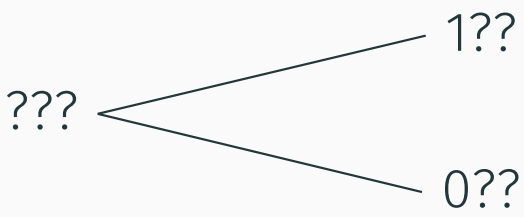
Qual è la complessità computazionale della soluzione?

- Intuitivamente: abbiamo due possibilità per la prima posizione del vettore. Quindi, fissata la prima posizione, ne abbiamo altre due per la seconda posizione (e così via). In tutto quindi

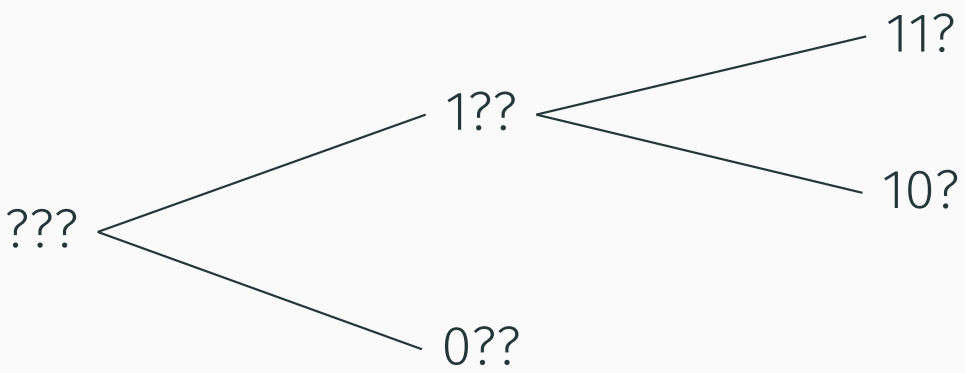
$$\underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{N \text{ volte}}$$

- Calcolo combinatorio: disposizioni con ripetizione di due elementi (0 - 1) su N posizioni $\implies 2^N$

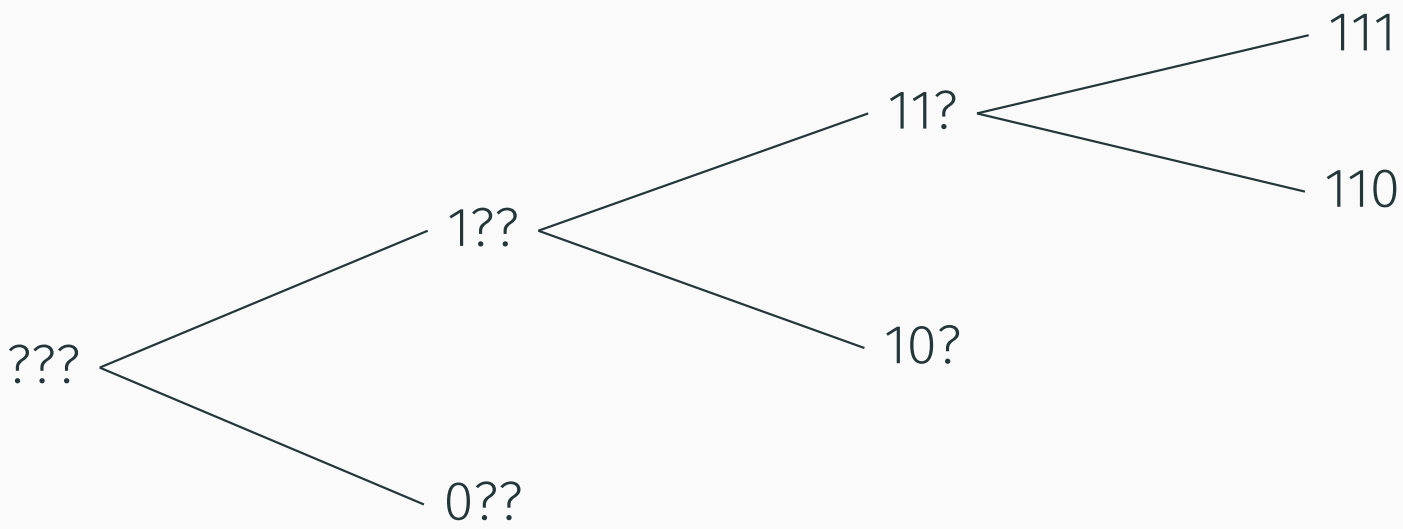
“Sommelier” - Soluzione naïve



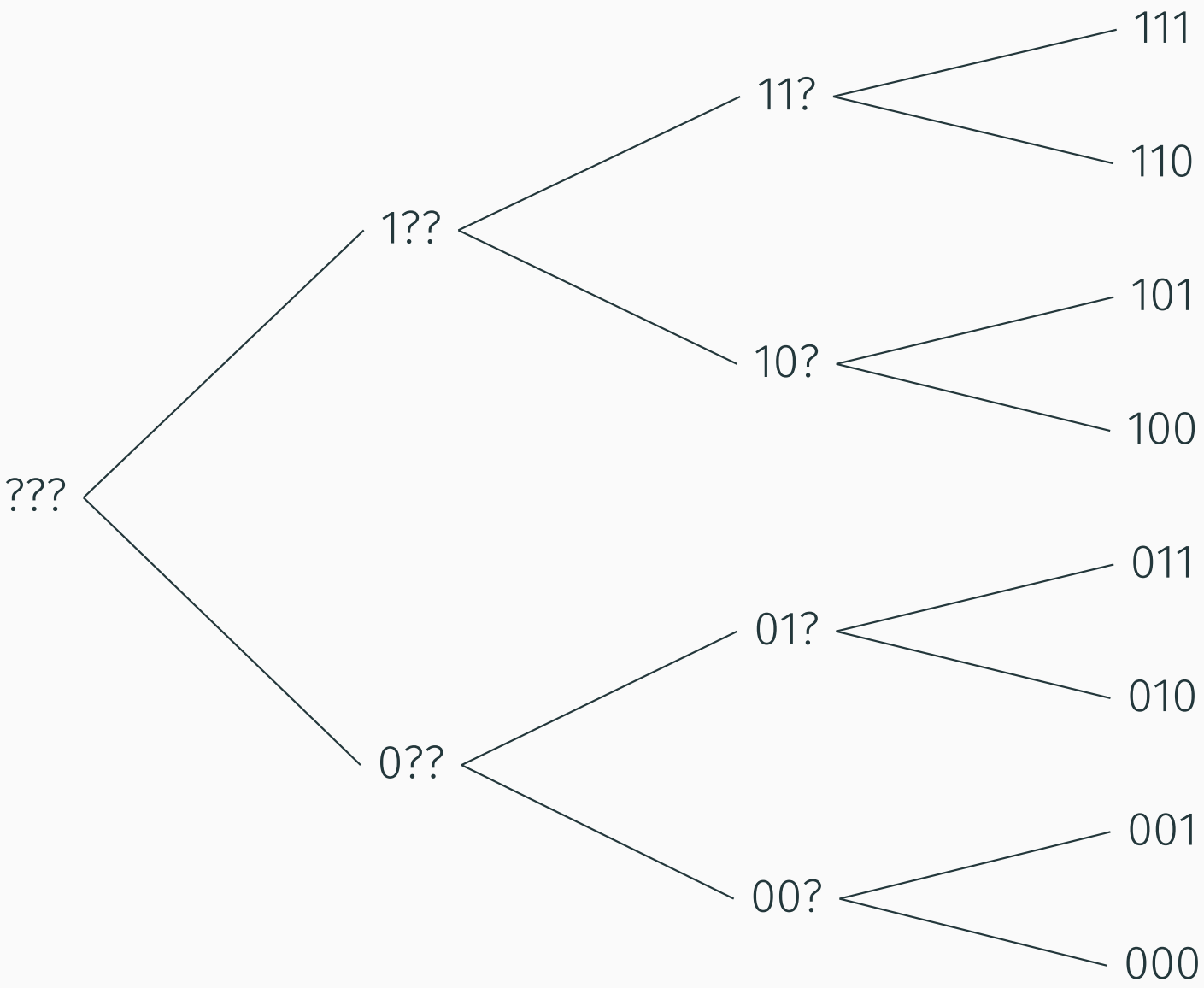
“Sommelier” - Soluzione naïve



“Sommelier” - Soluzione naïve



“Sommelier” - Soluzione naïve



“Sommelier” - Soluzione naïve

```
void genera_disposizioni(bool V[], int posizione)
{
    if (posizione == N)
    {
        if (controlla(V))
            // Conta vini e aggiorna massimo.
    }
    else
    {
        V[posizione] = false;
        genera_disposizioni(V, posizione + 1);
        V[posizione] = true;
        genera_disposizioni(V, posizione + 1);
    }
}
```

Soluzione naïve migliorata

“Sommelier” - Soluzione naïve migliorata

Lentezza della soluzione esponenziale

La precedente soluzione è esponenziale e per $N = 99$ è quindi troppo lenta. Viste le considerazioni fatte in precedenza sulla validità di alcune configurazioni, cosa potremmo migliorare?

“Sommelier” - Soluzione naïve migliorata

Lentezza della soluzione esponenziale

La precedente soluzione è esponenziale e per $N = 99$ è quindi troppo lenta. Viste le considerazioni fatte in precedenza sulla validità di alcune configurazioni, cosa potremmo migliorare? Evitiamo di generare configurazioni non valide:

- Configurazioni che prevedono due vini consecutivi
- Configurazioni che prevedono di bere un vino con una gradazione alcolica inferiore

“Sommelier” - Soluzione naïve

```
int conta(int pos, int minimo) {
    if (pos >= N) return 0;
    int conta_preso = 0, conta_non_preso = 0;

    // Provo a prenderlo, se mi è possibile
    if (vini[pos] >= minimo)
        conta_preso = 1 + conta(pos + 2, vini[pos]);

    // Oppure provo a non prenderlo
    conta_non_preso = conta(pos + 1, minimo);
    return max(conta_preso, conta_non_preso);
}
```

Spiegazione teorica

Numeri di Fibonacci

Numeri di Fibonacci

Sequenza di Fibonacci

I primi numeri della sequenza di Fibonacci sono
$1, 1, 2, 3, 5, 8, 13, 21, \ldots$

Numeri di Fibonacci

Sequenza di Fibonacci

I primi numeri della sequenza di Fibonacci sono
$1, 1, 2, 3, 5, 8, 13, 21, \ldots$

Formula ricorsiva

Si osserva abbastanza facilmente che vale la seguente formula ricorsiva per calcolare l'n-esimo numero di Fibonacci.

$$f(n) = \begin{cases} 1 & n \leq 2 \\ f(n-1) + f(n-2) & n > 2 \end{cases}$$

Fibonacci ricorsivo in C++

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

Fibonacci ricorsivo in C++

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

Complessità computazionale

Quante chiamate a `fibonacci()` vengono fatte questo algoritmo?

Fibonacci ricorsivo in C++

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

Complessità computazionale

Quante chiamate a `fibonacci()` vengono fatte questo algoritmo?

- La prima chiamata ne produce altre due

Fibonacci ricorsivo in C++

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

Complessità computazionale

Quante chiamate a `fibonacci()` vengono fatte questo algoritmo?

- La prima chiamata ne produce altre due
- Ciascuna di queste due ne produce altre due (quattro, in totale)

Fibonacci ricorsivo in C++

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

Complessità computazionale

Quante chiamate a `fibonacci()` vengono fatte questo algoritmo?

- La prima chiamata ne produce altre due
- Ciascuna di queste due ne produce altre due (quattro, in totale)
- Ciascuna di queste quattro ne produce altre due (otto, in totale)

Fibonacci ricorsivo in C++

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

Complessità computazionale

Quante chiamate a `fibonacci()` vengono fatte questo algoritmo?

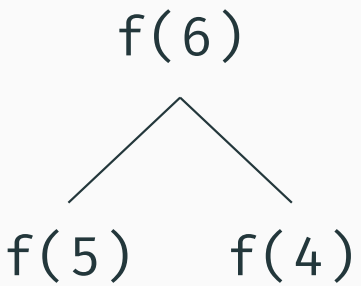
- La prima chiamata ne produce altre due
- Ciascuna di queste due ne produce altre due (quattro, in totale)
- Ciascuna di queste quattro ne produce altre due (otto, in totale)

La complessità computazione finale è quindi $\mathcal{O}(2^n)$

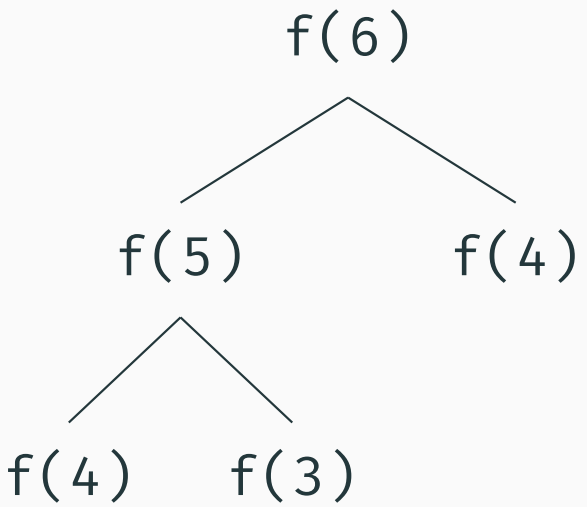
Calcolo di `fibonacci(6)`

`f(6)`

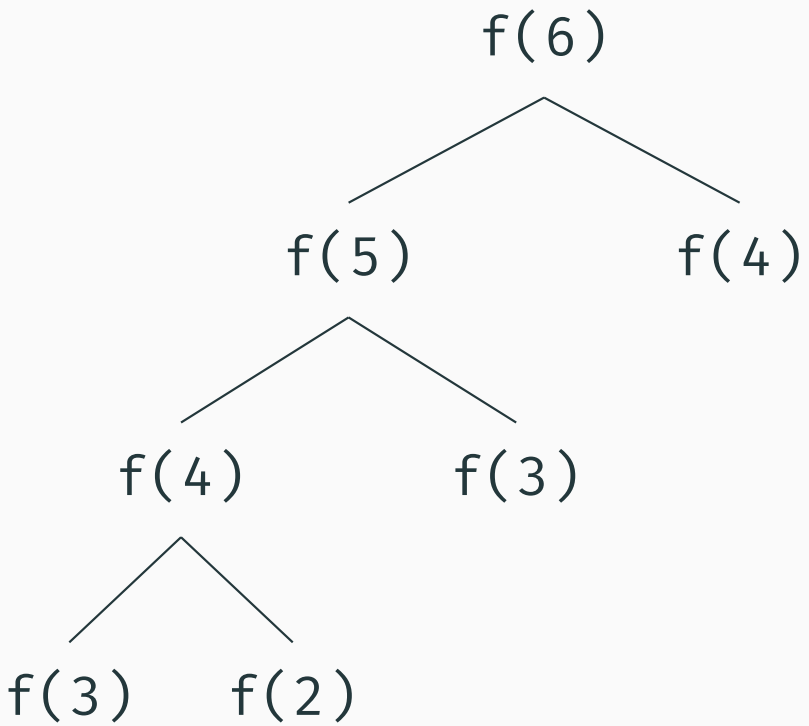
Calcolo di `fibonacci(6)`



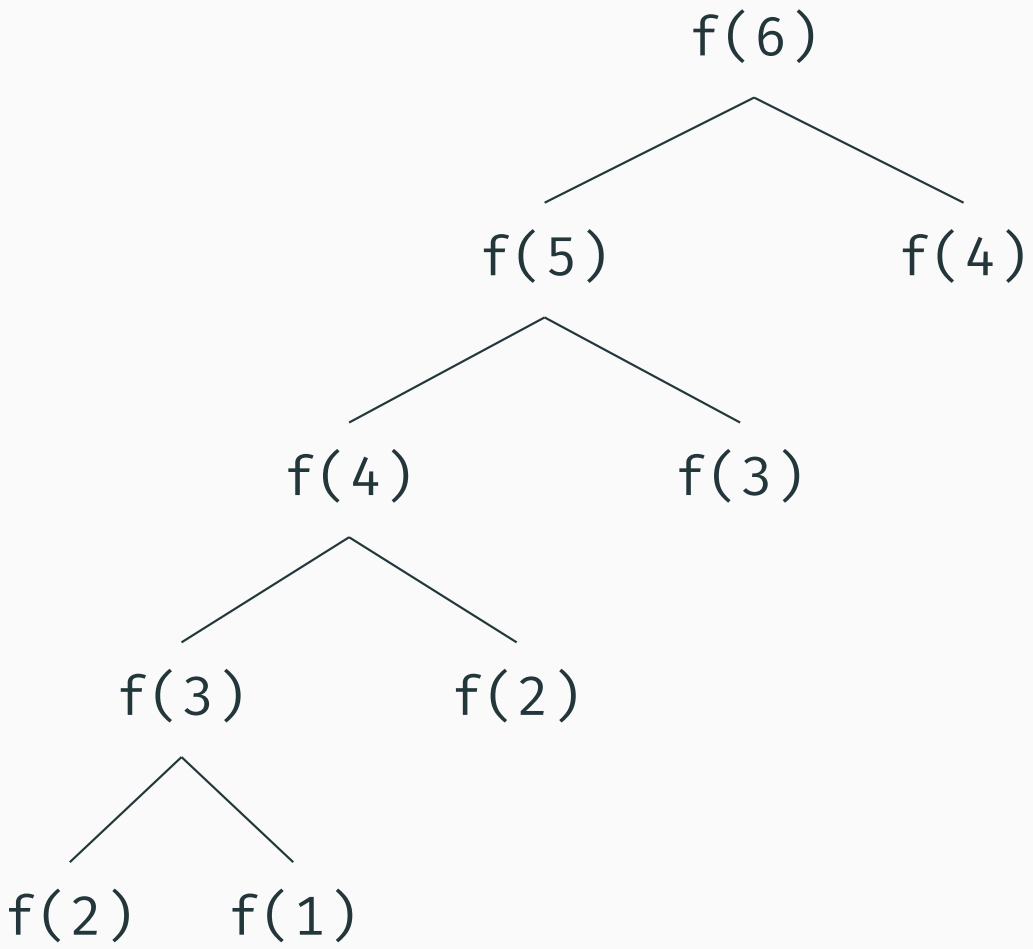
Calcolo di `fibonacci(6)`



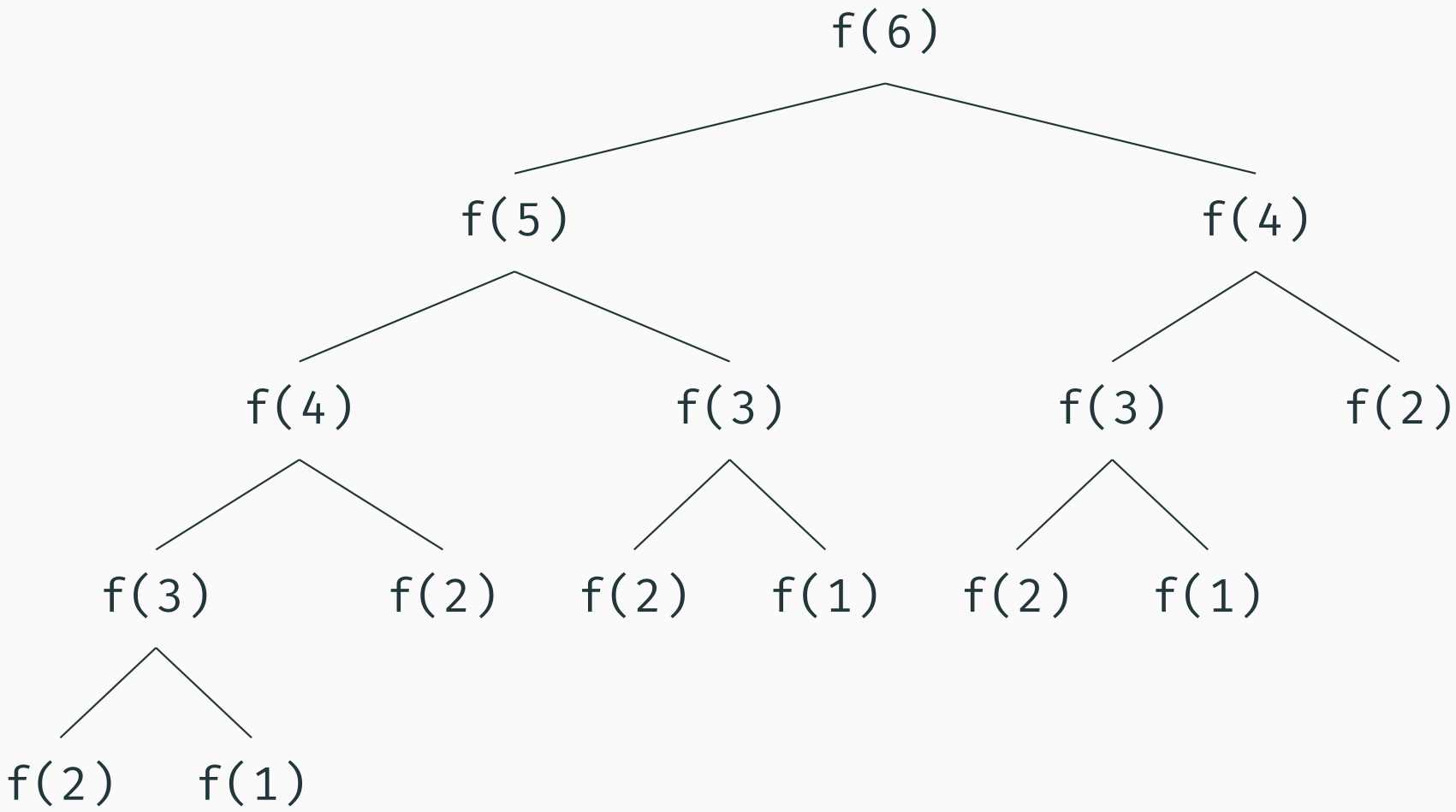
Calcolo di `fibonacci(6)`



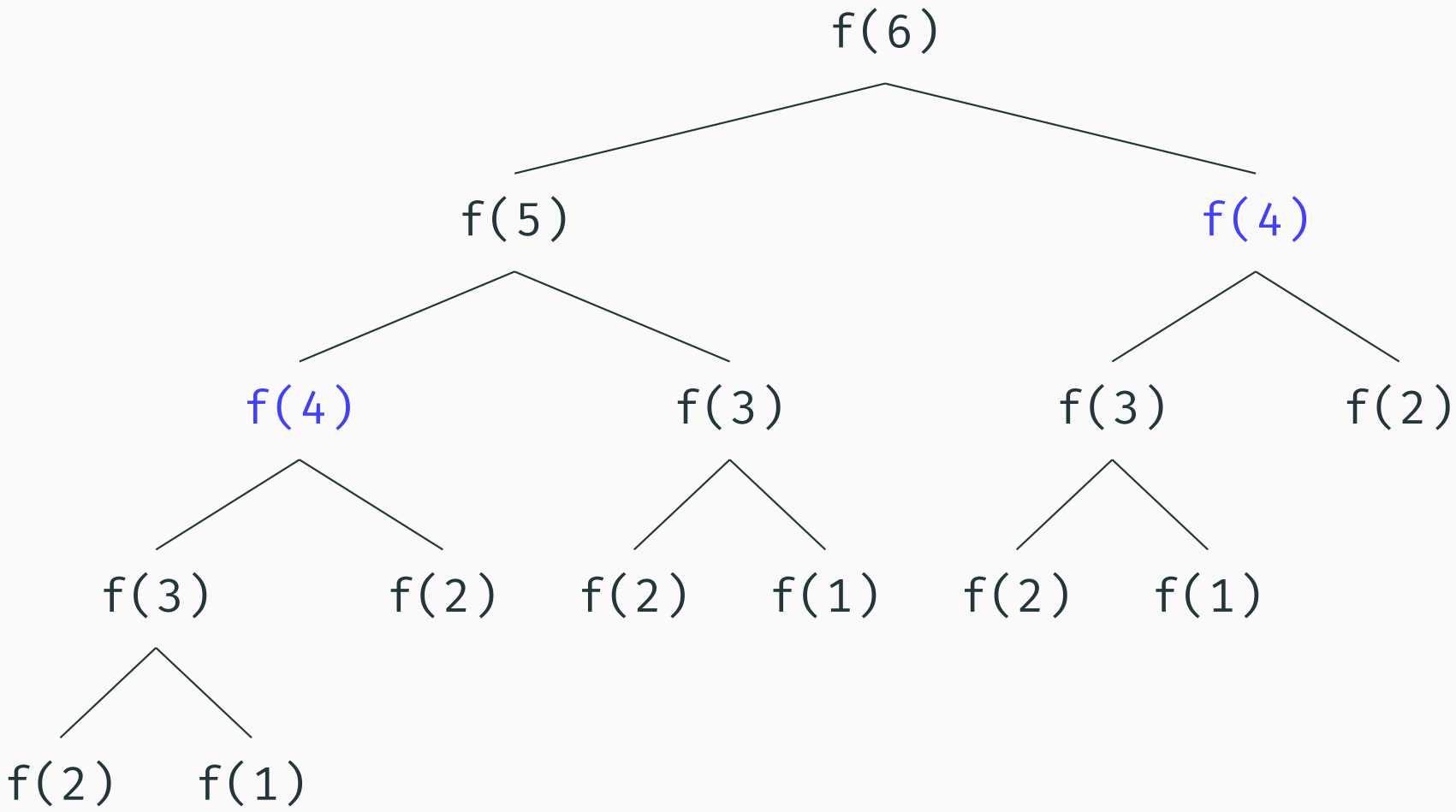
Calcolo di `fibonacci(6)`



Calcolo di `fibonacci(6)`

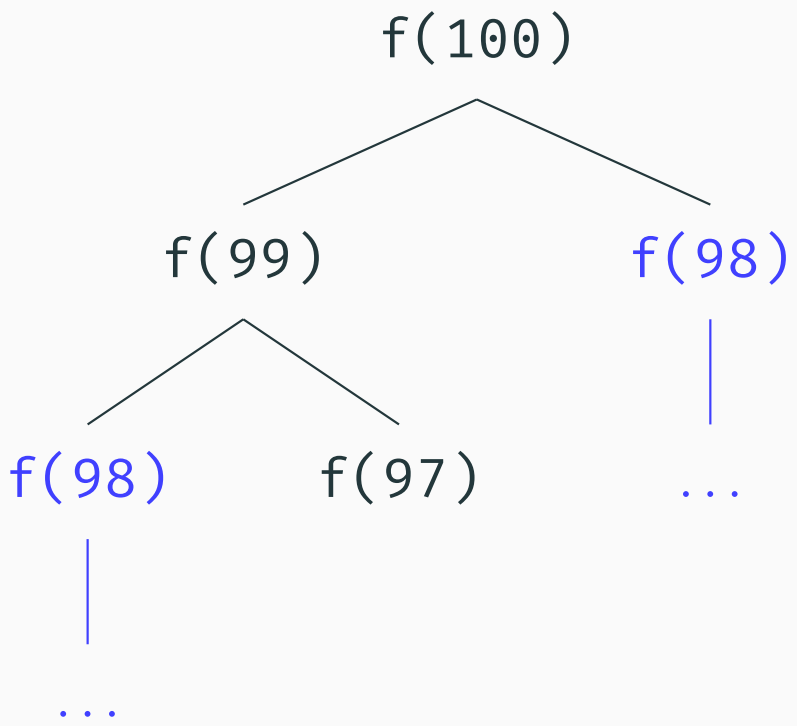


Ricalcolo di sottoproblemi in `fibonacci(6)`



Ricalcolo di sottoproblemi in `fibonacci(100)`

Ricalcolo di sottoproblemi in `fibonacci(100)`



Risoluzione di sottoproblemi

Inefficienza di `fibonacci` ricorsivo

È evidente che questa implementazione è particolarmente inefficiente perché calcola più volte "le stesse cose". Poiché per risolvere il problema `fibonacci(n)` ho bisogno di risolvere `fibonacci(n-1)` e `fibonacci(n-2)` (che sono "più facili" perché "più vicini" al caso base), chiamiamo questi ultimi *sottoproblemi*.

Risoluzione di sottoproblemi

Inefficienza di `fibonacci` ricorsivo

È evidente che questa implementazione è particolarmente inefficiente perché calcola più volte "le stesse cose". Poiché per risolvere il problema `fibonacci(n)` ho bisogno di risolvere `fibonacci(n-1)` e `fibonacci(n-2)` (che sono "più facili" perché "più vicini" al caso base), chiamiamo questi ultimi *sottoproblemi*.

Verso la soluzione

Dobbiamo quindi progettare un algoritmo che *non* ricalcoli più di una volta lo stesso sottoproblema.

Vettore dei sottoproblemi

Costruiamo un vettore *fib* dove fib_i contiene l'i-esimo numero di Fibonacci (che rappresenta la soluzione del nostro sottoproblema).

Soluzione

- Riempiamo i valori iniziali, che sono noti;
- A partire dalla prima cella bianca, calcoliamo il valore sommando i due elementi precedenti nel vettore.

1	1				

Vettore dei sottoproblemi

Costruiamo un vettore *fib* dove fib_i contiene l'i-esimo numero di Fibonacci (che rappresenta la soluzione del nostro sottoproblema).

Soluzione

- Riempiamo i valori iniziali, che sono noti;
- A partire dalla prima cella bianca, calcoliamo il valore sommando i due elementi precedenti nel vettore.

1	1	2			

Vettore dei sottoproblemi

Costruiamo un vettore *fib* dove fib_i contiene l'i-esimo numero di Fibonacci (che rappresenta la soluzione del nostro sottoproblema).

Soluzione

- Riempiamo i valori iniziali, che sono noti;
- A partire dalla prima cella bianca, calcoliamo il valore sommando i due elementi precedenti nel vettore.

1	1	2	3		

Vettore dei sottoproblemi

Costruiamo un vettore *fib* dove fib_i contiene l'i-esimo numero di Fibonacci (che rappresenta la soluzione del nostro sottoproblema).

Soluzione

- Riempiamo i valori iniziali, che sono noti;
- A partire dalla prima cella bianca, calcoliamo il valore sommando i due elementi precedenti nel vettore.

1	1	2	3	5	

Vettore dei sottoproblemi

Costruiamo un vettore *fib* dove fib_i contiene l'i-esimo numero di Fibonacci (che rappresenta la soluzione del nostro sottoproblema).

Soluzione

- Riempiamo i valori iniziali, che sono noti;
- A partire dalla prima cella bianca, calcoliamo il valore sommando i due elementi precedenti nel vettore.

1	1	2	3	5	8

Qual è la complessità computazionale?

Vettore dei sottoproblemi

Costruiamo un vettore *fib* dove fib_i contiene l'i-esimo numero di Fibonacci (che rappresenta la soluzione del nostro sottoproblema).

Soluzione

- Riempiamo i valori iniziali, che sono noti;
- A partire dalla prima cella bianca, calcoliamo il valore sommando i due elementi precedenti nel vettore.

1	1	2	3	5	8

Qual è la complessità computazionale? $\mathcal{O}(n)$.

Programmazione dinamica

Chiamiamo questa tecnica *programmazione dinamica*.

Scaletta per la programmazione dinamica

- Identificare i sottoproblemi che è necessario considerare.
- Trovare dei parametri che identifichino univocamente i sottoproblemi.
- Trovare un ordine in cui riempire il vettore (o la tabella) in modo che per calcolare qualsiasi elemento sia sufficiente “guardare” le caselle riempite in precedenza.

Soluzione ottima

Idea di soluzione

Dobbiamo utilizzare la *programmazione dinamica*. Riprendiamo la scaletta vista in precedenza.

Scaletta per la programmazione dinamica

- Identificare i sottoproblemi che è necessario considerare.
- Trovare dei parametri che identifichino univocamente i sottoproblemi.
- Trovare un ordine in cui riempire il vettore (o la tabella) in modo che per calcolare qualsiasi elemento sia sufficiente "guardare" le caselle riempite in precedenza.

Idea di soluzione

Dobbiamo utilizzare la *programmazione dinamica*. Riprendiamo la scaletta vista in precedenza.

Scaletta per la programmazione dinamica

- Identificare i sottoproblemi che è necessario considerare.
- Trovare dei parametri che identifichino univocamente i sottoproblemi.
- Trovare un ordine in cui riempire il vettore (o la tabella) in modo che per calcolare qualsiasi elemento sia sufficiente "guardare" le caselle riempite in precedenza.

Il problema generale, con tanti vini, è difficile...
Se ne avessimo di meno?

Idea di soluzione

Dobbiamo utilizzare la *programmazione dinamica*. Riprendiamo la scaletta vista in precedenza.

Scaletta per la programmazione dinamica

- Identificare i sottoproblemi che è necessario considerare.
- Trovare dei parametri che identifichino univocamente i sottoproblemi.
- Trovare un ordine in cui riempire il vettore (o la tabella) in modo che per calcolare qualsiasi elemento sia sufficiente "guardare" le caselle riempite in precedenza.

Il problema generale, con tanti vini, è difficile...
Se ne avessimo di meno? Se ne avessimo solo uno?

Riempimento del vettore `din`

Creiamo e riempiamo opportunamente un vettore `din` che conterrà le soluzioni dei sottoproblemi.

`vini`	11	13	10	16	12	12	13	11	13
`din`									

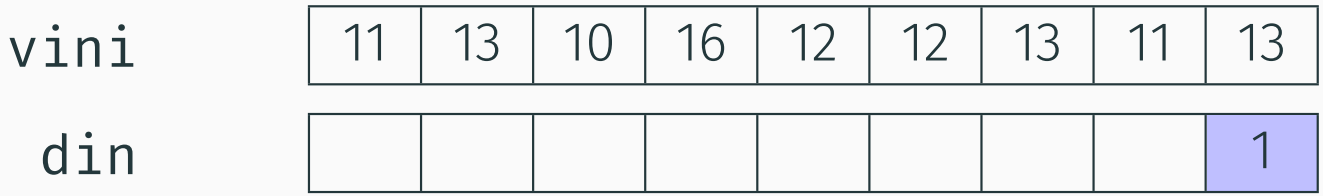
Riempimento del vettore `din`

Creiamo e riempiamo opportunamente un vettore `din` che conterrà le soluzioni dei sottoproblemi.

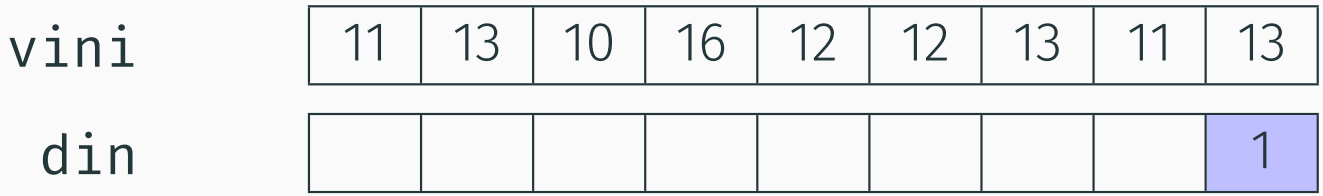
`vini`	11	13	10	16	12	12	13	11	13
`din`									

Quanti vini posso bere al massimo supponendo di partire dall'ultimo?

Riempimento del vettore `din`

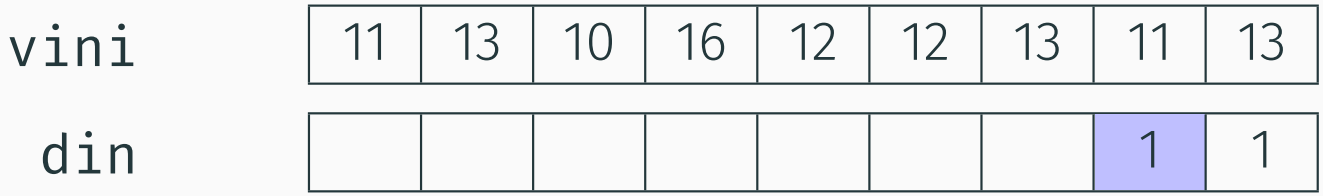


Riempimento del vettore `din`



Quanti vini posso bere al massimo supponendo di partire dal penultimo?

Riempimento del vettore `din`

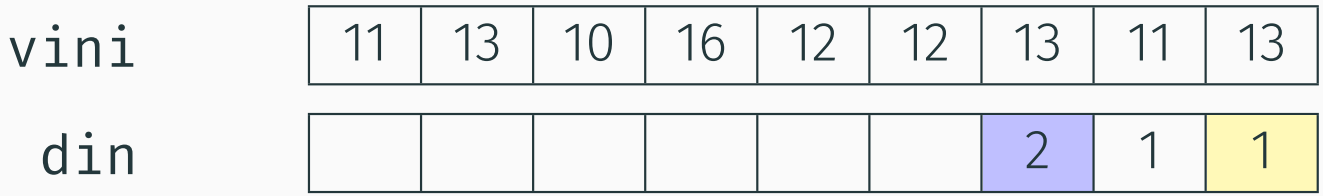


Riempimento del vettore `din`

`vini`	11	13	10	16	12	12	13	11	13
`din`								1	1

Quanti vini posso bere al massimo supponendo di partire dal terzultimo?

Riempimento del vettore `din`

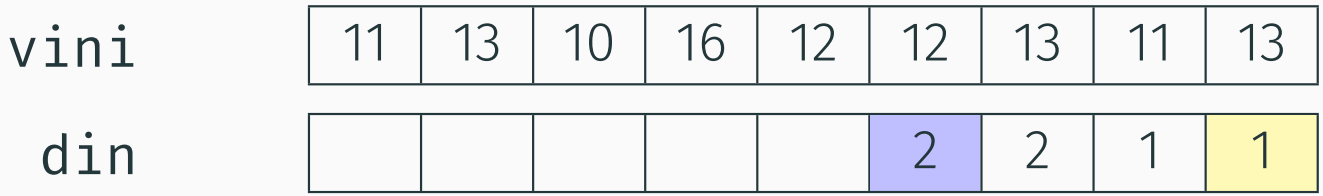


Riempimento del vettore `din`

`vini`	11	13	10	16	12	12	13	11	13
`din`							2	1	1

Quanti vini posso bere al massimo supponendo di partire dal quartultimo?

Riempimento del vettore `din`

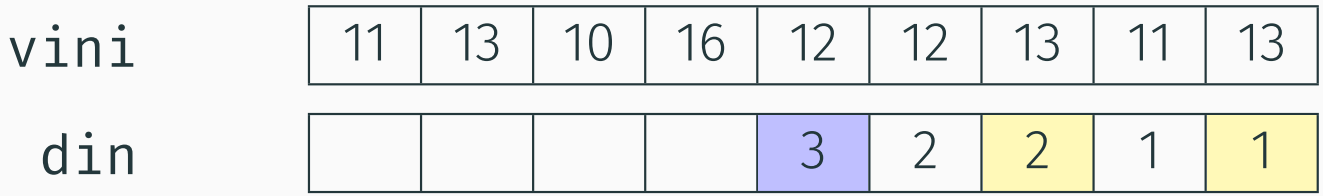


Riempimento del vettore `din`

`vini`	11	13	10	16	12	12	13	11	13
`din`						2	2	1	1

Quanti vini posso bere al massimo supponendo di partire dal quinto?

Riempimento del vettore `din`

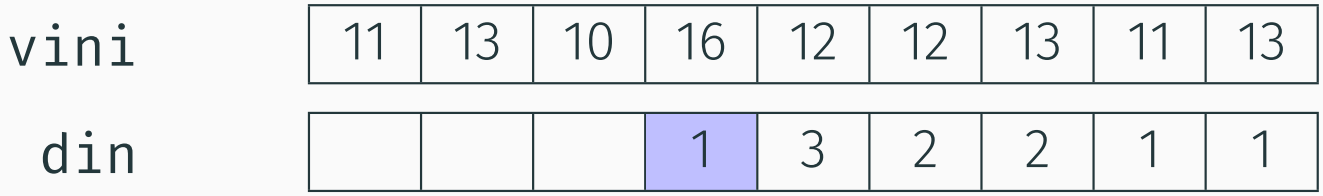


Riempimento del vettore `din`

`vini`	11	13	10	16	12	12	13	11	13
`din`					3	2	2	1	1

Quanti vini posso bere al massimo supponendo di partire dal quarto?

Riempimento del vettore `din`

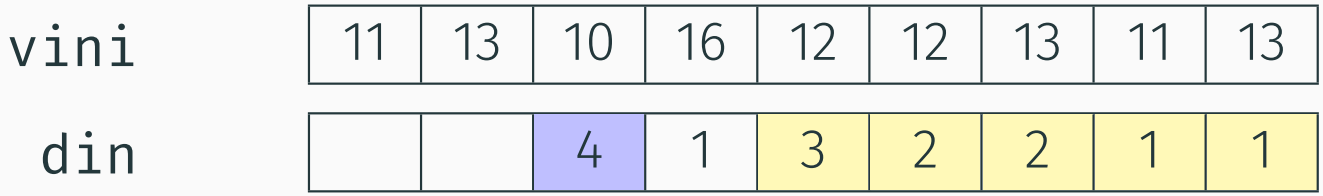


Riempimento del vettore `din`

`vini`	11	13	10	16	12	12	13	11	13
`din`				1	3	2	2	1	1

Quanti vini posso bere al massimo supponendo di partire dal terzo?

Riempimento del vettore `din`

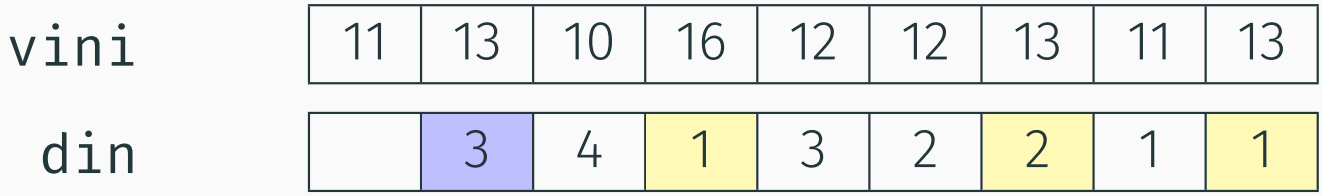


Riempimento del vettore `din`

`vini`	11	13	10	16	12	12	13	11	13
`din`			4	1	3	2	2	1	1

Quanti vini posso bere al massimo supponendo di partire dal secondo?

Riempimento del vettore `din`



Riempimento del vettore `din`

`vini`	11	13	10	16	12	12	13	11	13
`din`		3	4	1	3	2	2	1	1

Quanti vini posso bere al massimo supponendo di partire dal primo?

“Sommelier” - Soluzione ottima

vini	11	13	10	16	12	12	13	11	13
din	4	3	4	1	3	2	2	1	1

“Sommelier” - Soluzione ottima

`vini`	11	13	10	16	12	12	13	11	13
`din`	4	3	4	1	3	2	2	1	1

Soluzione

Ora che tutto il vettore è stato riempito, qual è la soluzione?

“Sommelier” - Soluzione ottima

`vini`	11	13	10	16	12	12	13	11	13
`din`	4	3	4	1	3	2	2	1	1

Soluzione

Ora che tutto il vettore è stato riempito, qual è la soluzione?
Il valore massimo presente nel vettore `din`. In particolare, l’i-esima posizione del vettore contiene il numero massimo di vini bevibili partendo dall’i-esimo vino.

Complessità computazionale

Qual è la complessità computazionale della soluzione?

“Sommelier” - Soluzione ottima

`vini`	11	13	10	16	12	12	13	11	13
`din`	4	3	4	1	3	2	2	1	1

Soluzione

Ora che tutto il vettore è stato riempito, qual è la soluzione?
Il valore massimo presente nel vettore `din`. In particolare, l’i-esima posizione del vettore contiene il numero massimo di vini bevibili partendo dall’i-esimo vino.

Complessità computazionale

Qual è la complessità computazionale della soluzione?
Calcolare il valore di ogni cella richiede $\mathcal{O}(n)$, quindi l’algoritmo ha complessità $\mathcal{O}(n^2)$.

Tocca a voi!

Esercizio

Implementare una soluzione per il problema “sommelier”.

Riferimenti

Questa presentazione e soluzione in C++ dell’esercizio:

`https://files.chiodini.org/OII_Territoriali_2017/`

- Piattaforma di allenamento con correttore e forum:
  `https://cms.di.unipi.it`
- Guida alle selezioni territoriali del prof. Bugatti:
  `http://www.imparando.net/sito/olimpiadi_di_informatica.htm`