Università degli Studi dell'Aquila

Anno Accademico 2024/2025

- Corso Integrato di Algoritmi e Strutture Dati con Laboratorio (12 CFU):
  - Modulo da 6 CFU di Algoritmi e Strutture Dati (Prof. Guido Proietti)
  - Modulo da 6 CFU di Laboratorio di ASD (Dott.ssa Giovanna Melideo)
- Orario: Martedì: 11.30 – 13.30 (A1.7)
  Mercoledì: 9.30 – 11.30 (A1.7)
- Ricevimento: Mercoledì 15.00-17.00 su appuntamento scrivendo a guido.proietti@univaq.it

Obiettivi del corso

Fornire le competenze necessarie per:

- analizzare le principali tecniche di progettazione e analisi degli algoritmi, e saperle valutare in termini di efficienza computazionale rispetto allo specifico problema che si vuole risolvere
- scegliere e realizzare strutture dati adeguate al problema che si vuole risolvere
- sviluppare un’intuizione finalizzata alla soluzione efficiente di problemi computazionali

Prerequisiti del corso

Cosa è necessario sapere:

- strutture dati elementari (array, liste, …)
- concetto di ricorsione
- avere dimestichezza con sommatorie
- dimostrazione per induzione e calcolo infinitesimale

Programma settimanale (13 settimane)

1. **Introduzione**: problemi, algoritmi, complessità computazionale.
2. **Notazione asintotica, problema della ricerca.**
3. **Ordinamento**: Insertion sort, Selection sort.
4. **Ordinamento ottimo**: Lower bound (*), Merge sort (*), Heapsort
5. **Ordinamento efficiente**: Quicksort (*), algoritmi di ordinamento lineari.
6. **Code di priorità**: heap binario, heap binomiale (*).

7. **Prova intermedia (settimana 4-8 novembre 2024)**

8. **Problema del dizionario:** alberi binari di ricerca e alberi AVL (*)
9. **Problema del dizionario:** rotazioni AVL, tavole hash.
10. **Grafi**: definizioni e visite.
11. **Cammini minimi**: Ordinamento topologico, Bellman&Ford.
12. **Cammini minimi**: Dijkstra (*), Floyd&Warshall.
13. **Insiemi disgiunti e Minimo albero ricoprente**: Kruskal (*), Prim, Boruvka.

(*): argomenti fondamentali

Libro di testo

C. Demetrescu, I. Finocchi, G. Italiano
Algoritmi e Strutture dati
McGraw-Hill

Slide e materiale didattico

http://people.disim.univaq.it/guido.proietti/2024.html

Altri testi utili

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein
Introduzione agli algoritmi e strutture dati
McGraw-Hill

P. Crescenzi, G. Gambosi, R. Grossi
Strutture di dati e algoritmi. Progettazione, analisi e visualizzazione
Pearson

Modalità d’esame: appelli

- Sei appelli (+1 a novembre 2025 per i fuori corso a.a. 2024/25)
  - 3 appelli a gennaio-febbraio
  - 2 appelli a giugno-luglio
  - 1 appello a settembre
- L’esame di ASDL (12 CFU) consiste in:
  - Teoria: prova scritta e prova orale **obbligatoria**
  - Laboratorio: prova scritta, seguita da un’eventuale prova orale da svolgersi a **discrezione** della docente o su **richiesta** dello studente
- Propedeuticità: Fondamenti di Programmazione con Laboratorio

Modalità d'esame: scritti e orali

- La prova scritta di teoria consiste in 10 test ragionati a risposta multipla (attenzione, non sono test mnemonici!)
- La prova orale di teoria può essere svolta solo dopo aver superato lo scritto di teoria ~~e lo scritto di laboratorio~~ e deve essere sostenuta nella stessa sessione in cui si è superato lo scritto
- I moduli di teoria e laboratorio possono essere svolti disgiuntamente, ma il voto viene mantenuto solo all'interno dello stesso anno accademico, ovvero fino alla sessione di gennaio-febbraio 2026
- Se si viene respinti all'esame orale di teoria, bisogna rifare il solo scritto di teoria
- Il voto finale sarà dato dalla media aritmetica arrotondata per eccesso dei voti conseguiti nei due moduli

Modalità d'esame: la prova orale di teoria

- La prova orale di teoria, oltre alla discussione degli esiti dello scritto, consta di due domande:
  - una prima domanda su un argomento a scelta del candidato;
  - una seconda domanda a scelta del docente
- Alcuni argomenti sono etichettati come fondamentali: la loro conoscenza all'orale sarà condizione necessaria per superare l'esame con profitto (anche in caso di punteggi massimi ottenuti negli scritti!)

Modalità d’esame: le prove parziali di teoria

- È una modalità riservata agli studenti **iscritti al secondo anno**, o a chi non ha mai sostenuto una prova parziale in passato; può essere svolto anche se non si è ancora superato l’esame di Fondamenti di Programmazione
- Il primo parziale (che conterrà 10 test a risposta multipla) ha un **unico appello** a Novembre e verte sugli argomenti 1-6; chi supera il primo parziale può accedere al secondo parziale
- Il secondo parziale (che conterrà 10 test a risposta multipla) ha un **unico appello** nella prima settimana della sessione di Gennaio-Febbraio, e verte sugli argomenti 8-13; chi supera anche il secondo parziale e ha acquisito la propedeuticità di Fondamenti di Programmazione può accedere al cosiddetto orale semplificato, da svolgere comunque entro Febbraio, e che consiste in una sola domanda a scelta del docente sulla seconda parte del programma (argomenti 8-13)

Algoritmi e Strutture Dati

Capitolo 1
Un'introduzione informale agli algoritmi

Etimologia

Il termine *Algoritmo* deriva da *Algorismus*, traslitterazione latina del nome di un matematico persiano del IX secolo, Muhammad al-Khwarizmi, che ne descrisse il concetto applicato alle procedure per eseguire alcuni calcoli matematici

Definizione (necessariamente informale) di algoritmo

Sequenza finita di *passi semplici* **(azioni)**, scelti tra un insieme (solitamente) finito di possibili azioni, che consente di **risolvere** un *problema* (ovvero di *ottenere una risposta* **(output)** ad un determinato *quesito* **(input)**).

Le quattro proprietà fondamentali di un algoritmo

- La sequenza di istruzioni deve essere **finita**
- Le istruzioni devono essere **eseguibili**
- Le istruzioni non devono essere **ambigue**
- Dopo un tempo finito, l'esecuzione delle istruzioni deve portare ad un **risultato univoco (corretto)**

Algoritmi e strutture dati

- In problemi di **natura computazionale**, un algoritmo è una procedura che prende dei dati in **input** e, dopo averli elaborati, restituisce dei dati in **output**

⇒ I dati devono essere organizzati e strutturati in modo tale che la procedura che li elabora sia "efficiente"

Tipo di dato *vs* struttura dati

Tipo di dato:

- Specifica la natura e l'insieme di valori che una variabile singola o composita può assumere (ad esempio, intero, carattere, insieme di record, etc.), e le **operazioni di interesse** su di essa (ad esempio: somma di due interi, ricerca di un elemento in un insieme, etc.)

Struttura dati:

- Organizzazione logica e relativa implementazione fisica di un certo tipo di dati che permette di supportarne le relative operazioni usando meno risorse di calcolo possibile (ad esempio, lista, array, etc.)

Algoritmi e programmi

- Algoritmo ≠ Programma: Un programma è la **codifica** (in un linguaggio di programmazione) di un certo algoritmo

⇒ Un algoritmo è l'essenza computazionale di un programma, ovvero rappresenta una procedura risolutiva depurata da dettagli riguardanti il linguaggio di programmazione, ambiente di sviluppo, sistema operativo

Problema: ricerca del massimo fra *n* numeri interi

- Input: una sequenza di *n* numeri $A=<a_1,a_2,\ldots,a_n>$
- Output: un numero a_i tale che $a_i \geq a_j \ \forall j=1,\ldots,n$

Algoritmo (ad altissimo livello): Inizializza il valore del massimo al valore del primo elemento. Poi, guarda uno dopo l'altro tutti gli elementi, e ad ogni passo confronta l'elemento in esame con il massimo corrente, e se maggiore, aggiorna il massimo corrente

Alcune codifiche classiche (con array)

```
int InC(int a[], int n){
  int i, max;
  max = a[0];
  for (i = 1; i < n; i++)
     if (a[i] > max) {
       max = a[i];
     }
  return max;
}
```

```
public static int InJava (int[] a){
int max=a[0];
for (int i = 1; i < a.length; i++)
   if (a[i] > max) max = a[i];
return max;
```

```
function InPascal(var A: array[1…Nmax] of integer): integer;
var k, max: integer;
begin
   max:=A[1];
   for k:= 2 to n do
   begin
      if A[k]>max then max:=A[k];
   end;
   InPascal:=max;
end;
```

Il nostro pseudo-codice

```
algoritmo Massimo (array A) → elemento
 max= A[1]
 for j=2 to n do
     if (A[j] ≥ max) then max=A[j]
 return max
```

Ma serve veramente il ≥ ?

Correttezza ed efficienza

Vogliamo progettare algoritmi che:

- Producano correttamente il risultato desiderato
- Siano efficienti in termini di tempo di esecuzione ed occupazione di memoria

Analisi di algoritmi

Correttezza:

- dimostrare formalmente che un algoritmo è corretto

Complessità (efficienza):

- Stimare la quantità di risorse (tempo di esecuzione e occupazione di memoria) necessarie all'algoritmo
- stimare il più grande input gestibile in tempi ragionevoli
- confrontare due algoritmi diversi
- ottimizzare le parti "critiche"

Un esempio giocattolo: i numeri di Fibonacci

L’isola dei conigli

Leonardo da Pisa (anche noto come Fibonacci) [1170-1240] si interessò di molte cose, tra cui il seguente problema di dinamica delle popolazioni:

> Quanto velocemente si espanderebbe una popolazione di conigli sotto appropriate condizioni?

In particolare, partendo da una coppia di conigli neonati in un’isola deserta, e data una certa regola di riproduzione, quante coppie si avrebbero nell’anno *n*?

Le regole di riproduzione

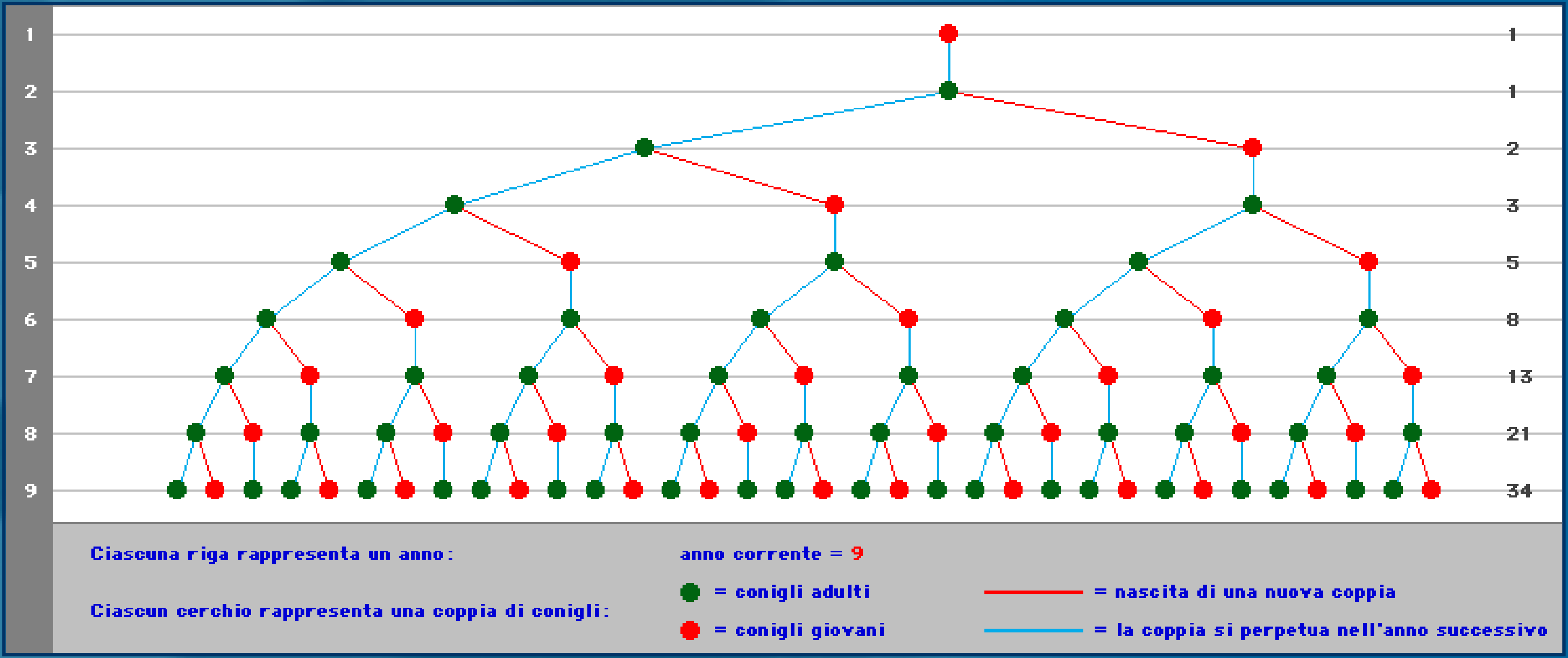
- Una coppia di conigli concepisce due coniglietti di sesso diverso ogni anno, i quali formeranno una nuova coppia
- La gestazione dura un anno (quindi un coniglietto concepito all'inizio dell'anno n nascerà all'inizio dell'anno n+1)
- I conigli cominciano a riprodursi soltanto al secondo anno dopo la loro nascita (quindi un coniglietto nato all'inizio dell'anno n diventa riproduttivo all'inizio dell'anno n+1, e genera prole negli anni n+2, n+3, n+4, etc.)
- I conigli sono immortali (!)

L'albero dei conigli

La riproduzione dei conigli può essere descritta usando un albero (genealogico) come segue:

Anno

coppie conigli



La regola di espansione

- All'inizio degli anni $n=1,2$ c'è una sola coppia di conigli
- All'inizio dell'anno $n \geq 3$ ci sono tutte le coppie dell'anno precedente, e una nuova coppia di conigli per ogni coppia presente due anni prima (ovvero tutte le coppie che sono in grado di riprodursi nell'anno $n-1$)
- Indicando con F_n il numero di coppie all'inizio dell'anno n, abbiamo la seguente relazione di ricorrenza:

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

Il problema

Primi numeri della sequenza di Fibonacci:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, F_{16}= 987, F_{17}= 1597, F_{18}= 2584,…

Problema computazionale:
Dato in input n, calcolare F_n

Digressione: la sezione aurea ϕ

Rapporto $\phi=a/b$ fra due grandezze disuguali $a>b$, in cui a è medio proporzionale tra $a+b$ e b

$(a+b) : a = a : b$

a

b

$$\frac{a+b}{a} = \frac{a}{b} = \phi$$

e ponendo $a=b\phi$

$$\frac{b\phi+b}{b\phi} = \frac{b\phi}{b} \quad\Rightarrow\quad \frac{\cancel{b}(\phi+1)}{\cancel{b}\phi} = \frac{\cancel{b}\phi}{\cancel{b}} \quad\Rightarrow\quad \phi+1=\phi^2$$

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1{,}618\,033 \qquad \hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -0{,}618\,033$$

Un approccio numerico

Keplero [1571-1630] osservò che

$$\lim_{n\to\infty} \frac{F_{n+1}}{F_n} = \phi$$

da cui si può dimostrare che la soluzione in forma chiusa della sequenza di Fibonacci, nota come formula di Binet [1786-1856], è:

$$F_n = \frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right)$$

ovvero $\lim_{n\to\infty} \frac{1/\sqrt{5}\left(\phi^n - \hat{\phi}^n\right)}{\phi^n} = 1/\sqrt{5}$ e cioè F_n cresce come Φ^n

Algoritmo `fibonacci1`

$$\textbf{algoritmo}\ \texttt{fibonacci1}(intero\ n) \rightarrow intero$$
$$\quad \textbf{return}\ \frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right)$$

È il nostro primissimo algoritmo: come detto, ne dobbiamo valutare la correttezza e l'efficienza: per quest'ultima, in questa fase iniziale ci limiteremo a contare il numero di linee di codice mandate in esecuzione

Correttezza ed efficienza

- Molto efficiente: apparentemente sì, **una sola linea di codice** mandata in esecuzione (sebbene stiamo trascurando la complessità dell'operazione in essa contenuta)!
- Ma siamo sicuri che sia corretto? Se adottassimo un modello di calcolo astratto avente celle di memoria di **dimensione arbitrariamente grande** sarebbe corretto, ma su un modello di calcolo reale, si può far vedere che fissata la dimensione della cella di memoria, e quindi l'accuratezza con cui rappresento √5, Φ e Φ, esiste sempre un n a partire dal quale il risultato restituito è SBAGLIATO a causa dell'arrotondamento. Ad esempio, se uso 3 cifre decimali:

$$\sqrt{5} \approx 2.236 \quad \phi \approx 1.618 \text{ e } \hat{\phi} \approx -0.618$$

n	fibonacci1(n)	arrotondamento	F_n
3	1.99992	2	2
16	986.698	987	987
17	1595.666	1596	1597

Algoritmo `fibonacci2`

Poiché `fibonacci1` non è corretto, un approccio alternativo consiste nell'utilizzare direttamente la definizione ricorsiva:

```
algoritmo fibonacci2(intero n) → intero
    if (n≤2) then return 1
    else return fibonacci2(n-1) +
                fibonacci2(n-2)
```

Correttezza? Corretto per definizione!

Efficienza?

Per valutare il tempo di esecuzione, ancora una volta calcoliamo il numero di linee di codice T(n) mandate in esecuzione

- Se n≤2: una sola linea di codice
- Se n=3: quattro linee di codice, due per la chiamata `fibonacci2(3)`, una per la chiamata `fibonacci2(2)` e una per la chiamata `fibonacci2(1)`, cioè

  $$T(3)=2+T(2)+T(1)=2+1+1=4$$

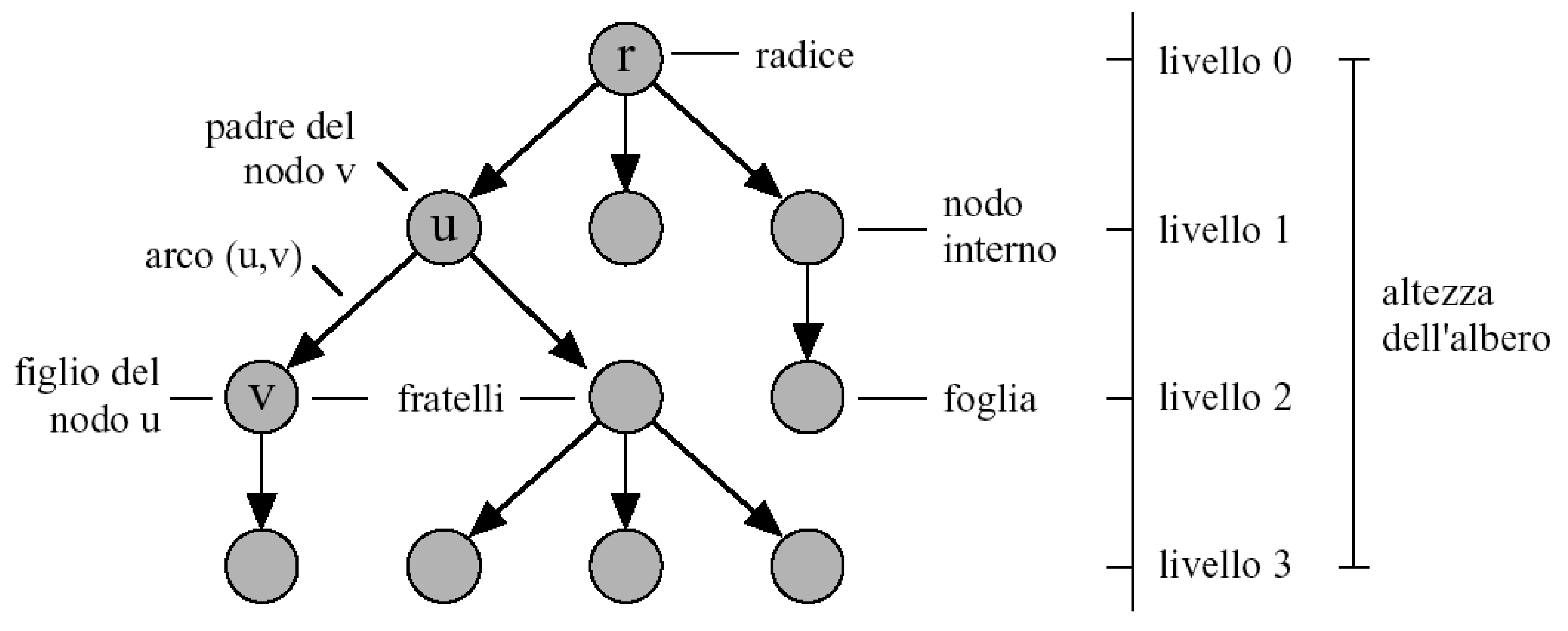
Relazione di ricorrenza

In generale, per $n \geq 3$, in ogni chiamata si eseguono due linee di codice, oltre a quelle eseguite nelle chiamate ricorsive

$$T(n) = 2 + T(n-1) + T(n-2) \qquad n \geq 3$$

Il tempo di esecuzione di un algoritmo ricorsivo è quindi pari al tempo speso all'interno della chiamata corrente più il tempo speso nelle chiamate ricorsive.
Vediamo in particolare come calcolare tale valore per `fibonacci2(n)` usando l'**albero della ricorsione**.

Alberi radicati: qualche definizione



albero d-ario: albero in cui tutti i nodi interni hanno (al più) d figli

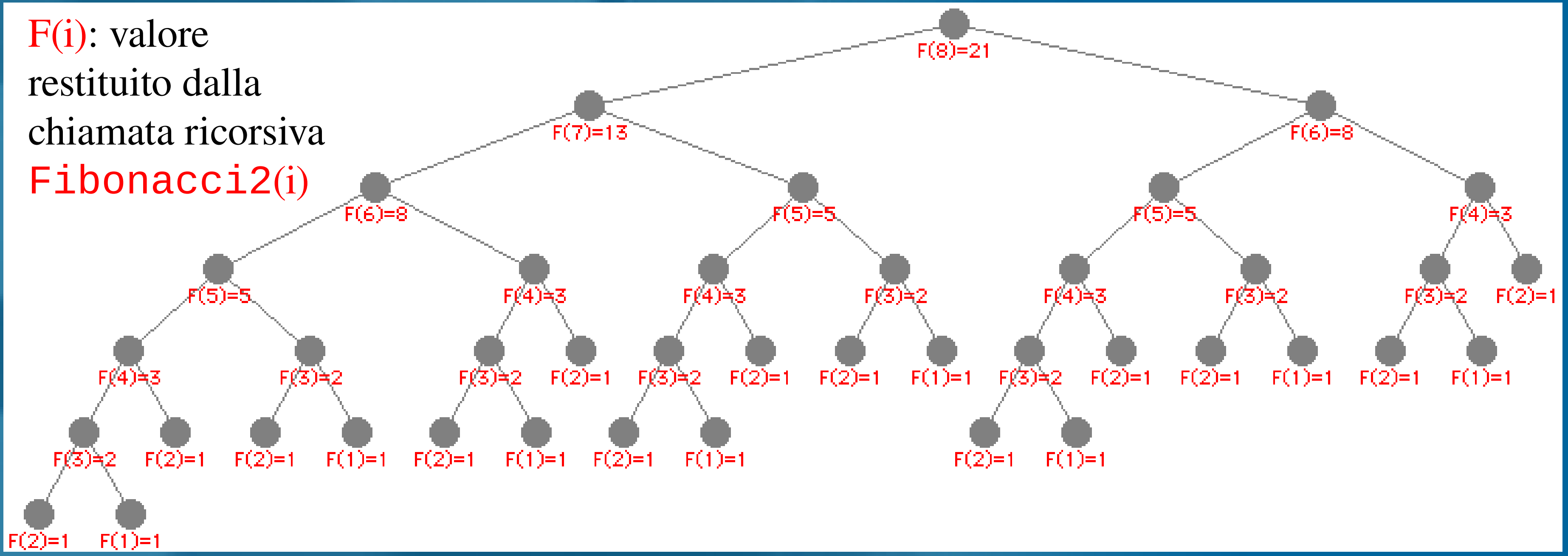
d=2 → albero binario

Un albero è strettamente binario se tutti nodi interni hanno esattamente 2 figli

Albero della ricorsione di `fibonacci2`

- Utile per risolvere la relazione di ricorrenza T(n)
- Ogni nodo corrisponde ad una chiamata ricorsiva
- I figli di un nodo corrispondono alle sottochiamate

F(i): valore restituito dalla chiamata ricorsiva `Fibonacci2`(i)



Calcolare T(n)

- Etichettando i nodi dell'albero con il numero di linee di codice eseguite nella chiamata corrispondente:
  - I nodi interni hanno etichetta 2
  - Le foglie hanno etichetta 1
- Per calcolare T(n):
  - Contiamo il numero di foglie
  - Contiamo il numero di nodi interni

Contare il numero di foglie

Lemma 1: Il numero di foglie dell'albero della ricorsione di `fibonacci2(n)` è pari a F_n

Dim: Per induzione su n:

- Caso base n=1 (e anche n=2): in questo caso l'albero della ricorsione è costituito da un unico nodo, che è quindi anche una foglia; poiché $F_1=1$, il lemma segue.
- Caso n>2: supposto vero fino ad n-1, dimostriamolo vero per n; osserviamo che l'albero della ricorsione associato ad n è formato da una radice etichettata F(n) e da due sottoalberi etichettati F(n-1) e F(n-2). Per l'ipotesi induttiva, tali sottoalberi hanno rispettivamente F_{n-1} ed F_{n-2} foglie, e quindi l'albero della ricorsione associato ad n avrà $F_{n-1} + F_{n-2} = F_n$ foglie, come volevasi dimostrare. □

Contare il numero di nodi interni

Lemma 2:
Il numero di nodi interni di un albero **strettamente binario** (come l'albero della ricorsione di `fibonacci2(n)`) è pari al numero di foglie -1.
Dim.:
(da fare a casa, per induzione sul numero di nodi interni dell'albero)

Abbiamo quindi F_n foglie (Lemma 1) e F_n-1 nodi interni (Lemma 2), per un totale di linee di codice eseguite pari a:

$$T(n) = F_n + 2\,(F_n-1) = 3F_n-2$$

Osservazioni

`fibonacci2` è un algoritmo lento, perché esegue un numero di linee di codice esponenziale in n:

$$T(n) = 3 \cdot F_n - 2 \approx 3 \cdot F_n \approx (3/\sqrt{5}) \cdot \Phi^n \approx 1{,}618^n$$

Alcuni esempi di numero di linee di codice eseguite

$n = 8$ $T(n)=3 \cdot F_8 - 2 = 3 \cdot 21 - 2 = 61$

$n = 45$ $T(n)=3 \cdot F_{45} - 2 = 3 \cdot 1.134.903.170 - 2 = 3.404.709.508$

$n = 100$, $T(n) \approx 10^{21}$. Con le attuali tecnologie, calcolare F_{100} richiederebbe circa 8000 anni!

Possiamo fare di meglio?