

Università degli Studi dell'Aquila

Anno Accademico 2015/2016

- **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: 11.15 – 13.00 (Aula A1.6)
Giovedì: 14.15 – 16.00 (Aula A1.6)
- **Ricevimento:** Martedì 16.00-18.00 o 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

1. **Introduzione:** problemi, algoritmi, complessità computazionale.
2. **Notazione asintotica,** problema della ricerca.
3. **Ordinamento:** Insertion sort, Selection sort.
4. **Ordinamento efficiente:** Lower bound (*), Merge sort (*), Quicksort (*), algoritmi di ordinamento lineari.
5. **Code di priorità:** heap binario, Heapsort, heap binomiale (*).
6. **Problema del dizionario.**
7. **Alberi di ricerca:** alberi AVL.
8. **Alberi di ricerca:** rotazioni AVL (*).
9. **Prova intermedia (settimana 23-27 novembre 2015)**
10. **Tabelle hash; tecniche algoritmiche.**
11. **Grafi:** visite.
12. **Cammini minimi:** Ordinamento topologico, Bellman&Ford.
13. **Cammini minimi:** Dijkstra (*), Floyd&Warshall.
14. **Insiemi disgiunti**
15. **Minimo albero ricoprente:** Kruskal (*), Prim, Boruvka.

(*): argomenti fondamentali

2016

Libro di testo

C. Demetrescu, I. Finocchi, G. Italiano

Algoritmi e Strutture dati

McGraw-Hill, prezzo di copertina Euro 36

Slide e materiale didattico

<http://www.di.univaq.it/~proietti/didattica.html>

Altri testi utili

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein
Introduzione agli algoritmi e strutture dati
McGraw-Hill, costo Euro 62.

P. Crescenzi, G. Gambosi, R. Grossi
Strutture di dati e algoritmi. Progettazione, analisi e visualizzazione
Pearson, costo Euro 27.

Modalità d'esame: appelli

- Sei appelli (+1 a novembre per i fuori corso)
 - 2 appelli a febbraio
 - 3 appelli ad aprile/giugno/luglio
 - 1 appello a settembre
- L'esame di ASDL (12 CFU) consiste in:
 - una prova scritta e una prova orale di **teoria**, da svolgersi **obbligatoriamente**
 - una prova scritta di **laboratorio**, 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 **sia lo scritto di teoria che lo scritto di laboratorio**
- Gli scritti di teoria e laboratorio possono essere svolti **disgiuntamente**, ma la loro validità è mantenuta solo **all'interno dello stesso anno solare**
- Se si viene **respinti** all'esame orale di teoria, bisogna rifare il **solo scritto** di teoria

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
- Durante il corso, alcuni argomenti verranno 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 ha un unico appello a Novembre** e verte sugli argomenti 1-7; chi supera il primo parziale può accedere al secondo parziale
- Il primo parziale conterrà **10 test a risposta multipla**, (e forse una **domanda aperta** su un argomento di teoria fatto a lezione)
- Il **secondo parziale** (che conterrà solo **10 test a risposta multipla**) **ha due appelli nella sessione di Gennaio-Febbraio**, e verte sugli argomenti 8-15; chi supera anche il secondo parziale e ha superato lo scritto di laboratorio (e 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 8-15)

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

Procedimento **effettivo** che consente di risolvere un *problema* (ovvero di **ottenere una risposta** ad un determinato *quesito*) eseguendo, in un determinato ordine, un insieme finito di *passi semplici* (**azioni**), scelti tra un insieme (solitamente) finito di possibili azioni.

Le quattro proprietà fondamentali di un algoritmo

- La sequenza di istruzioni deve essere **finita**
- Essa deve portare ad un **risultato corretto**
- Le istruzioni devono essere **eseguibili materialmente**
- Le istruzioni non devono essere **ambigue**

Algoritmi e strutture dati

- Il concetto di algoritmo è inscindibile da quello di **dato**
 - Da un punto di vista 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 composta 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 dei dati che permette di supportare le operazioni di un tipo di dato usando meno risorse di calcolo possibile (ad esempio, lista, array, etc.)

Algoritmi e programmi

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

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

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
```

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:

- Stimare la quantità di risorse (**tempo** e **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?

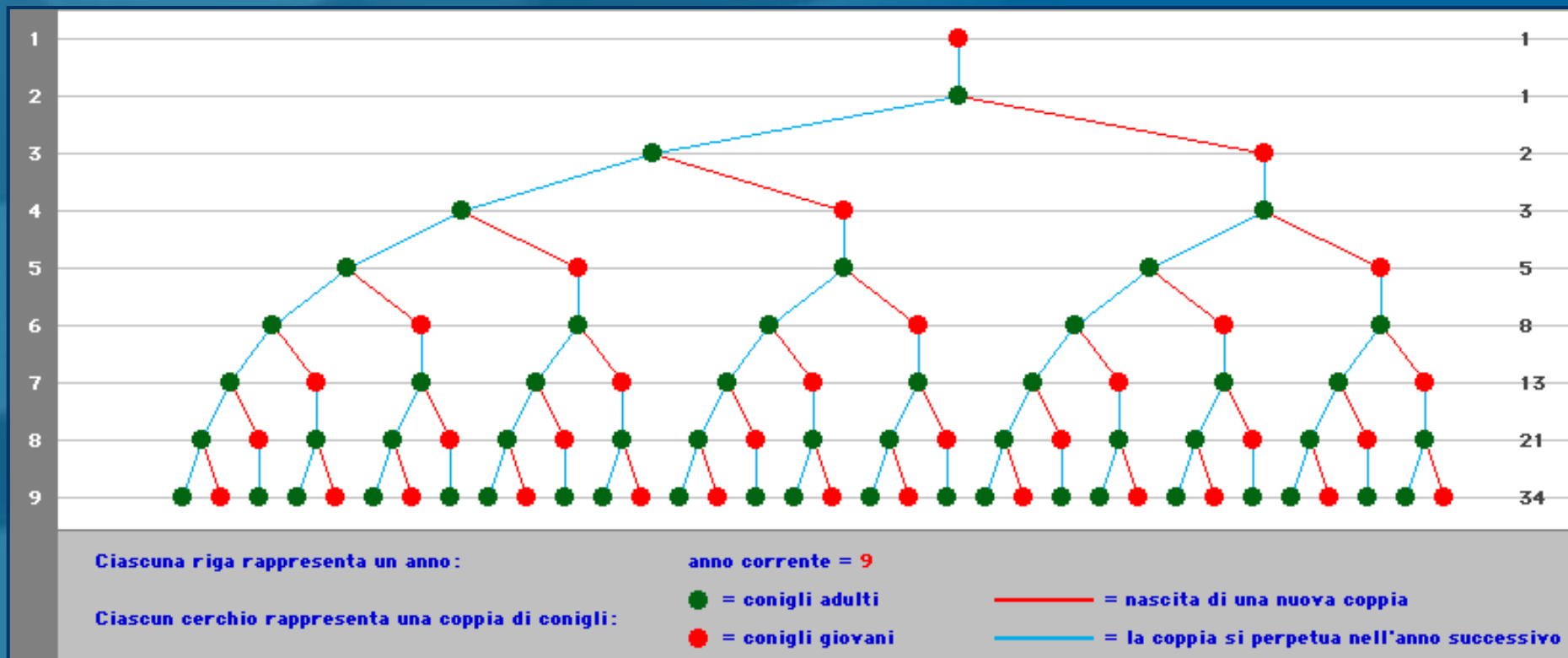
In particolare, partendo da una coppia di conigli in un'isola deserta, e data una certa regola di riproduzione, quante coppie si avrebbero nell'anno n ?

Le regole di riproduzione

- Una coppia di conigli concepisce **due coniglietti** di **sexso diverso** ogni anno, i quali formeranno una **nuova coppia**
- La gestazione dura **un anno**
- I conigli cominciano a riprodursi soltanto al **secondo anno** dopo la loro nascita
- I conigli sono **immortali (!)**

L'albero dei conigli

La riproduzione dei conigli può essere descritta in un **albero** come segue:



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
- 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, 987, 1597, $F_{18}=2584, \dots$

Come calcoliamo F_n ?

Digressione: la sezione aurea

Rapporto ϕ fra due grandezze disuguali $a > b$, in cui a è medio proporzionale tra b e $a+b$

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



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

e ponendo $a = b\phi$

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

$$\phi = \frac{1 + \sqrt{5}}{2} \approx 1,618033$$

$$\hat{\phi} = \frac{1 - \sqrt{5}}{2} \approx -0,618033$$

Un approccio numerico

Keplero [1571-1630] osservò che

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

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

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

Algoritmo fibonacci1

```
algoritmo fibonacci1(intero n) → intero  
return  $\frac{1}{\sqrt{5}} \left( \phi^n - \hat{\phi}^n \right)$ 
```

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**? Sì, se adottassi un modello di calcolo astratto avente celle di memoria infinite, ma su un modello di calcolo reale, con quale **accuratezza** devo fissare Φ e $\hat{\Phi}$ per ottenere un risultato corretto?
- Ad esempio, con 3 cifre decimali:

$$\phi \approx 1.618 \text{ e } \hat{\phi} \approx -0.618$$

n	fibonacci1(n)	arrotondamento	F_n
3	1.99992	2	2
16	986.698	987	987
18	2583.1	2583	2584

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**, calcoliamo il **numero di linee di codice** $T(n)$ mandate in esecuzione
- Se $n \leq 2$: 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

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

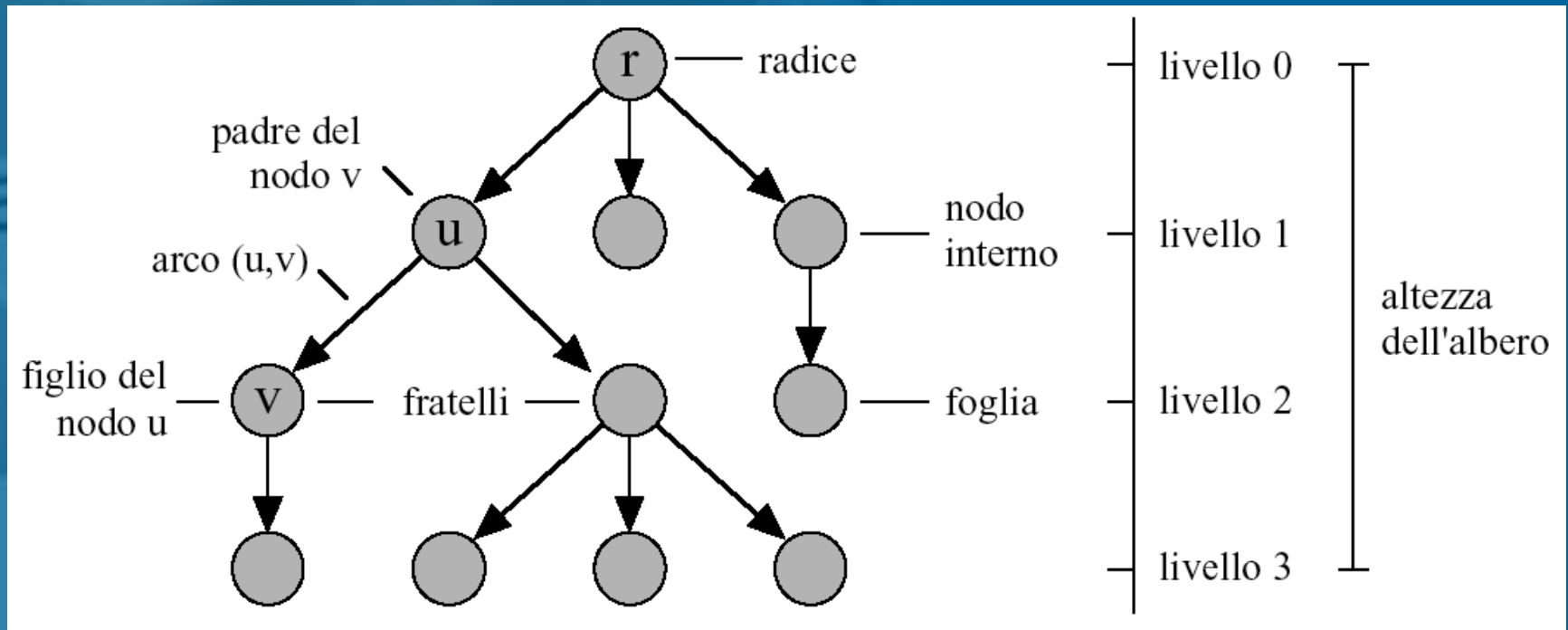
Relazione di ricorrenza

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

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

In generale, il tempo di esecuzione di un algoritmo ricorsivo è pari al tempo speso all'interno della chiamata più il tempo speso nelle chiamate ricorsive

Alberi radicati: qualche definizione



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

$d=2 \rightarrow$ **albero binario**

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

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

Calcolare $T(n)$

Lemma 1:

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

dim

(da fare a casa, per induzione su n)

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)



In totale le linee di codice eseguite sono

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

Osservazioni

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

$$T(n) = 3F_n - 2 \approx F_n \approx \Phi^n$$

Alcuni esempi di **linee di codice eseguite**

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

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

$n = 100 \dots$ con le attuali tecnologie, calcolare F_{100} richiederebbe circa 8000 anni!

Possiamo fare di meglio?