

# Olimpiadi **P**roblem **S**olving

Antonella Carbonaro

[antonella.carbonaro@unibo.it](mailto:antonella.carbonaro@unibo.it)

Dipartimento di Informatica - Scienza e Ingegneria  
Alma Mater Studiorum, Università di Bologna

Cervia – 27 aprile 2017



# Olimpiadi di Problem Solving

I vostri studenti non hanno difficoltà nell'eseguire esercizi di routine ma falliscono nel risolvere problemi che non hanno mai incontrato prima?

Il mercato del lavoro odierno richiede lavoratori in grado di risolvere problemi non routinari (a quelli fanno i fronte i computer!!)

World Economic Forum 2016



# Olimpiadi di Problem Solving

Le competenze di problem-solving sono una componente essenziale delle capacità richieste per affrontare analiticamente compiti interpersonali e non routinari.

In entrambi i casi le persone devono pensare a **come** affrontare la situazione, **monitorare** sistematicamente l'effetto delle loro azioni, e **modificarle** opportunamente.

**Pensiero computazionale:** pensiero critico nel valutare le situazioni e una capacità avanzata di problem-solving

Il pensiero computazionale è una abilità che andrebbe sviluppata sin da bambini, **a scuola**, perché aiuta a pensare meglio, in modo originale e mai ripetitivo.



# Olimpiadi di Problem Solving

Pensiero computazionale nella scuola dell'obbligo:

Grande valenza **didattica**: le prove proposte sono radicate nelle aree disciplinari di base (italiano, matematica ed inglese), ed intendono stimolare percorsi di ricerca in cui entrano in gioco le competenze proprie del **problem solving**: il pensare, il ragionare, il fare ipotesi ed operare scelte, per pervenire alla risoluzione dei problemi attraverso la logica.

Le attività proposte stimolano il **pensiero critico**, la **collaborazione**, la **comunicazione** e la **creatività**, riconosciute quali competenze del futuro per sostenere la crescita europea, l'occupazione, l'equità e l'inclusione sociale.

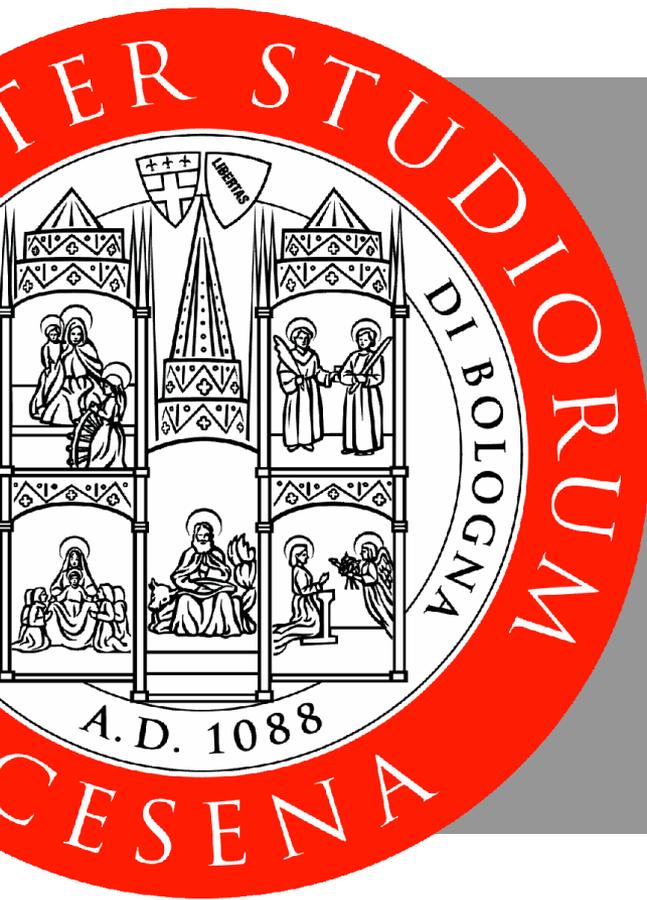


# Olimpiadi di Problem Solving

## Informatica e pensiero computazionale nella scuola dell'obbligo

Le Olimpiadi di Problem Solving offrono una occasione esplicita e operativa per fare **esperienze** di pensiero computazionale. Sono un mezzo per imparare linguaggi utili non solo all'informatico professionista ma a chiunque debba cimentarsi nel trovare la soluzione a problemi.

- **PERCHE'**
- **COME**
- **COSA**



# Olimpiadi **P**roblem **S**olving **PERCHE'**

Antonella Carbonaro  
[antonella.carbonaro@unibo.it](mailto:antonella.carbonaro@unibo.it)

Dipartimento di Informatica - Scienza e Ingegneria  
Alma Mater Studiorum, Università di Bologna

Cervia – 27 aprile 2017



# Problem Solving

- Ubiquitous computing
- Dimensioni ridottissime, con enorme potenza di calcolo
- Enorme disponibilità di informazioni
- ... e le conoscenze e le abilità connesse all'elaborazione di tale informazione??
- **Pensiero computazionale: OLTRE** l'uso della tecnologia, è indipendente da essa (anche se la può sfruttare intensamente):  
Processo di Problem Solving che consiste nel:
  - formulare problemi in modo tale che un esecutore (umano, artificiale) li possa risolvere
  - organizzare logicamente, rappresentare e analizzare dati
  - generalizzare il processo per trasferirlo ad altri problemi

**Computer Science is no more about computers than astronomy is about telescopes.  
E. W. Dijkstra**



# Problem Solving

Competenze generali di problem solving sono un obiettivo educativo riconosciuto e vengono acquisite nella scuola dell'obbligo in diverse discipline:

- Lingua madre,
- Lingue straniere,
- Matematica,
- Filosofia,
- Scienze,
- ...

«Le acquisizioni più valide nell'educazione scientifica e tecnologica sono quegli strumenti mentali di tipo **generale** che rimangono utili per tutta la vita. Ritengo che il linguaggio naturale e la matematica siano i due strumenti più importanti in questo senso, e l'informatica sia il terzo». (George Forsythe)



# Problem Solving

**L'informatica** offre un contributo linguistico al problem solving, ovvero lo strumento dei linguaggi formali per descrivere in modo preciso e chiaro **l'approccio risolutivo** rispetto a quanto si potrebbe fare usando il linguaggio naturale.

Usiamo l'espressione **Pensiero Computazionale** per evidenziare che, che quando si parla della necessità di “insegnare informatica nella scuola”, **l'obiettivo** non è insegnare l'uso di un certo strumento o applicazione o di una determinata tecnologia e sistema, quanto **l'apprendimento dei concetti scientifici di base**.

Il Pensiero Computazionale non compare con l'informatica, ha piuttosto una lunga storia!



# La storia dei dischi volanti

La storia dei dischi volanti inizia il 24 giugno del 1947.

Durante il pranzo, il discorso ritornò sulla recente ondata di dischi volanti ed **Enrico Fermi** all'improvviso chiese: "Dove sono tutti quanti?" La domanda del fisico italiano non era per nulla ingenua: egli infatti aveva calcolato che, poiché l'età dell'Universo è tre volte maggiore di quella del nostro pianeta, se girassero per lo spazio tutti gli extraterrestri di cui si parla essi avrebbero dovuto essere stati visti già da molto tempo ed anche più di una volta. La risposta immediata al dubbio di Fermi venne dal fisico ungherese Leo Szilard: "Sono qui fra noi e si fanno chiamare Ungheresi".

Il più "marziano" di tutti fu però il matematico, fisico e tecnologo John von Neumann.

Americano di origine ungherese, **John von Neumann** è stata uno delle menti più brillanti e straordinarie del secolo appena passato; mente che gli ha permesso di apportare contributi significativi, e talora assolutamente nuovi, praticamente in ogni campo della ricerca, dalla matematica alla meccanica statistica, dalla meccanica quantistica alla cibernetica, dall'economia all'evoluzione biologica, dalla teoria dei giochi all'intelligenza artificiale. E, naturalmente, alla bomba atomica.



# La storia dei dischi volanti

Janos Neumann nasce a Budapest il 28 dicembre del 1903 da una ricca famiglia di banchieri ebraici. Già a sei anni è una sorta di fenomeno da baraccone, e intrattiene gli ospiti di famiglia con la sua prodigiosa memoria, imparando a mente pagine dell'elenco telefonico o eseguendo divisioni con numeri di otto cifre. Inoltre si diverte con il padre conversando in greco antico, arrivando a padroneggiare, intorno ai dieci anni, quattro lingue.

Sarebbe stato interessante vederlo competere nel calcolo mnemonico con **Majorana** il quale però a quel tempo era già sparito nel nulla.

Viene nominato miglior studente di matematica dell'Ungheria.

A ventidue anni, infatti, si laurea in ingegneria chimica presso Zurigo e in matematica a Budapest, dopo aver seguito a Berlino anche i corsi Albert **Einstein**.

Si trasferisce poi a Göttingen, dove si occupa dei fondamenti della matematica e della meccanica quantistica che studia sotto la supervisione di **Hilbert**. In questo ambiente von Neumann entra nel pieno della maturità scientifica e i lavori che qui produrrà lo eleveranno a uno dei massimi matematici di ogni tempo.



# La storia dei dischi volanti

Tra il 1930 e il 1933 viene invitato a **Princeton**, dove mette in luce una vena didattica non proprio esemplare. Poco dopo, con l'arrivo dei nazisti al potere, abbandona la sua posizione accademica in Germania, considerando l'avventura americana ben più promettente. Terrà la cattedra di Princeton fino alla fine dei suoi giorni.

Il primo **incontro con un calcolatore** risale a poco tempo dopo, con la macchina Harvard Mark I (ASCC) di Howard Aiken, costruita in collaborazione con l'IBM; poi conosce **ENIAC** (Electronic Numerical Integrator and Computer), un ammasso enorme di valvole, condensatori e interruttori da trenta tonnellate di peso, progettato per far fronte alla enorme complessità dei calcoli balistici richiesti per le tavole di tiro di armamenti sempre più sofisticati (1943).



# La storia dei dischi volanti

Questo mastodonte è utile per eseguire calcoli balistici, meteorologici o sulle reazioni nucleari, ma è fondamentalmente una macchina **molto limitata**, quasi del tutto priva di memoria e di un briciolo di elasticità.

Per migliorare un simile marchingegno c'è bisogno di quell'intuizione che una decina d'anni prima aveva avuto **Turing** nel suo articolo sui numeri computabili, e cioè permettere al computer di **modificare il proprio comportamento**, o, in altre parole, imparare un software. Non appena ne venne a conoscenza, nell'agosto 1944, von Neumann vi si buttò a capofitto: nel giro di quindici giorni dalla sua entrata in scena, il progetto del calcolatore veniva modificato in modo da permettere la **memorizzazione interna del programma**.

Nel 1945 esce, così l'**Edvac** (Electronic Discrete Variables Automatic Computer) è la **prima macchina digitale programmabile tramite un software**: è nata "**l'architettura di von Neumann**".

La programmazione, che fino ad allora richiedeva una manipolazione diretta ed esterna dei collegamenti, era così ridotta ad un'operazione dello stesso tipo dell'inserimento dei dati, e l'ENIAC diveniva la prima realizzazione della **macchina universale** inventata da [Alan Turing](#) nel 1936: in altre parole, un computer programmabile nel senso moderno del termine.



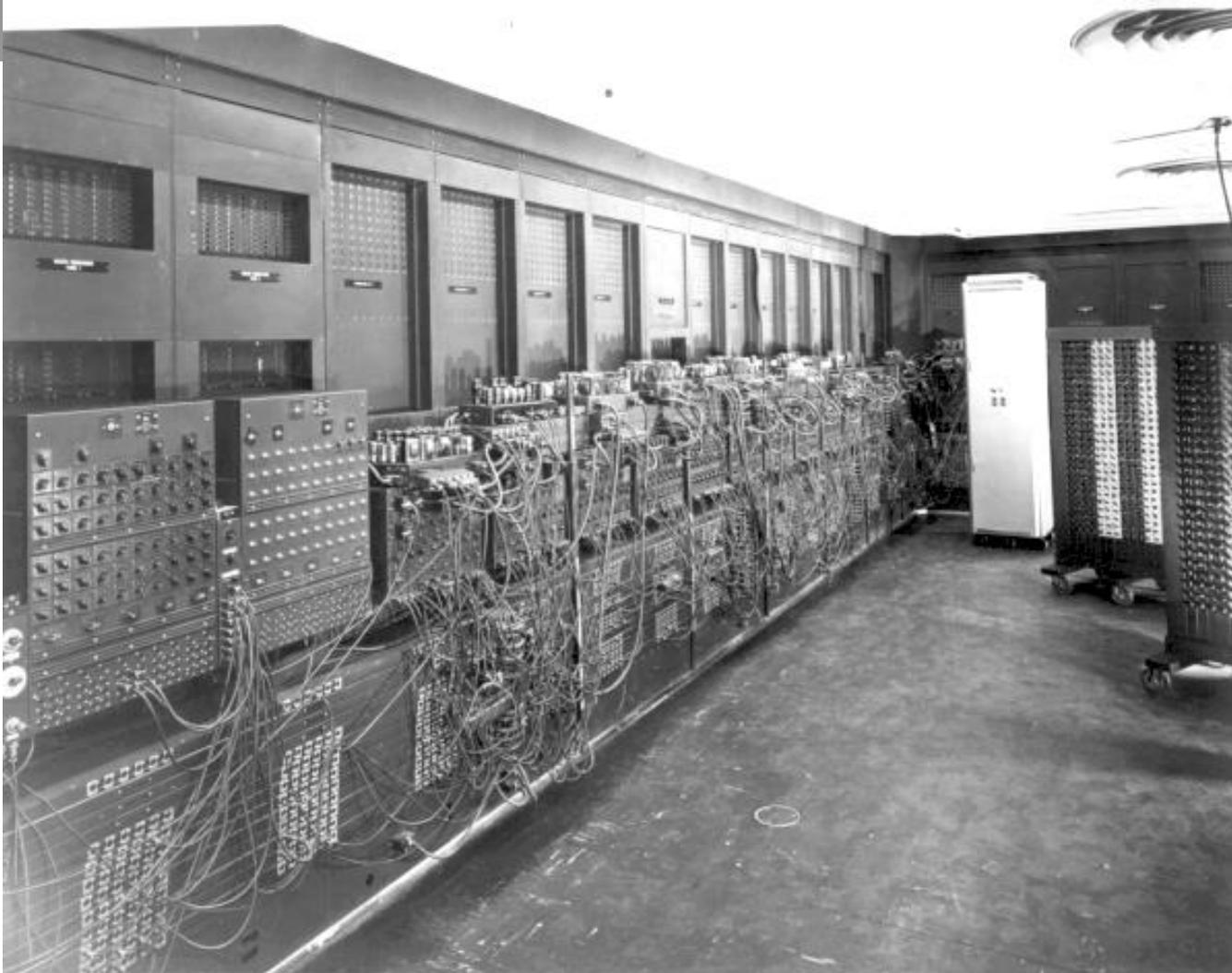
# La storia dei dischi volanti



MARK I – 1940-43



# La storia dei dischi volanti



ENIAC - 1943



# Pensiero Computazionale

Saper leggere la trama algoritmica (“effettiva”) della realtà saper descrivere tale trama in un linguaggio opportuno in modo che tale descrizione sia eseguibile su computer.

Il pensiero computazionale fornisce un mezzo per descrivere l’uno all’altro quello che sappiamo fare.

Fornisce **strumenti concettuali** per descrivere in modo effettivo le informazioni rilevanti per risolvere i problemi (dati e procedimenti).

Sapere, saper fare, **saper far fare**

*«In realtà una persona non ha **davvero** capito qualcosa fino a che non è in grado di insegnarla ad un **computer**» Donald Knuth*



# Strumenti concettuali

Data in input una sequenza di numeri,  
visualizzarla in ordine inverso



# Strumenti concettuali

Verificare che le parentesi tonde, quadre e graffe usate all'interno di una stringa siano correttamente annidate



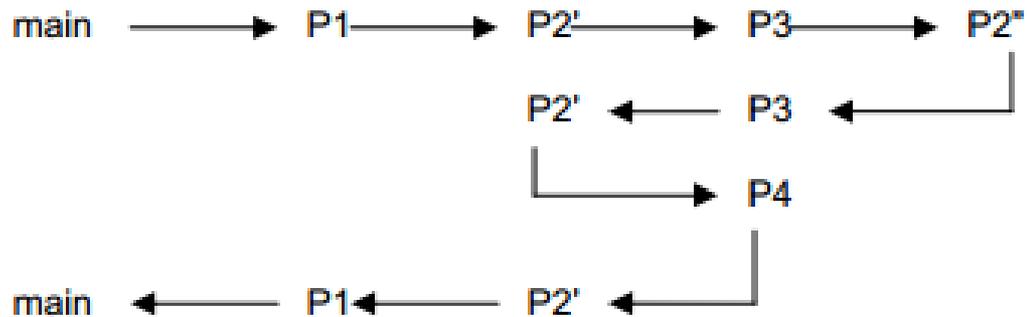
# Strumenti concettuali

Gestire la memoria di un computer memorizzando le informazioni rilevanti relative alla sequenza delle chiamate a funzione

**Problemi diversi?**



# Chiamate a funzione



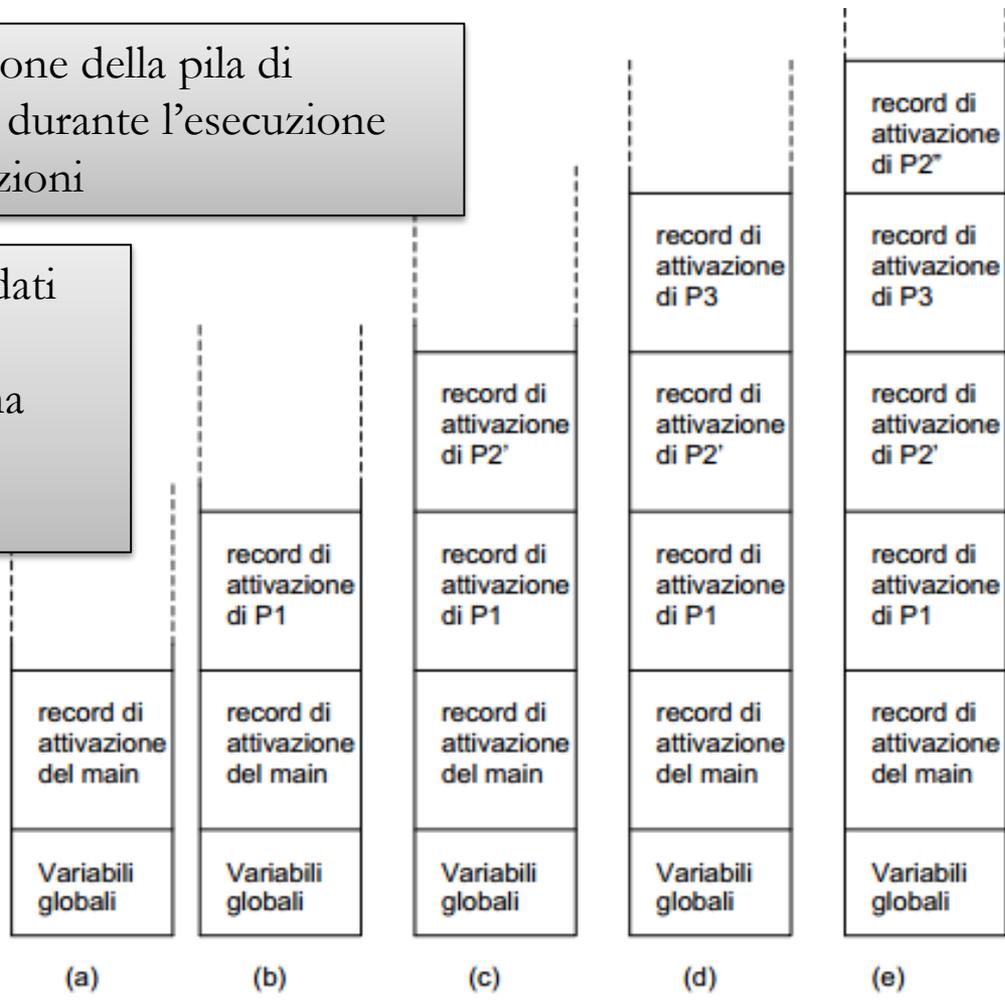
una freccia verso destra indica una chiamata di funzione;  
una verso sinistra indica il ritorno alla funzione chiamante.  
Le diverse attivazioni sono marcate da apici.



# Chiamate a funzione

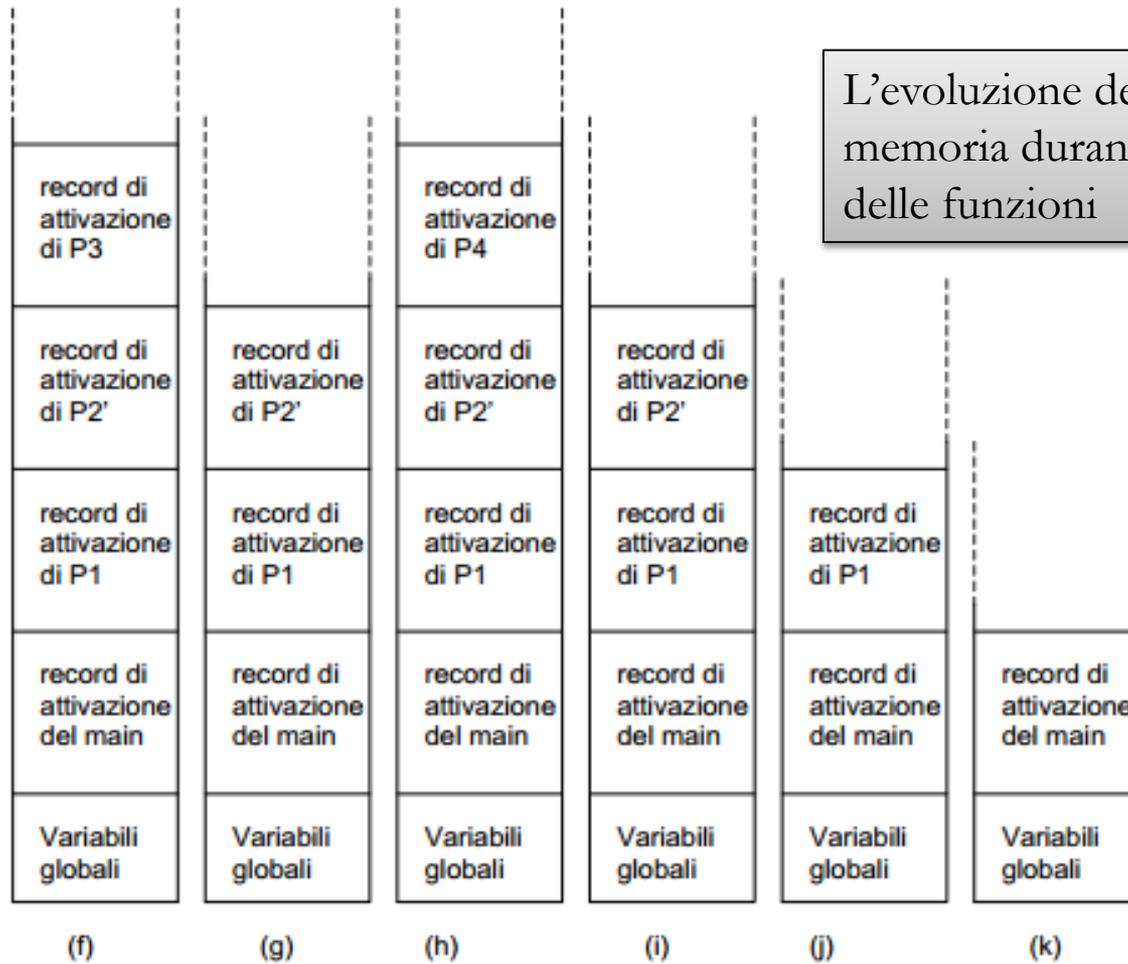
L'evoluzione della pila di memoria durante l'esecuzione delle funzioni

**record di attivazione:** area dati associata ad ogni chiamata di funzione; viene allocata in una zona contigua a quella immediatamente precedente





# Chiamate a funzione



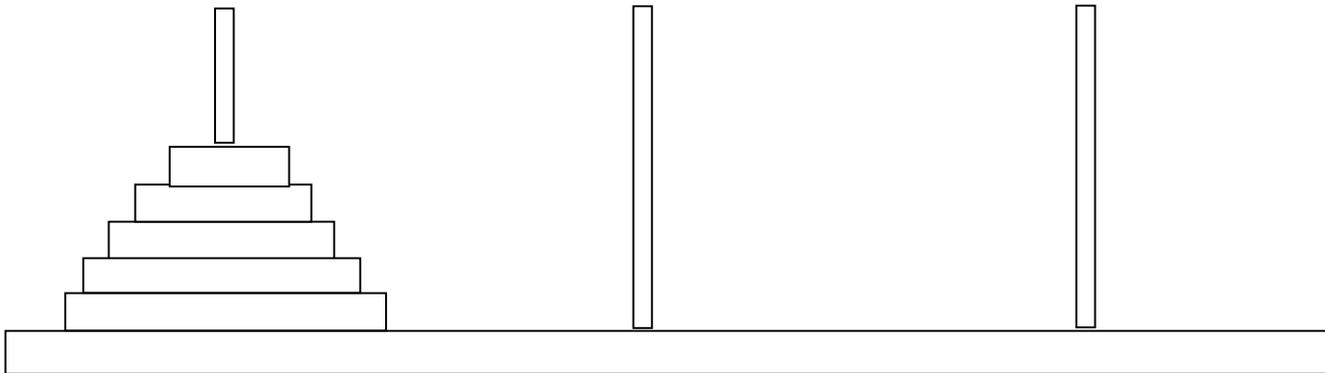
L'evoluzione della pila di memoria durante l'esecuzione delle funzioni



# Strumenti concettuali

Quello delle *Torri di Hanoi* è un gioco che si svolge con tre paletti e alcuni dischi di diametro differente con un foro al centro in modo da poter essere infilati nei paletti.

Inizialmente i dischi sono tutti impilati a piramide sul primo paletto. Il disco più grande è in basso, il più piccolo in alto.





# Le Torri di Hanoi

Narra la leggenda che nel grande tempio di Benares, in India, sotto la cupola che indica il centro del mondo, si trovi una lastra di bronzo sulla quale sono stati fissati **tre pioli** di diamante.

Su uno di questi pioli, al momento della creazione, Dio infilò **sessantaquattro dischi** di oro puro, il più grande a contatto diretto con la lastra di bronzo e poi via via gli altri fino ad arrivare al più piccolo in cima. È la Torre di Brahma.

Notte e giorno, senza sosta, **i monaci trasferiscono i dischi** da un piolo di diamante all'altro in conformità alle **leggi** fisse e immutabili di Brahma, per cui non si deve spostare più di un disco alla volta e bisogna infilarlo nell'altro piolo in modo tale che nessun altro disco più piccolo si trovi al di sotto.

Quando i sessantaquattro dischi saranno trasferiti nella corretta successione dal piolo in cui Dio li ha collocati a uno degli altri, la Torre, il tempio e i brahmani diverranno polvere e, con un gran fragore, **arriverà la fine del mondo**.

Per il momento, comunque, non c'è da preoccuparsi troppo perché, supponendo che il lavoro fosse svolto nel modo più efficiente possibile, sarebbe necessario effettuare un minimo di 18.446.744.073.709.551.615 movimenti che, a una media di una mossa al secondo, sarebbero compiuti in **seimila milioni di secoli**.

Provate a giocare su: <http://www.math.it/torrih/torri.htm>



# La ricorsione

Il problema  $P(n)$  di spostare  $n$  dischi sul paletto di destra può essere descritto in modo ricorsivo così:

- Spostare i primi  $n-1$  dischi dal paletto di sinistra a quello di centro.
- Spostare il disco  $n$ -esimo (il più grande) sul paletto di destra.
- Spostare i rimanenti  $n-1$  dischi dal paletto di centro a quello di destra.

In questo modo il problema può essere risolto per qualsiasi valore di  $n > 0$  ( $n=0$  è la condizione di terminazione della ricorsione).

$P(0)$



# Strumenti concettuali

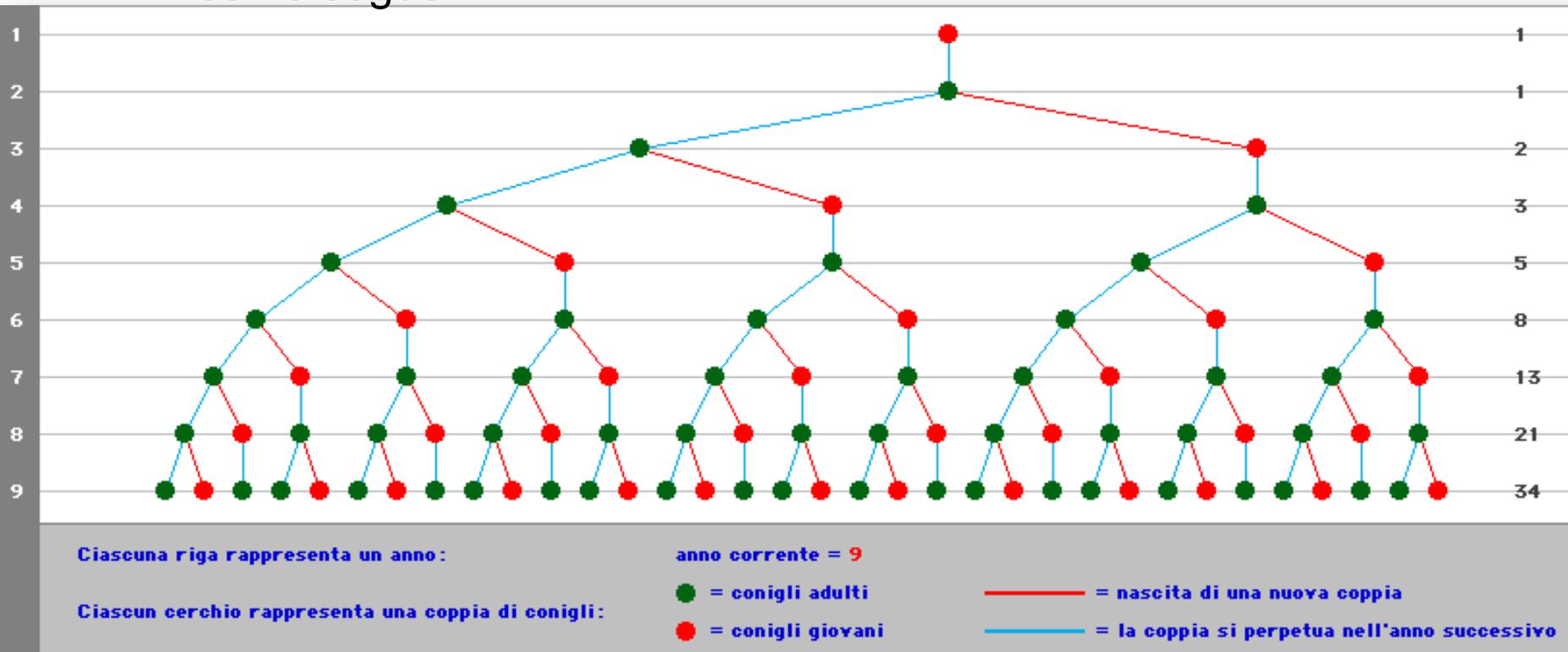
Quanto velocemente si espanderebbe una popolazione di conigli sotto appropriate condizioni?

- Una coppia di conigli genera due coniglietti ogni anno
- I conigli cominciano a riprodursi soltanto al secondo anno dopo la loro nascita
- I conigli sono immortali
- In particolare, partendo da una coppia di conigli in un'isola deserta, quante coppie si avrebbero nell'anno  $n$ ?
- La successione di **Fibonacci**



# Albero dei conigli

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



Nell'anno  $n$ , ci sono tutte le coppie dell'anno precedente, e una nuova coppia di conigli per ogni coppia presente due anni prima.



# Algoritmo fibonacci1

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



# Correttezza?

Qual è l'accuratezza su  $\phi$  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:

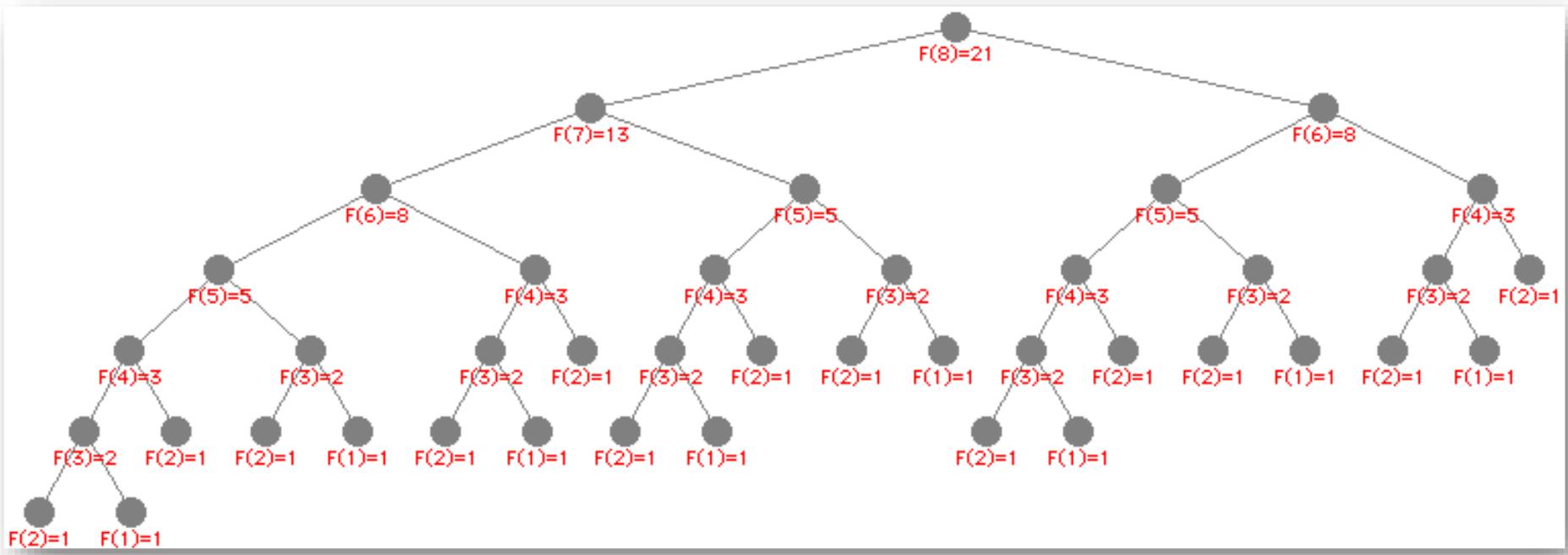
```
algoritmo fibonacci2(intero n) → intero  
if (n ≤ 2) then return 1  
else return fibonacci2(n-1) +  
                fibonacci2(n-2)
```

Opera solo con numeri interi.



# Albero della ricorsione

Utile per risolvere la relazione di ricorrenza.  
Nodi corrispondenti alle chiamate ricorsive.  
Figli di un nodo corrispondenti alle sottochiamate.





# Albero della ricorsione

Dall'albero della ricorsione e sfruttando alcuni teoremi sul numero di foglie e sul numero di nodi interni contenuti in un albero binario, si evince che l'algoritmo *fibonacci2* è **MOLTO** lento: il numero di linee di codice mandate in esecuzione a fronte di una generica chiamata alla funzione *fibonacci2(n)* cresce infatti ....

**!!!! come i conigli di Fibonacci !!!!**

Ad esempio:

- per  $n=8$  vengono mandate in esecuzione 61 linee di codice,
- per  $n=45$  vengono mandate in esecuzione 3.404.709.508 linee di codice.



# Algoritmo fibonacci3



Perché l'algoritmo *fibonacci2* è lento? Perché continua a ricalcolare ripetutamente la soluzione dello stesso sottoproblema →

memorizzare allora in un array le soluzioni dei sottoproblemi

```
algoritmo fibonacci3(intero n) → intero  
  sia Fib un array di n interi  
  Fib[1] ← Fib[2] ← 1  
  for i = 3 to n do  
    Fib[i] ← Fib[i-1] + Fib[i-2]  
  return Fib[n]
```



# tempo di esecuzione

L'algoritmo *fibonacci3* impiega un tempo proporzionale a  $n$  invece che *esponenziale in  $n$*  come *fibonacci2*.

Ad esempio:

- ◆ per **n=45** vengono mandate in esecuzione 90 linee di codice, risultando così 38 milioni di volte più veloce di *fibonacci2*!!!!
- ◆ per **n=58** *fibonacci3* è circa 15 miliardi di volte più veloce di *fibonacci2*!!!

	<i>fibonacci2</i> (58)	<i>fibonacci3</i> (58)
Pentium IV 1700MHz	15820 sec. ( $\simeq$ 4 ore)	0.7 milionesimi di secondo
Pentium III 450MHz	43518 sec. ( $\simeq$ 12 ore)	2.4 milionesimi di secondo
PowerPC G4 500MHz	58321 sec. ( $\simeq$ 16 ore)	2.8 milionesimi di secondo



# Algoritmo fibonacci4



*fibonacci3* usa un array di dimensione  $n$ .

In realtà non ci serve mantenere tutti i valori di  $F_n$  precedenti, ma solo gli ultimi due, riducendo lo spazio a poche variabili in tutto:

**algoritmo** fibonacci4(*intero*  $n$ )  $\rightarrow$  *intero*

$a \leftarrow b \leftarrow 1$

**for**  $i = 3$  **to**  $n$  **do**

$c \leftarrow a + b$

$a \leftarrow b$

$b \leftarrow c$

**return**  $b$

$a$  rappresenta  $Fib[i-2]$

$b$  rappresenta  $Fib[i-1]$

$c$  rappresenta  $Fib[i]$



# Potenze ricorsive

*fibonacci4* non è il miglior algoritmo possibile.

E' possibile dimostrare per induzione la seguente proprietà di matrici:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} = \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix}$$

Useremo questa proprietà per progettare un algoritmo più efficiente.



# Algoritmo fibonacci5

**algoritmo fibonacci5**(*intero n*)  $\rightarrow$  *intero*

1.  $M \leftarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
2. **for**  $i = 1$  **to**  $n - 1$  **do**
3.  $M \leftarrow M \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
4. **return**  $M[0][0]$

Il tempo di esecuzione è ancora  $O(n)$ .  
Cosa abbiamo guadagnato?



# Calcolo di potenze

Possiamo calcolare la n-esima potenza elevando al quadrato la (n/2)-esima potenza.

Se n è dispari eseguiamo una ulteriore moltiplicazione

Esempio:

$$3^2=9$$

$$3^4=(3^2)^2=(9)^2=81$$

$$3^8=(3^4)^2=(81)^2=6561$$



# Algoritmo fibonacci6

**algoritmo** fibonacci6(*intero*  $n$ )  $\rightarrow$  *intero*

1.  $M \leftarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
2. potenzaDiMatrice( $M, n - 1$ )
3. **return**  $M[0][0]$

**procedura** potenzaDiMatrice(*matrice*  $M$ , *intero*  $n$ )

4. **if** ( $n > 1$ ) **then**
5.     potenzaDiMatrice( $M, n/2$ )
6.      $M \leftarrow M \cdot M$
7.     **if** ( $n$  è dispari) **then**  $M \leftarrow M \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$



# Riepilogo

Trascurabile?



	Tempo di esecuzione	Occupazione di memoria
fibonacci2	$O(2^n)$	$O(n)$
fibonacci3	$O(n)$	$O(n)$
fibonacci4	$O(n)$	$O(1)$
fibonacci5	$O(n)$	$O(1)$
fibonacci6	$O(\log n)$	$O(\log n)$



# Ordine di grandezza dei problemi

Se ripiego un foglio di giornale 50 volte ottengo una pila di carta ...	così alta da arrivare da Roma sino a New York
Se piego un'altra volta ottengo una pila di carta ...	che arriva alla luna e oltre
Scacchiera 14x14: un chicco riso sulla prima casella, due sulla seconda, quattro sulla terza, ...	l'intera produzione mondiale di riso degli ultimi 100 anni non è sufficiente a soddisfare la richiesta
Un imprenditore sta valutando se emigrare all'estero. Due possibilità: 1) In Logaritmia le tasse sono il logaritmo in base 10 dello stipendio 2) In Percentualia esse sono lo 0,1% ovvero un millesimo dello stipendio	10 milioni o 10 euro?
Ricerca numero primo: 1) n 20 cifre, con algoritmo ordine n 2) n 20 cifre, con algoritmo ordine $\sqrt{n}$	$\approx 3000$ anni $\approx 10$ secondi



**Fatto 1** *Se ripiego un foglio di giornale 50 volte ottengo una pila di carta così alta da arrivare da Roma sino a New York.*

**Dimostrazione:** Un foglio di giornale é spesso (a occhio)  $0.1\text{mm} = 10^{-4}\text{m}$ . Piegando un foglio una volta lo spessore raddoppia, piegando ancora esso quadruplica, e così via. Piegando  $n$  volte lo spessore diventa

$$2^n.$$

Cinquanta piegamenti corrispondono ad  $n = 50$  per cui, ricordando che  $2^{10} = 1024 > 10^3$ ,

$$\begin{aligned}\text{spessore-pila-di-carta} &= 2^{50} \times \text{spessore-foglio-di-giornale} \\ &= 2^{50} 10^{-4} \text{ m} \\ &= (2^{10})^5 10^{-4} \text{ m} \\ &\geq (10^3)^5 10^{-4} \text{ m} \\ &= 10^{15} 10^{-4} \text{ m} \\ &= 10^{11} \text{ m} \\ &= 10^8 \text{ km} \\ &= 100 \text{ milioni di chilometri!}\end{aligned}$$

Piú che abbondante quindi per andare da Roma a NYC.





**Fatto 2** *L'intera produzione mondiale di riso degli ultimi 100 anni non è sufficiente a soddisfare la richiesta dell'asceta.*

**Dimostrazione:** Un chicco di riso pesa (a occhio) almeno un centesimo di grammo, cioè  $10^{-5}kg$ . Il numero totale di chicchi richiesti dall'asceta é

$$1 + 2 + 4 + 8 + 16 + \dots + 2^{196} = \sum_{i=0}^{196} 2^i = 2^{197} - 1 \approx 2^{197}$$

per cui, ricordando che  $2^{10} = 1024 > 10^3$ , il numero di kili di riso ammonterebbero a

$$2^{197} 10^{-5} kg > (10^3)^{19.7} 10^{-5} kg = 10^{59.1} 10^{-5} kg > 10^{54} kg$$

La massa dell'universo é stimata essere  $10^{55}kg$ . L'universo contiene senz'altro almeno dieci galassie per cui... ▽

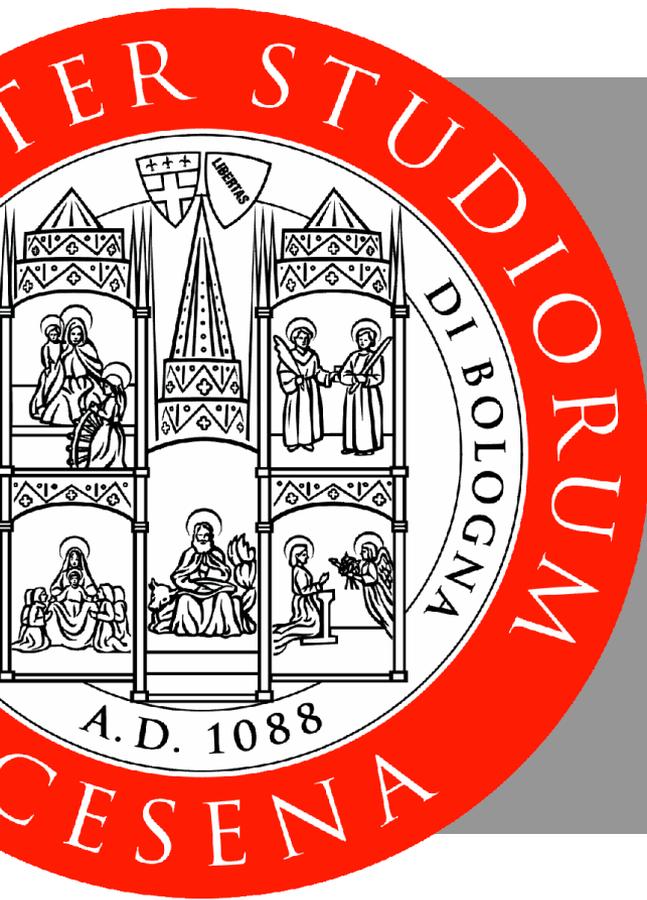


**Fatto 3** *Logaritmia forever!*

**Dimostrazione:** In Percentualia le tasse da pagare ammontano a €  $10^{10}10^{-3} = 10$  milioni.  
In Logaritmia invece

$$\log_{10} 10^{10} = 10 \text{ €}$$





# Olimpiadi **P**roblem **S**olving **COME**

Antonella Carbonaro  
[antonella.carbonaro@unibo.it](mailto:antonella.carbonaro@unibo.it)

Dipartimento di Informatica - Scienza e Ingegneria  
Alma Mater Studiorum, Università di Bologna

Cervia – 27 aprile 2017



# Olimpiadi

In genere s'intende:

- competizione
- rivolta a singoli
- per promuovere l'eccellenza
- tra i 16 e i 18 anni



# OPS

- Competizione → attività didattiche e competizioni
- rivolta a singoli → singoli e squadre
- per promuovere l'eccellenza → per tutti
- tra i 16 e i 18 anni → per tutta la scuola dell'obbligo (8-16), su tre livelli



# OPS – per tutti

- scuola primaria, classi IV e V
  - ◆ a squadre
- scuola secondaria di primo grado
  - ◆ a squadre
  - ◆ individuali
- scuola secondaria di secondo grado (primo biennio).
  - ◆ a squadre
  - ◆ Individuali

**Squadre:** costituite da quattro allievi (gender neutrality)

Partecipazione tramite registrazione dell'Istituzione scolastica

[www.olimpiadiproblemsolving.it](http://www.olimpiadiproblemsolving.it)



# Attività didattiche

- Attività ("giochi", sfide) locali sui tre livelli
- Prove erogate su web, da server centrale
- Con attività didattiche di supporto. Dopo ogni prova:
  - ◆ soluzione dei vari esercizi,
  - ◆ commenti alle soluzioni, che costituiscono una traccia per il percorso formativo
- Gli argomenti proposti sono allineati con quelli adottati nelle indagini e nelle competizioni nazionali e internazionali riguardanti la capacità di problem solving
- Per consentire la conoscenza dei contenuti e l'approccio metodologico della competizione sono state predisposte prove di **allenamento**.
- Agli allenamenti accedono tutti gli studenti, con le modalità ritenute più opportune dai rispettivi docenti.
- Le prove sono disponibili sul sito.
- Seguite da una competizione



# Competizioni

- Gestite da un sistema automatico sia per la distribuzione dei testi delle prove sia per la raccolta dei risultati e la loro correzione → vincoli alla formulazione dei quesiti e delle relative risposte
- Le prove di istituto hanno la durata di 120 minuti
- Le prove regionali e la finalissima avranno la durata di 90 minuti
- Il tempo assegnato per una prova non è sufficiente ad un singolo → sono necessari pianificazione, divisione dei compiti, organizzazione

Pianificazione, divisione dei compiti, organizzazione:  
sono già **problem solving!**



# Gare di istituto

- Per individuare:
  - ◆ la **squadra**,
  - ◆ **fino a 3 studenti** nel caso delle scuole secondarie di I e II grado,
- che rappresenteranno l'istituzione scolastica alla gara regionale, per ogni livello di competizione.
- E' opportuno che alle gare di istituto partecipi il **maggior numero** possibile di squadre/studenti.
- La partecipazione alle gare di istituto è **fortemente raccomandata** perché esse propongono un percorso di preparazione alle selezioni regionali.
- Per tutti gli studenti, anche non partecipanti → attività della classe, non della squadra
- Le Istituzioni scolastiche individuano, **entro il 15 marzo 2017**, le squadre e gli studenti che partecipano alla fase regionale



# Gare di istituto - calendario

## ■ ***I prova a squadre***

- ◆ 14 dicembre 2016 (S. Primaria)
- ◆ 12 dicembre 2016 (S. Sec. I gr.)
- ◆ 13 dicembre 2016 (S. Sec. II gr.)

## ■ ***II prova a squadre***

- ◆ 17 gennaio 2017 (S. Primaria)
- ◆ 18 gennaio 2017 (S. Sec. I gr.)
- ◆ 16 gennaio 2017 (S. Sec. II gr.)

## ■ ***III prova a squadre***

- ◆ 13 febbraio 2017 (S. Primaria)
- ◆ 14 febbraio 2017 (S. Sec. I gr.)
- ◆ 15 febbraio 2017 (S. Sec. II gr.)

## ■ ***IV prova a squadre***

- ◆ 6 marzo 2017 (S. Primaria)
- ◆ 7 marzo 2017 (S. Sec. I gr.)
- ◆ 8 marzo 2017 (S. Sec. II gr.)

## ***I prova individuale***

-----

- 16 dicembre 2016 (S. Sec. I gr.)
- 15 dicembre 2016 (S. Sec. II gr.)

## ***II prova individuale***

-----

- 20 gennaio 2017 (S. Sec. I gr.)
- 19 gennaio 2017 (S. Sec. II gr.)

## ***III prova individuale***

-----

- 16 febbraio 2017 (S. Sec. I gr.)
- 17 febbraio 2017 (S. Sec. II gr.)

## ***IV prova individuale***

-----

- 10 marzo 2017 (S. Sec. I gr.)
- 9 marzo 2017 (S. Sec. II gr.)



# Gare regionali

- Presso **scuole-polo** provinciali e/o regionali che saranno individuate e segnalate sul sito entro gennaio 2017.
- Partecipa:
  - ◆ una **squadra**
  - ◆ **fino a tre studenti** (Scuola Secondaria di I e II grado)
- per ogni Istituzione scolastica registrata sul sito.
- La selezione della squadra " regionale" è del coordinatore locale
- Nel caso di Istituti scolastici composti da più **plessi** (scuole I ciclo) e/o più **indirizzi** (scuole II ciclo) si consente la partecipazione di una squadra a plesso e/o indirizzo.
- Gli **Istituti comprensivi** partecipano con una squadra per ciascun livello previsto dalla competizione secondo il criterio sopradescritto.



# Gare regionali - calendario

## ■ Scuola secondaria di I ciclo

- ◆ **3 aprile 2017**
- ◆ h. 10.00 gara a squadre scuola primaria
- ◆ h. 12.00 gara a squadre scuola secondaria I grado
- ◆ h. 14.00 gara individuale scuola secondaria I grado

## ■ Scuola secondaria II ciclo

- ◆ **4 aprile 2017**
- ◆ h. 10.00 gara a squadre
- ◆ h. 12.00 gara individuale



# Finalissima nazionale

- **A squadre**
  - ◆ per ciascun livello scolastico la migliore squadra classificata nella selezione regionale, purché con punteggio superiore alla media nazionale.
- **Individuale**
  - ◆ il primo classificato di ogni regione, purché con punteggio superiore alla media nazionale.
- Il CTS si riserva di invitare alla finale le squadre/studenti con i migliori punteggi nazionali indipendentemente dalla regione di appartenenza.
- Le finalissime nazionali si terranno a Cesena, presso il Corso di Studi in Ingegneria e Scienze Informatiche - Dipartimento di Informatica, Scienza e Ingegneria dell' Università di Bologna - Sede di Cesena il **5 e 6 maggio 2017**



# Finalissima - calendario

## ■ Scuola Secondaria di II grado:

- ◆ **5 maggio 2017:**
- ◆ ore 11.00 finale a squadre;
- ◆ ore 15.00 finale individuale.
  
- ◆ Segue premiazione.

## ■ Scuola Primaria e Scuola Secondaria di I grado:

- ◆ **6 maggio 2017:**
- ◆ ore 11.00 finale a squadre primaria e secondaria I grado;
- ◆ ore 15.00 finale individuale I grado.
  
- ◆ Segue premiazione.



# Qualche numero a.s. 15/16

- 20800 studenti totali coinvolti, 2300 partecipano alla fase individuale
- 3700 squadre
  
- 830 scuole
  - ◆ 258 al nord
  - ◆ 177 centro
  - ◆ 395 sud
  
- 318 scuole superiori
  - ◆ 40% licei
  - ◆ 12% ITIS
  - ◆ 45% IIS
  - ◆ 3% professionali
  
- 62 primarie
- 84 secondarie primo grado
- 366 Istituti comprensivi



# Le finali a.s. 15/16

Corso di Laurea in Ingegneria e Scienze Informatiche dell'Università degli studi di Bologna, sede di Cesena:

- 21 squadre di scuola secondaria di II grado,
  - 20 squadre di scuola secondaria di I grado e
  - 20 squadre di scuola primaria.
- 
- 21 studenti di scuola secondaria di II grado e
  - 20 di scuola secondaria di I grado.
- 
- 285 gli studenti finalisti che si sono sfidati per la conquista del podio su problemi e algoritmi e circa 600 le persone arrivate da tutta Italia a Cesena per l'occasione (ad esempio, un'intera scolaresca della Calabria ha organizzato la gita di classe per accompagnare i suoi 4 finalisti).



# Prove

- tre livelli
- stesso tipo di prove
- differenze in dimensione o astrattezza



# Gara a squadre

L'articolazione dei problemi sarà, *usualmente*, la seguente:

- da quattro a sei problemi formulati in italiano e scelti, di volta in volta, tra l'insieme dei **"Problemi ricorrenti"** (si veda il successivo elenco);
- due o tre problemi formulati in italiano e relativi a uno **pseudo-linguaggio** di programmazione;
- un problema di comprensione di un testo in lingua **italiana**;
- due o tre problemi, in genere formulati in **inglese**, di argomento ogni volta diverso (almeno in linea di principio).



# Gara individuale

L'articolazione dei problemi sarà, *usualmente*, la seguente:

- due o tre problemi formulati in italiano e scelti, di volta in volta, tra l'insieme dei “**Problemi ricorrenti**” (si veda il successivo elenco);
- due o tre problemi formulati in italiano e relativi a uno **pseudo-linguaggio** di programmazione;
- due problemi, in genere formulati in **inglese**, di argomento ogni volta diverso (almeno in linea di principio)



# Le finali a.s. 16/17



[HOME](#) [EDIZIONE 2016/2017](#) [PARTECIPA](#) [LE OPS](#)

La Finalissima delle Olimpiadi di Problem Solving 2017 si svolgerà a Cesena, in collaborazione con il Dipartimento di Informatica - Scienza e Ingegneria (DISI), Sede di Cesena, Alma Mater Studiorum - Università di Bologna, e Ser.In.Ar. Forlì - Cesena, presso Palazzo Mazzini Marinelli, via Sacchi 3, Cesena secondo il seguente programma:

## **Venerdì 5 maggio 2017**

### **Scuola Secondaria di II grado:**

h. 9.30 ritrovo partecipanti

h. 9.45 saluto Autorità

h. 10.30 gara a squadre scuola secondaria II grado

h: 12.30 lunch

h. 14.00 gara individuale scuola secondaria II grado

h. 16.00 premiazione

h. 17.00 – 19.00 Tavola Rotonda presso la Sala Ex-Macello (via Mulini 23)

“Pensiero Computazionale e Coding nelle Scuole: Esperienze, Direzioni, Collaborazioni”

## **Sabato 6 maggio 2017**

### **Scuola Primaria e Scuola Secondaria di I grado:**

h. 9.30 ritrovo partecipanti

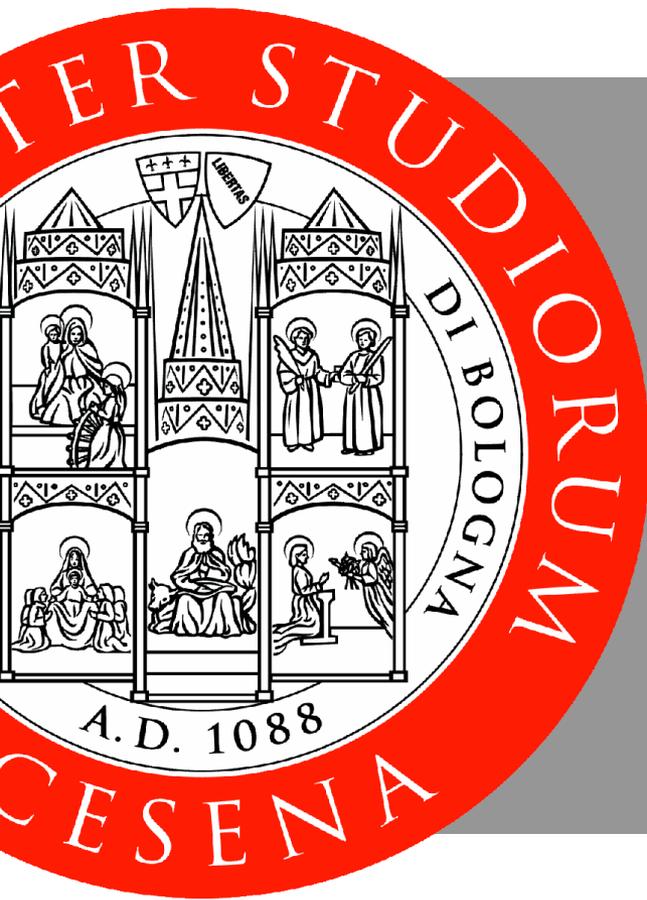
h. 9.45 saluto Autorità

h. 10.30 gara a squadre scuola primaria e secondaria I grado

h: 12.30 lunch

h. 14.00 gara individuale scuola secondaria I grado

h. 16.00 premiazione



# Olimpiadi **P**roblem **S**olving **COSA**

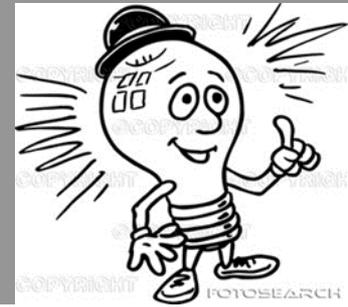
Antonella Carbonaro  
[antonella.carbonaro@unibo.it](mailto:antonella.carbonaro@unibo.it)

Dipartimento di Informatica - Scienza e Ingegneria  
Alma Mater Studiorum, Università di Bologna

Cervia – 27 aprile 2017



# Il coding alle OPS



- A partire dall'a.s. **2016/2017**, le OPS includono un'ulteriore modalità di partecipazione e competizione che riguarda il *Coding*.
- La modalità concerne l'ideazione e implementazione di un programma su piattaforma Scratch o Snap! a partire da un tema (o problema) assegnato, diverso a seconda della categoria (sp, ss I grado); la modalità di partecipazione per la ss di II grado verrà definita nelle prossime settimane.
- La partecipazione avviene inviando il programma mediante il sito delle OPS accedendo all'area riservata alle scuole e utilizzando l'apposita voce di menu.
- I programmi inviati verranno valutati dal CTS delle OPS in base a:
  - originalità
  - qualità del lavoro
  - qualità del codice e dell'implementazione
  - correttezza e completezza



# Coding - problemi



- Contestualmente alla valutazione, a tutte le Scuole partecipanti verrà rilasciato un **attestato di partecipazione** digitale, uno per la scuola e uno per ogni membro della squadra, personalizzato.
- I temi/problemi cambiano di anno in anno.
- I lavori valutati come migliori verranno selezionati per partecipare al "**Workshop sul Pensiero Computazionale e Coding**" che si terrà a Cesena contestualmente alla finale delle OPS.
- Ogni squadra avrà la possibilità di presentare e discutere il proprio lavoro al pubblico del workshop, eventualmente esteso/arricchito con ulteriori idee maturate nel tempo e i suggerimenti presenti nella valutazione.
- A partire dalla presentazione, dalla qualità del lavoro e dalla discussione prodotta, verranno scelti e premiati i 3 migliori lavori per ogni categoria.



# Coding - scadenze



- La scadenza per inviare i lavori è fissata per il 20 marzo 2017.
- Entro l'8 aprile 2017 verrà notificata la valutazione e contestualmente l'invito a presentare il lavoro al workshop per i lavori selezionati
- Ogni scuola può partecipare con una squadra
- Ogni squadra può essere composta da un numero variabile, non vincolato di alunni, eventualmente appartenenti a classi diverse, e uno o più insegnanti
- Ogni squadra selezionata per partecipare al workshop potrà inviare una delegazione di max 5 alunni e 1 docente



# Ecco il mio paese



- Realizzare un progetto Scratch/Snap! che implementi una sorta di guida turistica per il proprio paese/città. Esempi di possibili funzionalità:
  - ◆ visualizzare la mappa del paese con alcuni punti di interesse
  - ◆ fare in modo che l'utente possa clickare su un punto di interesse facendo comparire una breve descrizione del luogo, con foto e/o disegni e poi tornare alla mappa.
  - ◆ avere una modalità "Tour" in cui in automatico uno sprite funge da guida turistica, muovendosi fra i punti di interesse, compiendo un certo percorso a tappe all'interno della mappa
  - ◆ avere una modalità "macchina del tempo", che permetta di fare una guida su un percorso cittadino mostrando come erano i luoghi in un precedente momento, eventualmente scegliendo periodi storici diversi
  - ◆ ...



# In gita



- Realizzare un programma Scratch/Snap! che implementi un (video) gioco in cui è protagonista una corriera con alunni della scuola in gita per la propria regione, pilotata dal giocatore.
- Spunto: la corriera parte dal proprio paese e ha come obiettivo visitare il numero maggiore possibile di luoghi, fra quelli inclusi e visualizzati sulla mappa, entro un certo tempo massimo. Nel muoversi deve fare attenzione ad evitare ostacoli e insidie che possono esserci nel percorso (ad esempio buche), alcuni dei quali in movimento (ad esempio altre macchine). Il punteggio accumulato varia aumentando/diminuendo ogni volta che viene raggiunta una tappa o che non evita un ostacolo.



# In gita



## ■ Esempi di altri possibili elementi:

- ◆ la corriera ha a disposizione una certa quantità di carburante iniziale, che diminuisce man mano che la corriera si muove. Nel percorso è presente anche un certo insieme di distributori, passando per i quali è possibile aumentare il carburante disponibile
- ◆ la corriera ha una certa velocità che può essere aumentata o diminuita. Tuttavia se viene superato un certo limite, parte una macchina della polizia che insegue la corriera e se la raggiunge prima che abbia diminuito la velocità riportandola sotto il limite il punteggio viene diminuito di una certa quantità
- ◆ livelli di gioco diversi potrebbero variare la velocità e numero degli ostacoli
- ◆ in alcuni momenti del gioco potrebbero apparire (e poi scomparire) degli elementi “bonus” raggiungendo i quali si ottiene un qualche beneficio (es. Un cartello SOS che permette di fare il pieno di carburante o eliminare tutte le buche presenti...)
- ◆ ...



# Problemi ricorrenti

- a) Regole e deduzioni.
- b) Fatti e conclusioni.
- c) Grafi.
- d) Knapsack.
- e) Pianificazione.
- f) Statistica elementare.
- g) Relazioni tra elementi di un albero.
- h) Flussi in una rete.
- i) Crittografia.
- j) Movimento di un robot o di un pezzo degli scacchi.
- k) Sottosequenze.
- l) Pseudolinguaggio



# Termini

- Si consideri, ad esempio, la tabella seguente che si riferisce ad un magazzino di minerali: riporta la sigla il peso e il valore di ogni esemplare presente nel magazzino.

	<b>minerale</b>		
	<b>sigla</b>	<b>peso in chili</b>	<b>valore in euro</b>
	m1	30	213
	m2	35	200
	m3	38	230

- Il contenuto della tabella può essere descritto con dei **termini**; ogni termine è così *definito*:  
minerale(<sigla>,<peso in chili>,<valore in euro>)
- La definizione mostra il nome del termine, il numero degli argomenti e il loro significato



# Termini

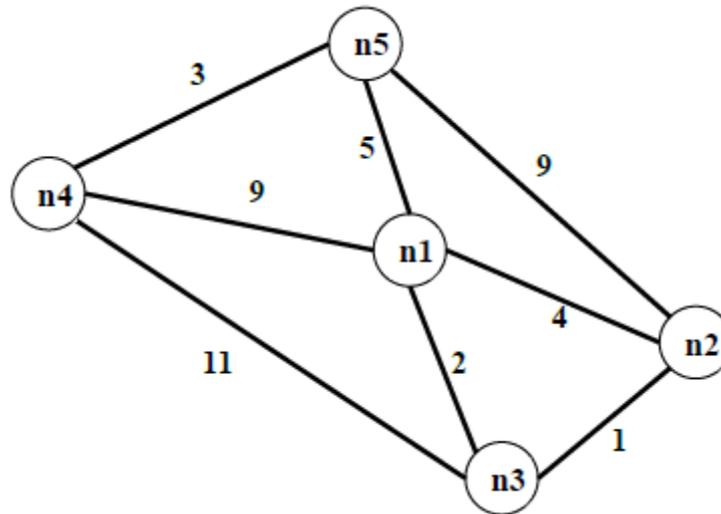
- Nel caso delle tabelle il nome del termine è il nome della tabella, il numero degli argomenti è il numero delle colonne e il significato di ogni termine è l'intestazione della colonna. Il contenuto della tabella può essere così descritto:

minerale(m1,30,213)

minerale(m2,35,200)

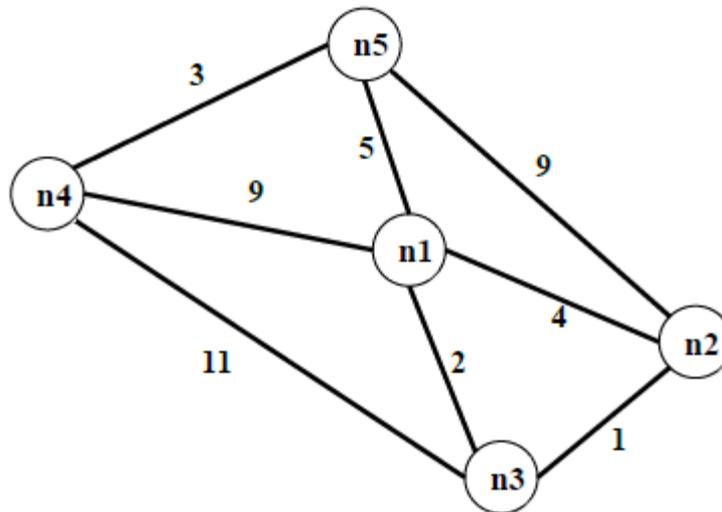
minerale(m3,38,230)

- Anche un grafo può essere descritto con dei termini; ogni termine è associato a un arco ed è così *definito*:  
arco(<nodo1>,<nodo2>,<lunghezza>)
- La definizione mostra il nome del termine e il numero degli argomenti (tre): i primi due si riferiscono ai nodi posti agli estremi dell'arco, il terzo descrive la lunghezza dell'arco.



- I termini che descrivono il grafo in figura sono:

arco(n1,n2,4) arco(n1,n3,2) arco(n1,n4,9) arco(n1,n5,5)  
arco(n4,n5,3) arco(n4,n3,11) arco(n2,n3,1) arco(n2,n5,9)





# Liste

- Una lista è una scrittura del tipo:  
[a1,2,56,b2,3,67]
- cioè una coppia di parentesi quadre che racchiudono un certo numero di elementi separati da virgole (se sono più di uno). Una lista può non contenere elementi: in tal caso si dice lista vuota.
- Gli elementi sono parole, stringhe, numeri o **altre liste**.



# Liste - esempio

- Per indicare la posizione sulla scacchiera dei quadrati neri si può usare una lista i cui elementi sono le coordinate delle caselle che li contengono; si ottiene, quindi, una lista di liste:

[[2,5],[3,1],[3,5],[4,8],[5,2],[7,4]]

♁			■			
	■	■		11		
					■	
				12		
		5	■		13	
		■				♁



# Liste - esempio

- Volendo indicare la posizione sulla scacchiera e il valore dei quattro numeri si può usare una lista di quattro elementi che, a loro volta, sono liste; ciascuna lista “interna” è formata dalle coordinate della casella che contiene il numero seguite dal valore:

[[3,2,5],[6,3,12],[6,5,11],[7,2,13]]

♞			■			
	■	■		11		
					■	
				12		
		5	■		13	
		■				♟



# Alcuni problemi ricorrenti

- a) **Regole e deduzioni.**
- b) Fatti e conclusioni.
- c) **Grafi.**
- d) **Knapsack.**
- e) Pianificazione.
- f) Statistica elementare.
- g) Relazioni tra elementi di un albero.
- h) Flussi in una rete.
- i) Crittografia.
- j) Movimento di un robot o di un pezzo degli scacchi.
- k) Sottosequenze.
- l) **Pseudolinguaggio**

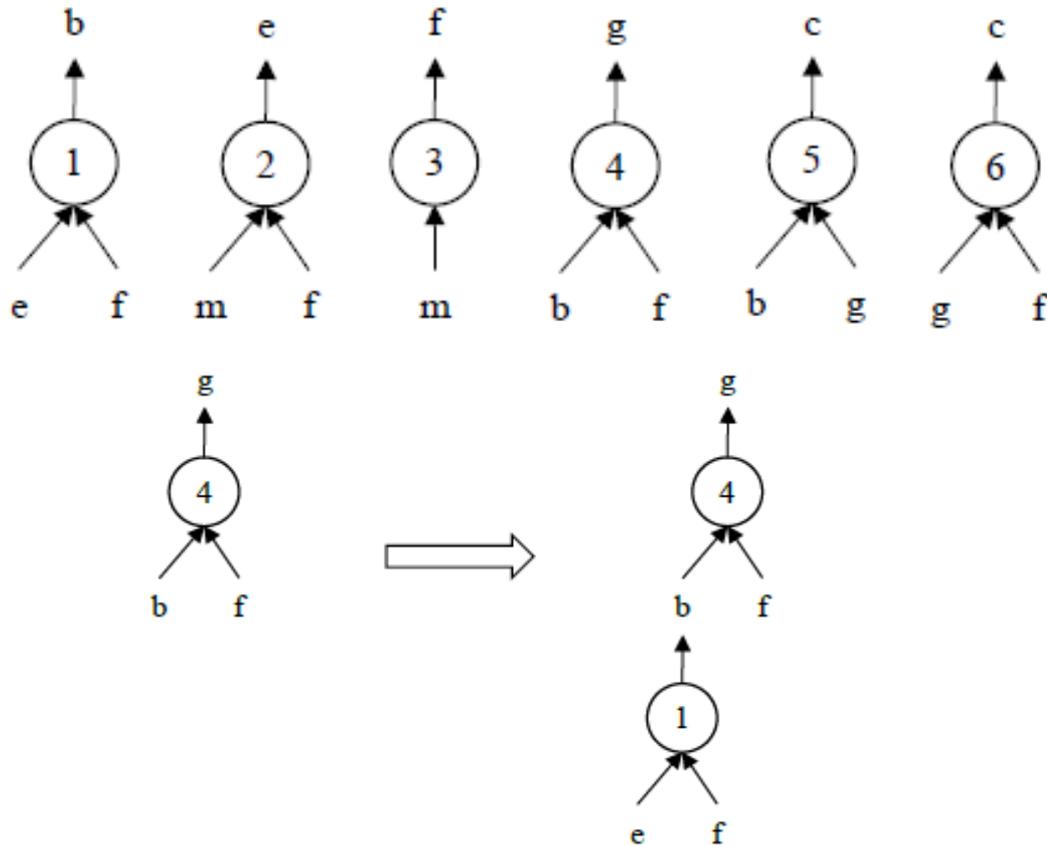


# Regole e deduzioni

- Si considerino le seguenti regole:  
regola(1,[e,f],b) regola(2,[m,f],e) regola(3,[m],f)  
regola(4,[b,f],g) regola(5,[b,g],c) regola(6,[g,f],c)
- Regola 1: si può calcolare (o dedurre) **b** conoscendo **e** ed **f**
- Conoscendo **b** ed **f** è possibile dedurre **g** con la regola 4.
- Quindi, a partire da **e** ed **f** è possibile dedurre prima **b** (con la regola 1) e poi **g** (con la regola 4).
- Un *procedimento di deduzione* (o deduttivo, o di calcolo) è rappresentato da un *insieme di regole da applicare in sequenza opportuna* per dedurre un certo elemento (incognito) a partire da certi dati. Il procedimento [1,4] descrive la soluzione del problema: “dedurre **g** a partire da **e** ed **f**”.

# Regole e deduzioni

- **radice** (in alto) è il conseguente, le **foglie** (in basso) sono gli antecedenti.





# Regole e deduzioni - problema

- Siano date le seguenti regole:  
regola(1,[b,c],a) regola(2,[c,d],a)  
regola(3,[b,c,d],a) regola(4,[b,a],f)
- Trovare:
  1. la sigla N della regola che consente di dedurre **a** da **d** e **c**;
  2. la lista L che rappresenta il procedimento per dedurre **f** da **b** e **c**.

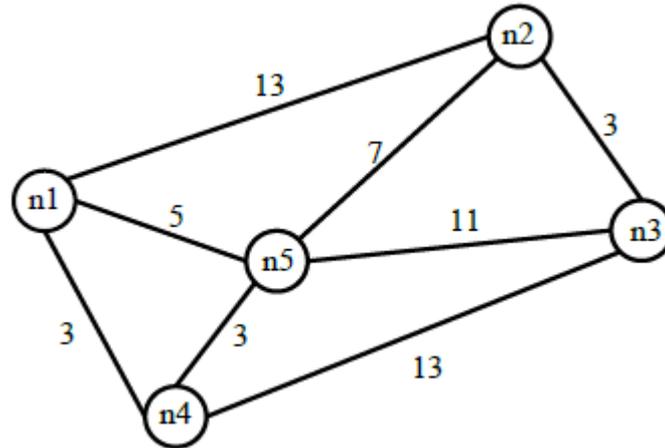
Per risolvere questo tipo di problemi si può usare il metodo *backward* (o *top down*) che consiste nel partire dalla incognita e cercare di individuare una regola per derivarla.



# Grafi - problema

- Un grafo (che corrisponde alla rete di strade che collegano delle città) è descritto dal seguente elenco di archi:  
 $a(n1, n2, 13)$   $a(n2, n3, 3)$   $a(n3, n4, 13)$   $a(n1, n4, 3)$   
 $a(n4, n5, 3)$   $a(n5, n1, 5)$   $a(n2, n5, 7)$   $a(n3, n5, 11)$
  
- Disegnare il grafo e trovare:
  1. la lista L1 del percorso semplice più breve tra  $n1$  e  $n3$ ;
  2. la lista L2 del percorso semplice più lungo tra  $n1$  e  $n3$ .

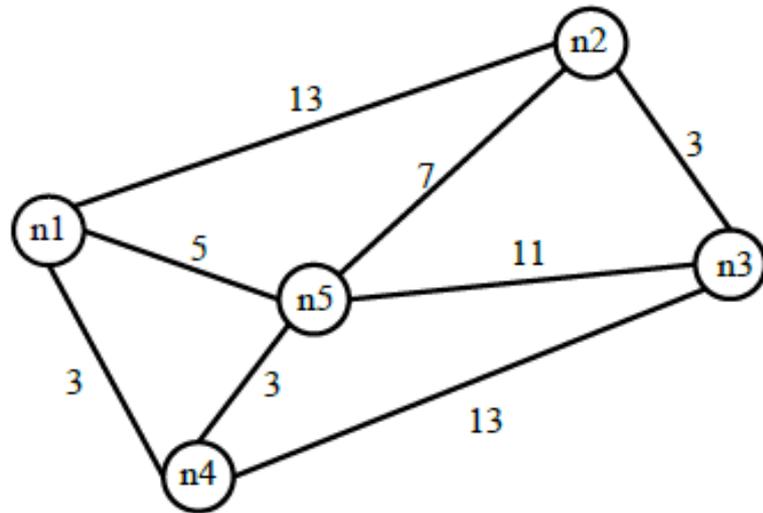
# Grafi - problema



- Per rispondere alle due domande occorre elencare sistematicamente *tutti* i percorsi, che non passino più volte per uno stesso punto, tra n1 e n3:

## ■ PERCORSO da n1 a n3 LUNGHEZZA

- [n1, n2, n3] 16
- [n1, n2, n5, n3] 31
- [n1, n2, n5, n4, n3] 36
- [n1, n5, n3] 16
- [n1, n5, n2, n3] 15
- [n1, n5, n4, n3] 21
- [n1, n4, n3] 16
- [n1, n4, n5, n3] 17
- [n1, n4, n5, n2, n3] 16



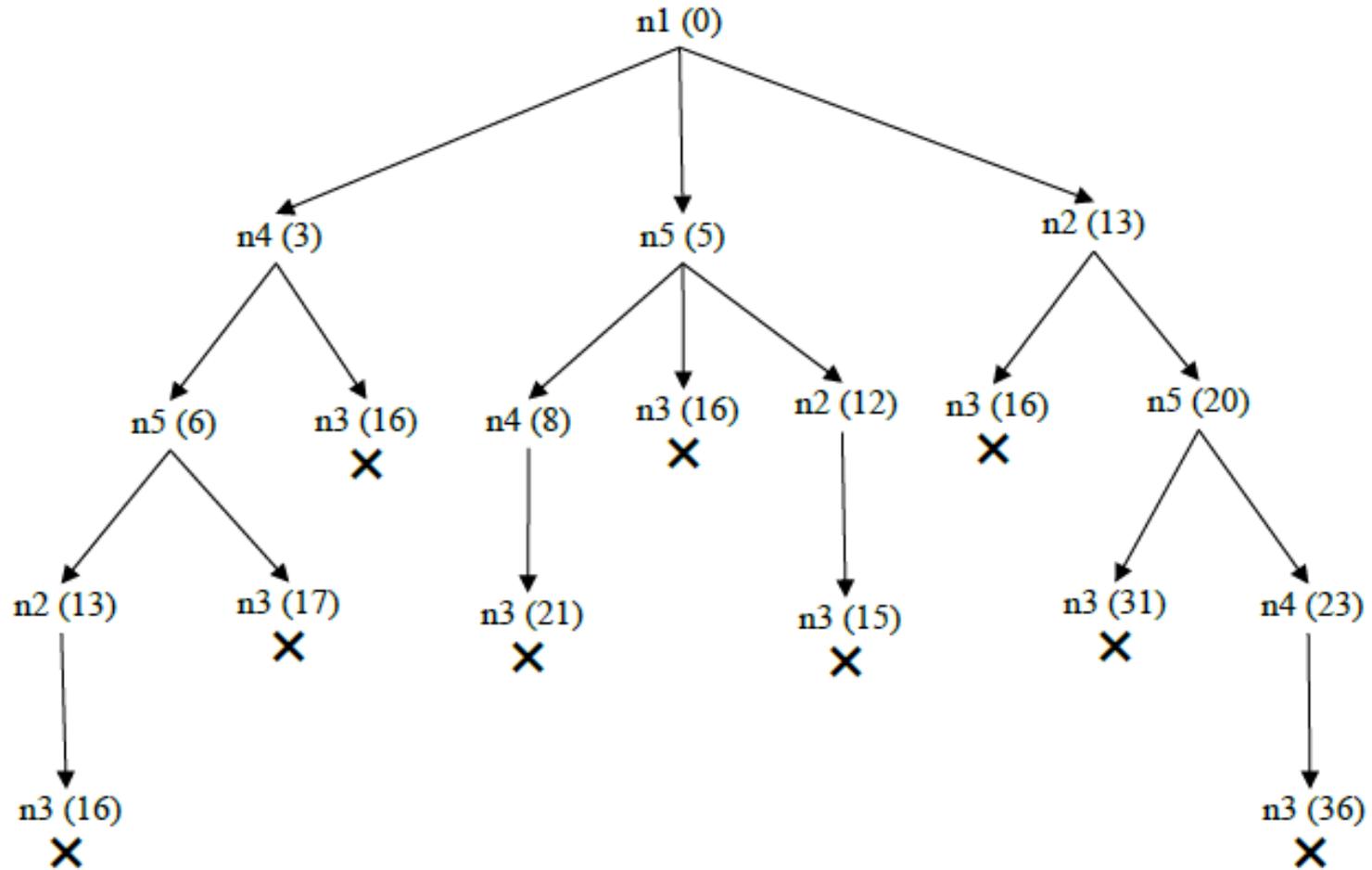


# Grafi - problema

- Si noti che, a partire dal nodo  $n_1$ , si “visitano” i tre nodi adiacenti ( $n_2$ ,  $n_5$ ,  $n_4$ ); a partire da ciascuno di questi si costruiscono tutti i cammini che arrivano a  $n_3$  senza passare per un nodo già visitato (cammini semplici).
- Una maniera grafica di chiara evidenza (ma anche concettualmente profonda) è illustrata dalla seguente figura che mostra un **albero** in cui la **radice** è il nodo di partenza ( $n_1$ ) del grafo, e ogni **nodo** dell'albero ha tanti figli quanti sono i nodi del grafo a lui collegati purché non compaiono come antenati (nell'albero).
- Le **foglie** dell'albero sono il nodo di arrivo ( $n_3$ ) (o un nodo da cui non ci si può più muovere perché il nodo successivo sarebbe un antenato).
- Ad ogni nodo (dell'albero) è stata aggiunta tra parentesi la distanza dalla radice.



# Grafi - problema





# Knapsack

- In un deposito di minerali esistono esemplari di vario peso e valore individuati da sigle di riconoscimento. Ciascun minerale è descritto da una sigla che contiene le seguenti informazioni:  
tab(<sigla del minerale>, <valore in euro>, <peso in Kg>).
- Il deposito contiene i seguenti minerali:  
tab(m1,6,10) tab(m2,7,12) tab(m3,9,50)  
tab(m4,21,32) tab(m5,13,22) tab(m6,14,24)
- Disponendo di un piccolo motocarro con portata massima di 48 Kg trovare la lista L delle sigle di tre minerali diversi che siano trasportabili contemporaneamente con questo mezzo e che abbiano il massimo valore complessivo; calcolare inoltre questo valore V.



# Knapsack

- Per risolvere il problema occorre considerare *tutte* le possibili *combinazioni* di tre minerali diversi, il loro valore e il loro peso.
- Le *combinazioni* corrispondono ai sottoinsiemi: cioè sono indipendenti dall'ordine;
- Per elencarle tutte (una sola volta) conviene costruirle sotto forma di liste i cui elementi sono ordinati, partendo dalle combinazioni che iniziano col “primo” minerale, poi tutte quelle che iniziano col “secondo” minerale, e così via, in modo da essere sicuri di averle considerate tutte
- Costruite le combinazioni occorre individuare quelle trasportabili (cioè con peso complessivo minore o eguale a 48) e tra queste scegliere quella di maggior valore. Nel problema presentato si evince immediatamente che le combinazioni che includono il minerale m3 non sono trasportabili in quanto il suo peso (50 kg) è già superiore a quello massimo trasportabile (48 kg). Di conseguenza tali combinazioni vengono immediatamente scartate senza calcolarne il valore e il peso complessivo.



# Knapsack

COMBINAZIONI	VALORE	PESO	TRASPORTABILI
[m1,m2,m3]	scartata	scartata	no
[m1,m2,m4]	34	72	no
[m1,m2,m5]	26	54	si
[m1,m2,m6]	27	46	si
[m1,m3,m4]	scartata	scartata	no
[m1,m3,m5]	scartata	scartata	no
[m1,m3,m6]	scartata	scartata	no
[m1,m4,m5]	40	64	no
[m1,m4,m6]	41	66	no
[m2,m3,m4]	scartata	scartata	no
[m2,m3,m5]	scartata	scartata	no
[m2,m3,m6]	scartata	scartata	no
[m2,m4,m5]	41	66	no
[m2,m4,m6]	42	68	no
[m3,m4,m5]	scartata	scartata	no
[m3,m4,m6]	scartata	scartata	no
[m4,m5,m6]	48	78	no



# Pseudolinguaggio

procedura	commento
<pre>procedure ESEMPIO1; variables A, B, C1, D integer; input A, B; C1 ← A+B; D ← C1+B-A; A ← C1+D; output A, C1, D; endprocedure;</pre>	<p>inizio della procedura di nome ESEMPIO</p> <p>si dichiara che si usano 4 variabili che assumono valori interi</p> <p>si acquisiscono dall'esterno valori per le variabili A e B</p> <p>la variabile C1 acquisisce valore (di una espressione)</p> <p>la variabile D acquisisce valore (di una espressione)</p> <p>la variabile A acquisisce (un nuovo) valore (di una espressione)</p> <p>si rendono disponibile all'esterno i valori delle variabili A, C1, D</p> <p>fine della procedura</p>



# Pseudolinguaggio - problema

```
procedure PROVA1;  
variables A, B, C, D integer;  
  input A, B;  
  C ← A + B;  
  D ← A × B;  
  A ← C+B;  
  B ← (A+B) ×(A- B);  
  output C, D, A, B;  
endprocedure;
```

- Input: A vale 4, B vale 2
- Determinare i valori di output



# Pseudolinguaggio

procedura	commento
<pre>procedure ESEMPIO2; variables A, B, K integer; A ← 0; B ← 0; for K from 1 to 4 step 1 do;     A ← A+K;     B ← B+K×K; endfor; output A, B;</pre>	<p>inizio del ciclo (che è ripetuto 4 <i>volte</i> con i valori di K: 1, 2, 3, 4) primo statement del ciclo (A assume via via i valori 1, 3, 6, 10) secondo statement del ciclo (B assume via via i valori 1, 5, 14, 30) segnala che il ciclo arriva fin qui</p>



# Pseudolinguaggio - problema

```
procedure PROVA2;  
variables A, B, M, N, K integer;  
input A;  
M ← 0;  
N ← 0;  
for K from 1 to 10 step 1 do;  
    input B;  
    if A > B then M ← M + A; endif;  
    if A < B then N ← N + A; endif;  
endfor;  
output M, N;  
endprocedure;
```

- Input: A vale 5, B vale rispettivamente: 9, 3, 7, 2, 8, 5, 1, 4, 4, 5.
- Determinare i valori di output.



# Pseudolinguaggio - problema

- Si consideri la seguente procedura (scritta in maniera sintatticamente scorretta: il simbolo X non è definito).

```
procedure P1;  
variables A, B, C, D integer;  
  D ← 0;  
  input A, B, C;  
  D ← A + B + C + X;  
  output D;  
endprocedure;
```

Input: A vale 2, B vale 5, C vale 7

Trovare, tra le variabili dichiarate nella procedura, il nome da sostituire a X per ottenere in output il valore 21



# Pseudolinguaggio - problema

```
procedure PROVA4;  
variables J, L, C integer;  
variables A, B string;  
  input A;  
  C ← 0;  
  L ← |?A;  
  for J from 1, step 1 to L do;  
    B ← |(J,J) A;  
    if B = 'a' then C ← C + 1; endif;  
  endfor;  
  output C;  
endprocedure;
```

|? A calcola la lunghezza della stringa A  
|(1,3) A estrae dalla stringa A la sottostringa che va dal primo al terzo carattere quindi  
|(J,J) A estrae dalla stringa A la sottostringa che contiene il J-simo carattere

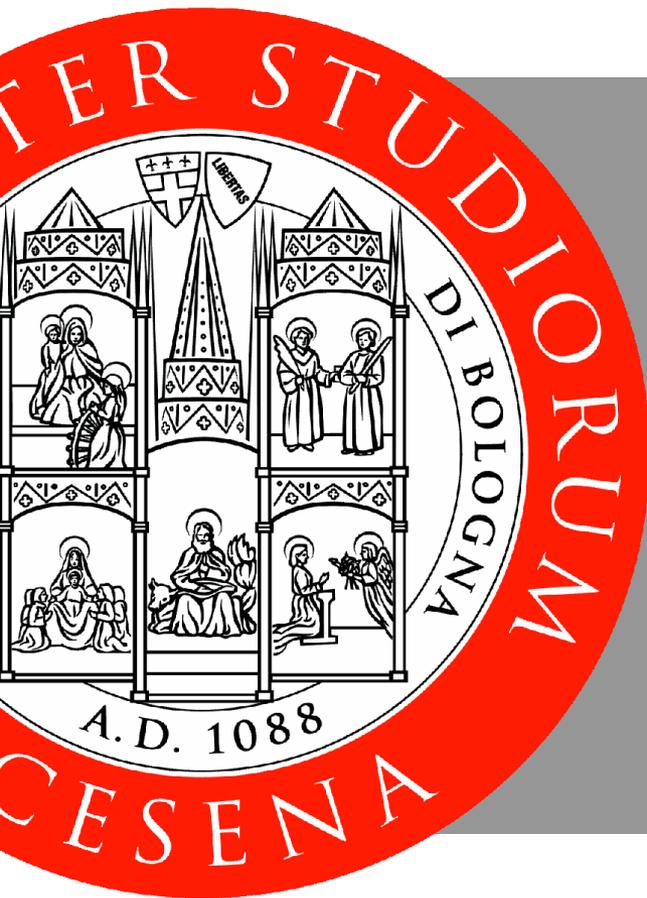
- Input: A vale '9, a, b, 2, ac, 5, a, 4, 4a, 5b'
- Determinare i valori di output.



# Riferimenti

- ◆ <http://www.olimpiadiproblemsolving.it>
- ◆ [info@olimpiadiproblemsolving.it](mailto:info@olimpiadiproblemsolving.it)
- ◆ <https://www.facebook.com/olimpiadiPS>
- ◆ [https://twitter.com/Oli\\_ProbSolving](https://twitter.com/Oli_ProbSolving)





# Olimpiadi **P**roblem **S**olving

Antonella Carbonaro

[antonella.carbonaro@unibo.it](mailto:antonella.carbonaro@unibo.it)

Dipartimento di Informatica - Scienza e Ingegneria  
Alma Mater Studiorum, Università di Bologna

Cervia – 27 aprile 2017