Olimpiadi Problem Solving

Antonella Carbonaro
antonella.carbonaro@unibo.it

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

Torino – 3 Febbraio 2020

Olimpiadi di Problem Solving

Informatica e pensiero algoritmico 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 algoritmico nella scuola dell'obbligo

Olimpiadi Problem Solving – **PERCHE'**
Olimpiadi Problem Solving – **COME**
Olimpiadi Problem Solving – **COSA**

Olimpiadi Problem Solving
PERCHE'

Antonella Carbonaro
antonella.carbonaro@unibo.it

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

Torino – 3 Febbraio 2020

Problem Solving

- Ubiquitous computing
- Dimensioni ridottissime, con enorme potenza di calcolo
- Enorme disponibilità di informazioni
- … e le conoscenza 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 PS 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, praticamene 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.**

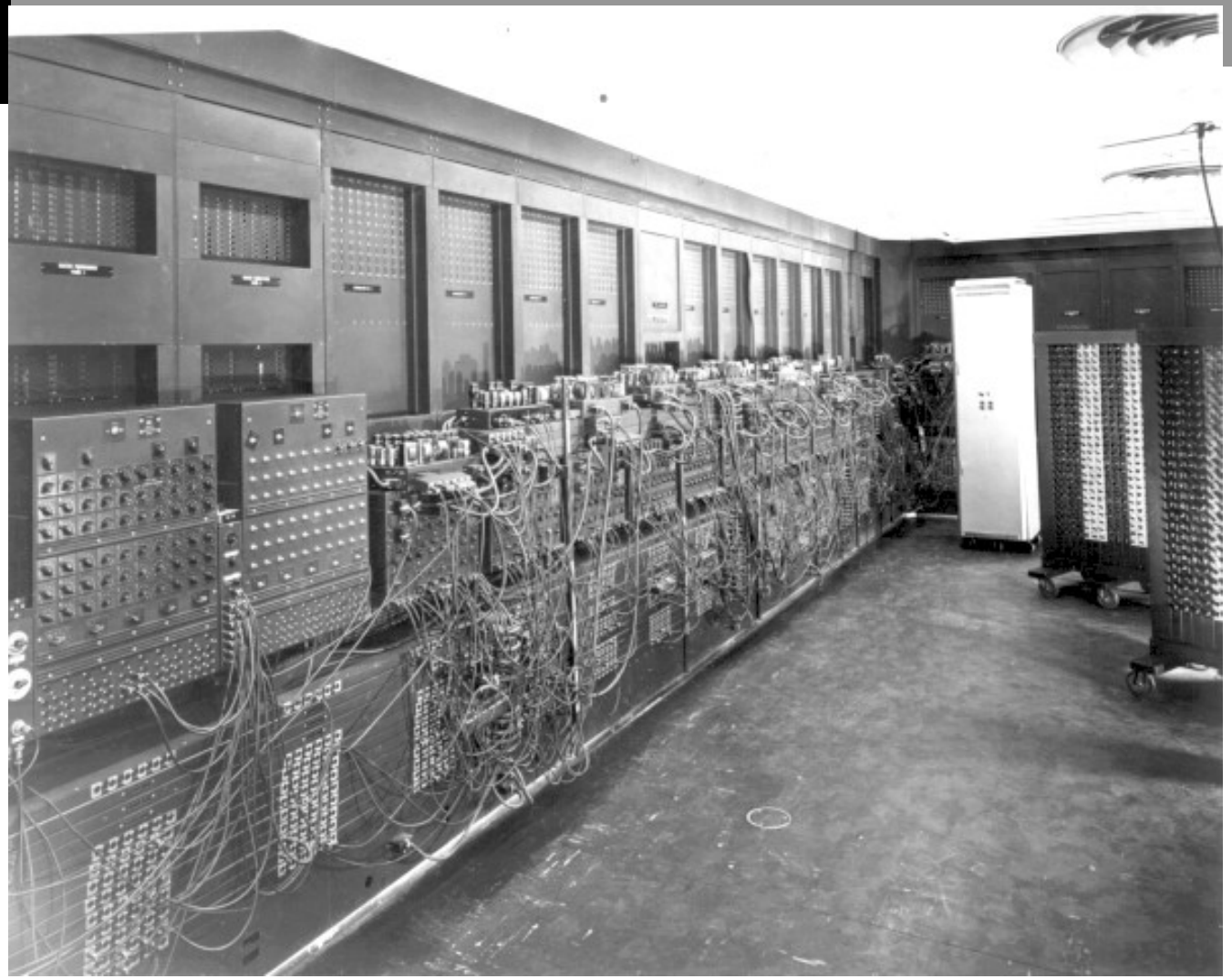
Nel 1945 esce, così l'**Edvac** (Electronic Discrete Variables Automatic Computer) è la **prima macchina digitale programmabile tramite un software**: è nata "**l'architettura di von Neumann**".

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

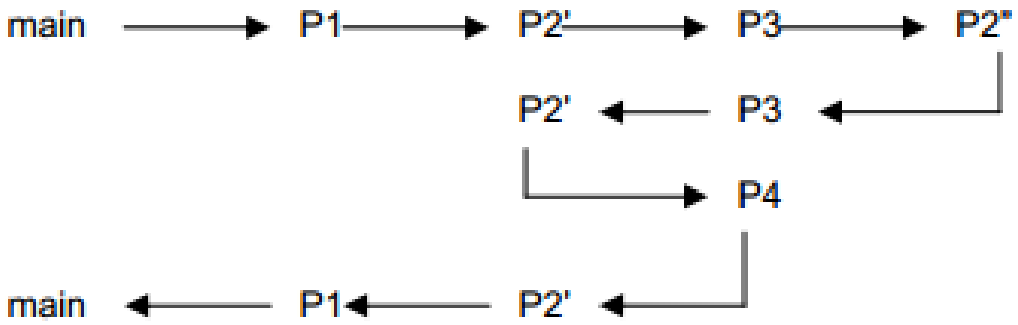
Verificare che le parentesi tonde, quadre e graffe usate all'interno di una espressione (matematica) siano correttamente annidate

Strumenti concettuali

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

Problemi diversi?

Chiamate a funzione

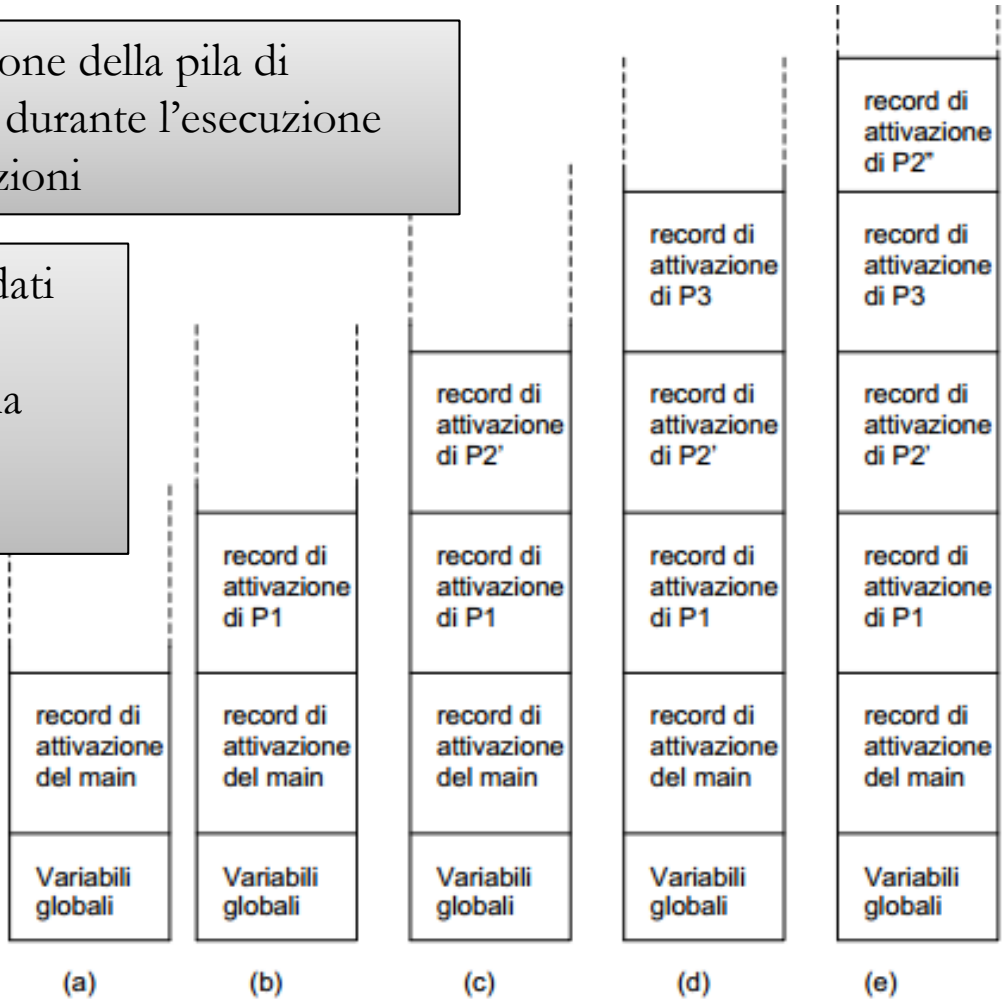


una freccia verso destra indica una chiamata di funzione;
una verso sinistra indica il ritorno alla funzione chiamante.
Le diverse attivazioni sono marcate da apici.

Chiamate a funzione

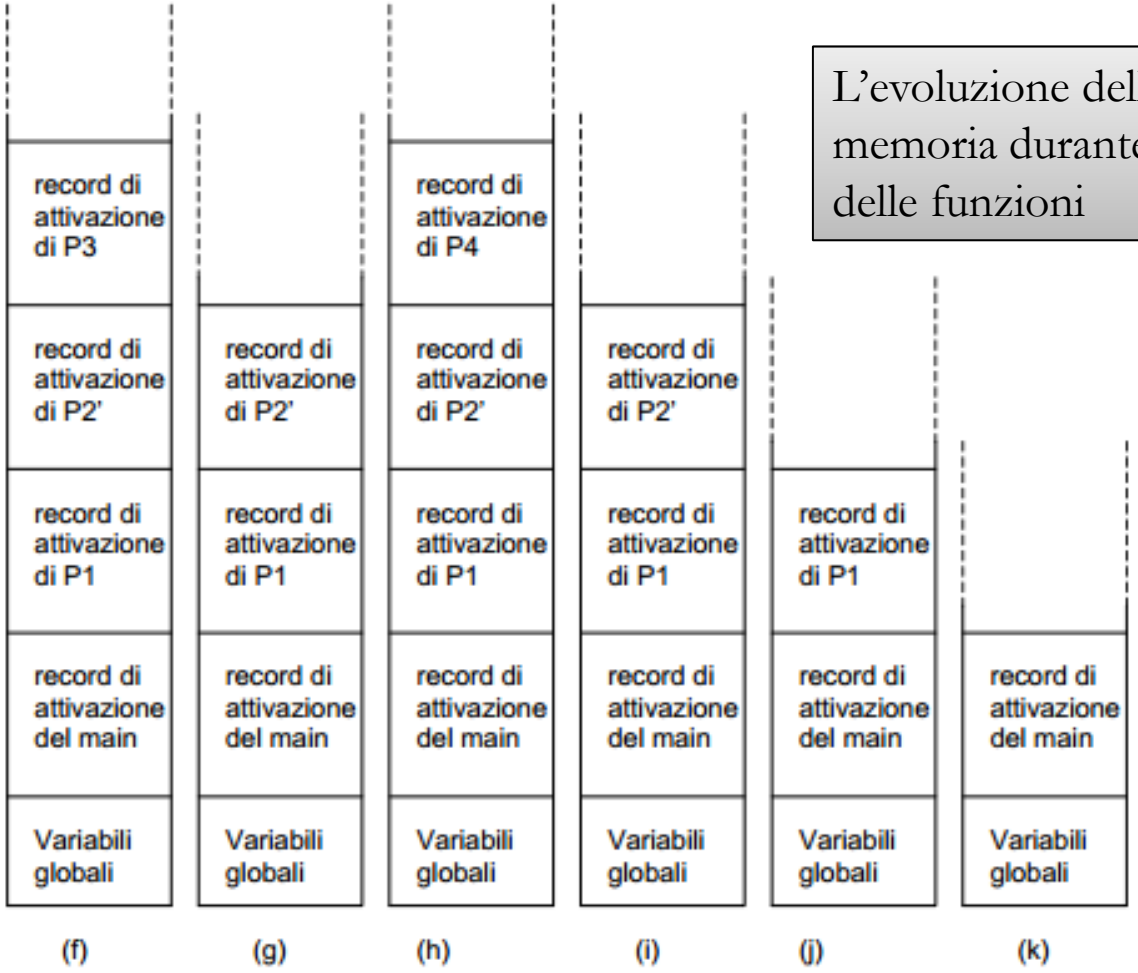
L'evoluzione della pila di memoria durante l'esecuzione delle funzioni

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



Chiamate a funzione

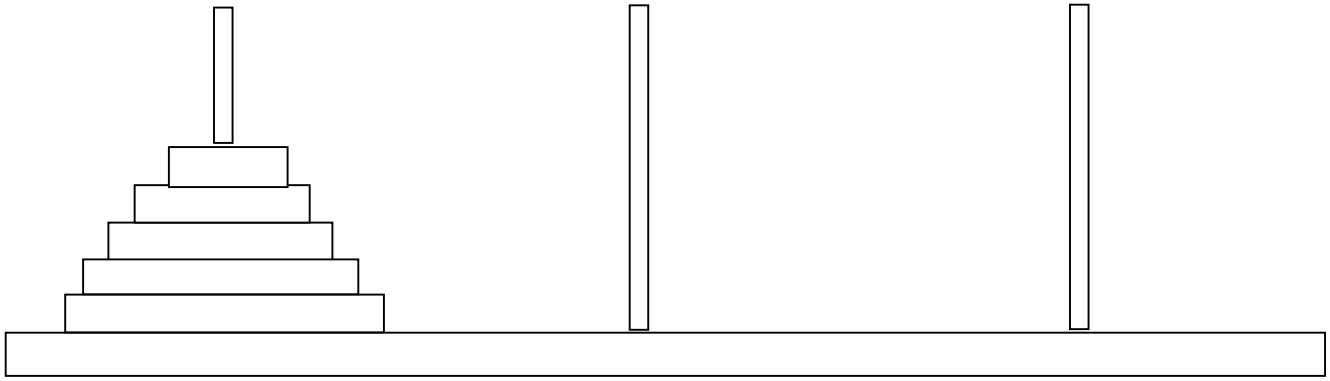
L'evoluzione della pila di memoria durante l'esecuzione delle funzioni



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:

P(n-1)

- 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)

OPS

Le Olimpiadi di Problem Solving offrono una occasione esplicita e operativa per fare **esperienze** di pensiero computazionale.

Quindi le OPS sono un mezzo per imparare questi linguaggi, utili non solo all'informatico professionista ma a chiunque debba cimentarsi nel trovare la soluzione a problemi.

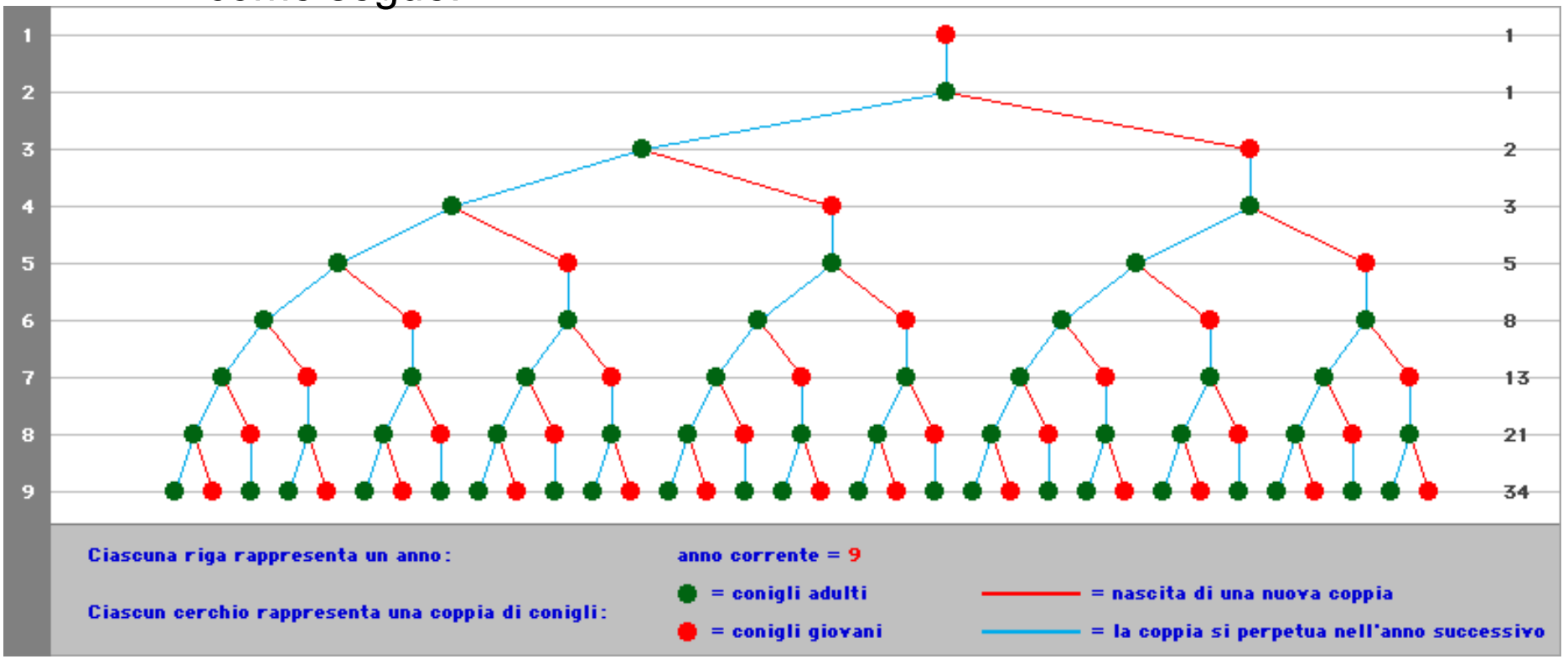
Strumenti concettuali

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**

L'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

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

Correttezza?

Qual è l'accuratezza su 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)* $\rightarrow$ *intero*

if (n≤2) **then return** 1

else return `fibonacci2`(n-1) +
`fibonacci2`(n-2)

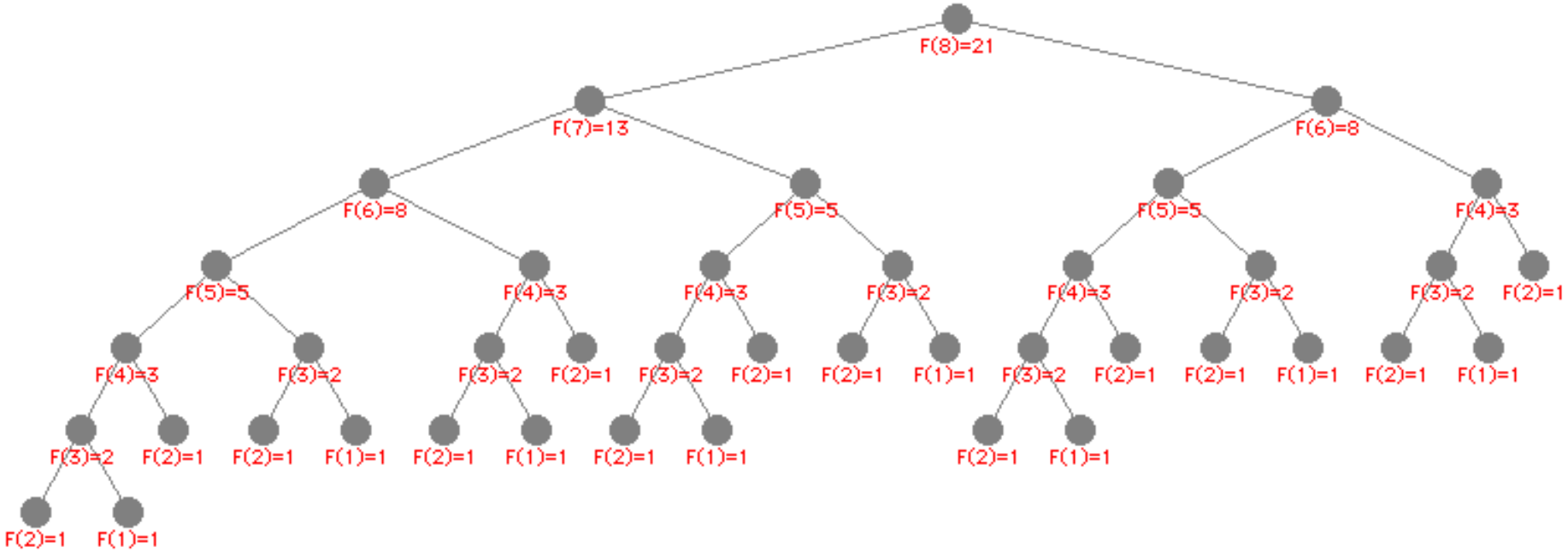
Opera solo con numeri interi.

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!!!*

	`fibonacci2`(58)	`fibonacci3`(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 Fn 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. $\quad M \leftarrow M \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
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 $\sqrt{n}$	 ≈ 3000 anni ≈ 10 secondi

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

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

$$2^n.$$

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

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

Piú che abbondante quindi per andare da Roma a NYC. ▽

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

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

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

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 Problem Solving COME

Antonella Carbonaro
antonella.carbonaro@unibo.it

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

Torino – 3 Febbraio 2020

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

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

Pianicazione, divisione dei compiti, organizzazione:

sono gia 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 27 febbraio 2020**, le squadre e gli studenti che partecipano alla fase regionale

Gare di istituto - calendario

GARA 1

4 novembre: a squadre secondaria di I grado
5 novembre: a squadre secondaria di II grado
6 novembre: a squadre primaria
7 novembre: individuale secondaria di II grado
8 novembre: individuale secondaria di I grado

GARA 2

2 dicembre: a squadre primaria
3 dicembre: a squadre secondaria di I grado,
4 dicembre: a squadre secondaria di II grado
5 dicembre: individuale secondaria di I grado
6 dicembre: individuale secondaria di II grado

GARA 3

13 gennaio: a squadre secondaria di II grado
14 gennaio: a squadre primaria
15 gennaio: a squadre secondaria di I grado
16 gennaio: individuali secondaria di II grado
17 gennaio: individuali secondaria di I grado

GARA 4

10 febbraio: a squadre primaria
11 febbraio: a squadre secondaria di I grado,
12 febbraio: a squadre secondaria di II grado
13 febbraio: individuale secondaria di I grado
14 febbraio: individuale secondaria di II grado

Gare regionali

- Presso **scuole-polo** provinciali e/o regionali che saranno individuate e segnalate sul sito entro gennaio 2020.
- 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

GARA 5 (regionale)

18 marzo: primaria e secondaria di I grado

19 marzo: secondaria di II grado

Finalissima nazionale

- A squadre
  - per ciascun livello scolastico, la migliore squadra classificata nella selezione regionale, purché con punteggio superiore alla media nazionale.
- Individuale
  - per i due livelli previsti, il primo classificato di ogni regione, purché con punteggio superiore alla media nazionale.

- Nel caso di ex-aequo, verrà scelta la squadra e/o alunno più giovane.
- 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

Finalissima - calendario

Scuola Secondaria di II grado:

- GARA 6 (finale)
  17 aprile: finale 1
  Segue la premiazione.

Scuola Primaria e Scuola Secondaria di I grado:

- GARA 6 (finale)
  18 aprile: finale 2
  Segue la premiazione.

Qualche numero a.s. 18/19

- 658 scuole
- 4480 squadre (17920 studenti)
- 4024 studenti individuale
- 64 progetti di coding/makers/progettazione
- TOTALE: 22200 studenti coinvolti
- 5 gare

Qualche numero a.s. 18/19

- Abruzzo 21
- Basilicata 15
- Calabria 125
- Campania 50
- Emilia Romagna 30
- Friuli Venezia Giulia 16
- Lazio 71
- Liguria 16
- Lombardia 32
- Marche 10
- Molise 17
- Piemonte 39
- Puglia 71
- Sardegna 8
- Sicilia 55
- Toscana 26
- Trentino Alto Adige (TN) 12
- Umbria 13
- Veneto 30
- Trentino Alto Adige (BZ) 1

Qualche numero a.s. 18/19

Anno scolastico	Gare	Insegnanti coinvolti	studenti coinvolti
2008/2009	3	550	10200
2009/2010	5	500	9850
2010/2011	6	400	7400
2011/2012	6	630	11400
2012/2013	4	880	16500
2013/2014	6	400	15100
2014/2015	7	500	12000
2015/2016	6	830	16700
2016/2017	6	850	23375
2017/2018	6	730	22225
2018/2019	5	720	22200

Prove

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

Gara a squadre

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

- circa sei problemi formulati in italiano e scelti, di volta in volta, tra l'insieme dei "**Problemi ricorrenti**" (si veda il successivo elenco);
- Circa due o tre problemi formulati in italiano e relativi a uno **pseudo-linguaggio** di programmazione;
- Circa un problema di comprensione di un testo in lingua **italiana**;
- Circa un problema, 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:

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

Durata gare

Le prove di istituto hanno la durata di 120 minuti e consistono nella risoluzione di 13 problemi per la gara a squadre e di 8 problemi per la gara individuale, scelti dal Comitato tecnico-scientifico.

Le prove regionali e la finalissima hanno la durata di 90 minuti

Olimpiadi Problem Solving
COSA

Antonella Carbonaro
antonella.carbonaro@unibo.it

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

Torino – 3 Febbraio 2020

Coding@OPS

- 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 (sp, ss I grado).
- 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@OPS

- 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@OPS

- La scadenza per inviare i lavori è fissata per il 15 febbraio 2020.
- Entro il 15 marzo 2020 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 4 alunni e 1 docente

Temi proposti per l'edizione 2019/2020:

- https://olimpiadiproblemsolving.it/web/coding_20192020.php

Programmazione@OPS

- Le OPS includono un'ulteriore modalità di partecipazione e competizione che riguarda la programmazione.
- Scuola secondaria di secondo grado (solo primo biennio)
- Per partecipare a questa gara le scuole dovranno inviare entro il 15/02/2020 tre programmi scritti da una squadra della scuola per risolvere tre problemi scelti dall'elenco sopra riportato. Il CTS delle OPS valuterà i primi 40 lavori pervenuti entro tale data, e le scuole che avranno inviato le migliori dieci proposte saranno invitate a partecipare alla finale inviando una squadra formata da 4 studenti e un docente.
- La sottomissione dei programmi dovrà avvenire usando l'apposita pagina del sito delle OPS

Programmazione@OPS

- La gara di programmazione a squadre delle OPS consiste nella risoluzione di un problema che deve essere risolto scrivendo un programma indifferentemente in uno dei seguenti linguaggi: C, Pascal, Python, C++. La prova ha la durata di 90 minuti e il programma deve essere implementato sui computer messi a disposizione dei finalisti nella sede di gara. Il problema sarà una variante dei seguenti problemi assegnati nelle tradizionali gare di Istituto di OPS.
- Regole e deduzioni.
- Grafi.
- *Knapsack*.
- Pianificazione.
- Crittografia.
- Programmazione dei movimenti di un robot.
- Sottosequenze.

Makers@OPS

- Le OPS includono un'ulteriore modalità di partecipazione e competizione che riguarda l'Ideazione e lìimplementazione di un progetto/prototipo basato su piattaforma Arduino compatibile a partire da un tema assegnato.
- Il tema per questa edizione è: **Il diorama 3.0**
- A differenza di un semplice diorama, il diorama 3.0 dovrà includere uno o più effetti di vario genere (movimenti, luci, suoni). I progetti realizzati potranno riguardare qualsiasi tema, reale o di fantasia, presente, passato o futuro.
- La partecipazione in questo caso è unicamente a squadre, senza più il vincolo dei 4 componenti.
- La partecipazione avviene inviando un pdf descrittivo ed un video (tramite link) del progetto mediante il sito delle OPS accedendo all'area riservata alle scuole e utilizzando l'apposita voce di menu.

Makers@OPS

- I progetti inviati verranno valutati dal Comitato Scientifico Tecnico delle OPS secondo i seguenti criteri:
  - originalità dell'idea
  - qualità del lavoro in generale
  - qualità, pulizia e originalità del codice, dell'implementazione
  - correttezza e completezza
  - basso costo della soluzione proposta (invito al riutilizzo)

Makers@OPS

- Entro il 15 marzo 2019, verranno selezionati i migliori progetti pervenuti per ogni categoria di scuola; questi progetti verranno presentati durante il workshop finale a Cesena.
- Durante il workshop il comitato tecnico esaminerà i progetti, ponendo ai componenti delle squadre domande relative alla parte fisica, elettronica ed al codice, commisurate all'ordine ed al grado della scuola. Potranno altresì essere richieste piccole modifiche al codice al fine di variare il comportamento del sistema.
- Oltre ai criteri precedentemente elencati, relativi alla fase preselettiva, l'esaustività delle risposte ed il soddisfacimento delle modifiche richieste saranno determinanti nell'attribuzione dei punteggi che porteranno alla selezione della squadre vincitrici.

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.

	minerale	
sigla	**peso in chili**	**valore in euro**
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)

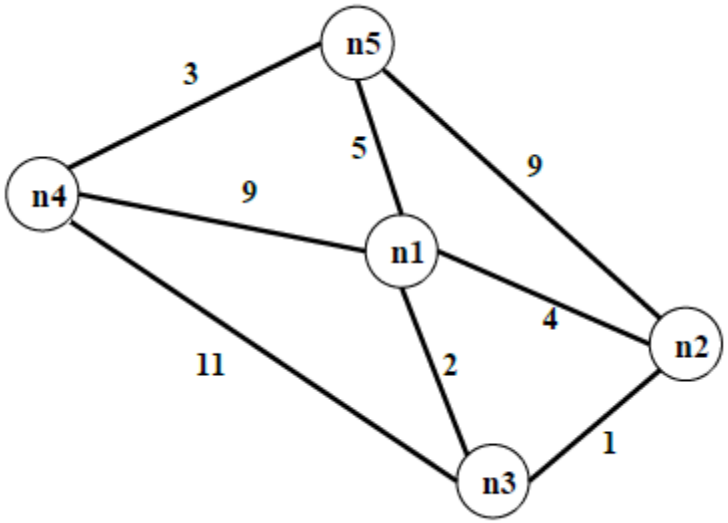
  minerale(m3,38,230)

Grafo

- Anche un grafo può essere descritto con dei termini; ogni termine è associato a un arco ed è così *definito*:

  arco(<nodo1>,<nodo2>,<lunghezza>)

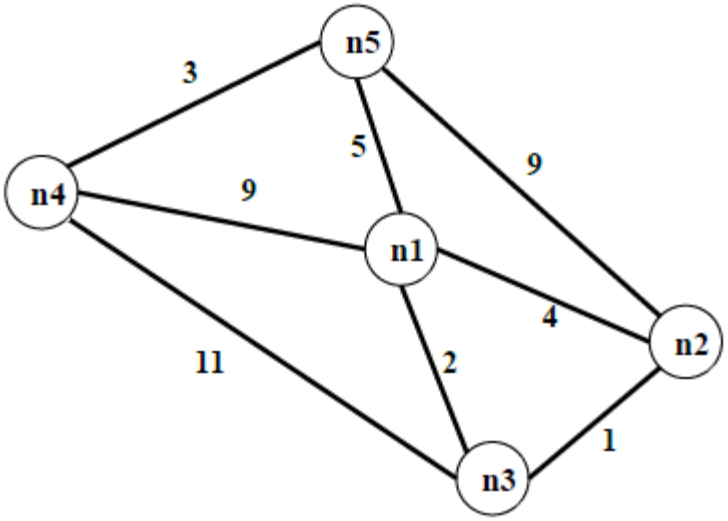
- La definizione mostra il nome del termine e il numero degli argomenti (tre): i primi due si riferiscono ai nodi posti agli estremi dell'arco, il terzo descrive la lunghezza dell'arco.



Grafo

- 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:

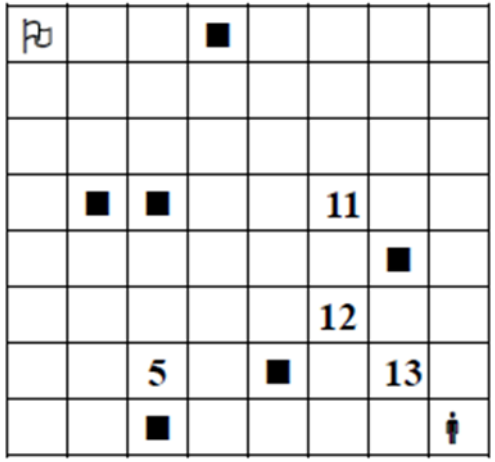
  [a1,2,56,b2,3,67]

- 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

- 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]]



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:

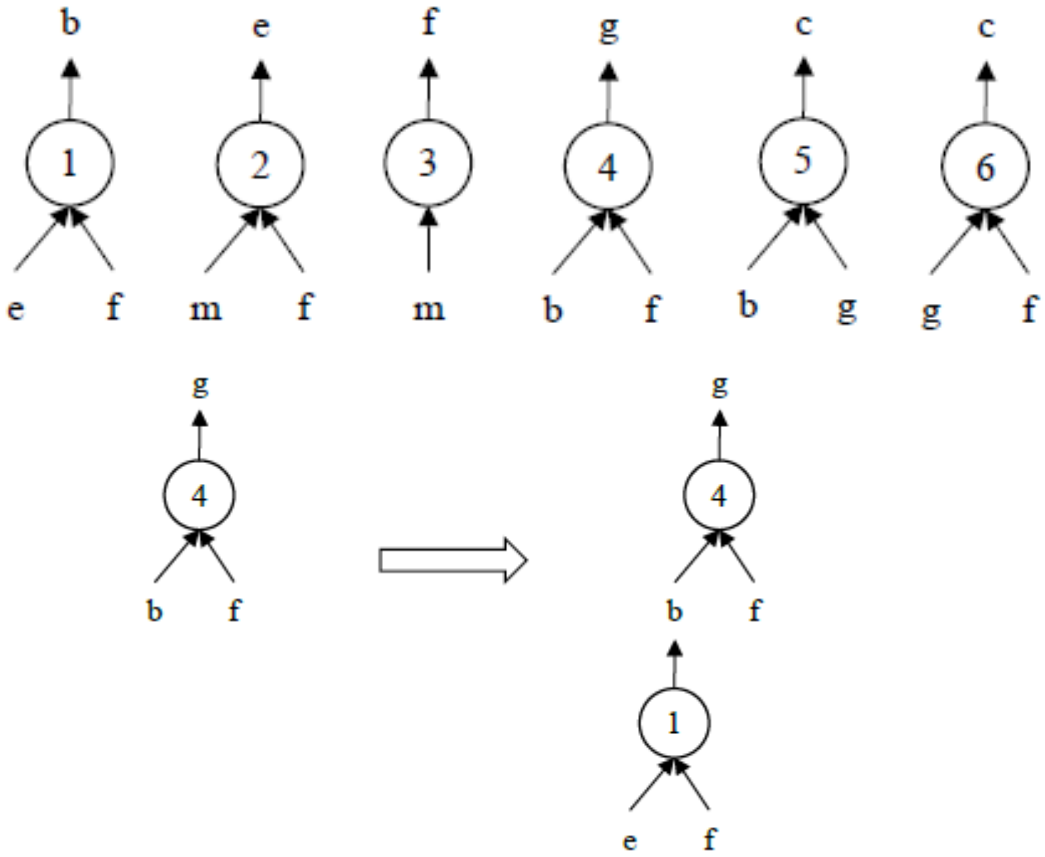
  regola(1,[e,f],b) regola(2,[m,f],e) regola(3,[m],f)

  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

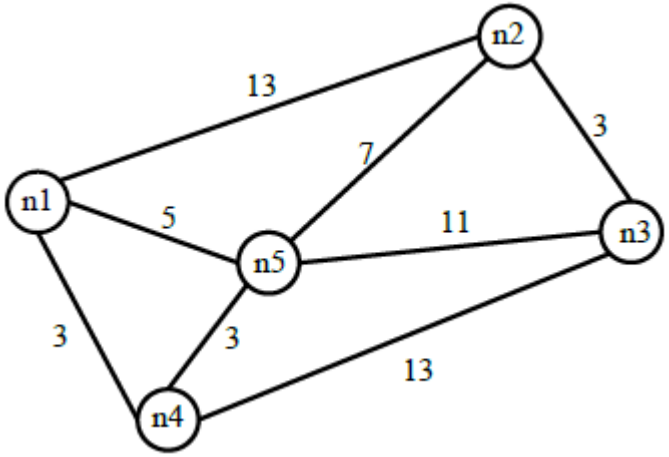
- Un grafo (che corrisponde alla rete di strade che collegano delle città) è descritto dal seguente elenco di archi:

  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.

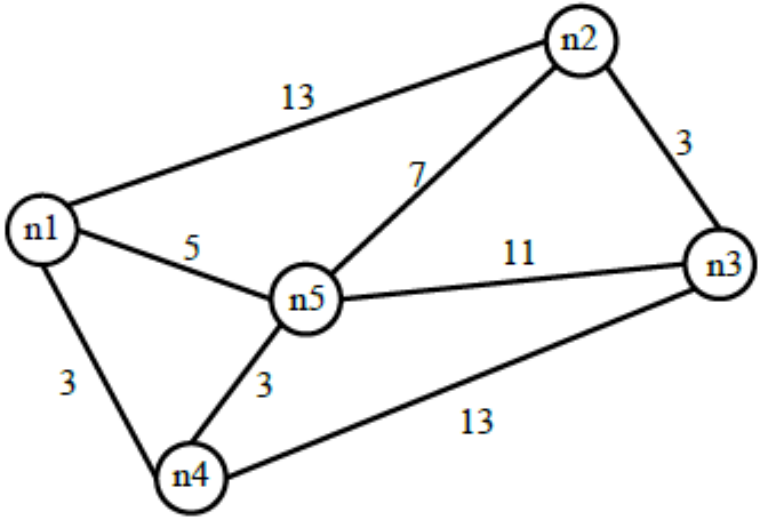
Grafi - problema



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

Grafi - problema

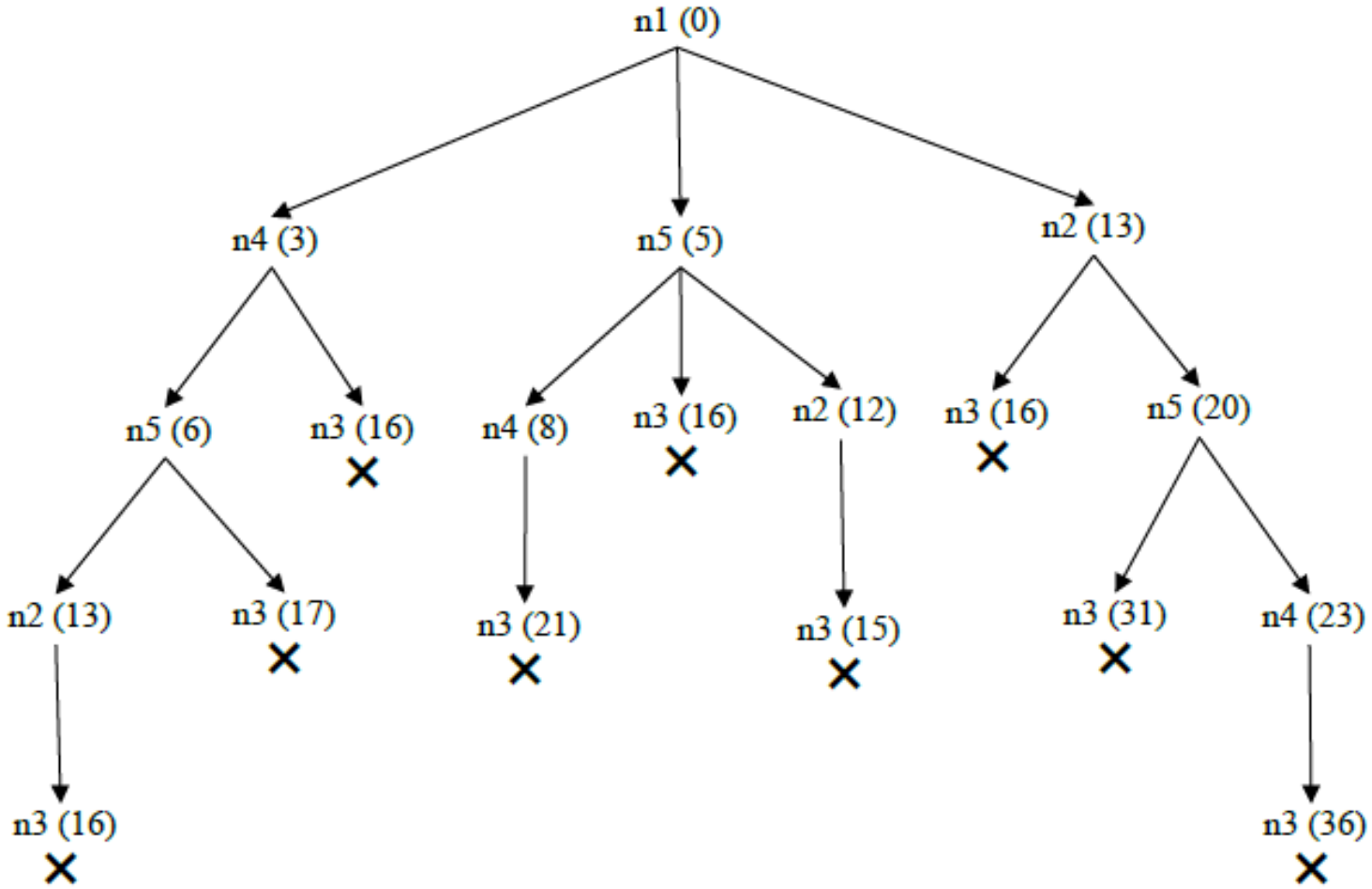
- PERCORSO da n1 a n3 LUNGHEZZA
  - [n1, n2, n3] 16
  - [n1, n2, n5, n3] 31
  - [n1, n2, n5, n4, n3] 36
  - [n1, n5, n3] 16
  - [n1, n5, n2, n3] 15
  - [n1, n5, n4, n3] 21
  - [n1, n4, n3] 16
  - [n1, n4, n5, n3] 17
  - [n1, n4, n5, n2, n3] 16



Grafi - problema

- Si noti che, a partire dal nodo n1, si “visitano” i tre nodi adiacenti (n2, n5, n4); a partire da ciascuno di questi si costruiscono tutti i cammini che arrivano a n3 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 (n1) 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 (n3) (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
procedure ESEMPIO1;	inizio della procedura di nome ESEMPIO
variables A, B, C1, D integer;	si dichiara che si usano 4 variabili che assumono valori interi
input A, B;	si acquisiscono dall'esterno valori per le variabili A e B
C1 ← A+B;	la variabile C1 acquisisce valore (di una espressione)
D ← C1+B-A;	la variabile D acquisisce valore (di una espressione)
A ← C1+D;	la variabile A acquisisce (un nuovo) valore (di una espressione)
output A, C1, D;	si rendono disponibile all'esterno i valori delle variabili A, C1, D
endprocedure;	fine della procedura

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
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;	inizio del ciclo (che è ripetuto 4 *volte* 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

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
- https://www.facebook.com/olimpiadiPS
- https://twitter.com/Oli_ProbSolving

Olimpiadi Problem Solving

Antonella Carbonaro
antonella.carbonaro@unibo.it

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

Torino – 3 Febbraio 2020