36º Convegno sulla Didattica della Matematica

Lucca, 9 Settembre 2019

Fabrizio Luccio. Algoritmi e Coding

La parola Algoritmo

Storicamente (sec. XIII): Algorismus è un procedimento di calcolo aritmetico (dal nome del matematico persiano del IX sec. Muḥammad ibn Mūsā al-Khuwārizmī)

Nel sec. XX: correlate al "Problema della decisione" di Hilbert (1928) nascono diverse definizioni formali di algoritmo, fino alla "Macchina di Turing" del 1936

In linguaggio corrente (e ineccepibile dal punto di vista formale) un algoritmo è una sequenza ordinata e finita di passi elementari per risolvere un problema, mediante azioni tratte da un insieme finito

La parola Code (inglese)

Dizionario di Oxford (UK): A system of words, letters, figures, or symbols used to represent others

Dizionario Merriman-Webster (USA): A system of signals or symbols for communication

In informatica: Coding, atto di trasformare un algoritmo secondo un code definito per una macchina. Può essere sostituito con la parola programmare

Ricerca dell'informazione associata a una chiave in un elenco di lunghezza arbitraria n.

La struttura dati naturale è composta di due vettori C e I per contenere chiavi e informazioni: alla chiave C[j] corrisponde l'informazione I[j], con $1 \leq j \leq n$.

La chiave di ricerca (input) e l'informazione trovata (output) sono contenuti in due variabili.

Immaginiamo che C e I contengano nomi di persone e i rispettivi numeri di telefono. Indichiamo con NOME e TEL le variabili di input/output.

Se NOME = C[j] porremo TEL = I[j]

Se NOME non appare in C porremo TEL = 0

1. L'elenco non ha alcuna strutturazione particolare

j	1	2	3	.....	n
C	Rossi	Bacci	Gotti		Carli
I	35822	42231	41061	.....	72909

2. L'elenco ha già una struttura coerente con l'operazione di ricerca da eseguire: per esempio i nomi vi appaiono in ordine alfabetico

j	1	2	3	.....	n
C	Bacci	Carli	Gotti	.....	Rossi
I	42231	72909	41061	.....	35822

Se i nomi appaiono in ordine qualsiasi un algoritmo ovvio è scandire l'intero vettore C confrontando ogni elemento con NOME.

La correttezza dell'algoritmo è evidente.

La sua complessità, ovvero il numero di passi che compie, è di ordine O(n) nel caso pessimo.

Non esiste algoritmo più veloce.

Se l'elenco dei nomi è ordinato possiamo adottare, anche per una macchina, l'algoritmo manuale di ricerca in una rubrica.

A questo scopo sono necessarie alcune precisazioni.

I nomi nella rubrica sono sequenze di caratteri e, nel caso manuale, sono disposte in ordine alfabetico. Nel calcolatore tali sequenze sono rappresentate in binario e possono essere trattate come numeri binari, dunque è legittimo impiegare la relazione aritmetica ≤ nel confronto.

In una rubrica ordinata gli accessi manuali sono eseguiti in modo euristico. Nel calcolatore possiamo calcolare in tempo costante un valore dell'indice j (per esempio il valore centrale tra 1 e n) e accedere in tempo costante a C[j].

L'algoritmo, noto come RICERCA-BINARIA, è *ricorsivo* e consiste nei passi seguenti:

- calcolare il valore centrale m dell'indice j
- confrontare NOME con C[j]: se coincidono, estrarre l'informazione I[j], altrimenti:

  se NOME < C[m] ripetere il punto precedente sul sottovettore di sinistra C[1] . . . C[m-1]

  se NOME > C[m] ripetere il punto precedente sul sottovettore di destra C[m+1] . . . C[n]

  se un sottovettore è vuoto, NOME non appare in C e l'algoritmo si arresta

Anche in questo caso la correttezza dell'algoritmo è evidente.

Quanto al tempo di calcolo, il caso pessimo è quando l'elemento NOME è in ultima posizione nella ricerca o non è contenuto in *C*:

> questo richiede tante operazioni di scansione e relativi confronti, per quanti sono i passi necessari a ridurre a un singolo elemento il sottovettore su cui si esegue via via la ricerca. Tali passi sono $\approx \log_2 n$.

La complessità in ordine di grandezza è $O(\log n)$

```
programma RIC-BIN(input NOME, output TEL)
    j ← 1; k ← n;
    while j ≤ k
        m ← inf{(j+k)/2};
              // inf{x} parte intera di x
        if NOME = C[m]
              TEL ← I[m];
              stop;
        if NOME < C[m] k ← m-1;
        if NOME > C[m] j ← m+1;
    TEL ← 0;
```

Un caso più sottile

$a^n + b^n = c^n$

non ha soluzione intera positiva per $n > 2$

è una famosa affermazione di Fermat dimostrata da Andrew Wiles nel 1995 in più di 130 pagine (ma la sua prima dimostrazione era sbagliata . . .)

Problema: si potrebbe dimostrare che l'affermazione è vera mediante un algoritmo ?

Costruiamo un algoritmo che prova tutti i valori possibili per a, b, c, n, si arresta se trova una quaterna per cui $a^n + b^n = c^n$, e stampa i valori a, b, c, n

questo ci permetterebbe di dimostrare che l'affermazione di Fermat è falsa e anche la nuova dimostrazione di Wiles è sbagliata

FERMAT

for (*i from 6 to* ∞, *i++*)

costruisci le quaterne $q_i = \{n,a,b,c\}$

con $n>2$, $a,b,c>0$, $n+a+b+c=i$

per ogni q_i:

if ($a^n+b^n=c^n$) *stampa* q_i *e termina*

FERMAT termina se e solo se l'affermazione di Fermat è falsa per una certa quaterna

Se l'affermazione di Fermat è vera l'algoritmo FERMAT non termina (dunque non può essere utilizzato per la dimostrazione).

Ma una dimostrazione potrebbe consistere nel formulare FERMAT e costruire un altro algoritmo che decida se FERMAT termina o no.

Nel suo famoso articolo del 1936 Turing dimostrò che costruire un simile algoritmo non è possibile !

La conseguenza, legata al programma di Hilbert, è che gli algoritmi non forniscono dimostrazioni gratis

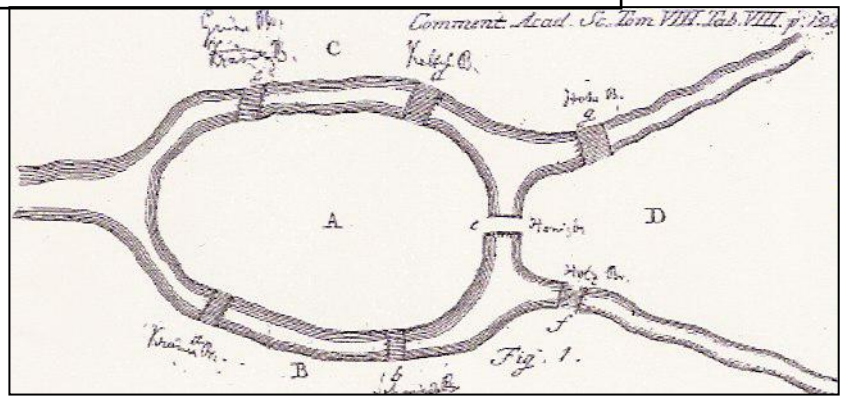
Problemi facili (o trattabili) e difficili (o intrattabili)

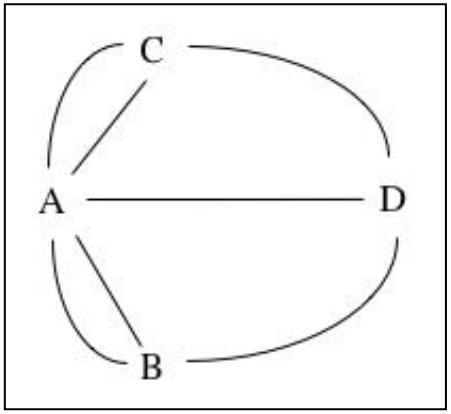
Nel mondo degli algoritmi questi termini non sono riferiti alla difficoltà incontrata nel risolvere un problema, ma al tempo di calcolo necessario per tale risoluzione, ovvero alla sua complessità di calcolo

Il tempo non si misura in secondi ma in numero di operazioni elementari necessarie, perché la valutazione sia indipendente dal mezzo di calcolo impiegato. Nei problemi facili o difficili tale tempo (in ordine di grandezza) è funzione polinomiale o esponenziale nella dimensione dell'input

Un esempio famoso

Königsberg, 1735



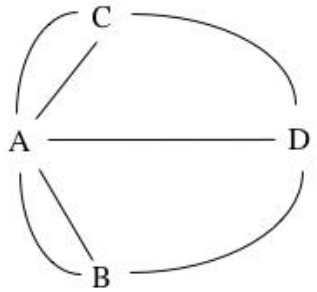


	A	B	C	D
A	0	2	2	1
B	2	0	0	1
C	2	0	0	1
D	1	1	1	0

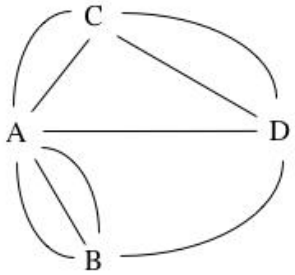
La condizione di Eulero:

Esiste un ciclo (detto Euleriano) che traversa tutti gli archi (ponti) esattamente una volta se e solo se tutti i nodi (zone della città) hanno grado pari.

Ovvero se è pari la somma su ogni riga della matrice.

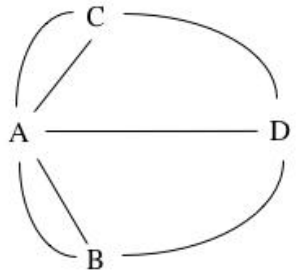


NESSUN CICLO

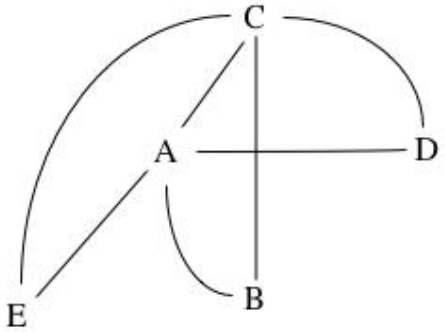


A C D B A C D A B A

Il ciclo Hamiltoniano traversa
tutti i nodi esattamente una volta



A B D C A



A B C D E ?
A B C E D ?
. . . . . .

In linea di principio, con *n* nodi si devono fare *n* ! prove

n ! cresce circa come $(n/e)^n$

Per n = 5 si ha 5! = 120. Per n = 10 si ha 10! > 3.5 M

Le due principali classi di complessità

P è la classe dei problemi che si risolvono in tempo polinomiale.

NP è la classe dei problemi che si verificano in tempo polinomiale (ma si sanno risolvere solo in tempo esponenziale).

P = NP ?

Per la risoluzione di questo problema è in palio un premio di 1 M $

Tre problemi tra i "più difficili" in NP

Biologia molecolare: dato un insieme arbitrario di frammenti di sequenze di DNA stabilire se esiste una sequenza incognita ma di lunghezza assegnata che contenga al suo interno ognuno dei frammenti dati.

Trasporti: stabilire se un insieme di casse di pesi diversi possono essere caricate su un assegnato numero di camion senza superare il peso massimo sopportabile da ciascun camion.

Algebra: stabilire se un'equazione algebrica di secondo grado in due variabili x, y, a coefficienti interi, ammette una soluzione in cui x e y hanno valori interi (se l'equazione è di primo grado il problema è in P).

Un campo di applicazione ove la distinzione tra problemi polinomiali e esponenziali è fondamentale è la crittografia, ove si utilizzano funzioni "one-way" che sono facili da calcolare e difficili da invertire.

Il calcolo diretto è richiesto per eseguire e interpretare una comunicazione cifrata, il calcolo dell'inverso è indispensabile per decifrare una comunicazione senza autorizzazione

Vediamo come si impiega questo concetto per consentire a due utenti A e B di costruire una chiave segreta condivisa, impiegando un algoritmo noto a tutti e ponendo che l'informazione tra i due possa essere completamente spiata.

Descriviamo il primo algoritmo proposto a questo scopo, noto come DH dalle iniziali degli inventori Whitfield Diffie e Martin Hellman (1976) e ancora largamente usato.

Ogni numero primo p ha un "generatore" g

cioè g^i mod p, per i = 1, 2,…, p-1, genera tutti i valori 1, 2,…, p-1 in ordine imprevedibile.

Per esempio g = 7 è un generatore di p = 11:

$7^1 = 7$ $7^2 = 5$ $7^3 = 2$ $7^4 = 3$ $7^5 = 10$
$7^6 = 4$ $7^7 = 6$ $7^8 = 9$ $7^9 = 8$ $7^{10} = 1$

ove i calcoli sono eseguiti mod 11

Scelti p e g

dato x calcolare $y = g^x \bmod p$ "è facile"

dato y calcolare x "è difficile"

perché si devono provare tutti i possibili valori di x tra 1 e p-1 fino a trovare quello per cui $g^x \bmod p = y$, e tali valori sono in numero esponenziale rispetto al numero di cifre di p

Per esempio calcolare $7^6 \bmod 11 = 4$ è facile, mentre trovare il valore 6 dell'esponente conoscendo il risultato 4 è difficile.

Poniamo che i valori p e g impiegati siano noti a tutti.

Nell'algoritmo DH i due utenti calcolano e si scambiano valori tipo $y = g^x \bmod p$, ove x è segreto e noto solo al mittente.
Un eventuale intruso non può ricostruire il valore di x dai valori di y (spiato) e g (noto).

Il protocollo DH per la creazione e lo scambio di una chiave segreta tra due utenti A, B

1. A e B si accordano su una coppia pubblica p, g e eseguono i calcoli mod p ;
2. A sceglie un intero casuale $x < p$, calcola $X = g^x$ e invia X a B ;
3. B sceglie un intero casuale $y < p$, calcola $Y = g^y$ e invia Y ad A ;
4. A calcola $K = Y^x = g^{yx}$; B calcola $K = X^y = g^{xy}$

K è la chiave segreta di A e B.