Informatica per le Biotecnologie
Algoritmica Lezione 1

Dixit Algorithmi

Nel XII secolo la studioso Adelardo di Bath pubblicava la traduzione in latino di alcuni libri scritti in arabo tre secoli prima da uno dei massimi matematici della storia:
Abu Ja‘far Muhammad Ibn Musa al-Khwarizmi,
noto come **al-Khwarizmi**, cioè proveniente dalla regione di Khwarizm in Asia centrale.

Uno di questi libri spiega come usare la numerazione indiana (posizionale con lo zero)

Adelardo apre la traduzione del libro con le parole:
Dixit Algorithmi

Tra i testi di al-Khwarizmi il più importante fu intitolato *Algebra,* dal metodo *al-jabr* (completamento) proposto da al-Khwarizmi per semplificare i termini di un'equazione. Il testo insegnava a risolvere le equazioni di primo e secondo grado e divenne la base della matematica europea che emergeva a piccoli passi dal medioevo.

Due esempi di algoritmo

L'algoritmo di Ahmes (1650 A.C.) per la moltiplicazione
$P = A \times B$ è basato sulle seguenti osservazioni:

se A è pari, $P = A/2 \times 2B$

se A è dispari e >1, $P = (A-1) \times B + B = (A-1)/2 \times 2B + B$

Il prodotto si costruisce iterando questi passi fino a che A diviene zero

uno spezzone di pseudo-codice

```
P = 0 ;
while A >= 1
    if (A è dispari) P = P + B ;
    A = ⌊A/2⌋ ;
    B = B x 2 ;
```

```
P    0
A   37
B   12
```

uno spezzone di pseudo-codice

```
P = 0 ;
while A >= 1
    if (A è dispari) P = P + B ;
    A = ⌊A/2⌋ ;
    B = B x 2 ;
```

P	0	12
A	37	18
B	12	24

uno spezzone di pseudo-codice

```
P = 0 ;
while A >= 1
    if (A è dispari) P = P + B ;
    A = ⌊A/2⌋ ;
    B = B x 2 ;
```

P	0	12	
A	37	18	9
B	12	24	48

uno spezzone di pseudo-codice

```
P = 0 ;
while A >= 1
    if (A è dispari) P = P + B ;
    A = ⌊A/2⌋ ;
    B = B x 2 ;
```

P	0	12		60
A	37	18	9	4
B	12	24	48	96

uno spezzone di pseudo-codice

```
P = 0 ;
while A >= 1
    if (A è dispari) P = P + B ;
    A = ⌊A/2⌋ ;
    B = B x 2 ;
```

P	0	12		60	
A	37	18	9	4	2
B	12	24	48	96	192

uno spezzone di pseudo-codice

```
P = 0 ;
while A >= 1
    if (A è dispari) P = P + B ;
    A = ⌊A/2⌋ ;
    B = B x 2 ;
```

P	0	12		60		
A	37	18	9	4	2	1
B	12	24	48	96	192	384

uno spezzone di pseudo-codice

```
P = 0 ;
while A >= 1
    if (A è dispari) P = P + B ;
    A = ⌊A/2⌋ ;
    B = B x 2 ;
```

P	0	12		60			444
A	37	18	9	4	2	1	0
B	12	24	48	96	192	384	768

Se i fattori A e B del prodotto (input) sono composti da n cifre il numero di operazioni elementari (tempo) richieste dall'algoritmo è proporzionale a n^2.

Poiché le cifre da esaminare sono 2n, ogni algoritmo di moltiplicazione deve richiedere un numero di operazioni elementari almeno proporzionale almeno a n (limiti inferiore al tempo).

Il problema delle dieci monete

Tra otto monete di identico aspetto una può essere falsa e quindi di peso diverso dalle altre: dobbiamo individuarla impiegando una bilancia a due piatti per il confronto tra gruppi di monete.

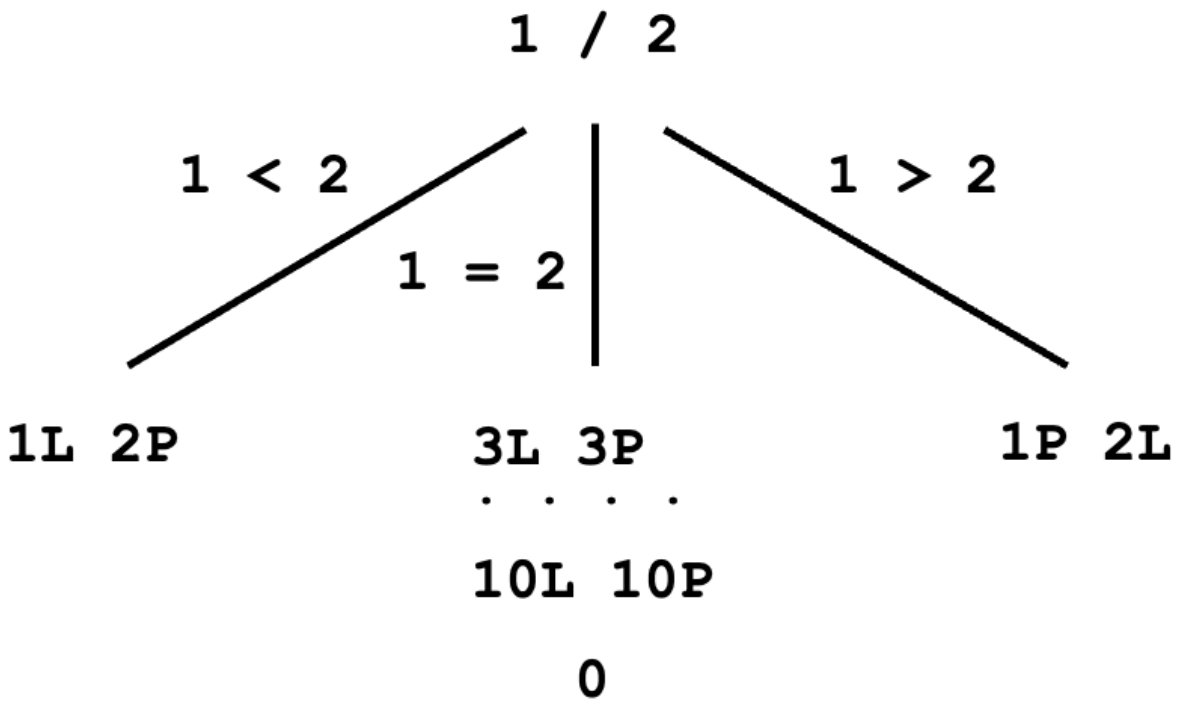
Come si deve procedere?
Quanti confronti occorrono?

Input: Le monete sono indicate con i numeri 1, 2, . . . , 10

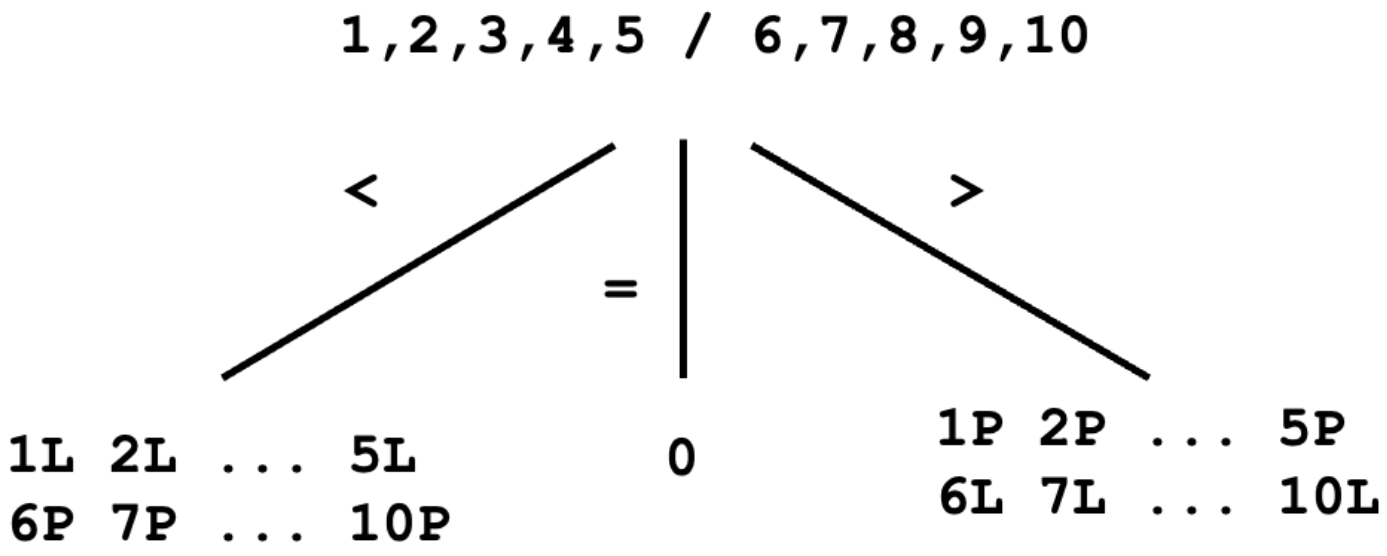
Output: I simboli xL, xP, indicano rispettivamente che la moneta x è falsa e più leggera o più pesante delle altre. Il simbolo 0 indica che la moneta falsa non c'è. Notiamo che le soluzioni possibili sono 21.

In questo caso il coding dell'algoritmo è un disegno in forma di *albero*

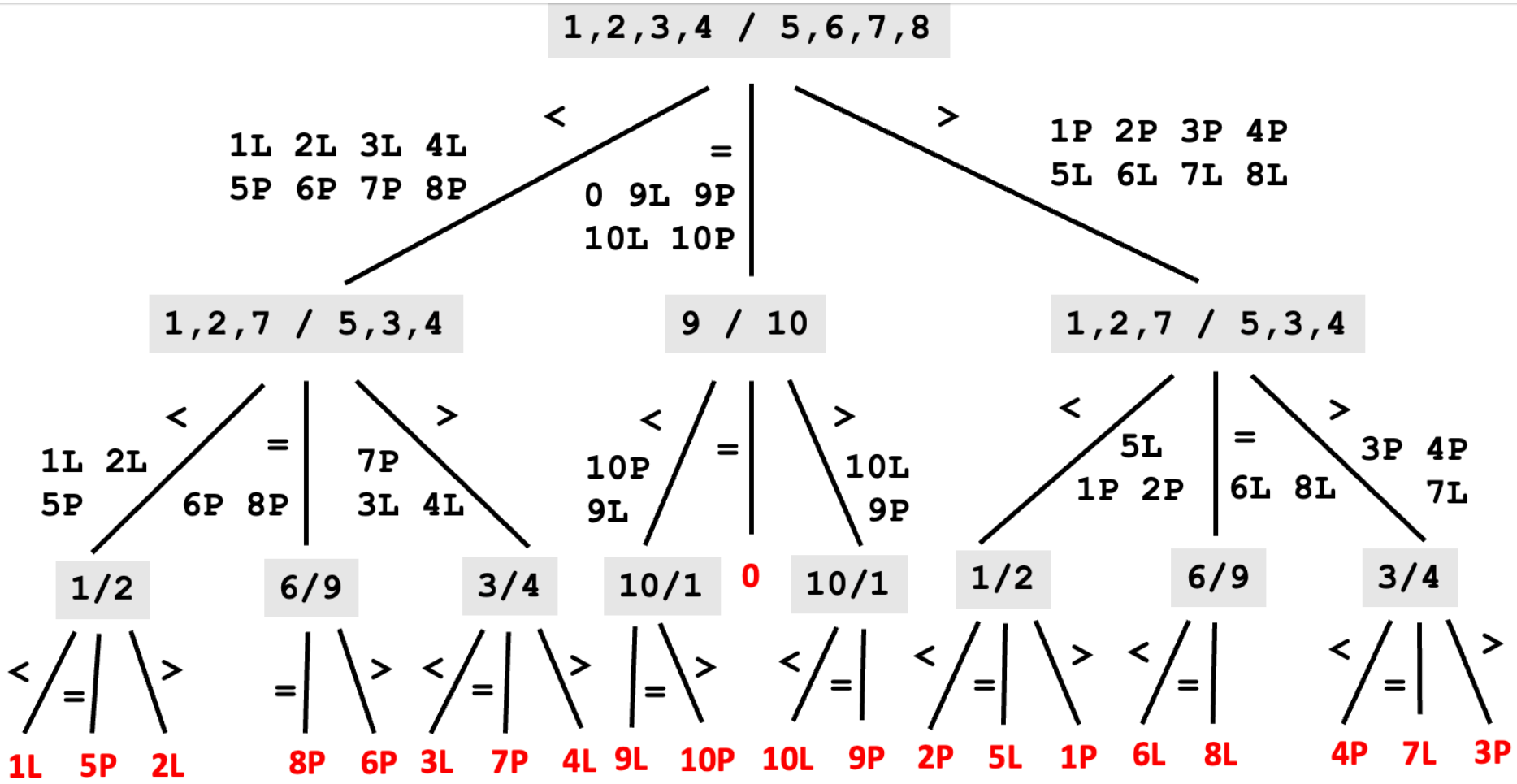
Rappresentazione delle operazioni:
confronto tra monete (per esempio 1/2)
risultato per esempio 1 < 2
informazione raccolta (1L, 2P)



E se iniziassimo con il confronto tra due gruppi di cinque monete?



Partendo con gruppi diversi di monete scopriamo che confrontando inizialmente due gruppi di quattro monete si bilancia il numero di soluzioni tra i tre rami dell'albero


1,2,3,4 / 5,6,7,8
<
1L 2L 3L 4L
5P 6P 7P 8P
=
0 9L 9P
10L 10P
>
1P 2P 3P 4P
5L 6L 7L 8L
1,2,7 / 5,3,4
9 / 10
1,2,7 / 5,3,4
<
1L 2L
5P
=
6P 8P
>
7P
3L 4L
<
10P
9L
=
>
10L
9P
<
5L
1P 2P
=
6L 8L
>
3P 4P
7L
1/2
6/9
3/4
10/1
0
10/1
1/2
6/9
3/4
1L 5P 2L
8P 6P 3L 7P 4L 9L 10P 10L 9P 2P 5L 1P 6L 8L 4P 7L 3P

Non si può risolvere il problema con meno di tre confronti.

Infatti le soluzioni sono 21 e ogni confronto genera al più tre casi corrispondenti a gruppi di soluzioni possibili in ognuno di essi.

Dopo un confronto si generano tre gruppi in cui il vi sono almeno 7 soluzioni. Dopo un secondo confronto le 7 soluzioni si dividono al meglio in 2, 2, 3 soluzioni. È dunque necessario un terzo confronto per generare gruppi di 1 soluzione.

Riassumendo:

La discussione su questi problemi introduce alcuni concetti fondamentali dell'informatica:

struttura dei dati (rappresentazione dei numeri o gruppi di monete)

algoritmo (la sequenza di operazioni numeriche o confronti)

coding (il linguaggio di programmazione o l'albero)

complessità in tempo (il numero di operazioni elementari – aritmetiche o confronti - eseguite per qualsiasi occorrenza iniziale dei dati: parliamo dunque di complessità nel caso pessimo)

limiti di complessità inferiore (almeno n operazioni o tre confronti sono necessari) e superiore (numero di operazioni o confronti eseguiti dall'algoritmo, nel secondo caso coincidente con il limite inferiore).

ALCUNE FORMULE MATEMATICHE

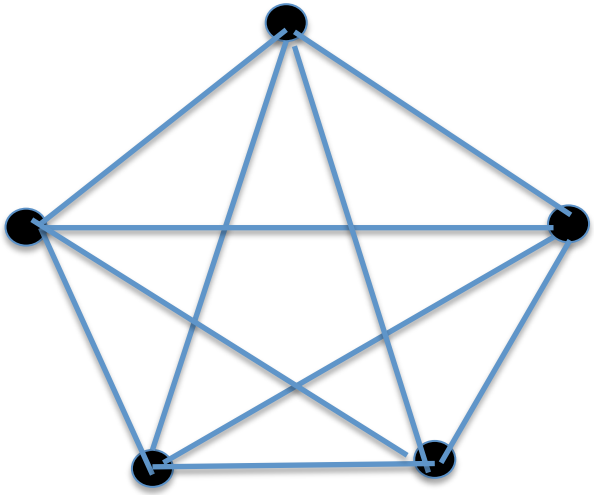
Somma dei primi *n* interi positivi

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \qquad 1+2+3+4+5+6 = \frac{6\times 7}{2} = 21$$

Numero di coppie tra gli *n* elementi di un insieme:
ogni elemento forma *n-1* coppie con gli altri, ma
l'incontro tra ogni coppia (A, B) va contato una sola volta

$$n(n-1) / 2$$

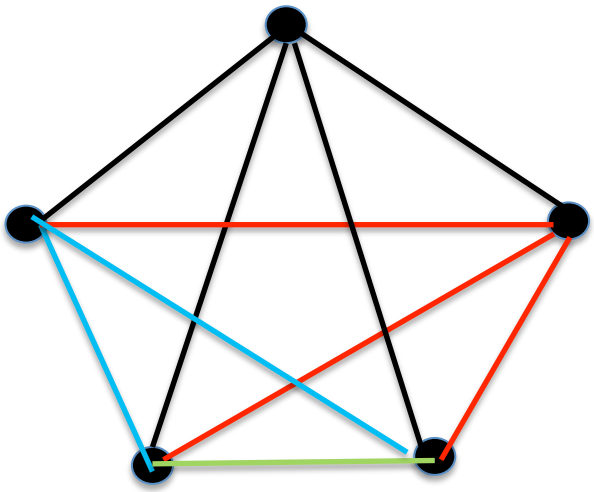
Oppure: numero di spigoli in un grafo completo di *n* vertici



Numero di coppie tra gli *n* elementi di un insieme:
ogni elemento forma *n-1* coppie con gli altri, ma
l'incontro tra ogni coppia (A, B) va contato una sola volta

$$n(n-1) / 2$$

Oppure: numero di spigoli in un grafo completo di *n* vertici



Vertici $n = 5$

Spigoli $(n-1)n/2 = 4 + 3 + 2 + 1$

$$\sum_{i=0}^{n-1} 2^i = 2^n - 1 \qquad\qquad 1 + 2 + 4 + 8 + 16 = 32 - 1 = 31$$

Crescita esponenziale !

n :	1	2	3	4	5	. . .	10	. . .	20	. . .	30	. . .	40	. . .
$1{,}3^n$:	1,3	1,7	2,2	2,9	3,7	. . .	13,8	. . .	190	. . .	2054	. . .	22190	. . .

k^n dà il numero di *disposizioni* (con ripetizione) di k elementi in gruppi di n. Per esempio per $k = 2, n = 3$ le disposizioni dei due elementi 0,1 in gruppi di tre sono $2^3 = 8$: 000 001 010 011 100 101 110 111.

Vi sono $2^{10} = 1024$ numeri binari di 10 bit, da 0 a 1023

e si possono contare con le dita . . .

Vi sono 2^n sottoinsiemi in un insieme di n elementi

`n = 5` `A B C D E`

k^n dà il numero di *disposizioni* (con ripetizione) di k elementi in gruppi di n. Per esempio per $k = 2, n = 3$ le disposizioni dei due elementi 0,1 in gruppi di tre sono $2^3 = 8$: 000 001 010 011 100 101 110 111.

Vi sono $2^{10} = 1024$ numeri binari di 10 bit, da 0 a 1023

e si possono contar con le dita . . .

Vi sono 2^n sottoinsiemi in un insieme di n elementi

```
n = 5        A   B   C   D   E

             1   0   0   1   0
```

fattoriale n! = 1×2×3× . . . n

$n! \approx \sqrt{2\pi n}(n/e)^n$, ove $e \approx 2,718\ldots$ è il numero di Eulero:

formula di Stirling che approssima la funzione fattoriale.

$n!$ è pari al numero di *permutazioni di n elementi.*

n = 4 n! = 24 insieme A B C D

ABCD ABDC ACBD ACDB ADBC ADCB
BACD . . . .
CABD . . . .
. . . . DCBA

$\binom{n}{k} = \frac{n!}{k!(n-k)!}$ **coefficiente binomiale**

esprime il numero di *combinazioni* di n elementi in gruppi di k.

Per esempio per $n = 5, k = 2$ le combinazioni dei cinque elementi a, b, c, d, e in gruppi di due sono $5!/(2!3!) = 120/12 = 10$:

$ab \;\; ac \;\; ad \;\; ae \;\; bc \;\; bd \;\; be \;\; cd \;\; ce \;\; de.$