Titolo unità didattica: Efficienza degli algoritmi [10]

Titolo modulo : Trattabilità dei problemi [03-T]

Ottimalità di algoritmi, trattabilità e intrattabilità di problemi

Argomenti trattati:

- ✓ analisi di complessità di algoritmi per la valutazione di polinomi
- ✓ algoritmi ottimali
- ✓ problemi trattabili e problemi intrattabili

Prerequisiti richiesti: P1-10-02-T

valutazione del valore di un polinomio

$$p_n(x) = \sum_{i=0}^{n} a_i x_i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

rappresentazione → array 1D dei coefficienti

forma standard

fissata un'ascissa c → calcolare $p_n(c)$

```
float valutazione_pol(float coef[], int n, float c) {
 int i;
 float potenza, valore_pol;
 valore_pol = coef[0] ;
 potenza = 1.0 ;
 for (i=1; i <= n; i++) {
    potenza = potenza*c ;
    valore_pol = valore_pol+coef[i]*potenza ;
  }
 return valore_pol ;
}
```

T(n) = 2n prodotti

notare che un polinomio di grado n ha $n+1$ coefficienti

osservazione:

$$p_n(x) = \sum_{i=0}^{n} a_i x_i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

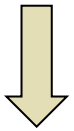
è (matematicamente) equivalente a

$$p_n(x) = \left(\left(\cdots\left(a_n x + a_{n-1}\right)x + a_{n-2}\right)x + \cdots\right)x + a_0$$

forma di Newton (forma innestata)

$$p_n(x) = \left(\left(\cdots\left(a_n x^2 + a_{n-1} x + a_{n-2}\right)x + \cdots\right)x + a_1\right)x + a_0$$

$$p_n(x) = \left(\left(\cdots\left(a_n x^3 + a_{n-1} x^2 + a_{n-2} x + a_{n-3}\right)x + \cdots\right) + a_1\right)x + a_0$$



$$p_n(x) = a_n x^n + \cdots + a_2 x^2 + a_1 x + a_0$$

osservazione:

$$p_n(x) = \sum_{i=0}^{n} a_i x_i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

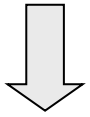
è (matematicamente) equivalente a

$$p_n(x) = \left(\left(\cdots\left(a_n x + a_{n-1}\right)x + a_{n-2}\right)x + \cdots\right)x + a_0$$

forma di Newton (forma innestata)

esempio:

$$p_3(x) = \sum_{i=0}^{3} a_i x_i = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$



$$p_3(x) = \left(\left(a_3 x + a_2\right)x + a_1\right)x + a_0$$

forma di Newton (forma innestata)

$$p_n(x) = (((\cdots(a_n x + a_{n-1})x + a_{n-2})x + \cdots)x + a_0$$

n-1 livelli di parentesi

```
for (i=1; i <= n-1; i++) {
      valutare il livello  i-simo  di parentesi
 }
```

```
for (i=n-1; i > 0; i--) {
      valutare il livello i-simo di parentesi
 }
```

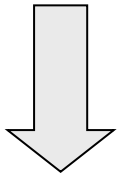
```
float val_pol_horner(float coef[], int n, float c) {
 int i;
 float valore_pol;
 valore_pol = coef[n] ;
 for (i=n-1; i >= 0; i--) {
    valore_pol = valore_pol * c + coef[i] ;
 }
 return valore_pol ;
}
```

$T(n) = n$ prodotti e somme

determinazione simultanea del massimo e del minimo (e dei loro indici) dei valori di un array 1D

strategia 1

richiamare successivamente una function per il calcolo del massimo valore (e del suo indice) e quindi una function per il calcolo del minimo valore (e del suo indice)



algoritmo con complessità di tempo

$T(n) = 2n-2$

operazione dominante: confronto tra due elementi dell'array

determinazione simultanea del massimo e del minimo (e dei loro indici) dei valori di un array 1D

strategia 2

approccio incrementale al calcolo simultaneo del massimo valore (e del suo indice) e del minimo valore (e del suo indice)

unico “passaggio” sull’array (unico ciclo `for`) e aggiornamento delle variabili per le due soluzioni parziali

```
/* massimo e minimo tra gli elementi di un
     array int - notazione standard          */
void max_min_arrayI(int a[],int n,
                    int *max, int *min)
{
 int i;
 *max = a[0];
 *min = a[0];
 for (i=1;i<n;i++)
    if(a[i] > *max)
          *max = a[i] ;
       else if(a[i] < *min)
             *min = a[i] ;
}
```

$T(n) = 2n-2$
confronti
(caso peggiore)

determinazione simultanea del massimo e del minimo (e dei loro indici) dei valori di un array 1D

strategia 2

algoritmo con complessità di tempo

(caso peggiore)

$T(n) = 2n-2$

operazione dominante: confronto tra due elementi dell'array

esistono valori dei dati di input che danno luogo solo a n-1 confronti

determinazione simultanea del massimo e del minimo (e dei loro indici) dei valori di un array 1D

strategia 3

approccio incrementale al calcolo simultaneo del massimo valore (e del suo indice) e del minimo valore (e del suo indice), **ma** attraverso un **confronto tra coppie di elementi successivi dell'array; il massimo elemento della coppia viene confrontato con** `max_a` **e il minimo elemento della coppia viene confrontato con** `min_a`

```
void max_min_3(in: float a[], int n;
                    out: float max_a, float min_a) {
 int i; max_a = a[0]; min_a = a[0];
 for (i=1; i < n; i=i+2) {
    if (a[i] > a[i+1]) {
/*  a[i] è il max  e  a[i+1]  è il min della coppia */
          if (a[i] > max_a)
             { max_a = a[i] ; }
          if (a[i+1] < min_a)
             { min_a = a[i+1] ; }
     else
/* a[i] è il min e a[i+1] è il max della coppia   */
          if (a[i] < min_a)
             { min_a = a[i] ; }
        if (a[i+1] > max_a)
             { max_a = a[i+1] ; }
    }
 }
}
```

$T(n) = 3n/2$ confronti

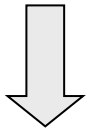
ATTENZIONE: da modificare in C

```c
/* massimo e minimo di un array - VERSIONE 3 */
void max_min_arrayI(int a[],int n,
                    int *max, int *min){
 int i;
 *max = a[0];
 *min = a[0];
 for (i=1; i<n; i=i+2)
   if(a[i] > a[i+1]){
      if(a[i] > *max)
          *max = a[i] ;
      if(a[i+1] < *min)
          *min = a[i+1] ;}
    else {
      if(a[i] < *min)
          *min = a[i] ;
      if(a[i+1] > *max)
          *max = a[i+1] ;}
}}
```

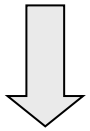
determinazione simultanea del massimo e del minimo (e dei loro indici) dei valori di un array 1D

strategia 3

1 confronto tra i 2 elementi di una coppia
(per determinare il massimo e il minimo della coppia);
1 confronto tra il max della coppia e max_a ;
1 confronto tra il min della coppia e min_a



3 confronti per coppia; ci sono n/2 coppie



complessità di tempo $T(n) = 3n/2$

determinazione simultanea del massimo e del minimo (e dei loro indici) dei valori di un array 1D

operazione dominante: confronto tra due elementi dell'array

strategia 1 $T(n) = 2n-2$

strategia 2

$T(n) = 2n-2$ (caso peggiore)

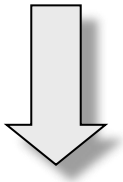
$T(n) = n-1$ (caso migliore)

strategia 3 $T(n) = 3n/2$

teoria della complessità computazionale

determinazione di
limitazioni inferiori di complessità
per **l'insieme degli algoritmi**
che risolvono uno **specifico problema**

individuare il **migliore algoritmo**
per risolvere uno **specifico problema**



algoritmi ottimali

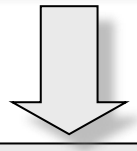
complessità computazionale intrinseca di un **problema**

=

complessità dell'algoritmo risolutore **ottimale**

esempio:

l'algoritmo di Horner è **ottimale**

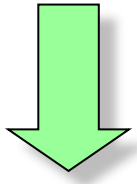


complessità computazionale **intrinseca del problema** della **valutazione di un polinomio di grado n in una fissata ascissa** (polinomio nella forma standard):

$T(n) = n$ (prodotti e somme) e $S(n) = n + 1$

teoria della complessità computazionale

determinare se un problema è
trattabile computazionalmente



esistono algoritmi risolutori del problema con complessità di tempo polinomiale **?**

si è dimostrato che l'algoritmo risolutore ottimale ha complessità di tempo polinomiale **?**

teoria della complessità computazionale

determinare se un problema è **intrattabile computazionalmente**

si conoscono solo algoritmi risolutori del problema con complessità di tempo **più** che polinomiale (esponenziale o fattoriale) **?**

si è dimostrato che l'algoritmo risolutore ottimale ha complessità di tempo **più** che polinomiale (esponenziale o fattoriale) **?**

teoria della complessità computazionale

problemi
trattabili

problemi di
complessità
intrinseca
polinomiale

problemi
intrattabili

problemi di
complessità
intrinseca
più che polinomiale
(**esponenziale** o
fattoriale)

teoria della complessità computazionale

una importante classe di problemi è quella dei **problemi NP**
cioè i problemi per i quali una **soluzione proposta** può essere **verificata** in **tempo polinomiale**

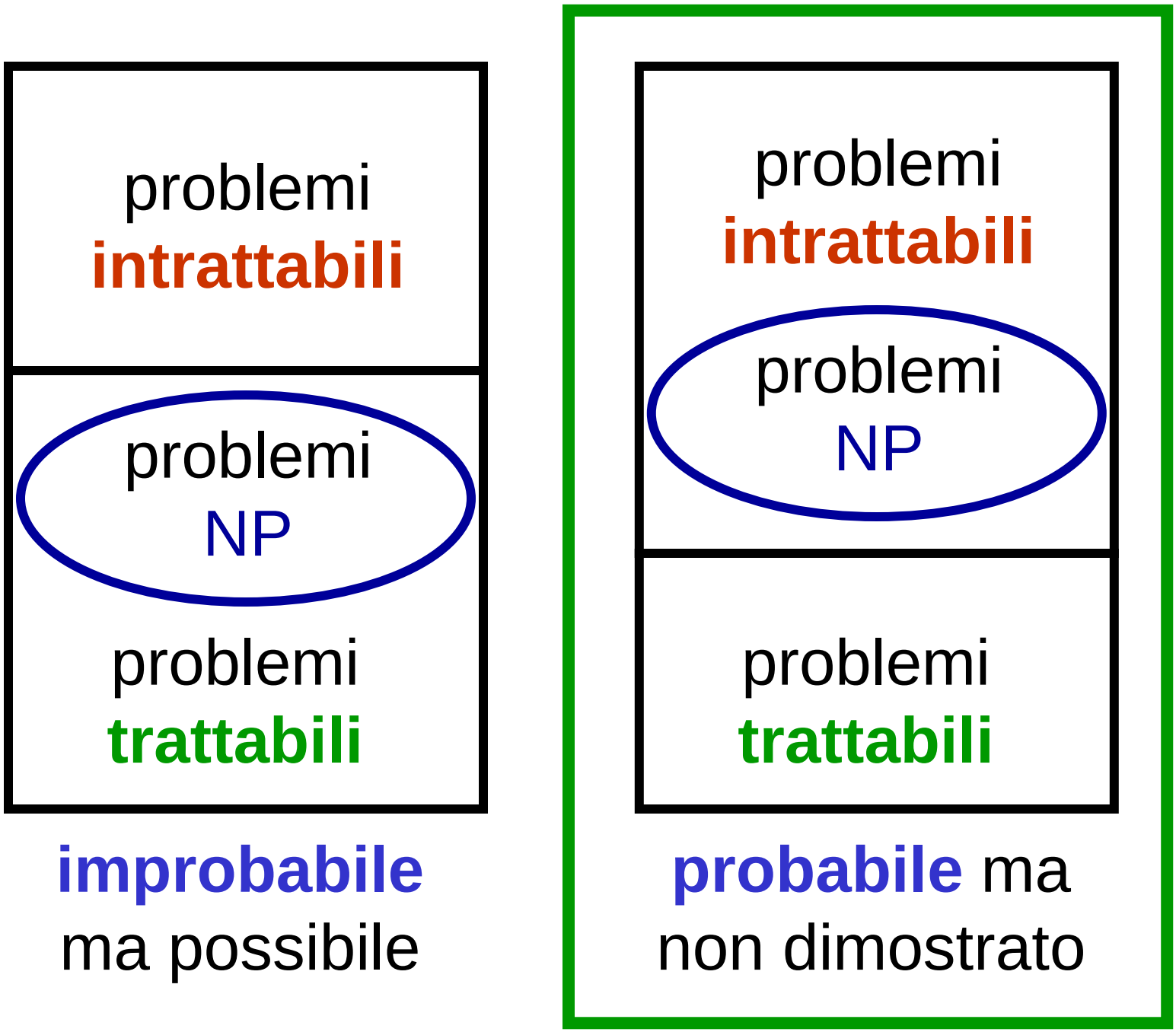
in altre parole, anche se potrebbe non essere facile trovare una soluzione, una volta che si abbia una soluzione proposta, si può **verificare** se è corretta o meno in **modo efficiente**

teoria della complessità computazionale

un esempio di
problema NP
è il problema della **fattorizzazione di un numero intero in fattori primi**

Esempio: 15 = 3 x 5, con 3 e 5 **numeri primi**, ovvero divisibili solo per sé stessi e per 1

dato un numero N e proposta una sua fattorizzazione in numeri primi, si può **verificare** se la fattorizzazione è semplicemente moltiplicando tra loro i numeri primi e confrontando tale prodotto con il numero N

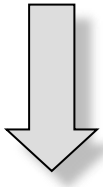

problemi
intrattabili
problemi
NP
problemi
trattabili
improbabile
ma possibile
problemi
intrattabili
problemi
NP
problemi
trattabili
probabile ma
non dimostrato

teoria della complessità computazionale

problema della **fattorizzazione di un numero intero N in fattori primi**

- la dimensione computazionale è il numero di cifre di N
- la verifica di una soluzione è di complessità polinomiale (verifica della primalità dei numeri proposti e poi loro moltiplicazione)
- al momento, l'algoritmo più efficiente per il calcolo dei fattori primi di N è a complessità esponenziale

trattabilità / intrattabilità
della classe dei
problemi NP



P = NP
P ≠ NP

Millennium problem

1 milione di dollari

Clay Mathematics Institute

http://www.claymath.org