Esempi di algoritmi

Lezione III

Scopo della lezione

- Implementare “da zero” algoritmi di media complessità.
- Verificare la correttezza di un algoritmo eseguendolo “a mano”.
- Imparare a valutare le prestazioni di un algoritmo.
- Scrivere alcuni algoritmi “tipici” da utilizzare come base per realizzarne altri.

Calcolo dei numeri primi

- Scriviamo un algoritmo che, letto un numero, verifica se questo sia o meno primo
- Un numero è primo se è divisibile solo per se stesso e per l'unità
  - 1 è primo
  - 2 è primo
  - 3 è primo
  - 4 NON è primo, ma è divisibile per 2
  - …

Calcolo dei numeri primi

- Quindi, per verificare la primalità di un numero n è sufficiente provare a dividerlo per tutti gli interi minori di esso
  - Se almeno uno di questi interi è un divisore di n allora n non è primo
  - Altrimenti n è primo

Calcolo dei numeri primi


```
leggi n
<inizializzazione>
    div ← 2
primo ← true
QUANDO div <= radice(n) OPPURE primo == false
    SE n è divisibile per div
        ALLORA
            primo ← false
    FINESE
    div ← div+1
RIPETI
<output del risultato>
SE primo == true
    ALLORA
        scrivi “il numero è primo”
    ALTRIMENTI
        scrivi “il numero non è primo”
FINESE
```

Calcolo dei numeri primi


```
leggi n
div = 2
primo ← true
QUANDO div < n ESEGUI
    SE n è divisible per div
        ALLORA
            primo ← false
    FINESE
    div ← div+1
RIPETI
SE primo == true
    ALLORA
        scrivi “il numero è primo”
    ALTRIMENTI
        scrivi “il numero non è primo”
FINESE
```

n	6
div	~~2 3 4 5~~ 6
primo	~~true~~ false

Scriviamo meglio l'algoritmo

- La condizione "n è divisibile per div" può essere scritta utilizzando l'operatore matematico MOD che ritorna il resto della divisione intera:
  - 10 MOD 3 = 1
  - 8 MOD 4 = 0
- Quindi a MOD b = 0 se e solo se a è divisibile per b

Scriviamo meglio l'algoritmo


```
leggi n
div = 2
primo ← true
QUANDO div < n ESEGUI
    SE n MOD div == 0
            ALLORA
                    primo ← false
    FINESE
    div ← div+1
RIPETI
SE primo == true
    ALLORA
            scrivi "il numero è primo"
    ALTRIMENTI
            scrivi "il numero non è primo"
FINESE
```

Miglioriamo l'algoritmo

- Nel momento in cui trovo un divisore proprio (cioè diverso da 1 e dal numero che sto esaminando), sono certo del fatto che il numero in esame non è primo
- Quindi è inutile proseguire il ciclo ESEGUI

Alternativa I


```
leggi n
div = 2
primo ← true
QUANDO (div < n) E (primo == true) ESEGUI
    SE n MOD div == 0
            ALLORA
                    primo ← false
    FINESE
    div ← div+1
RIPETI
```

Alternativa II


```
leggi n
div = 2
primo ← true
QUANDO div < n ESEGUI
    SE n MOD div == 0
            ALLORA
                    primo ← false
                    div ← n
    FINESE
    div ← div+1
RIPETI
```

Alternativa III


```
leggi n
div = 2
primo ← true
QUANDO div < n ESEGUI
    SE n MOD div == 0
            ALLORA
                    primo ← false
                    esci dal ciclo
    FINESE
    div ← div+1
RIPETI
```

Miglioriamo l'algoritmo

- Tutte e tre le alternative sono realizzabili
- La seconda inserisce un'istruzione "impropria", con il solo fine di forzare l'uscita dal ciclo
- Le altre due alternative sono più "leggibili"
- La prima è preferibile perché non ricorre a "salti"

L’abbiamo scritto bene?

- Criteri per valutare la bontà di un algoritmo
  - **Correttezza**: dimostrabilità del fatto che l’algoritmo esegua effettivamente le operazioni per cui è stato progettato.
  - **Complessità**: determinabilità del tempo limite di esecuzione.
- Per entrambi questi criteri esistono vari tipi di tecniche formali.

Correttezza

- L'algoritmo proposto
  - Termina sempre, al più in n iterazioni
  - Ha un output corretto, nel senso che se un numero non è primo uno dei suoi divisori propri attiverà necessariamente l'istruzione SE ALLORA che assegna false a primo.

Complessità

- Ovviamente il tempo effettivo di esecuzione dell'algoritmo proposto (anzi, di ogni algoritmo) dipende:
  - dal valore fornito in ingresso e
  - dalla velocità dell'esecutore
- Quindi questo tempo viene quantificato in termini del numero di operazioni che vengono eseguite in funzione del valore fornito in ingresso.

Quanto tempo ci mette?

- L'algoritmo consta di tre parti
  - Una costituita da tre operazioni che vengono sempre eseguite
  - Una costituita da un ciclo eseguito al massimo per tante iterazioni quanto è grande n. A ogni iterazione del ciclo vengono eseguite al massimo cinque operazioni (tre confronti e due assegnamenti)
  - Una costituita da un confronto e un'operazione di output, sempre eseguite

Sì,ma quanto tempo?

- Quindi, se n indica il numero inserito a inizio algoritmo, il numero di operazioni eseguito è al più 5+5n
- Trascurando i dettagli, il numero di operazioni dipende linearmente dal valore inserito
- Questa dipendenza viene formalmente indicata dicendo che la **complessità** dell'algoritmo è **O(n)**

Possiamo fare di meglio!

Lemma: se un numero n non è primo, almeno uno dei suoi divisori propri è minore o uguale della radice quadrata di n.

Dimostrazione:

n non primo $\Rightarrow n = ab$ per $1 < a,b < n$

per assurdo $a,b > \sqrt{n} \Rightarrow n = ab > \sqrt{n}\sqrt{n} = n$

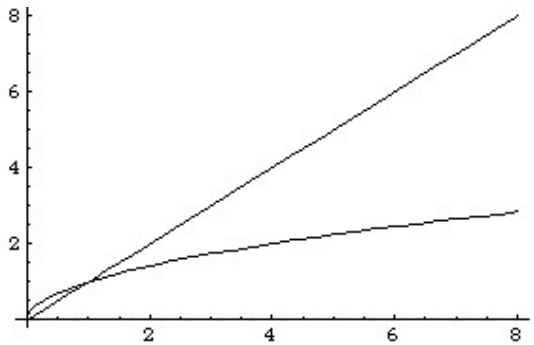
Quindi possiamo terminare prima il ciclo.

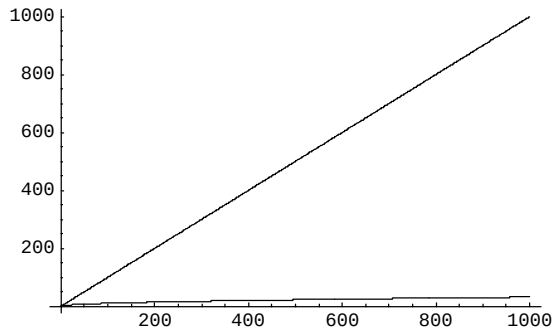
Un algoritmo migliore


```
leggi n
div = 2
primo ← true
QUANDO (div <sqrt(n)) E (primo == true) ESEGUI
    SE n MOD div == 0
        ALLORA
                primo ← false
    FINESE
    div ← div+1
RIPETI
SE primo == true
    ALLORA
        scrivi “il numero è primo”
    ALTRIMENTI
        scrivi “il numero non è primo”
FINESE
```

Complessità

- In questo modo la complessità dell'algoritmo è scesa da O(n) a O($\sqrt{n}$)





- Ad esempio, per un processore da 1 GHz, il tempo di esecuzione per n=12,345,678,901,234,567,890 passerebbe da più di 390 anni a 3.5 secondi!

Il migliore?

- Esistono algoritmi con prestazioni migliori per risolvere lo stesso problema, ma noi non li prenderemo in considerazione.

Modularità

- E’ possibile utilizzare un algoritmo già scritto per scriverne un altro che risolva un problema correlato
  - Usando l’algoritmo di partenza come base per costruire quello nuovo
  - Costruendo il nuovo algoritmo in modo che esegua uno o più volte quello di partenza

Esercizio

- Scrivete un algoritmo che, letto un intero positivo n, stampi la lista dei primi n numeri primi.

Giochiamo a carte

- Scrivete un algoritmo che, letti due caratteri indicanti una carta da gioco utilizzando la seguente codifica

A	asso	K	re
2-9	carte da 2 a 9	Q	quadri
0	dieci	C	cuori
J	fante	P	picche
Q	donna	F	fiori

stampi la descrizione completa

Esempi

- QP: donna di picche
- 0Q: dieci di quadri
- 4F: quattro di fiori
- AC: asso di cuori
- …

Giochiamo a carte


```
leggi punto
leggi seme
SE punto == ‘A’
    ALLORA scrivi “asso”
FINESE
SE punto == ‘2’
    ALLORA scrivi “due”
FINESE
SE punto == ‘3’
    ALLORA scrivi “tre”
FINESE
SE punto == ‘4’
    ALLORA scrivi “quattro”
FINESE
```

Giochiamo a carte


```
SE punto == ‘5’
    ALLORA scrivi “cinque”
FINESE
SE punto == ‘6’
    ALLORA scrivi “sei”
FINESE
SE punto == ‘7’
    ALLORA scrivi “sette”
FINESE
SE punto == ‘8’
    ALLORA scrivi “otto”
FINESE
SE punto == ‘9’
    ALLORA scrivi “nove”
FINESE
```

Giochiamo a carte


```
SE punto == ‘0’
    ALLORA scrivi “dieci”
FINESE
SE punto == ‘J’
    ALLORA scrivi “fante”
FINESE
SE punto == ‘Q’
    ALLORA scrivi “donna”
FINESE
SE punto == ‘K’
    ALLORA scrivi “re”
FINESE
scrivi “ di ”
```

Giochiamo a carte


```
SE seme == “Q”
    ALLORA scrivi “quadri”
FINESE
SE seme == “C”
    ALLORA scrivi “cuori”
FINESE
SE seme == “P”
    ALLORA scrivi “picche”
FINESE
SE seme == “F”
    ALLORA scrivi “fiori”
FINESE
```

Alcune osservazioni

- L'algoritmo si comporta correttamente solo se l'input rispetta la codifica segnalata. In caso contrario non viene stampato alcunché (nemmeno un messaggio di errore)
- Il numero massimo di operazioni eseguito è indipendente dall'input.

Alcune osservazioni

- Notate l'uso degli apici (‘’) e dei doppi apici (“”) in parti diverse dell'algoritmo.
- Servono per differenziare tipologie di dati differenti (un concetto che approfondiremo tra qualche lezione).
- Possiamo scrivere l'algoritmo utilizzando una sola istruzione scrivi?

Controllo dell'input

- Scrivere un algoritmo che, posta la domanda "Vuoi continuare?", legga un dato e continui a riproporre la domanda fino a quando il dato letto non sia uguale a "Si" oppure a "No"

Controllo input


```
ESEGUI
    scrivi “Vuoi continuare?”
    leggi risposta
QUANDO (risposta != “Si”) E (risposta != “No”)
```

- != è un operatore che indica quando i suoi due operandi sono diversi.
- Si può ottenere anche negando il risultato dell’operatore ==.

Esercizio

- Riscrivete questo algoritmo usando un ciclo QUANDO … RIPETI
- Modificate l'algoritmo in modo da avvisare l'utente in caso abbia immesso come risposta qualunque cosa che sia diversa da "Si" e "No".
- Cosa succede se l'utente immette come risposta "no"?

Ordinamento

- Scriviamo un algoritmo che, letti tre numeri, li stampa in ordine crescente

Ordinamento


```
leggi a
leggi b
leggi c
SE a<b ALLORA
    SE a<c ALLORA
        minimo ← a
        SE b<c ALLORA
            medio ← b
            massimo ← c
        ALTRIMENTI
            medio ← c
            massimo ← b
        FINESE
    ALTRIMENTI
```

Ordinamento


```
            minimo ← c
            medio ← a
            massimo ← b
        FINESE
ALTRIMENTI
        SE b<c ALLORA
            minimo ← b
            SE a<c ALLORA
                    medio ← a
                    massimo ← c
            ALTRIMENTI
                    medio ← c
                    massimo ← a
            FINESE
        ALTRIMENTI
```

Ordinamento


```
            minimo ← c
            medio ← b
            massimo ← a
        FINESE
    FINESE
    scrivi minimo
    scrivi medio
    scrivi massimo
```

Esercizi

- Un anno è bisestile (ha 366 giorni) se è divisibile per quattro (come il 1980) e non è divisibile per 100 (ad es. il 1900 non è bisestile).Fanno eccezione gli anni divisibili per 400, che sono bisestili (ad es. il 2000 è bisestile).
- Questa regola non si applica prima del 1582, anno di introduzione del calendario gregoriano.
- Scrivete un algoritmo che, letto un anno, decida se questo è o meno bisestile.

Esercizi

- Scrivete un algoritmo che riceva in input
  - il nome di un dipendente
  - il numero di ore che ha lavorato durante una settimana
  - la paga oraria
- Controllate che il numero di ore sia non inferiore a 40 e calcolate lo stipendio settimanale, tenuto conto che le ore di straordinarie sono pagate il doppio.