Palestra di algoritmi - day 9

10/01/2023

Efficienza di un algoritmo

Abbiamo tanto parlato del fatto che i nostri programmi devono essere veloci, ma non abbiamo mai discusso di come possiamo stabilire se un algoritmo è veloce.

Come misurare il tempo impiegato?

Il tempo reale impiegato per processare un programma dipende da troppi parametri:

- linguaggio di programmazione utilizzato
- codice generato dal compilatore
- processore, memoria (cache, primaria, secondaria)
- sistema operativo, processi attualmente in esecuzione

Insomma, dobbiamo trovare un modo più astratto.

Possiamo usare il numero di operazioni “rilevanti”

Ricerca del minimo

```
int minimo = vet[0];
for(int i=0;i<n;i++){
        if(vet[i]<minimo)
                minimo = vet[i];
}
```

Ricerca di un elemento

```
int pos = -1;
for(int i=0;i<n;i++){
        if(vet[i]==cercato)
                pos = i;
}
```

NB: le operazioni che consideriamo devono essere “semplici”

Possiamo fare meglio di così per questi due algoritmi?

Pensateci… Più avanti ne parleremo.

Notazione O(n)

n = dimensione del nostro input.

Nel caso della ricerca di un elemento, n = numero di elementi dell'array.

O(n) significa che vengono eseguite al massimo n operazioni.

Se scorriamo tutto l'array una volta -> O(n)

Se facciamo un doppio ciclo -> O(n^2)

E così via…

A cosa serve a noi tutto questo?

Solitamente

1 secondo = 1 milione di operazioni.

Quindi:

- con input molto piccoli: possiamo fare quello che vogliamo
- con input fino a n=1000 -> possiamo fare un algoritmo in O(n^2)
- con input più grandi dobbiamo fare algoritmi migliori, per esempio in O(n)

Selection sort

```
for(int i=0;i<n;i++) {
        //trovo il minimo
        int minimo = vet[i];
        int pos = i;
        for(int j=i;j<n;j++) {
            if(vet[j]<minimo) {
                minimo = vet[j];
                pos = j;
            }
        }
        //scambio minimo
        int temp = vet[pos];
        vet[pos] = vet[i];
        vet[i] = temp;
}
```

il ciclo esterno viene ripetuto n volte

il ciclo interno viene ripetuto al massimo n volte (la prima volta i va da 0 a n)

-> O(n^2)!!!

Bubble sort

```
bool notOrdinato = true;

while(notOrdinato) {
        notOrdinato = false;
        for(int i=0;i<numElem-1;i++) {
                if(vet[i]>vet[i+1]) {
                        notOrdinato = true;
                        int temp = vet[i];
                        vet[i] = vet[i+1];
                        vet[i+1] = temp;
                }
        }
}
```

quante volte viene ripetuto il while?

al massimo n volte

il ciclo interno viene ripetuto n volte

→ O(n^2)!!!

Ma quindi selectionSort e bubbleSort hanno la stessa complessità?

Cosa succede quando l’array è già ordinato?

Problemi di oggi

- 2 problemi su github
- easy3 su training.olinfo