Valutazione di Programmi

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Criteri di Valutazione

Un programma può essere valutato rispetto ai seguenti criteri (quando applicabili)

- Correttezza (le funzioni significative devono essere commentate con PRE POST): si fornisce evidenza che il codice realizzi le POST
- Efficienza: evitare codice inutile, cercare di trovare l'algoritmo più efficiente per risolvere i (sotto)problemi
- Organizzazione del codice: divisione logica del codice in funzioni. Evitare di risolvere una seconda volta problemi già risolti
- Stile: il codice deve essere leggibile dai vostri colleghi (evitare istruzioni non spiegate da me), commentare i frammenti di codice che non siano ovvi, usare nomi significativi per le variabili

Criteri di Valutazione

Correttezza

- le funzioni significative devono essere commentate con PRE POST
- il codice compila
- Se il codice non è corretto, si
- si fornisce evidenza che il codice realizzi le POST
- Efficienza: evitare codice inutile, cercare di trovare l'algoritmo più efficiente per risolvere i (sotto)problemi
- Organizzazione del codice: divisione logica del codice in funzioni. Evitare di risolvere una seconda volta problemi già risolti
- Stile: il codice deve essere leggibile dai vostri colleghi (evitare istruzioni non spiegate da me), commentare i frammenti di codice che non siano ovvi, usare nomi significativi per le variabili

Efficienza

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Efficienza

- L'efficienza di un algoritmo non viene calcolata cronometrando il tempo di esecuzione
  - perché dipendente dalla macchina
- Si usa una misura che dipende dal numero di istruzioni eseguite
- Ci interessa qual è la dipendenza rispetto all'input
  - Qual è il fattore dell'input che fa variare maggiormente il numero di istruzioni eseguite?
  - Es. la funzione che descrive il numero di istruzioni eseguite cresce in modo costante, linearmente, in modo quadratico rispetto alla dimensione dell'input?

Efficienza

- Ci interessa stimare quanto ci metterà per eseguire il programma con un input “molto grande”. Es. int X[10000000]; ordina_array(X)
  - Della funzione che conta il numero di istruzioni eseguite ci interessa il termine più grande
  - Es. se vengono eseguite 3n^2 + 50n+99999 istruzioni, il termine che domina è n^2
  - Indichiamo quindi il numero di istruzioni con O(n^2) (big-O notation), tralasciando gli altri termini (per compattezza e perché irrilevanti per grandi input)
- Un algoritmo è significativamente più efficiente di un altro se la sua complessità, utilizzando la notazione O() è minore

$$O(1) < O(n) < O(n^2)$$

- O(1) = complessità costante (numero di istruzioni costante che non dipende dall’input)

Cenni di Complessità Computazionale

- Problema: creare una funzione che, dato un array in input, verifichi se il primo elemento di un array è uguale al secondo

```
//PRE A ha dim>1 elementi
int f(int *A, int dim) {
  return A[0]==A[1];
}
```

- Il tempo di esecuzione è indipendente dalla dimensione dell'input, dim: O(1)

- Il tempo di esecuzione è costante: O(1)
  - O(1) non significa che viene eseguita esattamente una istruzione,
  - O(1) significa che il tempo di esecuzione è costante, ovvero che esiste una funzione costante che è un limite superiore al tempo di esecuzione
    - in altre parole che la complessità non cresce linearmente con la dimensione dell'input

Cenni di Complessità Computazionale

- Problema 2: creare una funzione che, dato un array in input, verifichi se il primo elemento di un array è uguale al secondo, terzo e quarto

- Tempo di esecuzione: O(1)

- Vengono eseguite più istruzioni rispetto al caso precedente, ma sono sempre un numero costante
  - Per un input enorme, eseguire 1 o 5 istruzioni non cambia molto

```
//PRE A ha dim>1 elementi
int f(int *A, int dim) {
   return (A[0]==A[1]) &&
          (A[0]==A[2]) &&
          (A[0]==A[3]);
}
```

Cenni di Complessità Computazionale

- Problema: creare una funzione che, dato un array in input e la sua dimensione, calcoli il valore massimo
- Si deve scorrere tutto l'array, confrontando ogni elemento di A con il max
- Il numero di istruzioni che vengono eseguite dipende dalla dimensione di A linearmente
- O(1) non è più un limite superiore al numero di istruzioni, ma O(n) lo è

```
int max_value(int A[], int n) {
   /*
      PRE: A ha n>0 elementi
      POST: calcola il valore massimo in A
    */
   int max = A[0];
   for(int i=1; i<n; i+=1) {
      if (A[i]>max)
         max = A[i];
   }
   return max;
}
```

Cenni di Complessità Computazionale

- Problema: funzione che verifichi se un elemento di un array è duplicato altrove nell'array.
  - Il primo elemento deve essere confrontato con ogni altro elemento dell'array;
  - il secondo elemento deve essere confrontato con ogni altro elemento tranne il primo (esso è già stato confrontato con il primo).
  - Il terzo elemento deve essere confrontato con ogni altro elemento eccetto i primi due.
- il totale dei confronti sarà: n-1 + n-2 + ... + 2 + 1
- La somma dei primi n-1 numeri è n*(n-1)/2 = n^2/2 – n/2.
- Il termine principale che influenza la complessità è quello più grande: n^2
- La complessità dell'algoritmo è perciò O(n^2)

Efficienza Computazionale

- Un algoritmo è significativamente più efficiente di un altro se la sua complessità è minore

$$O(1) < O(n) < O(n^2)$$

- nel corso di algoritmi imparerete come calcolare la complessità di un algoritmo
- Notate che nel corso abbiamo posto l'accento sul risparmio di singole istruzioni, il motivo, più che per efficienza, è per l'atteggiamento da tenere nella scrittura del codice (attenzione ai dettagli)

Efficienza Computazionale: Esempio


```
int somma1nV1(int n) {
        return n*(n+1)/2;
}
```

```
int somma1nV2(int n) {
        int somma=0;
        for(int i=0;i<n;i+=1)
                somma+=i;
        return somma;
}
```

- Complessità somma1nV1?
- Complessità somma1nV2?

Efficienza Computazionale: Esempio


```
int somma1nV1(int n) {
        return n*(n+1)/2;
}
```

```
int somma1nV2(int n) {
        int somma=0;
        for(int i=0;i<n;i+=1)
                somma+=i;
        return somma;
}
```

- Complessità somma1nV1 = O(1) // numero di operazioni costanti
- Complessità somma1nV1 = O(n) // il corpo del for viene eseguito n volte