Programmazione

Giovanni Da San Martino

Dipartimento of Matematica, Università degli Studi di Padova
giovanni.dasanmartino@unipd.it
A.A. 2021-2022

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Previously on Programmazione

- Una funzione ricorsiva può essere utilizzata quando la soluzione di un problema può essere espressa in termini della una soluzione di un'istanza più semplice dello stesso problema
- Se vogliamo applicare una soluzione ricorsiva dobbiamo

1. Chiarire cosa rende un'istanza di un problema più semplice di un'altra. E' possibile associare un numero naturale a tale complessità
2. Identificare i casi base, ovvero le istanze più semplici del problema, e come calcolare le loro soluzioni (caso base)
3. Chiarire come, una volta calcolata la soluzione di complessità n, calcolare da essa la soluzione del caso n+1 (caso ricorsivo/induttivo)

Esempio: Valore Minore di Array

- Definire una funzione ricorsiva che calcoli il valore minore in un array di interi.

1. Chiarire cosa rende un'istanza di un problema più semplice di un'altra. E' possibile associare un numero naturale a tale complessità

la complessità del problema può essere espressa in termini della dimensione dell'array

Esempio: Valore Minore di Array

- Definire una funzione ricorsiva che calcoli il valore minore in un array di interi.

1. la complessità del problema può essere espressa in termini della dimensione dell'array
2. il caso base è l'array con un solo elemento

Il minimo di un array con un solo elemento è l'unico elemento dell'array

Esempio: Valore Minore di Array

- Definire una funzione ricorsiva che calcoli il valore minore in un array di interi.

1. la complessità del problema può essere espressa in termini della dimensione dell'array
2. il caso base è l'array con un solo elemento
3. Se conoscessi il minore di un array di dimensione n, come potrei trovare la soluzione per un array di dimensione n+1, ovvero l'array di dimensione n con un elemento aggiunto?

   Se conosco min(X[0],..., X[n-1]) come posso calcolare min(X[0],..., X[n])?

Esempio: Valore Minore di Array

- Definire una funzione ricorsiva che calcoli il valore minore in un array di interi.

3. Se conoscessi il minore di un array di dimensione n, come potrei trovare la soluzione per un array di dimensione n+1, ovvero l'array di dimensione n con un elemento aggiunto?

se conosco min(X[0],..., X[n-1]) come posso calcolare min(X[0],..., X[n])?

min(X[0],..., X[n]) = min(X[0], min(X[0],..., X[n-1]))

Esempio: Valore Minore di Array


```
int minimo(int X[], int dim) {
   /*
      PRE: dim>=0 è il numero di elementi di X
    */
   if(dim==0)
      return -1; //ERRORE - vettore vuoto
   if(dim==1)
      return X[0];
   else {
      int min_vettore = minimo(X+1, dim-1);
      return (X[0]<min_vettore)? X[0]: min_vettore;
   }
}
```

Induzione Matematica

$$\frac{P(0) \qquad P(n) \Rightarrow P(n+1))}{\forall n \,.\, P(n)}$$

- Per dimostrare che una proprietà vale per tutti i numeri naturali, dimostriamo che
- vale per 0
- se assumiamo che vale per n, dimostriamo che vale per n+1

Induzione Matematica

$$\frac{P(0) \qquad P(n) \Rightarrow P(n+1))}{\forall n \,.\, P(n)}$$

- Per dimostrare che una proprietà vale per tutti i numeri naturali, dimostriamo che
- vale per 0
- se assumiamo che vale per n, dimostriamo che vale per n+1

Esempio: Valore Minore di Array


```
int minimo(int X[], int dim) {
/* POST restituisce Y=min(X[0],...,X[dim-1])*/
if(dim==0)
        return -1; //ERRORE - vettore vuoto
    if(dim==1)
        return X[0];
    else {
        int min_vettore = minimo(X+1, dim-1);
        return (X[0]<min_vettore)? X[0]: min_vettore;
    }
}
```

Caso base: X ha un elemento, X[0] è il minimo

Esempio: Valore Minore di Array


```
int minimo(int X[], int dim) {
if(dim==0)
        return -1; //ERRORE - vettore vuoto
    if(dim==1)
        return X[0];
    else {
        int min_vettore = minimo(X+1, dim-1);
        return (X[0]<min_vettore)? X[0]: min_vettore;
    }
}
```

Caso base: X ha un elemento, X[0] è il minimo

Esempio: Valore Minore di Array


```
int minimo(int X[], int dim) {
/* POST restituisce Y=min(X[0],...,X[dim-1])*/
...
   else {
      int min_vettore = minimo(X+1, dim-1); /* min_vettore=min(X[1],...X[dim-1) */
      return (X[0]<min_vettore)? X[0]: min_vettore;
   }
}
```

Caso ricorsivo: per induzione assumiamo che la POST sia vera per la chiamata ricorsiva, Y=min(X[1],...X[dim-1), e dimostriamo che minimo calcola il minimo di X[0],...,X[dim-1]: basta confrontare Y con X[0] e restituire il minimo dei due.

Esempio: Lunghezza Stringa


```
int lunghezza_stringa(char *p){
    /* PRE: p è un puntatore al primo carattere di una stringa (termina con \0)
        POST: restituisce la lunghezza della stringa puntata da p */
    if(*p == '\0')
        return 0;
    else
        return 1 + lunghezza_stringa(p+1);
}
```

Ricorsione in Coda

- Si ha una ricorsione in coda (tail recursion) quando ogni chiamata ricorsiva esegue il calcolo di valori intermedi e li passa all'invocazione successiva per un'ulteriore elaborazione.
- In questo modo è l'ultima chiamata quella che calcola il risultato finale.
- Questo deve poi solo essere passato indietro al chiamante con una semplice return alla terminazione di ciascuna delle chiamate.

La struttura tipica ha la forma seguente:

```
tipo funzione(tipo x) {
        y = espressione con x;
        return funzione(y);
}
```

Esempio di Ricorsione in Coda


```
int confronta_array(int *X, int *Y, int dim) {

        if (dim==0)
                return 1;
        else {
                if (X[0]!=Y[0])
                        return 0;
                else {
                        return confronta_array(X+1, Y+1, dim-1);
                }
        }
}
```

Ricorsione in Coda

- Nella ricorsione in coda, i dati automatici della chiamata a funzione, salvati sullo stack nei vari stack frame, non servono più alla funzione quando questa torna in esecuzione e alla chiusura vengono solo scartati: di ciascun stack frame serve solo l'indirizzo di ritorno (e il risultato finale)
- Una funzione che realizza la ricorsione in coda è facilmente riscrivibile come funzione iterativa con il vantaggio di consumare meno memoria e di essere più veloce (non ci sono chiamate a funzione)

Trasformazione da Ricorsione a Iterazione


```
tipo F_ric(tipo x) {

        if (casobase(x)) {
                istruzioni_casobase;
                return risultato;
        } else {
                istruzioni_nonbase;
                return F_ric(riduciComplessita(x));
        }
}
```

Trasformazione da Ricorsione a Iterazione


```
tipo F_iter(tipo x) {
while (!casobase(x)) {
        istruzioni_non_base;
        x = riduci_complessità(x);
}
istruzioni_casobase;
return caso_base;
```

Esempio di trasformazione


```
int confronta_array(int *X, int *Y, int dim) {
  if (dim==0)
          return 1;
  else {
    if (X[0]!=Y[0])
       return 0;
    else {
       return confronta_array(X+1, Y+1, dim-1);
    }
  }
}
```

```
int confronta_array(int *X, int *Y, int dim) {
int i=0;
while (i<dim) {
   if (X[i]!=Y[i])
       return 0;
   i+=1;
  }
  return 1;
}
```

Esercizio

Scrivere un programma che dato un numero in input verifichi con una funzione ricorsiva se è primo (restituire 1) oppure no (restituire 0).