TANDEM 2020 | Esercizi da Esame

Program. Avanzata, Algoritmi e Problem Solving

1 Simulazione

Tempo assegnato: 1 ora. Il candidato dovrà risolvere gli esercizi nel modo più rigoroso possibile, ma non è richiesta una soluzione formale: durante le lezioni abbiamo comunque trattato questi argomenti cercando di non eccedere in aspetti tecnici.

1.1 Esercizio: Correttezza.

> La **ricerca binaria** lavora su sequenze ordinate:
> data una sequenza `s` determina se un dato elemento `e`
> è presente in uno specifico intervallo della sequenza stessa.

L'algoritmo funziona secondo il metodo *"divide-et-impera"*: dato un intervallo `s[a:b]` della sequenza `s`
- calcola la posizione `m` dell'elemento di mezzo;
- se l'elemento di mezzo non è quello cercato;
- scarta la metà "sbagliata" dell'intervallo;
- e avvia una nuova ricerca con il nuovo intervallo.

Le *specifiche* del problema della ricerca binaria in versione ricorsivo possono essere così formaulate:

- **dominio**:
  - `s` è una sequenza di interi
  - `a, b, x` sono interi
- **codominio**: valori booleani (vero e falso)
- **pre-condizioni**:
  - `s` definita,
  - `0 <= a < b <= len(s)`
  - `x` intero
- **post-condizioni**
  - `ricerca(s,a,b,x) = true` se e solo se `x` compare in `s[a:b]`

Il procedimento può essere codificato mediante la sequente funzione `Python`:

```
def ricerca(s,a,b,x):
    m = (a+b)//2
    if x < s[m] :
        m = ricerca(s,a,m,x)
        elif s[m] < x:
            m = ricerca(s,m,b,x)
    return m
```

> Le specifiche e il codice contengono qualche errore!
> Prova ad elencarli sinteticamente.

1.1.1 Soluzione

- Specifica:
  - `b<len(s)` : l'estremo destro della ricerca non può coincidere con `len(s)` perché tale posizione non esiste;
  - la funzione restituisce un valore intero non negativo e non un booleano.
- Codice:
  - è un algoritmo ricorsivo, am manca il caso base: se la dimensione dell'intervallo di ricerca è nulla deve restituire un valore "impossibile", ad esempio `-1` ;
  - nelle chiamate ricorsive va la posizione `m` deve essere esclusa dalla ricerca e pertanto nella prima chiamata deve essere sostituita da `m-1` e nella seconda da `m+1` .

1.1.2 Algoritmo corretto

Specifiche.

- **dominio**:
  - `s` è una sequenza di interi
  - `a, b, x` sono interi
- **codominio**: valori booleani (vero e falso)
- **pre-condizioni**:
  - `s` definita,
  - `0 <= a < b <= len(s)`
  - `x` intero
- **post-condizioni**
  - `m = ricerca(s,a,b,x) ≥ 0` se e solo se `x` compare in `s[a:b]` ;

- in tal caso `s[m] == x`, altrimenti `m=-1`.

Codice.

```
def ricerca(s,a,b,x):
    if (a<=b):
            m = (a+b)//2
            if x < s[m] :
                    m = ricerca(s,a,m-1,x)
                    elif s[m] < x:
                        m = ricerca(s,m+1,b,x)
    else:
        m = -1
    return m
```

1.2 Esercizio: Numeri primi gemelli

> In matematica, si definiscono numeri primi gemelli due numeri primi che differiscono tra loro di due. Fatta eccezione per la coppia (2 , 3), questa è la più piccola differenza possibile fra due primi. Alcuni esempi di coppie di primi gemelli sono (3, 5), (5, 7), (11, 13), (17, 19).
> Fonte: [https://it.wikipedia.org/wiki/Numeri_primi_gemelli]

Scrivi il codice `Python` delle funzioni

```
    def is_primo(n):
        """Ritorna True se n è primo, False in caso contrario"""
        # TODO

    def print_gemelli(n):
        """Stampa le coppie di primi gemelli minori o uguali a
n"""
        # TODO
```

in modo tale che l'esecuzione dell'istruzione:

```
    print_gemelli(100)
```

produca il seguente *output*:

```
 3 : 5
5 : 7
11 : 13
17 : 19
29 : 31
41 : 43
59 : 61
71 : 73
```

1.3 Esercizio: Analisi dei Costi

> La successione di Fibonacci $F(n)$ (con n numero *naturale*, ossia intero non negativo) può essere così definita:

$$F(0) = F(1) = 1$$
$$F(n+2) = F(n) + F(n+1)$$

Il seguente codice traduce quasi esattamente il procedimento di calcolo suggerito dalla definizione:

```
def fib(n):
    f = 1
    if n>1:
        f = fib(n-1)+f(n-2)
    return f
```

Questo algoritmo, si sa, è molto inefficiente: richiede un tempo di esecuzione molto elevato.
Ma “consuma” anche risorse in spazio dal momento che ogni chiamata ricorsiva comporta la creazione di un ambiente locale di lavoro per la funzione.

> Stimare il costo computazionale in spazio motivando brevemente la risposta.

1.3.1 Soluzione

Osservazioni preliminari.

- L’algoritmo richiede uno spazio costante perché deve creare con le variabili locali l’ambiente di lavoro.
- Inoltre vanno considerate anche le chiamate ricorsive.
- Nella penultima linea ci sono due chiamate ricorsive, ma non sono valutate una alla volta!

Di conseguenza le richieste in spazio si possono esprimere per mezzo della sequente equazione (di ricorrenza):

$$S(n) = c + S(n-1)$$

che, sviluppata per sostituzione, dà

$$\ldots = c + c + S(n-2) = c + c + c + S(n-2) = \ldots = c \cdot n + S(0) = c(n+1)$$

Si può concludere che **il costo in spazio è lineare**.