Corso di Algoritmi e Strutture Dati
Anno Accademico 2018–2019
Prof. Giovanni Pighizzini

Pseudocodice

A. Ricerca in un array

A.1. Ricerca sequenziale

```
Algoritmo ricercaSequenziale (array A[0..n − 1], elemento x) → indice
  i ← 0
  while i < n and A[i] ≠ x do
  |  i ← i + 1
  if i = n then return −1
  else return i
```

A.2. Ricerca sequenziale

Ricercando dal fondo si evita la selezione finale presente nell'Algoritmo A.1.

```
Algoritmo ricercaSequenziale (array A[0..n − 1], elemento x) → indice
  i ← n − 1
  while i ≥ 0 and A[i] ≠ x do
  |  i ← i − 1
  return i
```

A.3. Ricerca binaria o dicotomica ricorsiva in un array ordinato

La funzione ricorsiva effettua la ricerca nella parte dell'array A che inizia all'indice sx e termina all'indice che *precede* dx (pertanto sx è l'indice del primo elemento della porzione di array da considerare, mentre dx è l'indice del primo elemento, tra quelli successivi, da non considerare).

```
Algoritmo ricercaBinaria (array A[0..n − 1], elemento x) → indice
  return ricercaRic(A, 0, n, x)

Funzione ricercaRic (array A, indice sx, indice dx, elemento x) → indice
  if dx ≤ sx then return −1
  else
  |  m ← (sx + dx)/2
  |  if x = A[m] then return m
  |  else if x < A[m] then
  |  |  return ricercaRic(A, sx, m, x)
  |  else
  |  |  return ricercaRic(A, m + 1, dx, x)
```

A.4. Ricerca binaria o dicotomica iterativa in un array ordinato

```
Algoritmo ricercaBinaria (array A[0..n − 1], elemento x) → indice
  sx ← 0
  dx ← n
  pos ← −1
  while sx < dx and pos = −1 do
      m ← (sx + dx)/2
      if x = A[m] then pos ← m
      else if x < A[m] then dx ← m
      else sx ← m + 1
  return pos
```

Note sullo pseudocodice

- *Indici degli array*
  Quando si definisce un array (in questo caso nei parametri degli algoritmi o della funzione), il range di indici viene indicato qualora sia rilevante per la scrittura dell'algoritmo. Quando non sia rilevante o sia chiaro dal contesto, il range viene omesso (come, in questo esempio, per il parametro A della funzione *ricercaRic*).

- *Operatori logici*
  Assumiamo che per congiunzione (*and*) e disgiunzione (*or*) sia utilizzata la *lazy evaluation*. Pertanto in una condizione della forma a **and** b, la condizione b viene valutata solo se la condizione a è vera. Analogamente in a **or** b la condizione b viene valutata solo se a è falsa. La parte di destra della condizione $i \geq 0$ **and** $A[i] \neq x$ nell'Algoritmo A.2 non è definita quando la parte di sinistra è falsa (non esistono posizioni di indice negativo). Pertanto senza la lazy evaluation la condizione non potrebbe essere calcolata (un discorso analogo può essere fatto per la condizione del ciclo nell'Algoritmo A.1).

- *Passaggio di parametri*

  – Tipi semplici.
  In assenza di indicazioni diverse, per i tipi semplici assumiamo che il passaggio avvenga sempre *per valore*: il parametro è una variabile locale in cui al momento della chiamata viene copiato il valore dell'argomento fornito. Pertanto una modifica del parametro non ha alcun effetto sull'argomento fornito nella chiamata. In questo esempio, *indice* è un intero ed è un tipo semplice. Se *elemento* è intero, anche per esso il passaggio avviene per valore.

  – Tipi strutturati.
  Per tipi strutturati, come gli array, in mancanza di altre indicazioni assumiamo il *passaggio per riferimento*: il parametro è una variabile puntatore in cui, al momento della chiamata, viene copiato un puntatore all'array passato come argomento. Pertanto il passaggio del parametro avviene in tempo costante (indipendente dalla lunghezza dell'array), lo spazio di memoria occupato per l'array nel record di attivazione è solo quello del puntatore, eventuali modifiche fatte dal sottoprogramma tramite il parametro (come negli algoritmi di ordinamento che studieremo successivamente) avvengono direttamente sull'array fornito tramite l'argomento. Anche nel caso di stringhe, oggetti (come nel linguaggio Java) o strutture (come nel linguaggio C), in mancanza di altre indicazioni assumiamo sempre che il passaggio avvenga *per riferimento* (e dunque in *tempo costante*).