Algoritmi e Strutture di Dati I

Massimo Franceschet

http://www.sci.unich.it/~francesc

Strutture di dati

Per risolvere un problema, un algoritmo rappresenta i valori di ingresso, l'informazione intermedia e i valori d'uscita mediante strutture di dati.

Una **struttura di dati** consiste di un insieme di **attributi** e un insieme di **operazioni**. Gli attributi sono valori che caratterizzano la struttura (ad esempio, dimensione, inizio, fine,...). Le operazioni sono procedure che manipolano la struttura (ricerca, inserimento, cancellazione, ...).

Strutture di dati

In questa parte del corso vedremo diverse strutture di dati per rappresentare insiemi di elementi. Un insieme è **statico** se non può variare la propria cardinalità; è **dinamico** se posso inserire nuovi elementi o cancellare elementi presenti nell'insieme.

Siamo interessati alle seguenti operazioni su insiemi:

- **Inserimento**: l'operazione inserisce un nuovo elemento nell'insieme;
- **Cancellazione**: l'operazione cancella un elemento dall'insieme;
- **Ricerca**: l'operazione verifica la presenza di un elemento in un insieme;
- **Massimo**: l'operazione trova il massimo dell'insieme;
- **Minimo**: l'operazione trova il minimo dell'insieme;

- **Successore**: l'operazione trova, se esiste, il successore di un elemento dell'insieme;
- **Predecessore**: l'operazione trova, se esiste, il prececessore di un elemento dell'insieme.

Inserimento e cancellazione vengono dette **operazioni di modifica**, e le altre **operazioni di interrogazione**. Le operazioni di modifica non sono ammesse su insiemi statici.

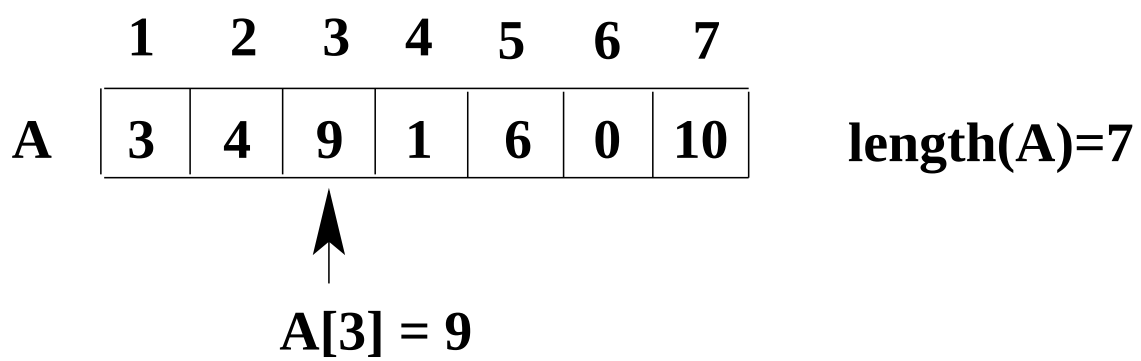
Un **dizionario** è un insieme dinamico che supporta le operazioni di modifica e l'operazione di ricerca.

Vettore (array)

La struttura di dati **vettore** (**array** in inglese) rappresenta un **insieme statico** di elementi.

Un vettore A possiede l'attributo $length(A)$ che contiene la **dimensione** del vettore. Gli elementi del vettore sono indicizzati da 1 a $length(A)$ e l'**operazione di accesso** $A[i]$ ritorna l'i-esimo elemento di A, qualora i sia un indice valido.

L'accesso agli elementi del vettore è **diretto**: posso accedere all'elemento i-esimo con una sola operazione senza dover passare per gli elementi che lo precedono. Il costo dell'operazione di accesso $A[i]$ è dunque **costante** $\Theta(1)$, indipendentemente dall'indice i.


1 2 3 4 5 6 7
A
3 4 9 1 6 0 10
length(A)=7
A[3] = 9

Scambio

Scambio *Sia A un vettore, i e j due indici di A. Scrivere una procedura $Exchange(A, i, j)$ che scambia gli elementi di $A[i]$ e $A[j]$.*

Qual è corretta?

Exchange(A,i,j)

$t \leftarrow A[i]$

$A[i] \leftarrow A[j]$

$A[j] \leftarrow t$

Exchange(A,i,j)

$A[i] \leftarrow A[j]$

$A[j] \leftarrow A[i]$

Copia

Copia *Scrivere una procedura* $Copy(A, B)$ *che copia* A *in* B.

Copia

Copia *Scrivere una procedura* $Copy(A, B)$ *che copia* A *in* B.

Copy(A,B)

```
length(B) ← length(A)
for i = 1 to length(A) do
    B[i] ← A[i]
end for
```

Copia

Copia *Scrivere una procedura $Copy(A, B)$ che copia A in B.*

Copy(A,B)

$length(B) \leftarrow length(A)$
for $i = 1$ **to** $length(A)$ **do**
    $B[i] \leftarrow A[i]$
end for

La complessità della procedura è $\Theta(n)$, con $n = length(A)$.

Inversione

Inversione *Sia* A *un vettore. Si scriva una procedura* $ArrayReverse(A)$ *che inverte la sequenza contenuta in* A.

Inversione

Inversione *Sia A un vettore. Si scriva una procedura $ArrayReverse(A)$ che inverte la sequenza contenuta in A.*

ArrayReverse(A)

for $i \leftarrow 1$ **to** $length(A)$ div 2 **do**
    $Exchange(A, i, length(A) - i + 1)$
end for

Inversione

Inversione *Sia A un vettore. Si scriva una procedura $ArrayReverse(A)$ che inverte la sequenza contenuta in A.*

ArrayReverse(A)

1: **for** $i \leftarrow 1$ **to** $length(A)$ div 2 **do**
2: $\quad Exchange(A, i, length(A) - i + 1)$
3: **end for**

La complessità della procedura è $\Theta(n/2) = \Theta(n)$, con $n = length(A)$.

Palindrome

Palindrome *Sia A un vettore. Si scriva una procedura ArrayPalindrome(A) che verifica se la sequenza contenuta in A è palindrome.*

Palindrome

Palindrome *Sia A un vettore. Si scriva una procedura ArrayPalindrome(A) che verifica se la sequenza contenuta in A è palindrome.*

ArrayPalindrome(A)

for $i \leftarrow 1$ **to** $length(A)$ *div* 2 **do**
    if $A[i] \neq A[length(A) - i + 1]$ **then**
        return FALSE
    end if
end for
return TRUE

Palindrome

Palindrome *Sia A un vettore. Si scriva una procedura* $ArrayPalindrome(A)$ *che verifica se la sequenza contenuta in A è palindrome.*

ArrayPalindrome(A)

for $i \leftarrow 1$ **to** $length(A)$ div 2 **do**
    if $A[i] \neq A[length(A) - i + 1]$ **then**
        return FALSE
    end if
end for
return TRUE

La complessità della procedura è $\Theta(n/2) = \Theta(n)$, con $n = length(A)$.

Concatenazione

Concatenazione *Si scriva una procedura* $ArrayConcat(A, B, C)$ *che mette in* C *la concatenazione delle sequenze contenute in* A *e in* B.

ArrayConcat(A,B,C)

```
length(C) ← length(B) + length(A)
if length(A) ≥ length(B) then
    n ← length(A)
else
    n ← length(B)
end if
for i ← 1 to n do
    if i ≤ length(A) then
        C[i] ← A[i]
    end if
    if i ≤ length(B) then
        C[length(A) + i] ← B[i]
    end if
end for
```

ArrayConcat(A,B,C)

$length(C) \leftarrow length(B) + length(A)$
if $length(A) \geq length(B)$ **then**
    $n \leftarrow length(A)$
else
    $n \leftarrow length(B)$
end if
for $i \leftarrow 1$ **to** n **do**
    if $i \leq length(A)$ **then**
        $C[i] \leftarrow A[i]$
    end if
    if $i \leq length(B)$ **then**
        $C[length(A) + i] \leftarrow B[i]$
    end if
end for

La complessità è $\Theta(max\{a, b\})$, con $a = length(A)$ e $b = length(B)$.

Cosa torna?

```
Copy(A, B)
Reverse(B)
Concat(A, B, C)
return Palindrome(C)
```

Massimo

Massimo *Scrivere una procedura* $ArrayMax(A)$ *che ritorna l'indice dell'elemento massimo di* A*.*

Massimo

Massimo *Scrivere una procedura* $ArrayMax(A)$ *che ritorna l'indice dell'elemento massimo di* A.

ArrayMax(A)

$max \leftarrow 1$
for $i \leftarrow 2$ **to** $length(A)$ **do**
    if $A[i] > A[max]$ **then**
        $max \leftarrow i$
    end if
end for
return max

Massimo

Massimo *Scrivere una procedura* $ArrayMax(A)$ *che ritorna l'indice dell'elemento massimo di* A.

ArrayMax(A)

$max \leftarrow 1$
for $i \leftarrow 2$ **to** $length(A)$ **do**
    if $A[i] > A[max]$ **then**
        $max \leftarrow i$
    end if
end for
return max

La complessità della procedura è $\Theta(n)$, con $n = length(A)$.

Massimo – Versione ricorsiva

Massimo *Sia A un vettore e $i \leq j$ due indici di A. Si scriva una procedura ricorsiva $ArrayMax(A, i, j)$ che restituisce l'indice dell'elemento massimo del sottovettore $A[i, \ldots, j]$, oppure* NIL *se tale sottovettore è vuoto.*

Uso la tecnica algoritmica nota come **dividi e conquista**: divido il problema in sottoproblemi sempre più piccoli. Problemi più piccoli sono in genere più facili da conquistare (risolvere).

ArrayMax(A,i,j)

```
if i > j then
    return NIL
else
    if i = j then
        return i
    else
        l ← (j − i + 1) div 2
        m_1 ← ArrayMax(A, i, i + l − 1)
        m_2 ← ArrayMax(A, i + l, j)
        if A[m_1] > A[m_2] then
            return m_1
        else
            return m_2
        end if
    end if
end if
```

Complessità della ricerca del massimo

La complessità $C(n)$ di $ArrayMax(A, i, j)$, ove $n = j - i + 1$, è espressa dalla seguente equazione ricorsiva:

$$\begin{aligned} C(1) &= 1 \\ C(n) &= 2C(n/2) + 1 \end{aligned}$$

La soluzione è

$$C(n) = \Theta(n)$$

Successore

Successore *Dato un vettore A e un suo elemento x, l'elemento successore di x è il più piccolo elemento di A che è più grande di x. Sia A un vettore e i un suo indice. Scrivere una procedura $ArraySuccessor(A, i)$ che ritorna l'indice dell'elemento successore di $A[i]$, oppure* NIL *se $A[i]$ è il massimo di A.*

Successore

ArraySuccessor(A,i)

$s \leftarrow ArrayMax(A)$
if $A[i] = A[s]$ **then**
  return NIL
end if
for $j \leftarrow 1$ **to** $length(A)$ **do**
  if $A[j] > A[i]$ **and** $A[j] < A[s]$ **then**
    $s \leftarrow j$
  end if
end for
return s

Successore

ArraySuccessor(A,i)

$s \leftarrow ArrayMax(A)$
if $A[i] = A[s]$ **then**
    return NIL
end if
for $j \leftarrow 1$ **to** $length(A)$ **do**
    if $A[j] > A[i]$ **and** $A[j] < A[s]$ **then**
        $s \leftarrow j$
    end if
end for
return s

La complessità della procedura è $\Theta(n)$, con $n = length(A)$.

Ordinamento

Ordinamento *Si scriva una procedura* $SlowSort(A)$ *che ordina in senso crescente il vettore* A *usando le procedure* $ArrayMin$ *e* $ArraySuccessor$.

Ordinamento

Ordinamento *Si scriva una procedura* $SlowSort(A)$ *che ordina in senso crescente il vettore* A *usando le procedure* $ArrayMin$ *e* $ArraySuccessor$.

SlowSort(A)

$length(B) \leftarrow length(A)$
$j \leftarrow ArrayMin(A)$
for $i \leftarrow 1$ **to** $length(B)$ **do**
    $B[i] \leftarrow A[j]$
    $j \leftarrow ArraySuccessor(A, j)$
end for
$Copy(B, A)$

Ordinamento

Ordinamento *Si scriva una procedura* $SlowSort(A)$ *che ordina in senso crescente il vettore* A *usando le procedure* $ArrayMin$ *e* $ArraySuccessor$*.*

SlowSort(A)

$length(B) \leftarrow length(A)$
$j \leftarrow ArrayMin(A)$
for $i \leftarrow 1$ **to** $length(B)$ **do**
    $B[i] \leftarrow A[j]$
    $j \leftarrow ArraySuccessor(A, j)$
end for
$Copy(B, A)$

La complessità della procedura è $\Theta(n^2)$, con $n = length(A)$.

Ricerca

Ricerca *Sia A un vettore e x un numero intero. Scrivere una procedura $ArraySearch(A, x)$ che ritorna l'indice dell'elemento che contiene x, oppure* NIL *se tale elemento non esiste.*

Ricerca

Ricerca *Sia A un vettore e x un numero intero. Scrivere una procedura $ArraySearch(A, x)$ che ritorna l'indice dell'elemento che contiene x, oppure* NIL *se tale elemento non esiste.*

ArraySearch(A,x)

```
i ← 1
while i ≤ length(A) and A[i] ≠ x do
    i ← i + 1
end while
if i ≤ length(A) then
    return i
else
    return NIL
end if
```

Ricerca

Ricerca *Sia A un vettore e x un numero intero. Scrivere una procedura $ArraySearch(A, x)$ che ritorna l'indice dell'elemento che contiene x, oppure* NIL *se tale elemento non esiste.*

ArraySearch(A,x)

$i \leftarrow 1$
while $i \leq length(A)$ **and** $A[i] \neq x$ **do**
  $i \leftarrow i + 1$
end while
if $i \leq length(A)$ **then**
  return i
else
  return NIL
end if

La complessità della procedura è $\Theta(n)$, con $n = length(A)$.

Ricerca binaria

Ricerca binaria *Sia A un vettore di n elementi* ***ordinato*** *in senso crescente e x un numero. Scrivere una procedura iterativa* ***efficiente*** $ArrayBinarySearch(A, x)$ *che ritorna l'indice dell'elemento che contiene x, oppure* NIL *se tale elemento non esiste.*

ArrayBinarySearch(A,x)

```
left ← 1
right ← length(A)
repeat
    i ← (left + right) div 2
    if x < A[i] then
        right ← i − 1
    else
        left ← i + 1
    end if
until x = A[i] or left > right
if A[i] = x then
    return i
else
    return NIL
end if
```

Complessità della ricerca binaria

Il calcolo della complessità si riduce alla seguente questione: quante volte occorre suddividere a metà una sequenza di lunghezza n prima di ottenere una sequenza di lunghezza inferiore a 1?

Ciò equivale a chiederci quante volte occorre moltiplicare 2 per se stesso prima di ottenere un numero maggiore o uguale a n. La risposta è $\lceil \log n \rceil$, in quanto $2^{\lceil \log n \rceil} \geq 2^{\log n} = n$.

Dunque la complessità di $ArrayBinarySearch(A, x)$ è $\Theta(\log n)$.

Ricerca binaria – Versione ricorsiva

Ricerca binaria *Sia A un vettore ordinato in senso crescente di lunghezza n, x un numero e i, j due indici del vettore A. Scrivere una procedura* ***ricorsiva*** *efficiente $ArrayBinarySearch(A, i, j, x)$ che ritorna l'indice dell'elemento che contiene x cercando nel sottovettore $A[i, \ldots, j]$, oppure* NIL *se tale elemento non esiste.*

Ricerca binaria – Versione ricorsiva

```
ArrayBinarySearch(A, i, j, x)
if i > j then
    return NIL
end if
if i = j then
    if A[i] = x then
        return i
    else
        return NIL
    end if
end if
```

$l \leftarrow (j - i + 1)\ div\ 2$
if $x = A[i + l]$ **then**
    return $i + l$
else
    if $x < A[i + l]$ **then**
        return $ArrayBinarySearch(A, i, i + l - 1, x)$
    else
        return $ArrayBinarySearch(A, i + l + 1, j, x)$
    end if
end if

Complessità della ricerca binaria

La complessità $C(n)$ di $ArrayBinarySearch(A, i, j, x)$, ove $n = j - i + 1$, è espressa dalla seguente equazione ricorsiva:

$$
\begin{aligned}
C(1) &= 1 \\
C(n) &= C(n/2) + 1
\end{aligned}
$$

La soluzione è

$$C(n) = \Theta(\log n)$$