Algoritmi e Strutture di Dati

Capitolo 8 - Insiemi e dizionari

Insiemi realizzati con vettori booleani

- **Insieme**
  - **Interi 1...*N***
  - **Collezione di *N* oggetti memorizzati in un vettore**
- **Rappresentazione dell'insieme:**
  - **Vettore booleano di *N* elementi**
- **Vantaggi**
  - **Notevolmente semplice**
  - **Efficiente verificare se un elemento appartiene all'insieme**
- **Svantaggi**
  - **Occupazione di memoria *O*(*N*), indipendente dalla dimensione dell'insieme**
  - **Operazioni inefficienti *O*(*N*)**

Insiemi realizzati con vettori booleani

SET (vettore booleano)

```
boolean[] V
integer dim
integer capacità

SET Set(integer N)
    SET t ← new SET
    t.dim ← 0
    t.capacità ← N
    t.V ← new boolean[1 ... N]
    for integer i ← 1 to N do V[i] ← false
    return t

integer size()
    return dim

boolean contains(integer x)
    return V[x]

insert(integer x)
    if not V[x] then
        dim ← dim + 1
    V[x] ← true

remove(integer x)
    if V[x] then
        dim ← dim − 1
    V[x] ← false

SET union(SET A, SET B)
    C ← Set(max(A.capacità, B.capacità))
    for integer i ← 1 to A.capacità do
        if A.contains(i) then C.insert(i)
    for i ← 1 to B.capacità do
        if B.contains(i) then C.insert(i)

SET intersection(SET A, SET B)
    C ← Set(min(A.capacità, B.capacità))
    for integer i ← 1 to min(A.capacità, B.capacità) do
        if A.contains(i) and B.contains(i) then C.insert(i)

SET difference(SET A, SET B)
    C ← Set(A.capacità)
    for integer i ← 1 to A.capacità do
        if A.contains(i) and (i > B.capacità or not B.contains(i)) then
            C.insert(i)
```

Insiemi realizzati con liste non ordinate

- **Vantaggi**
  - **Occupazione di memoria proporzionale alla dimensione del vettore**

- **Svantaggi**
  - **Operazioni di ricerca, inserimento e cancellazione: $O(n)$**
  - **Operazioni di unione, intersezione e differenza: $O(nm)$**

Insiemi realizzati con liste non ordinate

```
SET difference(SET A, SET B)
    C ← Set()
    p ← A.head()
    r ← C.head()
    while not A.finished(p) do
        if not B.contains(A.read(p)) then
            C.insert(A.read(p), r)
            p ← A.next(p)
    return C
```

Insiemi realizzati con liste ordinate

- **Vantaggi**
  - **Occupazione di memoria proporzionale alla dimensione del vettore**
  - **Operazioni di unione, intersezione e differenza: $O(n+m)$**
- **Svantaggi**
  - **Operazioni di ricerca, inserimento e cancellazione: $O(n)$**

Insiemi realizzati con liste ordinate

```
SET intersection(SET A, SET B)
    C ← Set()
    p ← A.head()
    q ← B.head()
    r ← C.head()
    while not A.finished(p) and not B.finished(q) do
        if A.read(p) = B.read(q) then
            C.insert(A.read(p), r)
            p ← A.next(p)
            q ← B.next(q)
        else if A.read(p) < B.read(q) then
            p ← A.next(p)
        else
            q ← B.next(q)
    return C
```

Realizzazione con strutture di dati complesse

- **Con alberi bilanciati di ricerca**
  - **Ricerca, inserimento, cancellazione: $O(\log n)$**
  - **Viene mantenuto l'ordinamento**
  - **Elencare tutti gli elementi: $O(n)$**

Operazioni insiemistiche (unione, intersezione e differenza)

- **Con alberi bilanciati di ricerca (Esercizio 6.7)**
  - **Unione: $O(n + m \log (n+m))$, dove $m = \min\{|A|, |B|\}$**
  - **Intersezione: $O(m \log n)$, dove $m = \min\{|A|, |B|\}$**
  - **Differenza ($C = A - B$): $O(\min\{n + m \log n,\ n(\log m + \log n)\})$, dove $n = |A|$**