

8 - Insiemi

Definizione

Un insieme è una **collezione di elementi dello stesso tipo** (omogenei).

A differenza delle liste, in un insieme:

1. Gli elementi **non hanno una posizione**.
2. Gli elementi **non possono apparire più di una volta** (non ci sono duplicati).

In informatica, gli insiemi sono memorizzati **estensionalmente**, cioè elencando i loro membri (es. `{giallo, rosso, blu}`), a differenza della matematica dove possono essere definiti per **intensionalmente**, ovvero per proprietà.

Operazioni

I concetti e le operazioni eseguibili sugli insiemi si basano sui concetti degli insiemi matematici:

- **Cardinalità:** È il numero di elementi nell'insieme, indicato come $|A|$.
- **Relazione Fondamentale:** È la relazione di **appartenenza** ($x \in A$), che verifica se un elemento x è membro dell'insieme A .
- **Operazioni Principali:** Vengono definite le tre operazioni binarie fondamentali:
 - **Unione ($A \cup B$):** L'insieme di tutti gli elementi che sono in A , o in B , o in entrambi.
 - **Intersezione ($A \cap B$):** L'insieme degli elementi che sono *sia* in A *che* in B .
 - **Differenza ($A \setminus B$):** L'insieme degli elementi che sono in A ma *non* sono in B .

Specifica Sintattica

- **Tipi:** `insieme`, `boolean`, `tipoelem`.
- **Operatori:**

Operatori Base dell'Insieme

Operatore	Input (Dominio)	Output (Codominio)	Descrizione
<code>Creainsieme</code>	<code>()</code>	<code>insieme</code>	Crea un nuovo insieme vuoto.
<code>Insiemevuoto</code>	<code>(insieme)</code>	<code>boolean</code>	Controlla se un insieme è vuoto.
<code>Appartiene</code>	<code>(tipoelem, insieme)</code>	<code>boolean</code>	Verifica l'appartenenza di un elemento.

Operatore	Input (Dominio)	Output (Codominio)	Descrizione
Inserisci	(tipoelem, insieme)	insieme	Aggiunge un elemento a un insieme.
Cancella	(tipoelem, insieme)	insieme	Rimuove un elemento da un insieme.

- **creainsieme** = A : L'insieme A è l'insieme vuoto, $A = \{\}$.
- **insiemevuoto**(A) = b : b è vero se $A = \{\}$, altrimenti falso .
- **appartiene**(x , A) = b : b è vero se $x \in A$, altrimenti falso .
- **inserisci**(x , A) = A' : Questa funzione necessita di non avere un elemento x presente nell'insieme, per non avere duplicati. Avendo ciò otterremmo un nuovo insieme A' è $A' = A \cup x$.
- **cancella**(x , A) = A' : Diversamente dalla funzione precedente, qui si necessita che l'elemento x sia presente nell'insieme per poter essere cancellato.

Attenzione: questi operatori potrebbero anche essere definiti senza pre-condizioni (ad esempio, inserire un elemento già presente non fa nulla).

Operazioni Binarie

Operatore	Input (Dominio)	Output (Codominio)
Unione	(insieme, insieme)	insieme
Intersezione	(insieme, insieme)	insieme
Differenza	(insieme, insieme)	insieme

Questi definiscono formalmente le operazioni matematiche:

- **unione**(A , B) = C :
 - **Post-condizione:** $C = A \cup B$.
- **intersezione**(A , B) = C :
 - **Post-condizione:** $C = A \cap B$
- **differenza**(A , B) = C :
 - **Post-condizione:** $C = A \setminus B$

Realizzazioni

Rappresentazione con Vettore Booleano

Conosciuto anche come **vettore caratteristico**.

Si usa un array di valori booleani (`true` / `false`). Questo approccio funziona solo se gli elementi dell'insieme appartengono ad un **universo conosciuto e limitato** (ad esempio, numeri interi da 1 a N). La k – *esima* posizione del vettore sarà `true` se l'elemento k **appartiene** all'insieme, e `false` altrimenti.

- **Esempio:** Per rappresentare l'insieme $\{1, 3, 5\}$ su un universo $[1, 5]$, il vettore sarebbe: `[true, false, true, false, true]`.

Rappresentazione mediante Lista

Si utilizza una struttura dati `Lista` (come quelle viste nei capitoli precedenti) per memorizzare gli elementi che appartengono all'insieme.

- **Vantaggio:** Questo metodo è molto più **flessibile**. Permette agli elementi di essere di qualsiasi tipo (non solo interi) e non richiede di conoscere in anticipo l'universo di appartenenza.
- **Svantaggio:** Poiché le liste permettono duplicati e gli insiemi no, l'implementazione deve **assicurare esplicitamente l'assenza di duplicati**. Questo introduce un costo aggiuntivo nelle operazioni.

Realizzazioni con Liste non Ordinate

In questa implementazione, l'insieme è rappresentato tramite **una lista concatenata semplice**, in cui i nuovi elementi vengono aggiunti **in testa**.

- **Occupazione di memoria:**
Utilizzando strutture dinamiche basate su puntatori, la memoria occupata risulta **proporzionale al numero di elementi effettivamente presenti**. Ciò rende la struttura molto efficiente in termini di spazio, soprattutto quando l'insieme contiene pochi elementi.
- **Operatore inserisci:**
L'inserimento di un nuovo elemento avviene in modo diretto e veloce, seguendo due passaggi principali:
 1. **Controllo di appartenenza:** si verifica prima che l'elemento non sia già presente nella lista.
 2. **Inserimento:** se l'elemento è assente, lo si aggiunge nel punto più rapido possibile, ossia **in testa alla lista**, ad esempio **in testa alla lista** (un'operazione $O(1)$ per la lista stessa).

Poiché la lista non ha un ordine, ogni ricerca richiede di controllarla potenzialmente tutta.

- **appartiene :** Per verificare se un elemento è presente, si deve **scorrere tutta la lista** nel caso peggiore (se l'elemento non c'è o è l'ultimo).
- **inserisci :** Deve prima chiamare `appartiene` (scansione della lista) e, se l'elemento non è presente, lo inserisce (solitamente in testa, un'operazione veloce $O(1)$). Il costo totale è dominato dalla scansione.
- **cancella :** È simile ad `appartiene`. Bisogna **scorrere la lista** per trovare l'elemento e, una volta trovato, rimuoverlo.
- **Operatori Binari (unione , intersezione , differenza):** Richiedono **scansioni multiple** della lista:

- **Intersezione ($A \cap B$):** Si scorre la lista A , per ogni elemento di A , si deve scorrere tutta la lista B per vedere se appartiene anche a B . Se sì, lo si inserisce in C .
- **Differenza ($A \setminus B$):** Si scorre la lista A . Per ogni elemento di A , si deve scorrere tutta la lista B per vedere se appartiene a B . Se **non** appartiene, lo si inserisce in C .
- **Unione ($A \cup B$):** Si può copiare tutta B in C . Poi si scorre A e per ogni elemento di A , si controlla se appartiene a C ; se non appartiene, lo si inserisce in C .

Realizzazioni con Liste Ordinate

Se è definita una relazione \leq di **ordinamento totale sugli elementi dell'insieme**, esso può essere rappresentato con una lista ordinata per valori crescenti degli elementi.

Impatto sulle Operazioni:

- **appartiene** : L'operazione di ricerca diventa più efficiente . Non è più necessario scorrere sempre tutta la lista; si può smettere di cercare non appena si trova un elemento più grande di quello cercato, se esiste.
- **inserisci / cancella** : Richiedono ancora una scansione (nel caso peggiore) per trovare la posizione corretta in cui inserire o rimuovere per mantenere l'ordine.
- **Operatori Binari (unione , intersezione , differenza)**: Questi diventano **molto più efficienti**. Invece di fare ricerche $n \times m$, è possibile scorrere le due liste ordinate in parallelo (in modo simile all'algoritmo di *merge*), completando l'operazione in un'unica passata, la complessità passa da $O(n \times m)$ a circa $O(n + m)$.

Altre Implementazioni

Con **dizionario** (soluzione ottimale), in questo caso il dizionario conterrà solo chiavi che sono gli elementi dell'insieme o con **albero bilanciato**. Li vedremo nei capitoli seguenti

Applicazione: Il Setaccio di Eratostene

Un applicazione classica dell'ADT degli insiemi per risolvere un problema: trovare tutti i numeri primi in un dato intervallo.

In quel contesto, **ADT (Abstract Data Type)**, o *Tipo di Dato Astratto*, è un modello **logico** per un tipo di dato:

1. **cosa può fare** (le operazioni, come **Inserisci** , **Cancella** , **Appartiene**).
2. **che tipo di dati contiene** (elementi unici, in questo caso).
3. **NON dice come è fatto dentro** (l'implementazione).

La parte fondamentale è **Astratto**: l'ADT nasconde la complessità.

Esempio:

- **Problema**: Trovare tutti i numeri primi nell'intervallo da 2 a n , con $n > 2$.
- **Algoritmo (Setaccio di Eratostene)**: L'algoritmo utilizza due insiemi:

1. Un insieme "*setaccio*".
2. Un insieme "*numeri primi*".

- **Procedura:**

1. **Inizializzazione:** Su *inserisci* sono tutti i numeri da 2 a n nell'insieme "*setaccio*".
2. **Ciclo("finché il setaccio non è vuoto"):** Finché l'insieme "*setaccio*" non è *insieme vuoto*. Si prende il numero minimo presente nel "*setaccio*" (che sarà il prossimo numero primo, p), e si includono i "*numeri primi*".
Si *cancella* p dal "*setaccio*" e lo si *inserisci* nell'insieme "*numeri primi*". Alla fine del ciclo si *cancella* dal "*setaccio*" tutti i **multipli** di p (cioè $2p, 3p, 4p, \dots$ fino a superare n). Se il "*setaccio*" non è vuoto bisogna ripetere i passi dal punto 2.
3. **Risultato:** Al termine del processo, l'insieme "*numeri primi*" conterrà tutti i numeri primi da 2 a n .