Alessandro Dubini 885957 a.dubini@campus.unimib.it

Relazione progetto C++

Progetto testato con:

- MSVC 19 (Windows 11)
- Clang++ 15.0.0 (MacOS)
- g++ 9.4.0 (Ubuntu VM)
- g++ 11.4.0 (Ubuntu)

Descrizione:

La classe Set implementa un set generico, con la possibilità di definire un funtore personalizzato per il confronto di uguaglianza tra elementi.

Fornisce operazioni comuni come l'aggiunta, la rimozione, la verifica della presenza di un elemento, l'iterazione, e supporta operatori di unione e intersezione tra set.

La classe è progettata per permettere di gestire un numero variabile di elementi.

Scelte di implementazione:

- **Struttura utilizzata per l'implementazione**: Array di tipo generico **T**

  La scelta è stata fatta per rispettare i requisiti progettuali.

  I principali vantaggi sono (rispetto ad un'implementazione basata sulle liste):

  1. Compattezza della memoria (cache locality).
  2. Tempi di accesso minori (non vi è necessità di attraversare più nodi per accedere all'i-esimo elemento).
  3. Tempo di aggiunta inferiore nel caso in cui non serva ridimensionare l'array (non vi è necessità di allocare altra memoria).
  4. Verifica di unicità dell'elemento più rapida (grazie al punto 1).

  Alcuni svantaggi di questo approccio:

  1. Tempo per l'aggiunta di un elemento maggiore nel caso in cui sia necessario ridimensionare l'array.

2. Tempo di rimozione maggiore dovuto alla necessità di shiftare a sinistra i dati posizionati alla destra dell'elemento rimosso.
3. Grande quantità di memoria inutilizzata nel caso in cui venisse rimosso un gran numero di elementi da un Set di grandi dimensioni.

NOTA BENE: Al fine di poter utilizzare un array come struttura e usufruire dei vantaggi sopra elencati è necessario che tutte le classi che vengono utilizzate con questo Set definiscano il costruttore di default (privo di parametri) o un costruttore che possa essere chiamato senza richiedere argomenti (con argomenti dotati di valore di default).

Questo è per garantire che, nel momento in cui viene creato un array di tipo **T** di mCapacity elementi, i suoi elementi si trovino in uno stato coerente.

Esistono altre soluzioni a questa problematica, come l'utilizzo di un array di puntatori a T, ma avrebbero comportato la perdita dei vantaggi derivanti dalla scelta di un array come struttura di base o violato l'indicazione del progetto che specifica di non utilizzare funzionalità di basso livello (malloc e free).

- **Scelta di non utilizzare add nel costruttore di copia e nell'operatore di assegnamento:**

  In entrambi questi metodi, dato che si sta creando un oggetto Set da un altro oggetto dello stesso tipo, l'unicità degli elementi è garantita.

  Per tale ragione, ho scelto di non utilizzare il metodo **add** al fine di evitare controlli di unicità ridondanti sugli elementi, preferendo effettuare la copia utilizzando direttamente le strutture interne dei due oggetti.

  Il metodo **add**, invece, è sfruttato nel costruttore con iteratori poiché questi potrebbero fornire accesso a una collezione che consente la presenza di elementi duplicati.

- **Fattore di crescita della capacità del set:**

$$\boldsymbol{mCapacity} = \begin{cases} \mathbf{2}, & \boldsymbol{mCapacity} = \mathbf{0} \\ \boldsymbol{mCapacity} + \left(\frac{\boldsymbol{mCapacity}}{\mathbf{2}}\right), & \boldsymbol{mCapacity} \geq \mathbf{2} \end{cases}$$

Scelto con l'obiettivo di fornire una quantità di spazio sufficiente per le operazioni di aggiunta successive, andando a limitare il numero di **resize** chiamate, che sia però abbastanza contenuto da evitare un eccessivo spreco di memoria.

- **Tipo di ritorno dei metodi add e remove:**

  Al fine di segnalare all'utente l'avvenuta aggiunta/rimozione di un elemento ho deciso di ritornare un bool il cui valore è true se l'operazione ha avuto successo e false altrimenti.
  Condizioni di fallimento del metodo **add**: l'elemento che si sta aggiungendo è già presente nel set.
  Condizioni di fallimento del metodo **remove**: l'elemento che si sta tentando di rimuovere non è presente nel set.

- **Aggiunta del metodo empty:**

  Il metodo **empty**, la cui funzione è quella di svuotare il set, è stato introdotto con l'ulteriore fine di lasciare la lista in uno stato coerente nel caso in cui si verificassero eccezioni (ad esempio durante l'allocazione della memoria tramite **new**).

- **Scelta del tipo di iteratore costante:** forward iterator:

  Il set è una struttura dati la cui unica caratteristica distintiva è l'unicità degli elementi in esso contenuti e non vi è alcuna garanzia dell'ordine con cui essi sono memorizzati.

  Il fatto che attualmente gli elementi siano memorizzati nello stesso ordine in cui vengono aggiunti al set è una mera coincidenza dovuta all'implementazione corrente.

  Pertanto, ho ritenuto privo di senso ed utilità introdurre altre tipologie di iteratori (bidirectional o random).