

Specifica assiomatica della "Lista Doppia-mente Concatenata"

Revisione basata su implementazione moderna in C++

1 Descrizione

Questa specifica presenta una descrizione formale della struttura dati *"Lista Doppia-mente Concatenata"*, presente nel repository, definendo sia gli aspetti sintattici (struttura e interfaccia) che semantici (comportamento delle operazioni) attraverso un approccio assiomatico, allineato a un'implementazione C++ che utilizza `std::unique_ptr`.

2 Definizione Sintattica

2.1 Tipi e Strutture

`DoublyLinkedList<T>` è una struttura dati parametrizzata su un tipo generico `<T>` che rappresenta una sequenza di elementi connessi in entrambe le direzioni.

2.1.1 Strutture Interne

- `Node<T>`: struttura interna che contiene:
 - `data` : `T` - il valore dell'elemento
 - `next` : `std::unique_ptr<Node<T>>` - puntatore proprietario al nodo successivo
 - `prev` : `Node<T>*` - puntatore non proprietario (raw) al nodo precedente
- `iterator`: classe interna che fornisce accesso in lettura/scrittura agli elementi.
- `const_iterator`: classe interna per l'accesso in sola lettura agli elementi.

2.1.2 Attributi Principali

`DoublyLinkedList<T>` mantiene i seguenti attributi:

- `head_` : `std::unique_ptr<Node<T>>` - puntatore proprietario al primo nodo
- `tail_` : `Node<T>*` - puntatore non proprietario (raw) all'ultimo nodo
- `size_` : `size_t` - numero di elementi nella lista

3 Interfaccia Sintattica

3.1 Costruttori, Distruttore, Assegnazione

```
// Costruttore di default
DoublyLinkedList();

// Costruttore e assegnazione di copia sono disabilitati
DoublyLinkedList(const DoublyLinkedList&) = delete;
DoublyLinkedList& operator=(const DoublyLinkedList&) = delete;

// Costruttore e assegnazione di spostamento
DoublyLinkedList(DoublyLinkedList&& other);
DoublyLinkedList& operator=(DoublyLinkedList&& other);

// Distruttore
~DoublyLinkedList();
```

3.2 Operazioni di Accesso e Stato

```
T& front();
const T& front() const;
T& back();
const T& back() const;
bool is_empty() const;
size_t size() const;
```

3.3 Operazioni di Modifica

```
void push_front(const T& value);
void push_front(T&& value);
void push_back(const T& value);
void push_back(T&& value);
void pop_front();
void pop_back();

template<typename... Args> T& emplace_front(Args&&... args);
template<typename... Args> T& emplace_back(Args&&... args);

iterator insert(iterator pos, const T& value);
iterator insert(iterator pos, T&& value);
iterator erase(iterator pos);
void clear();
void reverse();
```

3.4 Iteratori

```
iterator begin();
iterator end();
const_iterator begin() const;
```

```
const_iterator end() const;
const_iterator cbegin() const;
const_iterator cend() const;
```

4 Specifica Semantica Assiomatica

Definiamo il comportamento di ogni operazione tramite assiomi.

4.1 Notazione

- L denota una lista di tipo `DoublyLinkedList<T>`
- $L[i]$ denota l'elemento in posizione i (base 0)
- $|L|$ denota la lunghezza (size) della lista L
- $L = [e_0, e_1, \dots, e_{n-1}]$ rappresenta una lista di n elementi
- L' denota lo stato della lista dopo un'operazione
- \emptyset denota una lista vuota

4.2 Invarianti di Struttura

$\forall L : \text{DoublyLinkedList}\langle T \rangle :$

1. $L.\text{size_} = |L|$
2. Se $|L| = 0$ allora $L.\text{head_} = \text{nullptr} \wedge L.\text{tail_} = \text{nullptr}$
3. Se $|L| > 0$ allora $L.\text{head_} \neq \text{nullptr} \wedge L.\text{tail_} \neq \text{nullptr}$
4. Se $|L| = 1$ allora $L.\text{head_}.get() = L.\text{tail_}$
5. Se $|L| > 0$, il nodo contenente $L[0]$ ha il suo campo `prev` uguale a `nullptr`
6. Se $|L| > 0$, il nodo puntato da `tail_` ha il suo campo `next` uguale a `nullptr`

4.3 Costruttori

4.3.1 Costruttore di Default

`DoublyLinkedList<T>()` $\Rightarrow L'$, dove $|L'| = 0$

4.3.2 Costruttore di Spostamento

`DoublyLinkedList<T>(&&L)` $\Rightarrow (L'_{new}, L'_{old})$, dove:

- L'_{new} contiene tutti gli elementi originariamente in L
- L'_{old} (stato di L dopo l'operazione) è una lista vuota

4.4 Operazioni di Accesso e Stato

4.4.1 front

Pre: $|L| > 0$

$L.\text{front}() \Rightarrow e$, dove $e = L[0]$

4.4.2 back

Pre: $|L| > 0$

$L.back() \Rightarrow e$, dove $e = L[|L| - 1]$

4.4.3 is_empty

$L.is_empty() \Rightarrow b$, dove $b = (|L| = 0)$

4.4.4 size

$L.size() \Rightarrow s$, dove $s = |L|$

4.5 Operazioni di Modifica

4.5.1 push_front

$L.push_front(e) \Rightarrow L'$, dove:

- $|L'| = |L| + 1$
- $L'[0] = e$
- $\forall i \in \{1, \dots, |L|\} : L'[i] = L[i - 1]$

4.5.2 push_back

$L.push_back(e) \Rightarrow L'$, dove:

- $|L'| = |L| + 1$
- $L'[|L|] = e$
- $\forall i \in \{0, \dots, |L| - 1\} : L'[i] = L[i]$

4.5.3 pop_front

Pre: $|L| > 0$

$L.pop_front() \Rightarrow L'$, dove:

- $|L'| = |L| - 1$
- $\forall i \in \{0, \dots, |L'| - 1\} : L'[i] = L[i + 1]$

4.5.4 pop_back

Pre: $|L| > 0$

$L.pop_back() \Rightarrow L'$, dove:

- $|L'| = |L| - 1$
- $\forall i \in \{0, \dots, |L'| - 1\} : L'[i] = L[i]$

4.5.5 insert (con iteratore pos)

Pre: **pos** è un iteratore valido in $[L.begin(), L.end())$

$L.insert(pos, e) \Rightarrow L'$

4.5.6 erase (con iteratore pos)

Pre: pos è un iteratore valido e dereferenziabile in L
 $L.\text{erase}(\text{pos}) \Rightarrow L'$

4.6 Iteratori

4.6.1 begin/cbegin

$L.\text{begin}() \Rightarrow it$, dove:

- Se $|L| > 0$, allora it è un iteratore a $L[0]$
- Se $|L| = 0$, allora $it = L.\text{end}()$

5 Complessità Temporale

Operazione	Complessità	Descrizione
Costruttore default/move	$O(1)$	Operazioni a tempo costante.
Distruttore	$O(n)$	Deallocazione lineare di n elementi.
front/back	$O(1)$	Accesso diretto agli elementi tramite puntatori <code>head_</code> e <code>tail_</code> .
is_empty/size	$O(1)$	Lettura di attributi interni.
push_front/push_back	$O(1)$	Aggiornamento dei puntatori di testa/coda.
emplace_front/emplace_back	$O(1)$	Come sopra.
pop_front/pop_back	$O(1)$	Aggiornamento dei puntatori di testa/coda.
insert (con iteratore)	$O(1)$	Inserimento in tempo costante data la posizione.
erase (con iteratore)	$O(1)$	Rimozione in tempo costante data la posizione.
clear	$O(n)$	Richiede la deallocazione di tutti i nodi.
reverse	$O(n)$	Richiede una scansione lineare della lista.

Table 1: Tabella delle complessità computazionali