Progetto "Lista Concatenata con Zipper"

Appello del 7 Settembre 2020

Descrizione del progetto

Lo scopo di questo progetto è l'implementazione di una versione "zipper" delle lista concatenate, che chiameremo *lista concatenata con zipper*. Molte strutture dati (tra cui anche le liste) possono avere una nozione aggiuntiva di "posizione corrente" all'interno della struttura.

In particolare, noi vogliamo implementare una struttura dati che si comporti come una lista concatenata con l'aggiunta di un cursore che indica la posizione corrente nella lista e che possa essere aggiornato spostandolo avanti e indietro lungo la lista.

Si vuole inoltre permettere inserimento e rimozione nella posizione corrente in modo efficiente (i.e., tempo costante). Potete pensare a questa struttura come un modo di rappresentare un testo in un editor di testo in cui è presente un cursore che può essere mosso lungo il testo e con tutti gli inserimenti che vengono effettuati nel punto in cui si trova il cursore.

Supponiamo di avere una sequenza che contiene, in ordine, i valori $[x_1, x_2, \ldots, x_n]$ ed in cui il cursore è nella posizione $i \in \{1, \ldots, n\}$. Questa sequenza verrà rappresentata da due liste concatenate singole:

- Una lista concatenata singola dei valori precedenti la posizione i-esima in ordine inverso: $[x_{i-1}, \ldots, x_1]$.
- Una lista concatenata singola nella cui testa è il valore corrente (i.e., quello su cui è il cursore) e, a seguire, tutti i valori successivi: $[x_i, \ldots, x_n]$.

Notate come nel caso il cursore sia in prima posizione la lista dei valori precedenti sarà vuota, mentre la seconda lista sarà vuota solo se non sono contenuti elementi nell'intera lista concatenata (dato che, altrimenti, conterrebbe almeno il valore corrente).

Con tale struttura lo spostamento in avanti del cursore consiste nel rimuovere l'elemento corrente dalla testa della lista e spostarlo nella lista dei valori precedenti (Figura 1).

Lo spostamento all'indietro del cursore consiste invece nel prendere l'elemento di testa dalla lista dei valori precedenti e inserirlo in testa alla seconda lista come elemento corrente (Figura 2).

Si vuole implementare tale struttura dati come una classe `ListaConZipper` in modo che supporti le seguenti operazioni:

- `valore_cursore()`, che ritorna il valore indicato dal cursore o `None` se non sono presenti elementi.

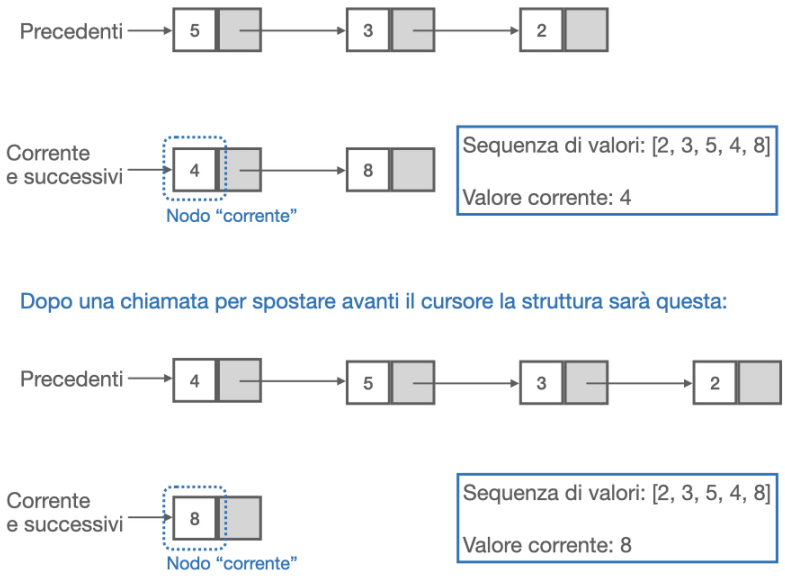

Figura 1: Esempio di lista concatenata con zipper e spostamento del cursore in avanti

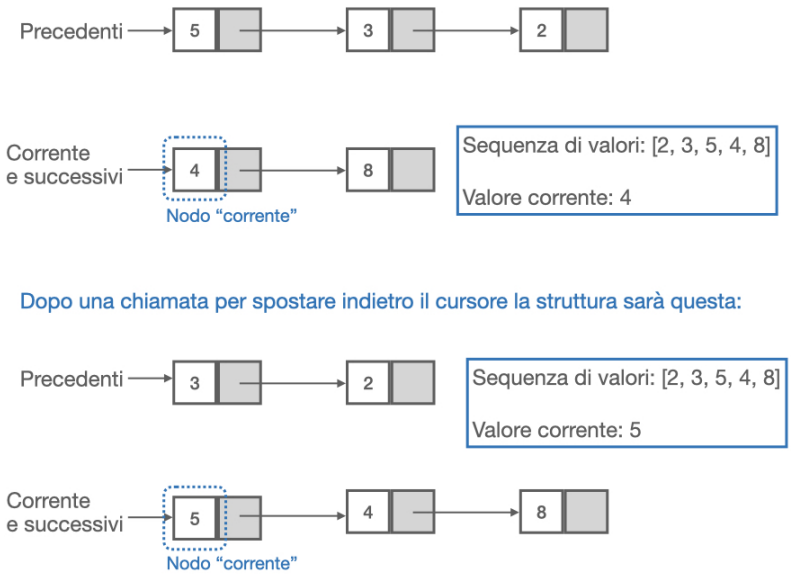

Figura 2: Esempio di lista concatenata con zipper e spostamento indietro del cursore

- `avanti()`, che sposta avanti il cursore. Se il cursore è già in ultima posizione questa operazione non ha effetto.
- `indietro()`, che sposta indietro il cursore. Se il cursore è già in prima posizione l'operazione non ha effetto.
- `inizio()`, che ritorna `True` nel caso il cursore sia in prima posizione della lista e `False` altrimenti.
- `fine()`, che ritorna `True` nel caso il cursore sia nell'ultima posizione della lista e `False` altrimenti.
- `inserisci_prima(valore)`, che inserisce il valore prima della posizione corrente.
- `inserisci_dopo(valore)`, che inserisce il valore dopo la posizione corrente. Il valore inserito deve diventare il nuovo valore corrente (i.e., il cursore viene spostato)
- `rimuovi()`, che rimuove il valore indicato dal cursore. Il cursore si sposterà in avanti a meno che non fosse già sull'ultimo elemento della lista. In quel caso deve rimanere sull'ultimo elemento della lista.

Tutte queste operazioni devono essere effettuate in *tempo costante*.

Si richiede inoltre l'implementazione di un metodo `__str___` che permetta di visualizzare tutto il contenuto della lista dall'elemento iniziale a quello finale e che racchiuda il valore corrente tra parentesi quadre. Per esempio, nelle lista iniziale di Figura 1 il metodo dovrà ritornare la stringa `"2 3 5 [4] 8`. Questa operazione *non* può essere effettuata in tempo costante.

Codice

Viene fornito uno scheletro del codice che deve essere implementato, con i metodi che devono essere scritti:

```
class ListaConZipper:

    def __init__(self):
        # ...

    def valore_cursore(self):
        # ...

    def avanti(self):
        # ...

    def indietro(self):
        # ...

    def inizio(self):
        # ...
```

```
    def fine(self):
        # ...

    def inserisci_prima(self, valore):
        # ...

    def inserisci_dopo(self, valore):
        # ...

    def rimuovi(self):
        # ...

    def __str__(self):
        # ...
```

Nel progetto è consentito avere funzioni e classi aggiuntive per l'implementazione. Per esempio, per rappresentare i nodi delle liste.

Si ricorda di commentare adeguatamente il codice, approfittandone per spiegare le scelte implementative effettuate.

Esempi d'uso

Il seguente frammento di codice chiama tutti i metodi la cui implementazione è richiesta dal progetto. Come commento è indicato il valore atteso in una implementazione funzionante:

```
def test():
    seq = ListaConZipper()
    seq.inserisci_dopo(3)
    seq.inserisci_dopo(4)
    print(seq)  # Stampa "3 [4]"
    seq.indietro()
    print(seq)  # Stampa "[3] 4"
    seq.inserisci_prima(2)
    print(seq)  # Stampa "2 [3] 4"
    seq.avanti()
    seq.inserisci_dopo(7)
    print(seq)  # Stampa "2 3 4 [7]"
    while not seq.inizio():
        seq.indietro()
    print(seq)  # Stampa "[2] 3 4 7"
    seq.inserisci_prima(1)
    print(seq)  # Stampa "1 [2] 3 4 7"
    while not seq.fine():
        seq.avanti()
    seq.inserisci_dopo(8)
```

```
print(seq)  # Stampa "1 2 3 4 7 [8]"
seq.indietro()
print(seq.valore_cursore())   # Stampa "7"
```

Suggerimenti

1. Quando viene richiesto l'utilizzo di liste concatenate **non** si intendono le liste di Python che, come visto a lezione, sono più assimilabili ad array.
2. Sebbene per risolvere lo stesso problema possano esistere altre strutture dati, per l'esame è da implementare la struttura dati descritta nel testo del progetto.
3. Rispettate i vincoli di complessità richiesti.
4. Fate più test possibile: assicuratevi che il codice implementato funzioni in tutti i casi richiesti.

Indicazioni

Il progetto deve essere svolto **individualmente**. La consegna dovrà avvenire entro le ore 23:59 del giorno 28/08/2020 secondo le seguenti modalità:

- Invio di una email a `lmanzoni@units.it` dal vostro account email istituzionale con oggetto *[informatica] consegna progetto appello del 07/09/2020*.
- L'email deve avere come allegato il progetto in un singolo file in codice sorgente Python versione 3, dal nome `Nome_Cognome_matricola.py`, quindi, per esempio Mario Rossi di matricola 12345 consegnerà un file dal nome `Mario_Rossi_12345.py`.
- Il file deve contenere sotto forma di commento le seguenti linee indicanti nome, cognome e numero di matricola:

  ```
  # Nome: Mario
  # Cognome: Rossi
  # Matricola: 12345
  ```

Note aggiuntive

Sono molteplici le strutture dati che possono essere dotate di zipper, per esempio gli alberi possono avere una nozione di "nodo corrente". Gli zipper sono simili a strutture date come i *gap buffer*, che sono utilizzati negli editor di testo (e.g., Emacs) in quanto supportano operazioni di modifica efficienti nelle vicinanze del cursore.