Laboratorio di algoritmi e strutture dati

Liste concatenate semplici, liste con puntatore a *tail*, liste bidirezionali*

Docente: Violetta Lonati

1 Implementazione di liste concatenate semplici

Scrivete un programma con l'implementazione di una lista concatenata semplice, seguendo i lucidi presentati a lezione. Definite i tipi `listNode` e `linkedList` e scrivete queste funzioni:

- `newNode`, che crea un nuovo nodo di lista;
- `addNewNode`, che inserisce un nuovo nodo in testa alla lista;
- `printList`, che stampa una lista;
- `searchList`, che cerca un elemento in una lista;
- `removeItem`, che cancella un *item* da una lista.

Per testare le vostre funzioni scrivete una funzione `main` che gestisca un insieme dinamico (che variano nel tempo) di interi. Il main deve leggere da standard input una sequenza di istruzioni secondo il formato nella tabella qui sotto, dove n è un intero. I vari elementi sulla riga sono separati da uno o più spazi. Quando una riga è letta, viene eseguita l'operazione associata; le operazioni di stampa sono effettuate sullo standard output, e ogni operazione deve iniziare su una nuova riga.

*Ultimo aggiornamento: 26 ottobre 2022 - 12:21:11

Istruzione in input	Operazione
`+` n	Se n non appartiene all'insieme lo inserisce, altrimenti non fa nulla
`-` n	Se n appartiene all'insieme lo elimina, altrimenti non fa nulla
`?` n	Stampa un messaggio che dichiara se n appartiene all'insieme
`c`	Stampa il numero di elementi dell'insieme
`p`	Stampa gli elementi dell'insieme (nell'ordine in cui compaiono nella lista)
`o`	Stampa gli elementi dell'insieme nell'ordine inverso
`d`	Cancella tutti gli elementi dell'insieme
`f`	Termina l'esecuzione

Si assume che l'input sia inserito correttamente. Conviene scrivere le istruzioni di input in un file `in.txt` ed eseguire il programma redirigendo lo standard input.

2 Questioni di efficienza

Nell'esercizio precedente avete implementato un insieme di interi usando una lista concatenata. Come valutate l'efficienza di questa implementazione? Quali sono le operazioni più onerose? Quale struttura dati alternativa si potrebbe usare per rendere tali operazioni più efficienti? Ci sarebbero altri svantaggi?

3 Lista con puntatore a *tail*

Considerate ora una lista con concatenata di interi, dotata di due riferimenti al primo e all'ultimo elemento della lista, come descritto dal tipo `linkedListwithTail` nella porzione di codice qui sotto. Quando la lista è vuota, sia `head` che `tail` sono `nil`. La funzione `newNode` alloca lo spazio per un nuovo nodo e ne inizializza il valore con l'argomento passato.

```
1  type listNode struct {
2    item int
3    next *listNode
4  }
5
6  type linkedListWithTail struct {
7    head, tail *listNode
8  }
9
10 func newNode(val int) *listNode {
11   return &listNode{val, nil}
12 }
```

3.1 Inserimento alla fine

Considerate la seguente funzione (incompleta) che aggiunge un elemento `e` in fondo alla lista:

```
1 func addNewNodeAtEnd(l *linkedListWithTail, val int) {
2   if l.tail == nil {
3     l.tail = newNode(val)
4     l.head = l.tail
5   } else {
6     // MISSING CODE
7   }
8 }
```

Quale dei seguenti frammenti di codice completa la funzione `addAtEnd`? Spiegate quali problemi si riscontrano scegliendo ciascuna delle altre opzioni.

A)
```
l.tail.next = val
l.tail = val
```

B)
```
temp := newNode(val)
l.tail.next = temp
```

C)
```
temp := newNode(val)
l.tail = temp
```

D)
```
l.tail.next = newNode(val)
l.tail = l.tail.next
```

E)
```
temp := l.head
for temp.next != nil {
  temp = temp.next
}
temp.next = newNode(val)
```

3.2 Confronto tra lista semplice e lista con `tail`

Nella lista concatenata `linkedListWithTail` abbiamo i riferimenti all'inizio e alla fine della lista. Confrontate questa implementazione con quella di una lista semplice che non ha un riferimento alla fine della lista, cioè definita come segue:

```
type linkedList struct {
  head *listNode
}
```

Per quali delle seguenti operazioni si ha un tempo migliore con `linkedListWithTail` piuttosto che con `linkedList`? Scegliete tutte le opzioni corrette e giustificate la risposta.

A) restituisci 1 se la lista contiene un dato elemento
B) cancella l'ultimo elemento della lista
C) aggiungi un elemento all'inizio della lista
D) aggiungi un elemento alla fine della lista

4 Lista semplice o bidirezionale?

Avete l'implementazione di una lista, ma non sapete se si tratta di una lista semplice (con solo il riferimento all'inizio della lista) oppure una lista bidirezionale (con riferimenti all'inizio e alla fine della lista, e in cui ogni nodo ha riferimenti al nodo successivo e al precedente).

Avete accesso alla lista solo tramite questa interfaccia:

- `size()` restituisce il numero di elementi nella lista
- `contains(e)` restituisce 1 se `e` è nella lista
- `removeAtStart()` cancella l'elemento all'inizio della lista
- `removeAtEnd()` cancella l'elemento alla fine della lista
- `addAtStart(e)` inserisce l'elemento `e` all'inizio della lista
- `addAtEnd(e)` inserisce l'elemento `e` alla fine della lista

Indicate quali dei seguenti esperimenti consente di stabilire quale sia l'implementazione sconosciuta. Se ci sono più opzioni valide, scegliete la migliore. Giustificate la risposta.

A) Realizzo una mia implementazione della lista semplice e confronto i tempi di esecuzione su tutte le funzioni. Se i tempi della mia implementazione coincidono con quelli della implementazione data per tutte le funzioni, allora si tratta di una lista semplice, altrimenti di una lista bidirezionale.

B) Eseguo N operazioni `addAtEnd` seguite da N operazioni `addAtStart`. Se le operazioni `addAtEnd` ci impiegano molto di più di quelle `addAtStart`, allora si tratta di una lista semplice, altrimenti una lista bidirezionale.

C) Eseguo N operazioni `addAtEnd` seguite da N operazioni `removeAtEnd`. Se le operazioni `removeAtEnd` ci impiegano molto di più di quelle `addAtEnd`, allora si tratta di una lista semplice, altrimenti di una lista bidirezionale.

D) Creo due istanze della lista e confronto il tempo di esecuzione della prima rispetto alla seconda, per tutte le operazioni. Se i tempi sono simili per tutte le operazioni, allora si tratta di una lista semplice, altrimenti di una lista bidirezionale.

E) Nessuno degli esperimenti precedenti mi consente di rispondere alla domanda. Bisogna esaminare il codice per vedere se l'implementazione usa un riferimento `prev` al nodo precedente.