Упражнение 7.2: Двусвязное отсортированное множество 36 Punkte

Verkettete Listen, Mengenoperationen

Эта задача заключается в реализации структуры данных для отсортированного набора записей типа int, то есть отсортированного списка без повторяющихся записей. Его следует сохранить как двусвязный список. Класс также должен овладеть типичными операциями над множеством, такими как пересечение, объединение и множество различий.

Элементы в односвязных списках знают только своего преемника; в двусвязных списках каждый элемент также знает своего предшественника (см. Рисунок 1). Ссылка на предшественник первого элемента списка и ссылка на преемник последнего элемента списка устанавливаются в ноль.

Класс количеств управляет ссылкой «head» на первый и ссылкой «tail» на последний элемент списка, оба из которых установлены в ноль в пустых списках.

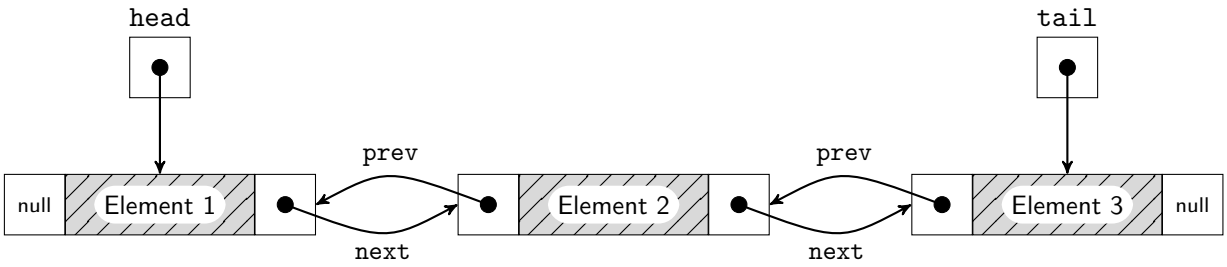

Рисунок 1: Схема двусвязного списка.

Примечание. Если не указано иное, видимость программы должна быть максимально ограничена.

Предупреждение: Избегайте использования классов из Java API в вашей реализации, за исключением String и NoSuchElementException. Разумеется, разрешены атрибуты примитивных типов, которые вам нужны для «управления» внутренним состоянием.

1. 1. Создайте новый проект 07-SortedSet.
2. В этом упражнении мы хотим использовать исключение, которое еще не является частью Java API. Поэтому создайте новый класс ElementExistsException, который должен наследовать от RuntimeException. Реализуйте стандартный конструктор и конструктор с сообщением об ошибке (в виде строки) в качестве параметров передачи. Вызвать соответствующие конструкторы суперкласса в конструкторах.
3. Реализуйте в новом классе `SortedSet` предоставленный нами 7- material.zip интерфейс `OrderedSet`.
4. В SortedSet создайте открытый внутренний класс ListItem, который будет представлять запись списка.

Обычно класс ListItem реализуется частным образом. Однако по техническим причинам абсолютно необходимо, чтобы ListItem был общедоступным!

a) Каждая запись списка хранит своего предшественника и преемника в двух ссылках, previous и next, а также свое значение в value.

b) Единственный конструктор должен сохранять значение, переданное в качестве параметра.

c) Переопределите метод toString (), унаследованный от Object. Он должен вернуть следующее строковое представление объекта ListItem, если есть преемник:

```
[⟨value⟩] -->
```

и следующее строковое представление, если нет преемника:

```
[⟨value⟩]
```

d) Кроме того, переопределите логический метод equals (Object other) для Object. Он должен возвращать истину тогда и только тогда, когда параметр также относится к типу ListItem и хранит то же значение.

5. Каждый экземпляр SortedSet хранит ссылку `head` на первую и ссылку `tail` auf den letzten Listeneintrag.

6. в SortedSet указанные там методы теперь должны быть реализованы с использованием комментариев JavaDoc в OrderedSet. По возможности используйте уже реализованные методы вместо копирования кода.

Важно: тщательно протестируйте все реализованные методы и оставьте их закомментированными в основном методе. Оценим и ваши тесты!

7. После этого основной метод доставки должен выполнять следующие действия:

a) Создайте два пустых набора типа OrderedSet и инициализируйте их объектами типа SortedSet.

b) Вывести количество на `stdout`.

c) Заполните первую сумму одну за другой пунктами 1, 6, 4, 2, 7, 5 и 12.

d) Заполните второе количество цифрами 5, 23, 22, 7 и 9 за один вызов.

e) Выведите обе величины на `stdout`.

f) e) Удалите элемент 22 из второго набора. Также попробуйте удалить элемент 77. Выявите ошибки и выведите сообщение об ошибке в stderr.

g) Найдите в первом наборе элементы в интервале [2, 6], создайте набор из них и выведите его на `stdout`.

h) h) Вычислить множества пересечений, объединений и различий наборов из 7a) и вывести их на stdout. Ловите любые исключения с сообщением об ошибке на stderr!

i) Создайте новый набор с элементами 18, 19 и 20. Вычислите объединение с первым набором.