Algoritmos e Estruturas de Dados - ISCTE, 2021/2002 – Projeto B

Queues / Filas

Escreva um tipo de dados genérico para um *deque* e uma *fila aleatória* (*randomized queue*) . O objetivo é implementar estruturas de dados elementares usando vetores (arrays) de dimensão variável e listas ligadas e introduzir a utilização de tipos genéricos e iteradores.

Dequeue. Uma *fila de extremidade dupla* ou *deque* ("deck") é a generalização de uma pilha e uma fila que suporta a adição e remoção de itens quer do cabeça ou da cauda da estrutura de dados. Crie um tipo de dados genérico `Deque` que implemente a seguinte API:

```
public class Deque<Item> implements Iterable<Item> {

    // construct an empty deque
    public Deque()

    // is the deque empty?
    public boolean isEmpty()

    // return the number of items on the deque
    public int size()

    // add the item to the front
    public void addFirst(Item item)

    // add the item to the back
    public void addLast(Item item)

    // remove and return the item from the front
    public Item removeFirst()

    // remove and return the item from the back
    public Item removeLast()

    // return an iterator over items in order from front to back
    public Iterator<Item> iterator()

    // unit testing (required)
    public static void main(String[] args)

}
```

Condições fronteira. Lance a exceção especificada para os seguintes casos

• `IllegalArgumentException` se o cliente chamar `addFirst()` ou `addLast()` com um argumento nulo,
• `java.util.NoSuchElementException` se o cliente chamar `removeFirst()` ou `removeLast()` quando a deque estiver vazia,
• `java.util.NoSuchElementException` se o cliente chamar o método `next()` no iterador quando não houver mais itens para retornar,

Testes unitários. O método `main()` deve chamar diretamente todos os construtores e métodos públicos para verificar se funcionam conforme prescrito (por exemplo, imprimindo resultados em `stdout`).

Requisitos de desempenho. A implementação deve atender aos seguintes requisitos de desempenho de pior caso:
• Um *deque* contendo n itens deve usar, no máximo, *48n + 192* bytes de memória, não incluindo a memória dos próprios itens,
• Cada operação de *deque* (incluindo construção) deve levar tempo constante.
• Cada operação do iterador (incluindo construção) deve levar tempo constante.

Fila aleatória. Uma fila aleatória é semelhante a uma pilha ou fila, exceto que o item removido é escolhido de maneira uniforme e aleatória entre os itens da estrutura de dados. Crie um tipo de dados genérico *RandomizedQueue* que implemente a seguinte API:

```
public class RandomizedQueue<Item> implements Iterable<Item> {

    // construct an empty randomized queue
    public RandomizedQueue()

    // is the randomized queue empty?
    public boolean isEmpty()

    // return the number of items on the randomized queue
    public int size()

    // add the item
    public void enqueue(Item item)

    // remove and return a random item
    public Item dequeue()

    // return a random item (but do not remove it)
    public Item sample()

    // return an independent iterator over items in random order
    public Iterator<Item> iterator()

    // unit testing (required)
    public static void main(String[] args)

}
```

Iterador. Cada iterador deve retornar os itens em ordem aleatória uniforme. A ordem de dois ou mais iteradores para a mesma fila aleatória deve ser mutuamente independente; cada iterador deve manter sua própria ordem aleatória.

Condições fronteira. Lance a exceção especificada para os seguintes casos:

• `IllegalArgumentException` se o cliente chamar `enqueue()` com um argumento nulo.
• `java.util.NoSuchElementException` se o cliente chamar `sample()` ou `dequeue()` quando a fila aleatória estiver vazia.
• `java.util.NoSuchElementException` se o cliente chamar o método `next()` no iterador quando não houver mais itens para retornar.

Testes unitários. O método `main()` deve chamar diretamente todos os construtores e métodos públicos para verificar se eles funcionam conforme prescrito (por exemplo, imprimindo resultados na `stdout`).

Requisitos de desempenho

• Uma fila aleatória contendo n itens deve usar no máximo 48n + 192 bytes de memória, sem incluir a memória dos próprios itens.
• Cada operação de fila aleatória (além de criar um iterador) deve levar um tempo amortizado constante. Ou seja, a partir de uma fila aleatória vazia, qualquer sequência misturada de `m` tais operações deve levar Θ (`m`) passos na pior das hipóteses.
• Um iterador sobre n itens deve usar no máximo 8n + 72 bytes de memória.
• A construção de um iterador deve levar Θ(n) passos; as operações `next()` e `hasNext()` devem levar tempo constante.

Cliente. Escreva um programa cliente `Permutation.java` que receba um inteiro k como argumento na linha de comando, lê uma sequência de *strings* de *stdin* usando `StdIn.readString()`, e imprime exatamente k deles, de forma uniformemente aleatória. Imprima cada item da sequência no máximo uma vez.

```
~/Desktop/queues> more distinct.txt
A B C D E F G H I

~/Desktop/queues> java-algs4
Permutation 3 < distinct.txt
C
G
A

~/Desktop/queues> java-algs4
Permutation 3 < distinct.txt
E
F
G
```

```
~/Desktop/queues> more duplicates.txt
AA BB BB BB BB BB CC CC

~/Desktop/queues> java-algs4
Permutation 8 < duplicates.txt
BB
AA
BB
CC
BB
BB
CC
BB
```

O programa deve implementar a seguinte API:

```
public class Permutation {
   public static void main(String[] args)
}
```

Argumento de linha de comando. Pode supor que $0 \le k \le n$, onde n é o número de *strings* na entrada padrão. Note que não recebe n.

Requisitos de desempenho. A implementação deve atender aos seguintes requisitos de desempenho de pior caso:
• O tempo de execução da `Permutation` deve ser linear relativamente ao tamanho da entrada.
• Pode usar apenas uma quantidade constante de memória mais um objeto Deque ou RandomizedQueue de tamanho máximo no máximo n.
• Como desafio extra, use apenas uma quantidade constante de memória mais um objeto `Deque` ou `RandomizedQueue` de tamanho máximo no máximo k.

Entrega. Envie of programa `RandomizedQueue.java`, `Deque.java`, and `Permutation.java`, junto com `readme.txt`. O trabalho deve apenas chamar as funções de `StdIn`, `StdOut`, `StdRandom`, `java.lang`, `java.util.Iterator`, e `java.util.NoSuchElementException`. Em particular, não use `java.util.LinkedList` nem `java.util.ArrayList`.

This assignment was developed by Kevin Wayne.
Original version
https://www.cs.princeton.edu/courses/archive/fall20/cos226/assignments/queues/specification.php