UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

Algoritmos e Estruturas de Dados

2021-2022

Projeto 1 – Coleções

Introdução

No 1.º Projeto pretende-se explorar a criação e utilização de estruturas de dados usadas para representar coleções de objetos.

Não é necessário ter um problema total implementado para obter a cotação correspondente no projecto. Mesmo que não tenha tempo para implementar algum dos métodos pedidos, deverá escrever o cabeçalho do mesmo, para que o *Mooshak* consiga compilar o código e executar os testes sobre os métodos que implementou.

Problema A: Implementação de Listas Ligadas Desenroladas (12 valores)

Implemente a classe UnrolledLinkedList, que implementa uma lista ligada desenrolada. Uma lista ligada desenrolada, é semelhante a uma lista ligada mas em que cada um dos nós da lista corresponde a um bloco de um determinado tamanho, e que pode guardar vários elementos. A figura seguinte ilustra uma lista desenrolada, composta por 3 blocos de tamanho 4[1] O 1.º bloco tem 3 elementos, o 2.º bloco tem 2 elementos, e o último bloco apenas um elemento. Para além dos elementos, cada bloco contém um contador com o número de elementos guardados nesse bloco, e um apontador para o próximo bloco (caso exista)[2].

Pesquisa de elementos: a pesquisa de um elemento numa determinada posição consegue ser feita de forma mais eficiente do que numa lista ligada tradicional (mas nunca será tão eficiente como a pesquisa num *array*), pois saltamos de bloco em bloco (subtraindo ao índice o número de elementos do bloco) até encontrar o bloco onde está guardado o elemento. Lembre-se que a posição 0 corresponde ao início da lista. Assim sendo, no exemplo acima, a pesquisa pelo índice 0 retornará o número 1 (bloco 0, índice 0), enquanto a pesquisa pelo índice 4 retornará o número 5 (bloco 1, índice 1).

Inserção de novos elementos: a operação de inserção começa por procurar o bloco correto onde inserir, inserindo o elemento. Se o bloco já estiver cheio, devemos criar e inserir um novo bloco a seguir a este, e mover metade dos elementos do bloco atual para o novo bloco.

Remoção de elementos: procuramos a posição a remover, e removemos o elemento do bloco correspondente, fazendo um "*shift*" para a esquerda de todos os elementos seguintes do bloco (caso existam). Se depois da remoção, o bloco ficar sem elementos, e se não for o único bloco

[1] Embora tenha mais dados, considera-se o tamanho do bloco o número de elementos que cada bloco consegue guardar.
[2] O bloco poderá ter outros campos adicionais, se acharem necessário (por exemplo um ponteiro para o bloco anterior).

existente, esse bloco deve ser "ignorado", bastando para isso fazer com que o bloco anterior passe a apontar para o seguinte (ver imagem seguinte).

A sua classe UnrolledLinkedList deve implementar a interface IList<T> fornecida, cujos métodos estão definidos na tabela seguinte. Para além destes métodos, deverá também implementar os construtores, e os método definidos abaixo.

IList<T> – representa uma lista de elementos do tipo *T.* Uma lista deve suportar obrigatoriamente as seguintes funcionalidades.		
`void`	`add(T item)`	Adiciona um elemento no fim da lista
`void`	`add(int index, T item)`	Adiciona um elemento numa determinada posição da lista.
`T`	`remove()`	Remove e retorna o item no fim da lista. Caso a lista esteja vazia deverá retornar *null*
`T`	`remove(int index)`	Remove e retorna o item numa determinada posição da lista. Se não existir nenhum elemento nessa posição, retorna *null*.
`T`	`get(int index)`	Retorna o elemento numa determinada posição da lista. Se não existe nenhum elemento nessa posição, retorna *null*.
`void`	`set(int index, T item)`	Substitui o elemento numa determinada posição pelo elemento recebido[3]. Se a posição recebida não for válida nada acontece.
`boolean`	`isEmpty()`	Retorna *true* se a lista estiver vazia e *false* caso contrário
`int`	`size()`	Devolve o tamanho (número de elementos) da lista
`IList<T>`	`shallowCopy()`	Retorna uma cópia superficial da lista. Uma cópia superficial copia a estrutura da lista sem copiar cada item individualmente
`Iterator<T>`	`Iterator()`	Retorna um iterador com estado para iterar sobre os elementos da lista. A lista deverá ser iterada do início para o fim. Este iterador deverá implementar os métodos hasNext() e next() obrigatoriamente.
UnrolledLinkedList<Item>		
	`UnrolledLinkedList()`	Cria uma nova lista enrolada vazia
	`UnrolledLinkedList(int blockSize)`	Cria uma nova lista enrolada vazia, usando blocos com o tamanho recebido
`void`	`main(String[] args)`	Método *main*, que deverá ser usado para testar empiricamente alguns dos métodos acima
`T[][]`	`getArrayOfBlocks()`	Método que retorna um array com todos os blocos da lista ligada desenrolada. Ou seja, cada posição contem um array de objetos do tipo T, e que corresponde aos items dos blocos. Por exemplo a lista na imagem inicial irá retornar o array: [[1,2,3,null],[4,5,null,null],[6,null,null,null]].

[3] A diferença entre este método e o método add, é que o set substitui o elemento anterior, enquanto que o add adiciona um novo elemento na lista (sem remover o que lá estava).

Testes empíricos

Utilize testes empíricos (por exemplo ensaios de razão dobrada) para determinar o tempo de execução e a ordem de crescimento temporal dos métodos add(int index) e get(int index). Utilize testes empíricos para determinar qual o valor mais apropriado para usar como tamanho *default* do bloco, tendo em conta a eficiência dos métodos *add* e *get*. Escreva um sumário dos testes e resultados, incluindo os valores de r (razão dobrada), e uma breve descrição como comentários no ficheiro Java a submeter

O método *main* deverá implementar os testes e ensaios utilizados. Embora este método não seja validado de forma automática pelo *Mooshak* será tido em consideração na validação do projeto, e contará para a nota final do mesmo.

Requisitos técnicos: Não poderá usar na sua implementação nenhuma das coleções nativas do Java (*ArrayList*,*LinkedList*,*Vector*,etc).

Implemente a classe *UnrolledLinkedList* no ficheiro *UnrolledLinkedList.java*.. A classe deverá estar definida na package *aed.collections*[4]. Outras classes auxiliares deverão ser definidas no mesmo ficheiro.

Submeta **apenas o ficheiro UnrolledLinkedList.java** no Problema A (não é necessário submeter a interface).

Problema B: Implementação de Filas através de Listas Ligadas Enroladas (8 Valores)

Para o problema B pretende-se implementar o tipo fila usando uma estrutura de dados baseada na ideia das listas ligadas enroladas. No entanto, a melhor forma de implementar isto não é usando a implementação efetuada no problema A, mas fazendo uma nova implementação tendo em conta as operações que irão ser usadas ao trabalhar com filas. Por exemplo, na implementação efetuada no problema A, tentou-se fazer com que os blocos da lista fiquem apenas meio cheios. Este detalhe tem como objetivo melhorar a eficiência de inserções de novos elementos no meio da lista (à custa de espaço adicional), pois apenas uma parte das inserções irão obrigar à criação de um novo bloco. Este detalhe de implementação faz sentido para a implementação genérica de uma lista ligada enrolada, mas não faz sentido (i.e. não traz qualquer benefício) para uma fila, pois numa pilha, estamos sempre a inserir no fim e remover elementos no início da fila e não no meio da mesma.

Desenhe e implemente uma estrutura de dados – baseada na lista ligada enrolada – que implemente as funcionalidades de uma fila. A classe a implementar deverá ter o nome ULLQueue, e deverá implementar a interface IQueue<Item> fornecida, apresentada de seguida.

IQueue<T> – representa uma fila de elementos do tipo *T*. Uma fila respeita a ordem de entrada dos elementos na mesma, sendo que o primeiro elemento a ser colocado (*enqueue*) será o primeiro a ser processado (*dequeue*)..		
`void`	`enqueue(T item)`	Coloca um item no início da fila.
`T`	`dequeue()`	Remove e retorna o item no fim da fila. Caso a fila esteja vazia deverá retornar *null*
`Item`	`peek()`	Retorna o item no fim da fila, mas não o remove. Retorna null caso a fila esteja vazia

[4] Poderá fazê-lo usando a instrução *"package aed.collections;"* (sem as aspas) no ínicio do ficheiro.

`boolean`	`isEmpty()`	Retorna *true* se a fila estiver vazia e *false* caso contrário
`Int`	`size()`	Devolve o tamanho (número de elementos) da pilha
`QueueArray<T>`	`shallowCopy()`	Retorna uma cópia superficial da fila. Uma cópia superficial copia a estrutura da fila sem copiar cada item individualmente
`Iterator<T>`	`Iterator()`	Retorna um iterador com estado para iterar sobre os elementos da fila. Este iterador deverá implementar os métodos hasNext() e next() obrigatoriamente.

Testes empíricos

Utilize testes empíricos (por exemplo ensaios de razão dobrada) para determinar o tempo de execução e a ordem de crescimento temporal dos métodos enqueue e dequeue. Escreva um sumário dos testes e resultados, incluindo os valores de r (razão dobrada), e uma breve descrição como comentários no ficheiro Java a submeter

O método *main* deverá implementar os testes e ensaios utilizados. Embora este método não seja validado de forma automática pelo *Mooshak* será tido em consideração na validação do projeto, e contará para a nota final do mesmo.

Requisitos técnicos: Esta fila deverá ser implementada obrigatoriamente através de uma lista ligada enrolada, usando blocos de *arrays* ligados entre si. Não poderá usar na sua implementação nenhuma das coleções nativas do Java (*ArrayList*,*LinkedList*,*Vector*,etc), mas poderá usar *Arrays*.

Implemente a classe *ULLQueue* no ficheiro *ULLQueue.java*. A classe deverá estar definida na package *aed.collections*. Outras classes auxiliares deverão ser definidas no mesmo ficheiro.

Submeta **apenas o ficheiro ULLQueue.java** no Problema B (não é necessário submeter a interface)

Condições de realização

O 1.º projeto vale 20% da nota final na componente de frequência da unidade curricular. O projecto deve ser realizado individualmente. Projectos iguais, ou muito semelhantes, originarão a reprovação na disciplina. O corpo docente da disciplina será o único juiz do que se considera ou não copiar num projecto.

O código do projecto deverá ser entregue obrigatoriamente por via electrónica, através do sistema Mooshak, **até às 23:59 do dia 29 de Outubro.** As validações terão lugar na semana seguinte, de **1-5 de Novembro**. Os alunos terão de validar o código juntamente com o docente **durante** o horário de laboratório correspondente ao turno em que estão inscritos. **A avaliação e correspondente nota do projecto só terá efeito após a validação do código pelo docente.**

A avaliação da execução do código é feita automaticamente através do sistema Mooshak, usando vários testes configurados no sistema. O tempo de execução de cada teste está limitado, bem como a memória utilizada. Em breve serão disponibilizadas mais informações sobre o registo e acesso ao sistema Mooshak.