Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

ALGORITMOS E ESTRUTURAS DE DADOS

Relatório da 2ª Série de Exercícios

Docente

Prof.ª Cátia Vaz

Alunos

36368 Mihail Cirja

43552 Samuel Costa

Abril 2018

Índice

Introdução

Este relatório, referente ao segundo de três trabalhos práticos, foi elaborado no âmbito da unidade curricular de Algoritmos e Estruturas de Dados e acompanha a resolução de uma série de exercícios. Esta série de exercícios constituiu uma aplicação dos conteúdos introdutórios das estruturas de dados, nomeadamente amontoados binários e listas. Foram também estudados os métodos e tipos de dados genéricos, as interfaces Comparable, Comparator.

Ao longo do documento apresentam-se a resolução de três exercícios com recurso aos conteúdos estudados nas aulas, envolvendo a escolha de estruturas de dados e a utilização de algoritmos adequados; e, por fim, uma breve reflexão sobre o trabalho realizado em que se discutem os resultados obtidos, explicitando-se, conclusões referentes ao trabalho desenvolvido.

1 Exercícios

Foi proposta a resolução de três exercícios. Remete-se a descrição dos problemas para o enunciado do trabalho.

Exercício 1
Para a definição do tipo dados MedianQueue precisaríamos de um Max-Heap, dos elementos menores que a mediana e um Min-Heap, dos elementos maiores que a mediana. A solução consiste em manter estas duas estruturas balanceadas tanto quanto possível.

As operações suportadas por este tipo de dados são:

- **`public void offer(int x)`** que adiciona o inteiro x à coleção;
- **`public int peek( )`** que permite obter a mediana dos elementos que pertencem à coleção;
- **`public int poll( )`** que retorna a mediana e remove os elementos correspondentes à mediana (1 ou 2) dos elementos que pertencem à coleção.

Para implementar a operação offer, tiveram-se em conta 3 casos:

- Tanto o Min-Heap como o Max-Heap estão vazios, i.e., o tamanho de ambos é zero. Neste caso, o elemento x é adicionado ao Min-Heap;
- O elemento a adicionar é maior ou igual à mediana, caso em que se insere no Min-Heap;
- O elemento a adicionar é menor que a mediana, caso em que se insere no Max-Heap.

Para implementar a operação peek tiveram-se em conta os diferentes casos possíveis. Assim, se o tamanho do Max-Heap e Min-Heap for igual, a mediana é a media aritmética das raízes dos dois amontoados, pois existe número par de amostras. Nos restantes casos, a mediana é obtida retornando o elemento na raíz da estrutura que tiver mais elementos, ou seja, da estrutura que tiver mais um elemento do que o outro, sendo esse elemento a mediana.

Ao remover ou adicionar elementos (offer ou poll) é feito o balanceamento das estruturas, isto é, se um dos amontoados tiver mais do que um elemento a mais que o outro, o elemento na raíz do amontoado de maior tamanho é transferido para o outro.

A operação poll pode ser obtida combinando peek com a remoção do elemento obtido, seguido de balanceamento.

Exercício 2

Para dar resposta ao problema, a coleção IntArrayList mantem uma variável de estado sum, bem como uma variável k, correspondente ao índice em que o próximo elemento deve ser inserido.

O argumento do método AddToAll é adicionado a sum. O elemento retornado é o elemento constante do array + sum. Para compensar esta soma, ao inserir um elemento na coleção, é colocado x-sum na posição k do array, por forma a compensar a soma efetuada no retorno.

Exercício 3.1

É entendido por ponto de interseção, o nó de list1, a partir do qual list1 e list2 sejam iguais até ao último nó. Assim, interessa percorrer as duas listas enquanto estas forem iguais, e encontrar o primeiro nó que seja diferente, passando este a constituir o último nó da lista.

Exercício 3.2

Considerou-se que lista vazia e lista singular se encontram ordenadas.

Como o algoritmo quickSort é do tipo "dividir para conquistar", considera-se que, dada uma determinada partição da lista, ordenar a lista é o mesmo que ordenar os dois conjuntos da partição. Assim, efetua-se uma partição da lista.

Em seguida, são efetuadas duas chamadas ao método quickSort com first = first e last= temp.prev e com first=temp.next e last = last.

Exercício 3.3

O algoritmo desenvolvido apresenta semelhanças relativamente ao problema da primeira série de exercícios, e consiste em colocar as cabeças das listas referenciadas por lists em minHeap, por forma a encontrar o menor elemento, segundo o critério cmp. Como se sabe que as listas estão ordenadas pelo mesmo critério de comparação, é garantido que o menor elemento é uma das cabeças de lista. Seguidamente, esse elemento é colocado na lista resultado e a sua posição na lista originária passa a ser ocupada pelo seu próximo. Quando uma das listas que compõem lists se encontrar vazia, é trocada com a última posição do Min-Heap e a dimensão deste é decrementada, sendo reorganizado o amontoado. O processo é repetido até a lista de listas se encontrar vazia. Este algoritmo não é estável, uma vez que não é garantido que dois elementos contíguos em lists[i] sejam contíguos na lista resultante.

Conclusões

Os métodos desenvolvidos foram testados e apresentam conformidade com os testes fornecidos.

Salienta-se que a utilização adequada das estruturas de dados estudadas permitiu (i)simplificar a escrita dos programas, como é o caso da resolução do primeiro exercício por recurso a um stack; (ii) diminuir consideravelmente o grau de crescimento das soluções apresentadas, como foi o caso da utilização de minHeap no exercício 2.1; (iii)apresentar soluções genéricas, como é o caso da resolução do exercício 2.2.

Por outro lado, as restrições postas pelas assinaturas dos métodos a desenvolver, particularmente a utilização de tipos genéricos e tipos compatíveis com as interfaces Comparator (ex. 2.1) e Iterable (ex. 3.1 e 3.2) permitiram (iv) desligar a implementação das estruturas de dados (p.ex. minHeap) do critério de comparação e do tipo de dados que

suportam; e (v) elaborar iteradores *lazy*, que encapsulam as regras de iteração e obtenção de novos elementos.

Por fim, acrescenta-se que a elaboração deste trabalho serviu para consolidar competências já adquiridas e para o seu aprofundamento, uma vez que constituiu uma oportunidade para o estudo das interfaces *Iterable*, *Iterator* e *Comparator*, no contexto dos métodos estáticos.