Aula Prática 12

Resumo:
- Aplicações de dicionários;
- Dicionários implementados como tabelas de dispersão.

Exercício 12.1

Acrescente à classe `HashTable` do pacote `p2utils` os seguintes métodos:

- `keys()` - devolve um array com todas as chaves existentes na tabela de dispersão.
- `toString()` - Devolve uma representação da tabela de dispersão em cadeia de caracteres de acordo com o seguinte formato: $\{(k_1, e_1), ..., (k_n, e_n)\}$. Dica: poderá invocar o `toString` generalizado para obter os elementos de cada `KeyValueList`.

Pode usar o programa `TestHashTable` para testar os novos métodos.

Exercício 12.2

O supermercado *DaEsquina* aceita encomendas e faz as respectivas entregas ao domicílio. O gerente decidiu instalar um sistema automático de processamento de encomendas que permita, não só registar encomendas recebidas e dar baixa de encomendas entregues, mas também saber em cada momento quantas unidades de cada produto estão pedidas. As encomendas, representadas pela classe `Order` (disponibilizada em anexo), são processadas por ordem de chegada. Implemente assim uma classe `SupermarketOrdering` com os seguintes métodos:

- `enterOrder(order)` - regista uma nova encomenda;
- `serveOrder()` - dá baixa da encomenda mais antiga e devolve-a;
- `query(product)` - devolve o número de unidades de um dado produto que estão pedidas nas encomendas actuais;
- `displayOrders()` - imprime a lista de encomendas, por ordem de chegada, e o número total de unidades encomendadas de cada produto;

Os métodos `enterOrder`, `serveOrder` e `query` devem ter uma complexidade temporal $O(1)$ no número de encomendas. O programa `TestSupermarket` permite testar a classe pedida.

Exercício 12.3

No exercício 11.2 obteve o número de ocorrências de palavras num texto e, com base nessa informação, a palavra mais comum e respectiva frequência relativa. Podemos generalizar este tipo de análise para obter estatísticas de ocorrência de sequências de várias palavras. Essa informação permite criar modelos para prever a próxima palavra num browser ou para auxiliar no reconhecimento de fala, por exemplo. Para simplificar, vamos considerar apenas sequências de 2 palavras consecutivas, também chamadas de *bigramas*.

a. Complete o programa `CountBigrams` para que determine o número de ocorrências de todos os bigramas existentes num conjunto de ficheiros de texto. Depois de processados os textos, o programa deve apresentar a lista completa de bigramas e respectivas frequências absolutas, sem qualquer exigência em termos de ordem. **Sugestão:** a chave para o dicionário pode ser uma combinação adequada das duas palavras do bigrama.

b. Altere o programa para apresentar a listagem por ordem alfabética dos bigramas.

c. Altere o programa para mostrar o bigrama mais comum.

Pode testar o programa com ficheiros de texto descarregados do Projeto Gutenberg, por exemplo.

Exercício 12.4

A solução proposta no problema anterior, usar o bigrama como chave, não é a mais conveniente se quisermos saber que palavras ocorrem com maior frequência antes ou depois de uma certa palavra.

Desenvolva uma nova solução para contagem de bigramas (`CountBigrams2`) baseada num dicionário que usa cada palavra como chave e o valor associado é uma tabela das palavras que lhe sucedem e respetivas contagens. Após processamento dos textos, o programa deve apresentar a informação no seguinte formato:

```
...
has -> {(been, 2), (access, 1), (appropriate, 1)}
...
to -> {(them, 1), (using, 1), (produce, 4), (my, 1), (the, 14), ... }
...
```

A seguir, o programa deve pedir uma palavra ao utilizador e indicar a palavra que mais frequentemente surge a seguir a essa nos textos. (Pode reutilizar a função `mostFrequent` feita no exercício 11.2 ou 12.3.)

Exercício 12.5

O ficheiro `groups1.csv` tem uma lista de alunos que fizeram a primeira prova de avaliação de Programação 2 no ano de 2017-2018. Cada linha contém o número de um aluno e o computador onde fez a prova, separados por um TAB (`"\t"`). Como a prova foi feita em grupo, vários alunos partilharam o mesmo computador e o nome do computador serve de identificador do grupo respetivo. Para a segunda avaliação, os grupos tinham de ser diferentes.

a. Dado um ficheiro nesse formato, o programa `CheckGroups` deve pedir números de alunos dois-a-dois e verificar se estiveram no mesmo grupo ou não. Complete a função de leitura do ficheiro e a função principal para atingir esse objetivo.

```
aula12$ java -ea -jar CheckGroups.jar groups1.csv
Student1? 71904
Student2? 84672
Students 71904 and 84672 were NOT in the same group!

Student1? 75762
Student2? 88888
Students 75762 and 88888 were NOT in the same group!

Student1? 75762
Student2? 76442
Students 75762 and 76442 WERE in the same group!

Student1? <Ctrl+D>
```

b. Escreva um programa `ListGroups` que leia um ficheiro de grupos e mostre os alunos de cada grupo, um grupo por linha. Sugestão: convém criar um dicionário que associe cada grupo à respetiva lista de alunos. Pode criar uma nova função de leitura ou reutilizar a função de leitura da alínea anterior e acrescentar outra função para converter o dicionário obtido.

```
aula12$ java -ea -jar ListGroups.jar groups1.csv
l040106-ws10: [88970, 88826]
l230213-ws03: [88750, 88998]
l230213-ws04: [84983]
l040106-ws03: [89189, 88906]
l230213-ws06: [76442, 88932, 75762]
...
```

c. O ficheiro `groups2.csv` tem os pares (aluno, computador) da segunda prova de avaliação. Escreva um programa `FindGroupCollisions` que, dados os ficheiros de duas avaliações, descubra se na segunda avaliação algum aluno fez grupo com outro aluno com quem já tivesse feito grupo na primeira. Note que pode ter havido grupos de 1, 2 ou mais alunos em qualquer das avaliações e que alguns alunos foram apenas a uma das avaliações. Teste o programa também com os ficheiros `groups1.csv` e `groupsX.csv` que têm algumas "colisões".

```
aula12$ java -jar FindGroupCollisions.jar groups1.csv groups2.csv

aula12$ java -jar FindGroupCollisions.jar groups1.csv groupsX.csv
In groupsX.csv, group l230204-ws06: [88857, 85123]
includes students that were in same group
in groups1.csv: {(l230214-ws02, [88857, 85123])}

In groupsX.csv, group l230214-ws04: [80248, 89280, 79996]
includes students that were in same group
in groups1.csv: {(l230211-ws02, [80248, 79996]), (l230205-ws10, [89280])}
```