Estruturas de Dados e Algoritmos: Trabalho #2

20 Janeiro 2016

Hiago Oliveira - 29248

Introdução

O objectivo do trabalho é resolver um problema do tipo "Sopa de Letras", utilizando vários tipos abstractos de dados leccionados durante o semestre. O problema consiste num tabuleiro de letras e encontrar as palavras possíveis de se formar a partir de cada letra, sem repetir nenhuma posição.
São providênciados dois ficheiros para o problema, um com o dicionário de palavras e outro com o tabuleiro de jogo (`allWords.txt` e `tabuleiroXxY.txt` respectivamente).
O programa necessita do nome dos dois ficheiros providenciados como argumentos para que o ficheiro java possa ser executado. Por exemplo `java Boggle allWords.txt board4x4.txt`

Resolução do problema

Começa-se por chamar o método `lerDicionário()` da classe `Boggle`, onde será lido o ficheiro com todas as palavras. Ao ler, cada palavra (linha) será inserida numa hashtable com acesso fechado, e resolução de problemas de forma quadrática. Aproveita-se também para inserir em outra hashtable separada os prefixos da palavra lida actualmente. Por exemplo, ao ler a palavra `bola` do dicionário, seria inserido na hashtable das palavras `bola` e em outra hashtable para os prefixos `bol, bo, b`. Mais tarde servirá para saber se a combinação actual das letras começa a formar alguma palavra do dicionário, poupando assim tempo.
Após ler o dicionário é chamado outro método para ler o tabuleiro de jogo, `lerBoggle()`, onde se passa a ter acesso ao tabuleiro através de um array de duas dimensões chamado `boggleBoard`. Por exemplo, o tabuleiro de jogo

```
S E L D
O U M O
O M E T
I N K Y
```

É possível aceder a qualquer letra através do número da sua linha e coluna. Ex: `boggleBoard[1][1]` obtém-se a letra `U`. No trabalho também é criada a classe `Position` que contém o conceito de linha e coluna. É aproveitada aqui para se ter o tamanho do tabuleiro de jogo facilmente num objecto desta classe chamado `boardSize`, que será utilizado mais tarde para validar as posições vizinhas. Após se ter tratado do tabuleiro de jogo e do dicionário, pode-se começar a encontrar as palavras no tabuleiro, portanto é chamado o método `solve()`.
Começa-se por preencher um array com as posições/movimentos possivéis da vizinhança de uma letra. Estas são `N, S, E, W, NE, NW, SE, SW`. A partir de cada letra vão se formando palavras. Para saber as posições adjacentes calcula-se a posição actual com cada um dos possiveis movimentos, por exemplo para a posição (2,2), a letra a seu nordeste estará em (1,3), ou (2 + (-1), 2 + 1) (relembre-se que o movimento para ir a NE é (-1,1). Isto é feito para todos os movimentos possiveis, e é sempre validado através de um método auxiliar chamado`validPos`, onde é verificado se o vizinho é válido (por exemplo a primeira posição do tabuleiro (0,0) não tem vizinhos a Norte ou a Oeste). Aproveita-se também este método, que recebe uma lista de posições já visitadas, para verificar se a nova posição já foi visitada ou não, não se repetindo assim posições.
A lista de posições usadas neste caso trata-se de uma `ArrayList`, e não de uma `LinkedList` implementada no âmbito da cadeira pela razão de que através da estratégia utilizada para resolver o problema, é necessário fazer uma cópia (clone) da lista de posições utilizadas. Ao inicio tentou se utilizar a `LinkedList` porém sem sucesso, visto ao se fazer uma `shallow copy` não servir para o pretendido, pois iria estar a alterar a lista original ao mesmo tempo. Tentou-se implementar o método `clone` na classe `LinkedList`, mas sem sucesso também. Neste passo por isso utiliza-se uma `ArrayList` por providenciar o método de clonagem necessário.

No seguimento, vê-se se a palavra actual a ser formada faz parte dos prefixos, e se fizer, continua-se a formar, e caso seja uma palavra que se encontre no dicionário junta-se à lista das palavras encontradas, e continua-se a formar a palavra, pois pode ser também ser prefixo de alguma outra palavra.
No final é mostrado ao utilizador a lista das palavras encontradas e as respectivas posições das suas letras.