Mastermind com palavras

Relatório

Universidade do Porto

Faculdade de Engenharia

FEUP

Mestrado Integrado em Engenharia Informática e Computação
Concepção e Análise de Algoritmos

José Carlos Rocha Lima – ei10012
Nuno Filipe Dinis Cruz – ei10082
Vasco Manuel Pérola Filipe – ei10031

2 de Maio de 2014

Índice

Introdução

No âmbito da disciplina de Concepção e Análise de Algoritmos, temos como segundo trabalho prático a elaboração de um projeto de programação em C++ de algoritmia em pesquisa e comparação de strings.

Pretende-se desenvolver um programa capaz de emular o jogo mastermind com palavras, assim como inteligência artificial capaz de encontrar a solução.

No jogo, o utilizador começar por escolher quem vai jogar, ele ou o computador. De seguida escolhe um modo de jogo e um tema (categoria), e apartir daí é escolhida uma palavra de forma aleatória. Conforme o modo de jogo, o jogador saberá ou não o tema da palavra, assim como o seu tamanho. A partir daí tem 10 tentativas para adivinhar a palavra, sendo que recebe *feedback* ao fim de cada uma delas, *feedback* esse que é dado em forma de *string*, onde as letras correctas aparecem como "B" (de *Black*), as pertencentes à palavra mas na posição errada como "W" (de *White*) e as restantes como "_".

Implementação

Numa primeira fase, estudámos o algoritmo de *Levenshtein* (pesquisa aproximada) para cálculo de distâncias entre *strings*. Um problema com esse algoritmo é que não prevê a troca de posições das letras (apenas suporta adição, subtração e substituição). Para além disso, a inteligência artificial não tem acesso à solução, pelo que é impossível calcular a distância entre a tentativa e a palavra certa.

Decidimos então implementar algoritmos de raíz, tanto para o desenvolvimento do jogo como para a inteligência artificial.

Jogo

Para cada jogada, o algoritmo recebe uma tentativa, comparando-a com a solução. No caso de serem iguais, o jogador ganha. Senão, é utilizado um *map* para guardar a quantidade de cada letra. Por exemplo, a palavra algoritmo teria (a-1), (o-2), etc. Assim, o algoritmo garante que não são dadas falsas indicações, no caso de palavras com várias letras iguais. Um exemplo seria ao comparar "cal" com "aaa": sem o contador daria "WBW"; com ele daria "_B_", uma dica mais correcta, visto só existir um 'a'.

A partir daí, é percorrida a palavra à procura de letras na posição certa (solução[i]=tentativa[i]), pois um "B" tem prioridade sobre um "W". Por fim, percorre-se as duas palavras à procura de "W" (letras em comum mas na posição errada). O resultado da tentativa é dado numa *string* do tipo "B_W_WWB".

Análise temporal :

n= tamanho da palavra

Criação do contador – n

Verificação de letras "B" – n

Verificação de letras "W" – n^2

$O(2n+n^2)$

Inteligência Artificial

No modo de jogo em que joga o computador, este começa por procurar palavras que tenham o tamanho igual ao da palavra a acertar, dando como tentativa a primeira que encontrar. De acordo com o *feedback* recebido, são então guardadas as letras corretas, as que existem em posições diferentes e as que não existem na palavra a adivinhar. Nas iterações a seguir, são adicionados estes parâmetros à pesquisa por uma palavra válida. Cada palavra processada é eliminada para acelerar a próxima pesquisa.

Dados

Dados de Entrada:

Ficheiro de texto contendo todas as palavras reconhecidas pelo jogo, separadas por temas ;

Tentativa do jogador ou do computador.

Dados de Saída:

String do tipo "B_W_WWB" no caso de tentativa falhada;

Palavra certa, no caso de acabarem as tentativas.

Os dados de entrada são lidos a partir do ficheiro de texto "*dictionary.txt*". A estrutura do ficheiro de texto é a seguinte:

{tema}

{palavra}

...

{palavra}

{tema}

{palavra}

...

{palavra}

Estrutura do programa

Este trabalho tinha um foco maior na lógica, pelo que apenas foi preciso criar uma classe tema para o agrupamento das palavras do dicionário.

class Theme:

String name – nome do tema;

Vector <string> words – vector que contém todas as palavras do tema.

Vector<Theme> readDictionary()

Lê e analisa um ficheiro de texto, retornando um vector de temas.

void play(string word, Theme theme, int player)

Função responsável pelo desenvolvimento do jogo, tanto com um jogador humano como com o computador.

main()

Menu Inicial, inicializa o jogo consoante o modo escolhido pelo utilizador.

Conclusão

Uma boa gestão do tempo permitiu-nos desenvolver o trabalho por completo, sendo que não foram encontrados erros na execução do mesmo. Decidimos melhor trabalhar em conjunto, pelo que todos trabalharam um pouco em cada tarefa, facilitando a resolução dos problemas com que nos deparámos.

O trabalho decorreu sem grandes entraves, surgindo apenas o problema das palavras com várias letras iguais, que foi resolvido com a implementação dum contador de letras.

Bibliografia

http://pt.wikipedia.org/wiki/Dist%C3%A2ncia_Levenshtein

http://www.cplusplus.com/