Documentação TP2 - Sugestões Doodle

Bruno M. Monteiro

03/06/2018

1 Introdução

O problema consistiu na implementação de um algoritmo para, dado um dicionário e uma palavra digitada, encontrar as possíveis palavras que o digitador quis escrever. Para isso, foram definidas 3 operações:

- **Inserção**: Inserir um caractere em uma posição de uma palavra;
- **Remoção**: Remover um caractere de uma palavra;
- **Substituição**: Substituir um caractere de uma palavra por outro caractere.

Com isso, dado um inteiro n, precisávamos descobrir as palavras do dicionário que estão a uma "distância" de no máximo n da palavra digitada, sendo a distância o número mínimo de operações necessárias para que as palavras fiquem iguais.

Além disso, nos foi requisitado que ordenássemos as palavras do dicionário (que distanciam de no máximo n da palavra digitada) pela distância, e fazer o desempate pela ordem lexicográfica.

2 Solução do Problema

Para solucionar o problema, foi feita uma programação dinâmica no intuito de encontrar a distância entre duas palavras. Seja $f(i, j)$ a menor distância entre duas palavras a e b de tamanhos i e j, respectivamente.

Vamos comparar o último caractere de cada palavra. Observa-se que $f(i, j)$ é o custo de uma operação mais o mínimo entre as soluções ótimas após realizar cada uma das 3 operações: remoção do caractere de a, $f(i-1, j)$, inserção do caractere em a, $f(i, j-1)$, e substituição, $f(i-1, j-1)$. Se os caracteres forem iguais, temos que $f(i, j) = f(i-1, j-1)$. Quando i ou j for zero, a distância é o tamanho da palavra não nula. Assim, temos:

$$f(i,j) = \begin{cases} \max(i,j), & \text{se } \min(i,j) = 0 \\ f(i-1,j-1), & \text{se } a_i = b_i \\ 1 + \min(f(i-1,j), f(i,j-1), f(i-1,j-1)), & \text{caso contrário} \end{cases}$$

Foi feita então uma memorização para que um mesmo valor não fosse recalculado. Dessa forma, calcula-se a distância entre a palavra digitada e todas as palavras do dicionário. Todas as palavras cuja distância é menor ou igual a n são colocadas em um vetor e depois ordenadas, fazendo uso do *quick sort*.

3 Análise de Complexidade Assintótica

3.1 Complexidade Assintótica de Tempo

Para cada palavra do dicionário, calcula-se a distância dela para a palavra digitada, ou seja, chama-se a função de distância uma vez.

A função calcula a distância para um par de limites i, j, e depende de valores menores ou iguais de i e j. Se os valores menores já estão calculados, a função tem custo $\mathcal{O}(1)$. Caso contrário, os valores são calculados, e são então combinados em tempo constante.

Tem-se, graças à memorização, que a complexidade da programação dinâmica é proporcional ao número de estados que ela possui, que no caso é proporcional ao tamanho das palavras.

Supondo-se que cada palavra do dicionário tem tamanho m, a complexidade de calcular a distâncias é $\mathcal{O}(Dm|q|)$, pois são feitos D cálculos de distância, um por palavra do dicionário. Depois disso, é feito um *quick sort*, que é $\mathcal{O}(mD\log(D))$, uma vez que a comparação de duas palavras é $\mathcal{O}(m)$. A complexidade de exibir a resposta é dominada pela complexidade de ordenar.

Portanto, a complexidade do algoritmo é $\mathcal{O}(Dm|q|) + \mathcal{O}(mD\log(D)) = \mathcal{O}(mD(|q| + \log(D)))$.

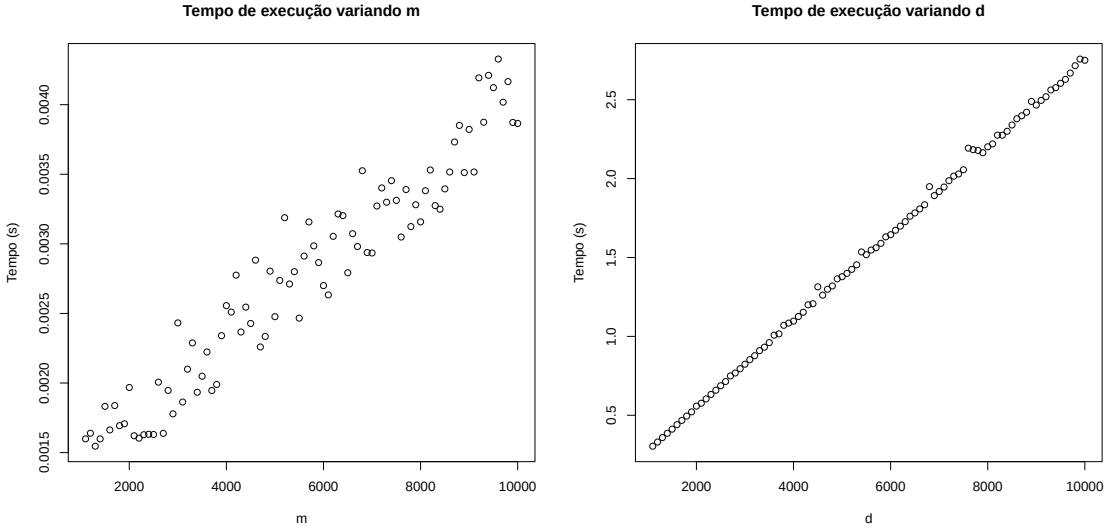
3.2 Complexidade Assintótica de Espaço

Em relação ao espaço utilizado, para memorizar o dicionário gastamos $\mathcal{O}(mD)$. Para armazenar a matriz de memorização, precisamos de $\mathcal{O}(m|q|)$. Dessa forma, a complexidade de espaço é $\mathcal{O}(m(D + |q|))$.

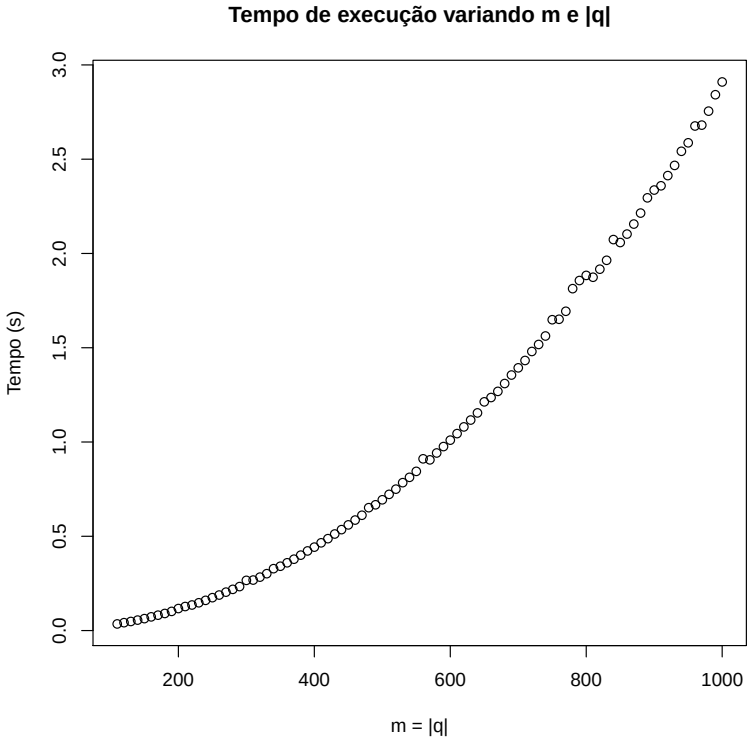
4 Análise Experimental

Para fazer a análise experimental, foram criados *scripts* para gerar e cronometrar automaticamente casos de teste. Os resultados acabam sendo afetados por otimizações feitas para aumentar a velocidade de execução.

Inicialmente, foi medido o tempo de execução do programa variando o tamanho das palavras do dicionário, m, e mantendo $d = |q| = n = 100$. Em seguida, foi variado o tamanho do dicionário, d, mantendo $m = |q| = n = 100$. Os resultados obtidos foram colocados nos gráficos abaixo.



Para demonstrar o comportamento quadrático da função de distância, foi medido o tempo de execução fazendo $m = |q|$, e variando m, mantendo d constante:



5 Conclusão

Neste trabalho, foi implementado um algoritmo para calcular a distância de edição entre duas palavras. Utilizando o paradigma de programação dinâmica, observou-se que a complexidade do mesmo acabou sendo quadrática no tamanho das palavras.

As complexidades foram verificadas experimentalmente, e em certos casos foram observadas variações no tempo de execução, que provavelmente se devem pelas várias otimizações que foram feitas no trabalho para atingir um tempo de execução baixo.