TP2 de Algoritmos e Estruturas de Dados III

Lucas O. Silvestre[1]

[1]Sistemas de Informação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, MG – Brazil

lsilvs@dcc.ufmg.br

1. Objetivo inicial

O trabalho prático em questão visa avaliar os conhecimentos do aluno no que diz respeito a paradigmas de programação. Para tanto, foi proposto o desenvolvimento do famoso problema do palíndromo. Palíndromo é toda e qualquer palavra que lida da esquerda para direita é a mesma que lida da direita para esquerda. O desafio é descobrir qual o menor número de letras devemos acrescentar à palavra original para que ela se transforme em um palíndromo. O algoritmo deve ser implementado utilizando dois diferentes paradigmas de programação. Primeiramente uma solução ótima utilizando Programação Dinâmica e outra solução, não necessariamente ótima, utilizando paradigma Guloso. Abaixo serão discutidas as soluções propostas, vantagens e desvantagens de cada um deles.

2. Modelagem e Solução proposta

A modelagem segue o padrão de leitura de um arquivo de entrada especificado na chamada do programa. A primeira linha do arquivo de entrada representa o número de palavras que o arquivo contém. As linhas seguintes são as palavras propriamente ditas, sempre com letras maiúsculas, sem espaço e separadas por quebra de linha.

A solução proposta consiste em encontrar o meio palíndromo já existente dentro da palavra dada e subtrair da palavra o número de letras desse palíndromo encontrado. Isso porque, se a palavra já possui um palíndromo em sua estrutura, a única coisa que precisamos fazer é acrescentar as outras letras restantes para formar o palíndromo de toda a palavra. Conforme o enunciado do problema, foram aplicadas soluções para dois diferentes paradigmas. Cada paradigma será discutido com maior detalhe logo abaixo.

O arquivo de saída imprime o número de letras que são necessárias para que, adicionando-as em seus devidos lugares na palavra, a original se transforme um palíndromo.

2.1. Programação Dinâmica

A Programação Dinâmica consiste em resolver um problema a partir de resoluções menores já calculadas e armazenadas. A partir desses pequenos resultados, faz-se a combinação deles de forma a poupar esforços que por ventura pudessem ser aplicados.

A solução implementada utiliza o conceito do LCS (Longest Commom Subsequence). A ideia do LCS é encontrar a maior subsequência de duas strings quaisquer. Para adaptar ao nosso caso, as duas strings passadas no exercício serão a palavra em sua ordem natural e a mesma palavra invertida, ou seja, de trás para frente. Dessa forma, o LCS irá retornar o tamanho da maior subsequência comum entre a palavra e sua inversa, o que corresponde ao maior palíndromo existente dentro da palavra dada.

O paradigma aplicado tem solução ótima, uma vez que o menor número de letras necessárias para transformar dada palavra em um palíndromo é seu tamanho subitraído do maior palíndromo existente dentro da palavra.

```
int prog_din(char * palavra) {
    int j, k, l;
    int array[2][5001];
    int length = strlen(palavra);

    // Seta os valores iniciais em zero para o array
    memset(array, 0, sizeof(array));

    // Percorre a palavra de trás para frente
    for(j = length-1; j >= 0; j--) {
        char ch = palavra[j];

        // Percorre a palavra em seu sentido normal
        for (k = 0; k < length; k++) {
            if (ch == palavra[k]) {
                array[1][k+1] = array[0][k] + 1;
            } else {
                array[1][k+1] = max(array[1][k], array[0][k+1]);
            }
        }

        // Salva na posição zero do array os dados calculados
        for (l = 0; l <= length; l++) {
            array[0][l] = array[1][l];
        }
    }

    // Retorna o tamanho da palavra subitraindo-se
    // o maior palíndromo encontrado
    return (length - array[1][k]);
}
```

2.2. Paradigma Guloso

O Paradigma Guloso busca fazer a melhor escolha no momento da decisão. Assim, ele quer fazer a melhor escolha local e espera que, com a soma das escolhas locais ótimas, o algoritmo final tenha a melhor decisão global.

Aplicando o paradigma para o nosso exercício, o algoritmo escolhido realiza a busca do maior palíndromo dentro da palavra ao compará-la com sua inversa e avaliando se cada caracter da palavra original existe, (na mesma ordem, mas não necessariamente consecutivamente), na palavra invertida. Uma comparação é realizada de dois em dois caracteres e movendo o ponteiro para frente do caracter encontrado. É um algoritmo guloso pois ele toma decisões locais e não garante solução ótima.

3. Análise de Complexidade

Aqui, iremos analisar ambas as funções que retornam o número de caracteres necessários para se transformar uma dada palavra em um palíndromo. A primeira função é a que aplica o algoritmo de Programção Dinâmica. Nesse algoritmo, temos dois laços que percorrem toda a palavra cada um. O primeiro deles percorre a palavra de trás para frente e o segundo no sentido normal. Sendo assim, temos que a complexidade dessa função é $O(n^2)$, onde n é o número de letras da palavra recebida por parâmetro.

O segundo algoritmo, Guloso, já possui uma complexidade bem menor (e isso será comprovado pelos experimentos descritos abaixo). Nele, temos um laço que percorre toda a palavra original e evoca uma função que possui outro laço que recebe uma referência de onde continuar. Dessa forma, o laço interno mantém complexidade linear e o externo também, sendo a complexidade total $O(n) + O(n)$, o que nos dá uma complexidade total igual a $O(m)$.

4. Experimentos

Como não foram disponibilizados arquivos de entrada para teste, geramos dois tipos de arquivos para tal finalidade. O primeiro teste realizado foi rodar os dois algoritmos com arquivos de strings com o tamanho máximo especificado pelo enunciado (5000 caracteres) e aumentando-se o número de instâncias, ou seja, o número de palavras.

Esse teste tinha como objetivo calcular o tempo gasto por ambos algoritmos e compará-los quando à eficiência de tempo de execução. Pelo resultado obtido (ver gráfico abaixo), concluímos que o algoritmo Guloso é extremamente mais rápido que o algoritmo de Programação Dinâmica. E isso já era esperado.

Um segundo experimento foi realizado com objetivo de avaliar o desempenho dos algoritmos no que diz respeito a exatidão dos resultados. Para tanto, o programa foi rodado tendo como arquivo de entrada um dicionário de 6045 palavras curtas. De todas as palavras avaliadas, 2113 tiveram valores maiores quando aplicado o algoritmo Guloso. Ou seja, aproximadamente 35% não obteve a solução ótima do algoritmo Dinâmico. Dentre os que não obtiveram a solução ótima, a diferença do número de caracteres da solução ótima (Programação Dinâmica) e o número de caracteres da

solução não ótima (Guloso) foi de 24,5%. Em outras palavras, a média do número de caracteres a mais retornado pelo algoritmo Guloso foi de 24,5% do número de caracteres retornado pelo algoritmo de Programação Dinâmica.

5. Especificação

Todo o desenvolvimento do código foi realizado utilizando um MacBook Air com as seguintes especificações:

Processador: Intel Core i5 1,7 GHz

Memoria: 4 GB 1333 MHz DDR3

Sistema Operacional: OS X 10.8.2

IDE: Sublime Test 2

GCC: i686-apple-darwin11-llvm-gcc-4.2 (GCC) 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2336.11.00)

6. Conclusão

O presente trabalho foi de extrema importância para alinhar os conceitos apresentados durante a aula expositiva. Após todo desenvolvimento, fica claro que o algoritmo de Programação Dinâmica possui a solução ótima e ainda é otimizado por salvar cálculos já realizados anteriormente. Embora ele tenha solução ótima e seja otimizado, podemos ainda, em alguns casos, utilizar um algoritmo Guloso a fim de ter um desempenho de tempo de execução ainda melhor. Nesse caso, estaríamos abrindo mão da solução ótima para ter um algoritmo altamente eficiente no que diz respeito ao tempo de execução.

Ou seja, cada caso é um caso e fazendo-se análizes sistemáticas de ambos algoritmos e seus resultados, podemos escolher o que melhor se adapta a um cenário específico.