TRABALHO PRÁTICO 2:
Paradigma Guloso e Programação Dinâmica

Sandro Miccoli - 2009052409 - smiccoli@dcc.ufmg.br

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

7 de novembro de 2012

Resumo. *Este relatório descreve como foi solucionado o problema de encontrar o menor palíndromo possível dada uma palavra. Será descrito como foi modelado o problema, pois duas soluções foram propostas, uma abordagem de paragima guloso e outra de programação dinâmica. Finalmente será detalhado a análise de complexidade dos algoritmos e uma análise comparativa entre os dois paradigmas e uma breve conclusão do trabalho implementado.*

1. INTRODUÇÃO

Um palíndromo é uma palavra, frase ou qualquer outra sequência de unidades que tenha a propriedade de poder ser lida tanto da direita para a esquerda como da esquerda para a direita [Wikipedia b].

O problema deste trabalho é encontrar qual o menor palíndromo possível de ser gerado inserindo caracteres na palavra original. No caso da palavra CASA, por exemplo, a solução seria CASAC.

Para solucionar essa questão foi criada duas abordagens, uma de solução ótima, utilizando programação dinâmica (PD) e outra de solução não-ótima, utilizando o paradigma guloso (PG).

O restante deste relatório é organizado da seguinte forma. A Seção 2 descreve como foi feita a modelagem do problema e manipulação das palavras. A Seção 3 descreve as propriedades de sub-estrutura ótima e sobreposição dos problemas do paradigma guloso, e também discute a otimalidade da solução gulosa. A Seção 4 trata de detalhes específicos da implementação do trabalho: quais os arquivos utilizados; como é feita a compilação e execução; além de detalhar o formato dos arquivos de entrada e saída. A Seção 5 contém a avaliação experimental, que faz uma análise de quantos porcento a mais de caracteres que a solução gulosa utiliza em relação à solução ótima. A Seção 6 conclui o trabalho.

2. MODELAGEM

Para solucionar o problema do palíndromo, o modelamos como o problema da **Maior Subsequência Comum**, também conhecido como *Longest Common Subsequence* (LCS). Uma subsequência de uma sequência de símbolos X é uma sequência Y com zero com maios símbolos de X removidos. Exemplo: $Z = \{B, C, D, B\}$ é uma subsequência de $X = \{A, B, C, B, D, A, B\}$.
Definição mais formal:

Longest Common Subsequence (LCS): Datas as sequências $X[1..m]$ e $Y[1..n]$ encontrar uma sequência Z que seja subsequência de X e Y e tem comprimento (número de símbolos) máximo.
$X = \{A, B, C, B, D, A, B\}$
$Y = \{B, D, C, A, B, A\}$
$LCS(X, Y) = \{B, C, B, A\}$

Como o problema específico neste trabalho é encontrar palíndromos, então nossa sequência Y é o inverso da nossa sequência X.

3. SOLUÇÃO PROPOSTA

3.1. Abordagem Dinâmica

Uma primeira abordagem escolhida para resolver este problema é o de Programação Dinâmica. Programação dinâmica é um método para a construção de algoritmos para a resolução de problemas computacionais, em especial os de otimização combinatória. Ela é aplicável a problemas nos quais a solução ótima pode ser computada a partir da solução ótima previamente calculada e memorizada - de forma a evitar recálculo - de outros subproblemas que, sobrepostos, compõem o problema original [Szwarccfiter 1984].

O que um problema de otimização deve ter para que a programação dinâmica seja aplicável são duas principais características: subestrutura ótima e superposição de subproblemas. Um problema apresenta uma subestrutura ótima quando uma solução ótima para o problema contém em seu interior soluções ótimas para subproblemas. A superposição de subproblemas acontece quando um algoritmo recursivo reexamina o mesmo problema muitas vezes [Szwarccfiter 1984].

O *LCS* tem **subestrutura ótima**, pois pode ser quebrado em problemas menores, mais simples, os quais também podem ser divididos em subproblemas menores ainda, e assim por diante, até, finalmente, a solução for trivial. O *LCS* também tem **sobreposição de problemas**: a solução de um subproblema maior depende da solução de vários subproblemas menores [Wikipedia a].

Para calcular o *LCS* utilizamos uma tabela para armazenar os valores calculados a cada passo da execução. As implementações mais encontradas na internet utilizavam uma tabela $(n+1, m+1)$, sendo n e m o tamanho de cada uma das strings. A abordagem dinâmica escolhida aqui foi utilizar uma tabela com apenas 2 linhas e n colunas, sendo n o tamanho da *string* que encontraremos o menor palíndromo, pois como o trabalho pede apenas o tamanho do *LCS*, então só precisamos do estado atual e anterior de cada coluna da matriz, por isso utilizamos apenas duas linhas para realizar os cálculos [Hirschberg].

A seguir um pseudocódigo do algoritmo implementado:

```
int LCS(X)
    M = array(n, 2) // Matriz de estados do LCS

    for i = n ate 0
        charRev = X[i] // Le palavra reversamente

        for j = 0 ate n
            charCon = X[j] // Le palavra continuamente

                if charCon = charRev
                    M[i,j+1] = M[0,j] + 1
                else
                    M[i,j+1] = max(M[1,j], M[0,j+1])

        for k = 0 ate n
            M[0][k] = M[1][k] // Atualiza estado atual

return n - M[1,n]
```

3.1.1. Algoritmo implementado

- *int lcs(char * palavra, int len, Matriz * estados)*

  Descrição: Calcula o *LCS* da palavra e retorna seu valor.
  Parâmetros: Vetor da palavra, inteiro contendo o tamanho da palavra, e uma matriz de estados.
  Complexidade: $O(n^2)$, onde n é o tamanho da palavra.

3.2. Abordagem Gulosa

A abordagem gulosa escolhida foi a seguinte: o algoritmo percorre a palavra letra por letra e a cada momento verifica se essa palavra é um palíndromo ou não. Ele percorre a palavra inteira para encontrar o maior palíndromo contido nela, então de acordo com seu tamanho e o tamanho do maior palíndromo, podemos deduzir que a quantidade de letras a serem inseridas para transformar a palavra inicial em um palíndromo.

O algoritmo tem **subestrutura ótima** porém não pode ser considerado ótimo. A cada iteração ele verifica se a atual substring é um palíndromo ou não, caso seja, ela verifica se ela é maior que o palíndromo encontrado anteriormente, se for então é armazenado a quantidade de caracteres desse novo palíndromo encontrado.

O algoritmo supõe que o maior palíndromo estará no início ou no final da palavra, em outras palavras, o algortimo apenas encontra palíndromos que são sufixos ou prefixos da palavra. Então para casos em que o maior palíndromo esteja no meio da palavra o algoritmo guloso não será nem um pouco ótimo.

Logo abaixo segue um exemplo para o algoritmo ficar mais claro:

```
Palavra: ovobanana
```

```
a 1 // Marca como sendo o maior palindromo
o 1
na 0
vo 0
ana 1 // Palindromo maior do que o encontrado anteriormente
ovo 1
nana 0
bovo 0
anana 1 // Palindromo maior do que o encontrado anteriormente
abovo 0
banana 0
nabovo 0
obanana 0
anabovo 0
vobanana 0
nanabovo 0
ovobanana 0
ananabovo 0
```

Então o algoritmo encontrou o maior palíndromo (anana), que contém 5 caracteres. A resposta que ele retorna é a quantidade de letras da palavra original, menos a quantidade de letras do palíndromo. Nesse caso, o algoritmo irá precisar inserir quatro caracteres para transformar a palavra toda em um palíndromo.

3.2.1. Algoritmos implementados

- *int palindromo(char *palavra, int len)*

  Descrição: Verifica caso a palavra em questão é um palíndromo ou não
  Parâmetros: Vetor da palavra e um inteiro contendo o tamanho da palavra.
  Complexidade: $O(n)$, onde n é o tamanho da palavra.

- *int guloso(char * palavra, int len)*

  Descrição: Algoritmo que vasculha a palavra para encontrar o maior palíndromo que seja sufixo ou prefixo da palavra.
  Parâmetros: Vetor da palavra e um inteiro contendo o tamanho da palavra.
  Complexidade: $O(n^2)$, onde n é o tamanho da palavra.

4. IMPLEMENTAÇÃO

4.1. Código

4.1.1. Arquivos .c

- **tp2pd.c** Arquivo principal do programa com a abordagem de programação dinâmica, chama a função LCS e insere cada resultado no arquivo de saída.

- **tp2g.c** Arquivo principal do programa com a abordagem gulosa, chama a função ? e insere cada resultado no arquivo de saída.
- **lcs.c** Contém a implementação da função LCS, utilizada na abordagem de programação dinâmica.
- **guloso.c** Contém a definição da função guloso, utilizada na abordagem de paradigma guloso, e da função palindromo.
- **matriz.c** Contém todas as funções de manipulação, leitura e escrita de matrizes.
- **arquivos.c** Um tipo abstrado de dados de manipulação de arquivos, contendo funções de abertura, leitura, escrita e fechamento.

4.1.2. Arquivos .h

- **lcs.h** Contém a definição da função LCS, utilizada na abordagem de programação dinâmica.
- **guloso.h** Contém a definição da função guloso, utilizada na abordagem de paradigma guloso, e da função palindromo.
- **matriz.h** Além de definir a estrutura de matriz, contém o cabeçalho todas as funções de manipulação, leitura e escrita de matrizes.
- **arquivos.h** Definição da das funções utilizadas para ler, escrever e fechar corretamente um arquivo.

4.2. Compilação

O programa deve ser compilado através do compilador GCC através dos seguintes comandos

Pkara programação dinâmica:

```
gcc -Wall -Lsrc src/tp2pd.c src/arquivos.c src/matriz.c src/lcs.c -o tp2pd
```

E para o algortimo guloso:

```
gcc -Wall -Lsrc src/tp2g.c src/arquivos.c src/matriz.c src/guloso.c -o tp2g
```

Ou através do comando $make$.

4.3. Execução

A execução do programa tem como parâmetros:

- Um arquivo de entrada contendo várias instâncias de palavras.
- Um arquivo de saída que irá receber o resultado dos cálculos do menor palíndromo possível a ser gerado.

O comando para a execução do programa é da forma:

Programação Dinâmica:

```
./tp2pd <arquivo_de_entrada> <arquivo_de_saída>
```

Paradigma Guloso:

```
./tp2g <arquivo_de_entrada> <arquivo_de_saída>
```

4.3.1. Formato da entrada

A primeira linha do arquivo de entrada contém o valor k de instâncias que o arquivo contém. As próximas k linhas contém as palavras.

A seguir um arquivo de entrada de exemplo:

```
5
BOLO
XBOX
BANANA
SUCESSO
WOW
```

4.3.2. Formato da saída

O arquivo de saída consiste em k linhas com o número mínimo de caracteres necessários para gerar um palíndromo com a palavra dada de entrada na linha correspondente. Um exemplo é mostrado abaixo:

```
1
1
1
4
0
```

5. AVALIAÇÃO EXPERIMENTAL

Para testar o algoritmo realizamos quatro testes, separados em dois grupos. O primeiro grupo de testes se refere à grafos esparsos e o segundo a grafos densos.

5.1. Máquina utilizada

Segue especificação da máquina utilizada para os testes:

```
model name:     Intel(R) Core(TM) i3 CPU       M 330  @ 2.13GHz
cpu MHz:        933.000
cache size:     3072 KB
MemTotal:       3980124 kB
```

5.2. Resultado

O resultado geral dos testes ocorreu como esperado.

6. CONCLUSÃO

Conclusão

Referências

Hirschberg, D. S. *A linear space algorithm for computing maximal common subsequences*.

Szwarccfiter, J. L. (1984). *CGrafos e Algoritmos Computacionais*.

Wikipedia. Longest common subsequence problema. `http://en.wikipedia.org/wiki/Longest_common_subsequence_problem`.

Wikipedia. Palíndromo. `http://pt.wikipedia.org/wiki/Pal%C3%ADndromo`.