Discente: João Vítor Antunes Ribeiro **NUSP:** 9744910

Relatório Técnico – Projeto "*Programação Dinâmica 2 (Word Wrap)*"

- **Requisitos:** Os requisitos de *software* e *hardware* utilizados para o desenvolvimento do projeto foram, respectivamente: sistema operacional Debian 8.4.0 amd64 (2016), 4GB de memória RAM e processador Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz 2.60GHz.

- **Interface gráfica:** O projeto foi desenvolvido utilizando o *Eclipse Java EE IDE for Web Developers*, versão 4.5.2. (2016). Para implementar a interface gráfica, foi utilizada a extensão *WindowBuilder*, disponível especialmente para o *Eclipse*. A **Figura 1** ilustra a interface gráfica da ferramenta *Word Wrap – Dynamic Programming*. O menu *File* possui as opções de abrir um arquivo (*Open*) de texto (atualiza a área de "Entrada de texto"), salvar (*Save as...*) ou limpar (*Clear*) o conteúdo atual da área de "Entrada de texto". Já o menu *Compiler* contém as opções de compilar (*Compiler*) e configurar (*Word Wrap*) a compilação do texto de entrada. Em destaque na Figura 1, a janela "*Word Wrap*" ilustra a interface de configuração de compilação do texto, que permite ao usuário escolher o algoritmo de *Word Wrap* (Guloso ou Programação Dinâmica, este último implementando o conceito de "soma mínima dos quadrados dos espaços remanescentes de cada linha").

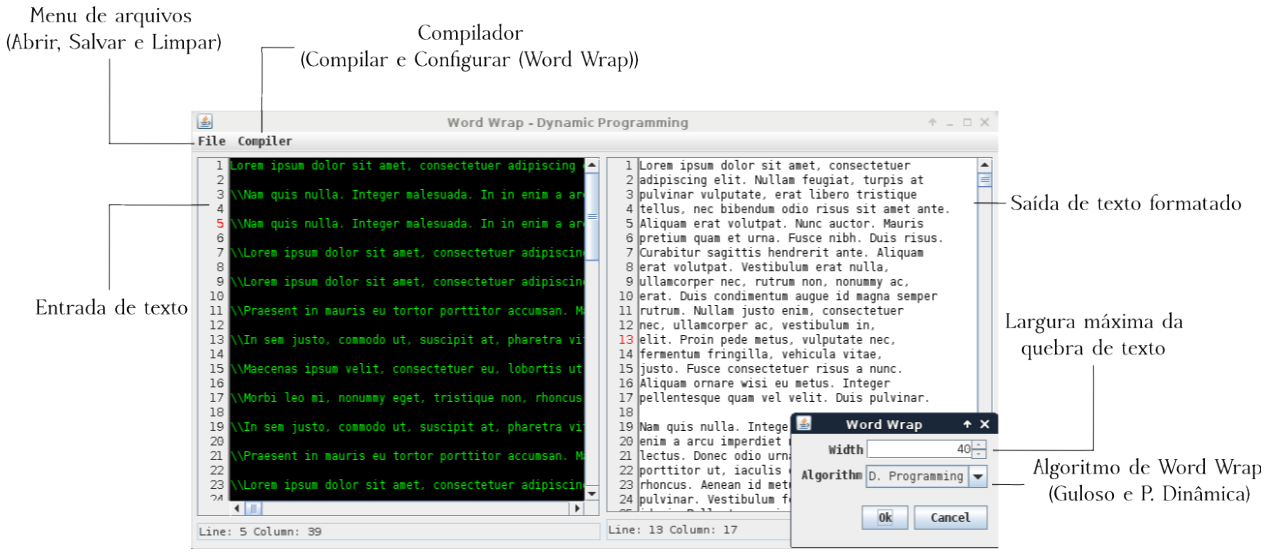

Figura 1: Interface gráfica da ferramenta *Word Wrap - Dynamic Programming*.

- **Funcionamento:** Para processar corretamente um arquivo de texto, as entradas devem estar formatadas em texto puro, com quebras de linhas indicadas por duas barras invertidas (\\), similar ao modelo adotado pelo processador de textos LaTeX. Dada qualquer entrada de texto, o usuário pode requisitar a compilação do mesmo no menu *Compiler* → *Compile*, que realizará um pré-processamento do texto de entrada, removendo espaços em branco e quebras de linhas usuais (utilizando *\n*), e exibindo (de forma não editável) o texto formatado segundo o algoritmo de *Word Wrap* (o padrão é o algoritmo guloso) e a largura máxima de quebra de texto (o padrão é 40, podendo variar entre 1 e 81) escolhidas.

- **Relações de Recorrência:** Assumindo que uma palavra é uma série contínua (sem espaços em branco e quebras de linha) de caracteres, e que nenhuma palavra ou linha l_i de palavras é maior do que a largura L máxima de quebra de linhas definida pelo *Word Wrap*, isto é, $l_i \leq L$, então a quantidade de

espaços extras ou em branco que sobram ao final de uma linha contendo as palavras de *i* à *j* podem ser definidos como a fórmula **(1)**, que pode ser um número negativo:

$$extras[i][j] = L - j + i - \sum_{k=i}^{j} l_k \quad \textbf{(1)}$$

Assim, a relação de recorrência do custo de inclusão das palavras de *i* à *j* pode ser expressa como os quadrados dos espaços extras de cada linhas do texto, ou conforme ilustrado pela equação **(2)**. O custo infinito para arranjos (*i...j*) que não cabem em uma linha evita que eles façam parte da minimização, já que o custo nesse caso seria negativo. Fazendo com que o custo da última linha seja 0 (quando um arranjo cabe na linha), evita-se que o mesmo seja influenciado pela soma minimizada.

$$lc = \begin{cases} \infty & se\ extras[i][j] < 0 \ (as\ palavras\ i \dots j\ não\ cabem\ na\ linha) \\ 0 & se\ j = n \wedge extras[i][j] \geqslant 0 \ (a\ última\ linha\ tem\ custo\ 0) \\ (extras[i][j])^2 & caso\ contrário \end{cases} \quad \textbf{(2)}$$

Como o objetivo é minimizar a soma de *lc* de todas as linhas do texto, os subproblemas são arranjar de forma ótima as palavras de *i* à *j*, com *j* variando de 1 à *n* (todas os possíveis arranjos de palavras). Considerando *c[j]* como o custo do arranjo ótimo das palavras de 1 à *j*, então pode-se afirmar que *c[j] = c[i-1] + lc{i][j]*, pois a última linha contém as palavras de *i* à *j*. Sabendo que *c[0] = 0*, têm-se que *c[1] = lc[1][1]*. Logo, o problema de minimização se resume à seguinte relação de recorrência:

$$c[j] = \begin{cases} 0 & se\ j = 0 \\ min\,(c[i-1] + lc[i][j]) & se\ j > 0, 1 \leqslant i \leqslant j \end{cases} \quad \textbf{(3)}$$

Dessa forma, sabe-se que que todos os arranjos escolhidos caberão nas linhas, pois os arranjos que não cabem foram definidos como infinito (fórmula **(2)**) e não serão escolhidos na minimização. Para mapear quais arranjos de palavras devem ser inseridos em cada linha, foi utilizado um vetor auxiliar *substringRanges[]*, com base no vetor *c*. Quando *c[j]* é calculado, *c[j]* é baseado no valor de *c[k-1]*, então pode-se fazer *substringRanges[j] = k*. A última linha começa na palavra *substringRanges[n]*, e a anterior começa na palavra *substringRanges[substringRanges[n]]* e vai até *substringRanges[n]-1*. A relação de recorrência expressão por (3) implica que a complexidade de tempo do algoritmo é $O(n^2)$, dado que é necessário testar todos os arranjos *i...j*. Já a complexidade de espaço também é $O(n^2)$, pois *lc* precisa armazenar os $(n^2-n)/2$ custos dos arranjos em uma matriz triangular superior, para evitar recálculos e fazer com que o tempo de acesso à informação dos custos seja constante.

➢ **Referência Bibliográficas:**

[1] ***Line wrap and word wrap***, Disponível em: https://en.wikipedia.org/wiki/Line_wrap_and_word_wrap, Acessado em: 30/06/16.

[2] TENG, Shang-Hua, ***Solution to Word Wrap Problem***, Disponível em: http://www-bcf.usc.edu/~shanghua/teaching/Spring2010/public_html/files/HW3_Solutions_A.pdf, Acessado em: 30/06/16.