MAC0323 - Algoritmos e Estruturas de Dados II
Segundo semestre de 2020

Quarto Exercício-Programa – Devolução: 19 de dezembro

Grafos de textos[1]

O objetivo deste exercício-programa é fazer um estudo de dicionários de palavras. Usaremos para isso uma modelagem por meio de grafos, e os experimentos envolverão algoritmos que vimos nas aulas de processamento de grafos.

O grafo em que serão feitos os experimentos será construído a partir de um texto dado como entrada do programa. Você poderá fazer uso de uma das implementações de tabela de símbolos que fez no seu EP3. Além do programa, você deverá entregar um relatório com os resultados dos seus experimentos.

Construção do grafo

O seu programa deve ler da linha de comando um número k e o nome de um arquivo texto. Em seguida, deve ler o texto dado no arquivo e montar a lista de palavras com pelo menos k letras que ocorrem no texto.

O grafo que estudaremos tem como vértices o conjunto das palavras do texto com pelo menos k letras. As arestas do grafo representam as três possíveis situações:

- *remoção ou inserção de letra:* duas palavras são vizinhas se uma é igual à outra com uma de suas letras removidas ou inseridas. Por exemplo, `alta` e `ata` são vizinhas no grafo, assim como `casa` e `casca`.

- *troca de letras de uma mesma palavra:* duas palavras são vizinhas se uma pode ser obtida da outra a partir da troca de posição de duas letras da outra. Por exemplo, `nato` e `nota` são vizinhas, assim como `nota` e `tona`.

- *substituição de uma letra:* duas palavras são vizinhas se uma é obtida da outra substituindo-se uma de suas letras por outra. Por exemplo, `gato` e `rato` são vizinhas, assim como `prata` e `preta`.

Numa primeira etapa, o seu programa deve construir esse grafo a partir do texto lido do arquivo. A seguir, descrevemos as rotinas que serão utilizadas nos experimentos que faremos sobre este grafo.

[1]Esse EP foi preparado pelo Prof. Carlos Eduardo Ferreira, e adaptado para esse oferecimento da disciplina.

Investigação dos grafos

Você deve executar experimentos pelo menos para os seis arquivos de teste disponibilizados no diretório de dados do EP4. Acrescente outros arquivos de teste se achar interessante.

Usando as rotinas da interface que será descrita na próxima seção, e eventualmente outras que você decida implementar, você deverá investigar propriedades de cada um dos grafos construídos.

Primeiramente seu programa deve imprimir as seguintes informações básicas sobre cada grafo:

- número de vértices e arestas;
- grau mínimo, máximo e médio dos vértices do grafo;
- densidade do grafo, que é o número de arestas do grafo dividido pelo número máximo de arestas que ele poderia ter, considerando quantos vértices ele tem.

Numa segunda fase, ele deve determinar e imprimir informações relacionadas à conexidade:

- número de componentes conexas do grafo;
- tamanho mínimo, máximo e médio das componentes conexas do grafo.

Na terceira fase, ele deve determinar e imprimir informações relacionadas ao diâmetro das componentes do grafo:

- distância máxima e média em cada componente do grafo.

Um vértice cuja remoção aumenta o número de componentes conexas do grafo é chamado de *ponto de articulação*. É possível determinar os pontos de articulação de um grafo usando uma DFS, ou seja, em tempo linear no tamanho do grafo. Leia mais sobre isso na página Articulações e componentes biconexas, do Prof. Paulo Feofiloff.

Na quarta e última fase, seu programa deve determinar e imprimir a seguinte informação extra relacionada a conexidade do grafo:

- quantidade de pontos de articulação;
- lista das palavras associadas aos pontos de articulação, caso o número de pontos de articulação seja no máximo 20.

Além disso, para auxiliar nos testes de correção do seu programa, você deve implementar duas rotinas que constam da interface pedida, que determinam a componente conexa em que um determinado vértice está, e se um tal vértice é ou não um ponto de articulação. Note que estas rotinas não fornecerão o modo mais eficiente para calcular as informações pedidas acima. Elas apenas nos auxiliarão a verificar se o seu programa está correto, e podem auxiliá-lo a determinar aspectos interessantes destes grafos, a serem reportados no relatório.

Relatório

No relatório, além de apresentar as informações mencionadas acima para cada um dos arquivos de teste disponibilizados, e sobre outros que você ache interessantes, gostaríamos que você fizesse uma reflexão do que observou nos testes.

Os seis arquivos disponibilizados vêm em pares: dois arquivos mais longos, com uma peça e um romance (`les_miserables.txt` e `tales.txt`), dois artigos mais curtos (`turing.txt` e `lovelace.txt`), e dois arquivos com programas em java (`avl.java` e `red-black.java`).

Tente identificar se existem similaridades e diferenças entre arquivos de um mesmo tipo, faça outros testes, com outros arquivos, tentando identificar semelhanças ou diferenças nos grafos obtidos. Relate, no relatório, semelhanças ou diferenças observadas nos testes que chamaram a sua atenção, aspectos destes grafos que você achou interessantes ou inesperados, e outras observações que você ache relevante reportar. Elabore, reflita e relate o que observou.

Esqueleto do EP e interface para os experimentos

Disponibilizamos um esqueleto do programa, que já contém algumas rotinas prontas, bem como uma interface `graph.h` que você deve implementar, e usar nos seus experimentos.

Segue uma versão enxugada da interface.

```
#define TOTAL 0
#define ARTICULACAO 1

/*
 * void inicializa(int k);
 *
 * Inicializa um grafo com parametro k.
 */
void inicializa(int k);

/*
 * int vizinho(char * a, char * b);
 *
 * Recebe duas strings, a e b.
 * Retorna 1 caso a e b sejam vizinhas e 0 caso contrario.
 */
int vizinho(char * a, char * b);

/*
 * int insere(char * palavra);
 *
 * Insere a string palavra no grafo e retorna
 * a quantidade de arestas que foram adicionadas ao grafo.
 *
 * Retorna -1 caso a palavra nao tenha sido adicionada
 * (seja por ja estar no grafo ou por nao possuir tamanho >= k)
 */
int insere(char * palavra);
```

```
/*
 * int vertices(int tipo)
 *
 * Se tipo == TOTAL,
 * retorna a quantidade total de vertices do grafo
 *
 * Se tipo == ARTICULACAO,
 * retorna a quantidade de vertices de articulacao do grafo
 */
int vertices(int tipo);

/*
 * int arestas()
 *
 * Retorna a quantidade total de arestas do grafo
 */
int arestas();

/*
 * int grau(char * a)
 *
 * Retorna o grau do vértice representado pela string a
 */
int grau(char * a);

/*
 * double densidade()
 *
 * Retorna a densidade do grafo
 */
double densidade();

/*
 * int componentes()
 *
 * Retorna a quantidade de componentes conexas do grafo
 */
int componentes();

/*
 * int articulacao(char * a)
 *
 * Retorna 1 se o vertice da string a é de articulação,
 * 0 se não é, -1 se a string a nao esteja no grafo
 */
int articulacao();

/*
 * int tamanhoComponente(char * a)
 *
 * Retorna o tamanho da componente conexa que contem
 * o vertice da string a, -1 se a string a nao esteja no grafo
 */
int tamanhoComponente(char * a);

/*
 * int distancia(char * a, char * b)
 *
 * Retorna a quantidade de arestas no caminho do vertice que
 * contem a string a para o vertice que contem a string b
 */
int distancia(char * a, char * b);
```