Algoritmos e Estruturas de Dados II

EP2: Grafos de Texto

Profº Carlos Ferreira
Atenágoras Souza Silva (nºUSP: 5447262)
athenagoras@gmail.com

1 Introdução

Neste trabalho foi implementado um grafo de texto, obedecendo os seguintes critérios para estabelecer as arestas do mesmo:

Os vértices do grafo são palavras, e as arestas devem ligar palavras parecidas dentro do seguinte critério:

- remoção de uma letra: Duas palavras são vizinhas caso tornem-se iguais após a remoção de uma letra de uma delas
- inserção de uma letra: Duas palavras são vizinhas caso tornem-se iguais após a inserção de uma letra em uma delas
- troca de letras de uma mesma palavra: Duas palavras são vizinhas caso tornem-se iguais após a troca de letras em uma delas
- subistituição de letra: Duas palavras são vizinhas caso tornem-se iguais após a substituição de uma letra em uma delas

Com o grafo construido, puderam ser feitos experimentos sobre medidas do grafo, conexidade, caminhos e ciclos do mesmo.

2 Compilação e modo de uso

Para compilar o projeto, basta digitar

$ make -k

no prompt e ter o g++ (compilador de C++ da coleção de compiladores gcc) disponível.

Uma vez pronto, para gerar um grafo de texto com base nos critérios acima, é necessário chamar o programa tendo como argumento um arquivo de texto.

Exemplo:

$ ./grafosT manifesto.txt

Chama o programa grafosT com o texto manifesto.txt para que um grafo de texto e os experimentos sejam realizados.

O programa solta resultados em arquivos terminados com -GrafoAnalise. No caso do exemplo, seria manifesto.txt-GrafoAnalise

3 Métodos

Para executar os experimentos, foi necessário compilar o programa como exposto acima, desenvolvendo métodos para a classe de grafos, utilizando-se de outras estruturas de dados de suporte como uma tabela de símbolos, estruturas para extrair texto de livros, fila (para a busca em largura) e a criação de uma interface de realização dos experimentos propostos.

Foram utilizados quatro textos em inglês para rodar com o programa gerador de grafos de texto:

- Manifesto Comunista (Marx)
- Crítica da Economia Política (Marx)
- Os Lusíadas (Camões)

- Textos de Sherlock Holmes (Doyle)

A utilização de textos em inglês é preferível, pois neste trabalho, a construção de grafos em textos com caracteres em UTF-8 poderia não ser bem sucedida.

Para a realização dos experimentos, foram construidas funções na interface de experimento que recebem o grafo, extraem as métricas, conexidade e informações quanto a caminhos e ciclos dele, e imprimem os resultados em um relatório para cada texto.

3.1 Fluxo do Programa

O programa passa pelo seguinte fluxo enquanto é executado:

1. Extrai as palavras do livro selecionado
2. Armazena-as em uma tabela de símbolos implementada por árvores binárias
3. Armazena as palavras da tabela de símbolos no grafo de texto
4. Cria as arestas com base nas regras expostas na introdução 1
5. Executa os experimentos (Métricas, Conexidade, Caminhos e Ciclos)

3.2 A interface de experimento

A interface de experimento é chamado pelo arquivo principal do programa (grafosT) com a TSO em que as palavras do livro foram armanezadas, e apresenta o seguinte código:

```
void Experimentos(char *nomeArq, AB *tso){
  char saida[256];
  strcpy(saida, nomeArq);
  strcat(saida, "-GrafoAnalise");
  FILE *Saida = fopen(saida, "w");
  // Grafo G = Grafo(tso);
  Grafo G(tso);
  G.vizinhas(); // Gera as arestas ao procurar palavras vizinhas
  medidas M = Medidas(Saida, &G);
  printMedidas(nomeArq, &M);
  conexidade C = Conexidade(Saida, &G);
  printConexidade(nomeArq, &C);
  caminhosCiclos Ca = CaminhosCiclos(Saida, &G);
  printcaminhosCiclos(nomeArq, &Ca);
  fclose(Saida);
}
```

As estruturas e funções que fazem os experimentos tem os seguintes cabeçalhos:

```
#include "ab.h"
#include "grafo.h"

typedef struct {
  char nomeArq[256];
  int vertices;
  int arestas;
  char mesmaOrdGrandeza[10]; // Mesma Ordem de grandeza? Sim ou Não
  float den; // Densidade
  float mediaGraus;
} medidas;
void printMedidas(char *nome, medidas *M);
```

```
typedef struct {
  char nomeArq[256];
  int componentes;
  int maior; // maior número de componentes
  int menor; // menor número de componentes
  float mediaComp; // média de compomentes
} conexidade;
void printConexidade(char *nome, conexidade *C);

typedef struct {
  char nomeArq[256];
  float distMedia; // distância média entre duas palavras
} caminhosCiclos;
void printcaminhosCiclos(char *nome, caminhosCiclos *Ca);

medidas Medidas(FILE *fp, Grafo *g);
conexidade Conexidade(FILE *fp, Grafo *g);
caminhosCiclos CaminhosCiclos(FILE *fp, Grafo *g);
void Experimentos(char *nomeArq, AB *tso);
```

As estruturas acima foram utilizadas para criar um arquivo .csv menos prolixo, destinado a virar tabelas que foram inseridas posteriormente neste relatório

As funções Medidas(), Conexidade(), CaminhosCiclos() fizeram o processamento dos experimentos.

Em CaminhosCiclos(), foram sorteadas duas palavras para verificar se existe caminho entre elas, e se iniciam um circuito.

Para checar a distância médias entre duas palavras, foram sorteadas duas palavras repetidamente ($\frac{V}{2}$, onde V é o número de Vértices do grafo), e calculadas as distâncias entre elas, quando da existência de um caminho ligando-as.

3.3 Classe Grafo

```
#include "livro.h"
#include "ab.h"
// #include "Fila.h"

class Adjacencia {
public:
  int v; // vértice de destino
  Adjacencia *prox;
  Adjacencia(int i);
};

class Vertice {
public:
  palavra p; // Informação do Vértice, no caso a palavra do dicionário
  int grau;
  Adjacencia *cab; // Cabeça da lista de adjacências
  Vertice();
  ~Vertice();
};

class Grafo {
private:
  bool removeLetra(int v1, int v2);
  bool insereLetra(int v1, int v2);
  bool trocaLetras(int v1, int v2);
```

```
  bool substituiLetra(int v1, int v2);
  void dfsR_imprime(int v, bool *marked);
  void dfsR_circuito(int vif, int v, int *flag, bool *marked);
  int dfsR(int v, bool *marked, int *tamC); // auxilia o dfs e algoritmos que dependem disso e inform
public:
  int V; // Nº de Vértices
  int A; // Nº de Arestas
  Vertice *adj; // Adjacências do grafo (cada vértice e suas adjacências)

  Grafo(AB *tso);
  Grafo(int N, palavra *palavras); // Cria um gráfico com N vértices a partir de um livro de palavras
  ~Grafo();
  void novaAresta(int v1, int v2);
  void vizinhas(); // procura por palavras vizinhas. Quando encontradas, são produzidas arestas
  // Busca em profundidade: Visita todos os vértices em profundidade
  void dfs_imprime();
  void dfs();
  bool circuito(int v); // Verifica se há um circuito para o qual o vértice v é o início e fim
  bool *circuitos(); // Procura por circuitos em todos os vértices
  palavra devolve(int v); // devolve a palavra no vértice V

  // Conta quantas componentes conexas tem um grafo. Retorna o vetor tamCon com o tamanho das compone
  int compConexas(int *tamCon);
  int *caminhoMaisCurto(int v);
};
```

Os métodos privados removeLetra, insereLetra(), trocaLetras() e substituiLetra() foram utilizados para auxiliar o método vizinhas() a construir as arestas do grafo.

Quando vizinhas() é chamado, chama essas funções auxiliares para gerar as arestas, chamando o método novaAresta() para tal. Este, por sua vez, cria 2 arestas entre os vértices (a criação de um grafo não dirigido está implícita nas regras para construção de aresta).

Os métodos dfsRimprime, dfsRcircuito e dfsR() auxiliam métodos de busca em profundidade associados, respectivamente, à impressão dos caminhos a partir de um vértice, ao checar se um vértice é o início e o fim de um circuito, e à própria busca em profundidade completa.

4 Resultados

4.1 Medidas

Tabela 1: Medidas dos grafos de texto

Arquivo	Vértices	Arestas	Mesma Ordem de Grandeza	Densidade (A/V^2)	Média de Graus em um vértice
criticaeconomiapoliticaIngles.txt	6497	2092	Não	0.000050	0.321995
manifestocomunistaIngles.txt	2625	518	Não	0.000075	0.197333
sherlock.txt	8067	2310	Não	0.000035	0.286352
lusiadasIngles.txt	14461	6002	Não	0.000029	0.415047

4.2 Conexidade

Tabela 2: Conexidade dos grafos de texto

Arquivo	Componentes	Maior Componente	Menor Componente	Média de Componentes por vértice
criticaeconomiapoliticaIngles.txt	5927	429	1	1.096170
manifestocomunistaIngles.txt	2460	141	1	1.067073
sherlock.txt	7339	455	1	1.099196
lusiadasIngles.txt	12773	1046	1	1.132154

4.3 Caminhos e Ciclos

- criticaeconomiapoliticaIngles.txt-GrafoAnalise

Palavras Sorteadas:
palavra 1: colonies vértice 1016
palavra 2: deals vértice 1457
Não existe caminho da palavra 1 à palavra 2.
Não existe caminho da palavra 2 à palavra 1.
A palavra 1 não inicia um circuito.
A palavra 2 não inicia um circuito.
- manifestocomunistaIngles.txt-GrafoAnalise

Palavras Sorteadas:
palavra 1: fantastic vértice 883
palavra 2: fashion vértice 886
Não existe caminho da palavra 1 à palavra 2.
Não existe caminho da palavra 2 à palavra 1.
A palavra 1 não inicia um circuito.
A palavra 2 não inicia um circuito.

- sherlock.txt-GrafoAnalise

Palavras Sorteadas:
palavra 1: writings vértice 8029
palavra 2: blundering vértice 757
Não existe caminho da palavra 1 à palavra 2.
Não existe caminho da palavra 2 à palavra 1.
A palavra 1 não inicia um circuito.
A palavra 2 não inicia um circuito.

- lusiadasIngles.txt-GrafoAnalise

Palavras Sorteadas:
palavra 1: writings vértice 8029
palavra 2: blundering vértice 757
Não existe caminho da palavra 1 à palavra 2.
Não existe caminho da palavra 2 à palavra 1.
A palavra 1 não inicia um circuito.
A palavra 2 não inicia um circuito.

Tabela 3: Caminhos e Ciclos para os grafos de texto

Arquivo	Distância Média entre duas palavras
criticaeconomiapoliticaIngles.txt	0.019397
manifestocomunistaIngles.txt	0.017530
sherlock.txt	0.009422
lusiadasIngles.txt	0.022130

5 Análise e Conclusão

No que diz respeito às medidas brutas, nos quatro textos, a ordem de grandeza de Arestas e Vértices^2 não é a mesma, pois todos os grafos apresentaram uma muito baixa densidade. Na verdade, com as regras dadas para construir arestas neste grafo de texto, o número de arestas é da ordem de grandeza dos vértices, apenas (com exceção de lusiadasIngles.txt). A média de graus por vértice também ficou abaixo de 1.

Quanto à conexidade, a média de componentes por vértice foi, em média, 1 para todos os grafos construidos, ou seja, a maioria dos vértices não estão ligados a nenhum outro, como sugere, aliás, a proximidade entre o número de vértices e o número de componentes em todos os grafos de texto construidos.

Analisando os Caminhos e ciclos nenhuma das palavras sorteadas inicialmente nos grafos de texto apresentaram um caminho entre elas, nem formam um ciclo. Entretanto, ao sortear várias delas para calcular a distância entre elas, notou-se que a distância entre uma e outra é próxima de zero (na verdade, entre 0,009 e 0,022). Com este resultado, pode-se inferir que as palavras que estão ligadas, o fazem diretamente, praticamente sem outras palavras entre elas.

Os textos submetidos com os quais foram construiram os grafos de texto possum tamanhos variando entre 2625 e 14461 vértices contendo, cada um, uma palavra diferentes em inglês. Dois versam sobre política e economia, e outros dois são textos ficcionais.

Pode-se concluir que apresentam números semelhantes na análise dos grafos, ou quando não tão semelhantes, apresentam a mesma qualidade, como por exemplo o de serem pouco densos, apesar dos diferentes números de densidade.

Uma sugestão que encareceria o processo de construir as arestas do grafo, por possivelmente aumentá-las, além de exigir mecanismos de inteligência artificial seria classificar as palavras gramaticalmente, por exemplo, verbos, substantivos, adjetivos etc. Isto permitiria uma melhor diferenciação de textos técnicos de outros textos mais ficcionais ou históricos, caso isto seja necessário.