TEORIA DOS GRAFOS

COS242

25 de setembro de 2013

Dupla: Guilherme Sales e Thiago Barroso Perrotta

Relatório da parte 1 do Trabalho

1. Do objetivo

O objetivo deste trabalho de disciplina é projetar e desenvolver uma biblioteca para manipular grafos. A biblioteca deverá ser capaz de representar grafos assim como implementar um conjunto de algoritmos em grafos. Esta biblioteca deverá ser projetada e desenvolvida de forma que possa ser facilmente utilizada em outros programas.

2. Das decisões de projeto

Optamos por desenvolver a nossa biblioteca inteira **na linguagem C++,** mais especificamente com o compilador gcc (g++). Dentre alguns motivos para essa decisão, destacam-se:

- Linguagem de **médio** nível:
  - não ser em baixo nível implica que não é uma linguagem tão difícil de ser entendida e utilizada (como Assembly) por nós humanos: código legível;
  - não ser em alto nível implica que é uma linguagem relativamente rápida; o gcc (g++) possui muitas otimizações para gerenciamento de memória e aumento de velocidade;
- Existência da biblioteca STL (*Stantard Templates Library*), que implementa eficientemente diversos algoritmos e estruturas de dados usados que utilizamos com frequência;
- Suporta orientação a objetos, permitindo que nosso código seja melhor estruturado através de classes (métodos e atributos);
- Possui uma grande comunidade de usuários e desenvolvedores, tornando fácil a consulta a referências e tira-dúvidas (especialmente no *website* de Q&A *Stack Overflow*);
- Linguagem de código aberto e de natureza bastante compatível e integrada com um ambiente GNU/Linux e com os editores GNU Emacs e Gedit, que é os editores de texto que estamos utilizando, acompanhados com um Makefile;
- Nós dois já tínhamos conhecimento prévio da linguagem, o que facilita bastante o trabalho de escrever o código.
- É relativamente fácil realizar testes **simples** do código, utilizando a função *assert* da biblioteca *cassert*.

Outra decisão relevante para o projeto foi a de usar o *software* aberto **GIT**. Tivemos as seguintes principais vantagens e razões para usá-lo:

- Sistema de **controle de versão**, de modo que poderíamos voltar para qualquer ponto do projeto quando desejássemos (caso cometêssemos erros futuros);
- Solução melhor integrada para **trabalhar em paralelo** em várias partes diferentes do mesmo código e depois **juntar as modificações**. Em particular, a facilidade de ver os *diffs* entre cada *commit* facilitaram enormemente saber o que foi modificado.
- Integração com a **nuvem**, de modo que era bastante fácil editar/obter o código de qualquer lugar; e também sistema de notificações de *commits* por e-mail.
- Sempre tivemos vontade de aprender e trabalhar com um sistema CVS. Achamos que essa era uma ótima oportunidade para fazer isso.

Também utilizamos o *debugger* GDB algumas vezes, principalmente para detectar erros como falha de segmentação no código.

Outra decisão para o projeto: desenvolvê-lo completamente com ferramentas livres (pelo menos até então, e o máximo que pudermos), o que é mais filosófico do que relevante para o trabalho, mas significa que acreditamos na qualidade do software livre. Algumas ferramentas livres já foram citadas, como o git, o gedit, o utilitário make, e o GNU Emacs. Além disso, estamos desenvolvendo o código completamente em um ambiente **GNU/Linux**, e editando este documento com o LibreOffice.

3. Da estrutura do projeto

Para essa primeira parte do trabalho, dividimos nossa biblioteca em 3 arquivos:

- graph.h
- graph.cpp
- graph_test.cpp

O primeiro inclui simplesmente o *header* do projeto (assinaturas de função etc) e a definição da classe principal do grafo. Ele é uma das chaves para manter o código organizado e modularizado. Resolvemos fazer assim pensando no futuro, pois isso tornará mais fácil, posteriormente, acrescentar novas funcionalidades à nossa biblioteca.

O segundo contém a implementação de todas as funções listadas no *header*. É o maior arquivo de todos. Esse arquivo poderia ser dividido em vários outros mas, para simplificar, preferimos manter um arquivo só por enquanto.

O terceiro arquivo possui um papel fundamental para a biblioteca, que é assegurar a sua **validade**! Nele incluímos uma pequena suíte de testes para a biblioteca, o que significa que incluímos vários casos de testes de grafos como entrada, visando testar vários métodos e funções da nossa classe. Note que os

testes (no que diz respeito a sua implementação) são bastante simples, constituindo basicamente de vários *asserts*.

Também temos um Makefile para compilar o projeto adequadamente.

Eventualmente abusamos de vários arquivos de texto para servir como entrada de testes durante o desenvolvimento do projeto. No entanto, após refinação do código, esses arquivos passam a ser integrados no graph_test.cpp.

4. Estudo de Caso 1

1. Medimos a quantidade de memória utilizada por cada representação usando o comando free -m antes e durante a duração de um programa que só guardava o grafo na estrutura de dados pedida. Os resultados foram os seguintes (em megabytes):

Matriz: 1243, 1225, 1233, 1213, 1234, 1228, 1256, 1243, 1244 ==> MÉDIA: **1235 MB**

Lista: 10, 0, 0, 3, 3, 2, 2, 3, 8, 2 ==> MÉDIA: **3 MB**

2. Usamos a função clock() do <ctime>, medimos o tempo antes e depois de rodar bfs(source). Obtivemos os seguintes resultados:

Vértice 1:

- matriz: 5800, 5840, 5860, 5840, 5860 ==> MÉDIA 5840 ms
- lista: 20, 0, 0, 0, 20 ==> MÉDIA 8 ms

Vértice 10:

- matriz: 5800,5860,5800,5840,5800 ==> MÉDIA 5820 ms
- lista: 20, 0,20,0,0 ==> MÉDIA 8 ms

Vértice 100:

- matriz: 5800,5860,5860,5800,5800 ==> MÉDIA 5824 ms
- lista: 0,20,0,20,0 ==> MEDIA 8 ms

Vértice 1000:

- matriz: 5840, 5800, 5880,5860,5840 ==> MÉDIA 5844 ms
- lista: 20,0,0,0,20 ==> MEDIA 8 ms

Vértice 42:

- matriz: 5880,5860,5780,5840,5800 ==> MÉDIA 5832 ms
- lista: 0,0,0,20,0 ==> MEDIA 4 ms

3. Ver arquivo answers/a1-3 para a distribuição empírica. O maior grau do grafo é o do vértice 3691, como podemos ver no arquivo anterior. Basta ver a primeira linha de answers/a1-3sorted.

```
cat a1-3 | awk '{print $2 " " $1}' | sort -r > a1-3sorted
```

Esse grau vale delta(G) =

Comando: cat as_graph.txt | awk '{ if($1=="1") print $1 " " $2}' | wc

O menor grau do grafo é zero (existem vários nós isolados).

Isso significa que existe um contraste grande entre os graus de nós encontrados nesse grafo: uns poucos nós com grau [relativamente] alto, e muitos nós isolados.

4. Existem, no total, 14386 componentes conexas.

5. Estudo de Caso 2

1. Ver arquivo answers/a2-1 para a distribuição empírica. O maior grau do grafo é o do vértice 1, como podemos ver no arquivo anterior. Esse grau vale delta(G) = 2161;
Comando: cat as_graph.txt | awk '{ if($1=="1") print $1 " " $2}' | wc

O menor grau do grafo é zero (existem vários nós isolados).

Isso significa que existe um contraste grande entre os graus de nós encontrados nesse grafo: uns poucos nós com grau [relativamente] alto, e muitos nós isolados.