SCC0503 (Algoritmos e Estruturas de Dados II) — Prof. Moacir P. Ponti Junior

Trabalho 1

Implemente sua atividade sozinho sem compartilhar, olhar código de seus colegas, ou buscar na Internet. Procure usar apenas os conceitos já vistos nas aulas.

Modelagem de Problemas e Busca em Grafos

Este trabalho é uma adaptação de um trabalho proposto pelo Prof. M. Böhlen.

Uma rede sem fio pode ser modelada como um grafo onde os vértices são dispositivos de computação localizados em um espaço físico. O grafo contém uma aresta (u, v) se o dispositivo v está perto o suficiente para receber e trasnmitir sinais para um dispositivo u. Dois dispositivos estão conectados de modo não confiável, se a conexão pode ser destruída através da remoção de apenas um dispositivo da rede. Tal dispositivo é chamado de vértice crítico.

Na Figura 1, os vértices **1** e **4** são exemplos de dispositivos conectados de modo não confiável, pois deixarão de se comunicar caso o vértice **3** seja removido (**3** é, portanto, um vértice crítico).

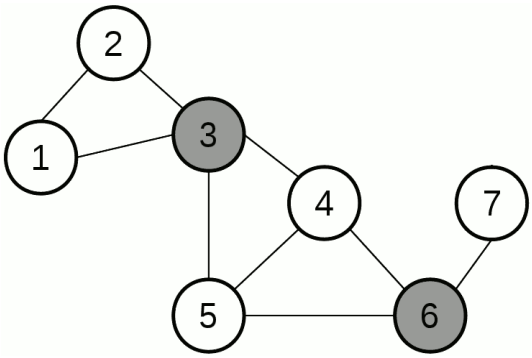

Figura 1: Exemplo de rede com dois vértices críticos: 3 e 6.

Tarefa

Escreva um programa que testa se dois vértices v e u são conectados de modo não confiável. Utilize, para isso, o algorimto de busca em largura (BFS). Utilize como estrutura de dados para a conectividade dos vértices uma lista de adjacência. A entrada do programa será um grafo não dirigido conforme o ilustrado anteriormente. A complexidade de sua implementação não deve ser maior que $\mathcal{O}(|V| \cdot (|V| + |E|))$, sendo $|V|$ o número de vértices e $|E|$ o número de arestas.

Exemplo de entrada

A entrada do programa será o número de vértices seguido do número de arestas, seguidos das arestas que representam a conexão entre dois dispositivos. Na forma:

```
<n Vertices>
<m Arestas>
<u_1 v_1>
...
<u_M v_M>
```

Por exemplo, para o grafo da Figura 1, temos:

```
7
9
1 2
1 3
2 3
3 4
3 5
4 5
4 6
5 6
6 7
```

Note que estamos trabalhando com grafos não dirigidos. Portanto, é necesário fazer o tratamento correto da entrada pois, caso seja fornecido a aresta `1 2`, subintende-se que `2 1` também exista.

Exemplo de saída

A saída deve mostrar todos os vértices críticos da rede, **um em cada linha**. No caso da Figura 1, temos:

```
3
6
```

Caso não haja vértices críticos, deve ser mostrado `-1` na saída padrão.

Critérios

O projeto será avaliado principalmente levando em consideração:

1. Processamento correto das entradas e saídas do programa;
2. Realização das tarefas descritas;
3. Bom uso das técnicas de programação;
4. Boa endentação e uso de comentários no código;
5. Boa estruturação e modularização do código.

Implementação

As seguintes funções devem, **obrigatoriamente**, ser implementadas:

- `int is_unreliably_connected(Vertex u, Vertex v);`
- `int is_empty();`
- `void print_critical_vertices();`
- `void enqueue(Vertex v);`
- `Vertex dequeue();`

Você poderá criar outras funções se quiser ou precisar lembre-se de tornar a função criada útil para os propósitos de reuso e abstração, e de comentá-la corretamente.

Restrições:

- Não deverá ser utilizada qualquer variável global.
- Não poderão ser utilizadas bibliotecas com funções prontas (a não ser aquelas para entrada, saída e alocação dinâmica de memória).

Uso de cabeçalhos e Makefile:

- O(s) arquivo(s) de cabeçalho (`.h`) deverá(ão) conter toda a implementação das funções para manipulação do grafo, incluindo estrutura de dados, percurso e outros.
- O arquivo fonte (`.c` ou `.cc`) deverá conter apenas o processamento referente à entrada e saída dos dados.
- Mais informações sobre como usar Makefiles e gerar bibliotecas pode ser encontrado em: `http://wiki.icmc.usp.br/images/0/0a/ApostilaMakefiles2011.pdf`

Dicas

As estruturas podem ser modificadas, de forma que o grafo contenha a lista de adjacência, e haja um arranjo de vértices para armazenar as informações de cada um. Uma sugestão:

```
struct digraph {
   int V;              // total de vertices
   int A;              // total de arestas
   VertexList *list;  // arranjo de vertices (a ser alocado)
};

typedef struct {
  int visited;   // -1 se vertice nao foi visitado, ou >=0 caso contrário
  int pred;      // armazena indice do vertice predecessor no percurso
  int critical;  // 1 se for vertice critico, 0 se nao for
  Link adj;      // ponteiro para a lista de adjacencia
} VertexList;
```

ATENÇÃO

- O projeto deverá ser entregue apenas pelo Sistema de Submissão de Programas (SSP)[1], escolhendo a opção `Trabalho1`. Há um casos de teste para serem baixados na página da disciplina.O sistema receberá trabalhos **apenas** entre os dias 31/03 as 0h00 e 07/04 as 23h59. Não serão aceitos trabalhos com atraso.

- Leia as observações contidas na tela de submissão de trabalhos do sistema.

- Todos os arquivos do projeto (`Makefile`, arquivos `.h` e arquivos `.c`) deverão ser armazenados no mesmo diretório. Esse diretório deverá ser compactado com a extensão `.zip`.

- Dúvidas conceituais deverão ser colocadas nos horários de atendimento. Dificuldades em implementação, por favor, envie e-mail para o estagiário PAE ou para o monitor da disciplina com o assunto `[trab1]duvida`, anexando o código e especificando o problema.

- A detecção de cópia de parte ou de todo código-fonte, de qualquer origem, implicará reprovação direta no trabalho. Partes do código cujas **ideias** foram desenvolvidas em colaboração com outro(s) aluno(s) devem ser devidamente documentadas em comentários no referido trecho. O que **NÃO** autoriza a cópia de trechos de código nem a codificação em conjunto. Portanto, compartilhem ideias, soluções, modos de resolver o problema, mas não o código. Qualquer dúvida entrem em contato com o professor.

[1] `http://netuno.icmc.usp.br/ssp01/`