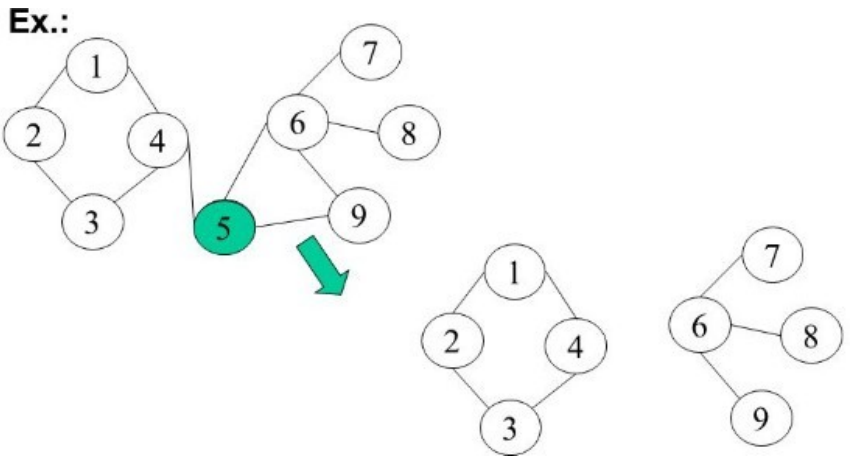
ACH2024 – Algoritmos e Estruturas de Dados II - Profa: Ariane Machado Lima

EXERCÍCIO PROGRAMA 1

Data de entrega: 05 de maio de 2024

Neste EP1, vocês devem resolver o problema abaixo usando busca em profundidade (DFS) e busca em largura (BFS) que aprendeu na aula. O objetivo deste exercício-programa é implementar os algoritmos básicos e usá-los em um problema de visualização.

Um vértice de articulação é um vértice que, se for removido do grafo, aumenta o número de componentes conectados do grafo em pelo menos um. Uma possível abordagem para identificar se um dado vértice é um ponto de articulação é identificar se o número de componentes conectados do grafo (não direcionado) original é menor que o número de componentes conectados do grafo sem aquele



dado vértice e as arestas que saem/entram nele. De maneira similar é para grafos direcionados, considerando apenas que neste caso fala-se em “componentes fortemente conectados”.

Essa abordagem simples, descrita no parágrafo anterior, será aceita no EP. Contudo essa estratégia é ineficiente O(V*(V+A)). Há um algoritmo O(V+A) que é capaz de identificar os pontos de articulação apenas analisando a árvore da busca em profundidade executada sobre o grafo original. DICA: Na árvore de busca em profundidade:

- um vértice folha .... (pode ou não pode ser um ponto de articulação?);
- o vértice raiz é um ponto de articulação se e somente ...;
- um vértice *x* que não é folha nem raiz é um ponto de articulação se e somente se *x* possui um vértice filho *y* tal que ... (alguma coisa a ver com a sub-árvore com raiz em *y*...)

Este algoritmo eficiente para grafos direcionados é proposto no artigo publicado no periódico *Theoretical Computer Science*, vol 447, ano 2012, páginas 74–84.

Vocês podem discutir a solução entre vocês, só não podem trocar códigos.

Você tem que implementar um algoritmo que recebe a entrada e retorna a saída no formato especificado.

Este EP é uma simplificação do problema sugerido em SPOJ JUDGE: http://www.spoj.com/problems/GRAPHS12/

Descrição:

Neste problema, seu programa terá que ler um grafo ponderado **não direcionado** de um arquivo cujo nome é entrada.txt e executar alguns algoritmos básicos no grafo:

- Busca em largura:
  - após a linha "BL:" devem ser impressos os vértices na ordem em que eles foram descobertos durante a busca em largura (separados por um espaço);
  - após a linha "Caminhos BL": deve ser impresso, em cada linha i, o caminho do vértice raiz em questão [1] até o vértice i durante a BL (vértices separados por um espaço).

- Busca em profundidade:
  - após a linha "BP:" devem ser impressos os vértices na ordem em que eles foram descobertos durante a busca em profundidade (separados por um espaço);
  - após a linha "Caminhos BP": deve ser impresso, em cada linha i, o caminho do vértice raiz em questão[2] até o vértice i durante a BP (vértices separados por um espaço).

- Visualização de componentes conectados: cada componente conectado *i* deve ser impresso em uma linha que se inicia com "C*i*: ". A impressão de um componente significa imprimir seus vértices (separados por um espaço) em ordem crescente. A ordem dos compomentes é dada pela ordem crescente dos vértices de menor número de cada componente. Por exemplo, um componente que contém o vértice 0 será o C1. Se houver outros dois componentes, C2 será aquele que tem o menor vértice com número menor que o menor vértice do terceiro componente.

- Visualização dos vértices de articulação (os números dos vértices que são pontos de articulação): uma única linha contendo todos os vértices que são vértices de articulação (separados por um espaço).

Sobre o programa fonte e forma de execução:

Seu programa deve funcionar tanto para implementação usando matriz de adjacência e lista de adjacência. Para facilitar o processo de teste, iremos informar qual implementação iremos usar (matriz ou lista) via diretiva do compilador. Para isso, você deverá entregar exatamente **cinco** arquivos com essas especificações:

- 2 arquivos com a implementação de grafos por matriz de adjacência (**grafo_matrizadj.c** e **grafo_matrizadj.h**);
- 2 arquivos com a implementação de grafos por lista de adjacência (**grafo_listaadj.c** e **grafo_listaadj.h**);
- 1 arquivo contendo a função main (**ep1_XXXX.c**) e as rotinas de buscas, etc (utilizando apenas as funções definidas na interface de grafos, ou seja, sem assumir explicitamente a estrutura por matriz ou lista de adjacência), em que **XXXX** é seu nr USP; **este arquivo deve incluir em suas primeiras linhas os includes dos arquivo .h de grafos EXATAMENTE DESTA MANEIRA:**

```
#ifdef MATRIZ
#include "grafo_matrizadj.h"
#else
#include "grafo_listaadj.h"
#endif
```

O arquivo **Makefile** não deverá ser entregue, pois seu EP será testado com o Makefile disponilizado como modelo. Este Makefile é capaz gerar dois executáveis: **ep1_matriz.exe** e **ep1_lista.exe** por meio dos seguintes comandos, respectivamente:

- $ make CFLAGS+="-DMATRIZ" ep1_matriz
- $ make ep1_lista

Obs: o arquivo Makefile precisa ter exatamente esse nome (case-sensitive), sem NENHUMA extensão. Cuidado para não ter nem mesmo uma extensão oculta, senão não será identificado pelo programa *make*.

Cada um desses executáveis deve receber na linha de comando dois strings: os nomes dos arquivos de entrada e de saída, nesta ordem. Ou seja, eles devem ser executados da seguinte forma:

- $ ep1_matriz.exe <nome do arquivo de entrada> <nome do arquivo de saída>
- $ ep1_lista.exe <nome do arquivo de entrada> <nome do arquivo de saída>

Por exemplo:

- $ ep1_matriz.exe entrada1.txt saida1.txt
- $ ep1_lista.exe entrada1.txt saida1.txt

Não utilize redirecionamento da entrada-padrão com o caractere “<” (menor) ou da saída-padrão como caractere “>” (maior);

Especificação do arquivo de entrada:

Você deverá ler o grafo a partir do arquivo de entrada.

Na primeira linha você terá um par de números: V e A, sendo o primeiro (V) a quantidade de vértices no grafo, e o segundo (A) a quantidade de arestas no grafo.

As próximas “A” linhas conterão uma tripla em cada linha representando uma aresta: o, d e p, na qual o é o vértice origem de uma aresta, d é o vértice destino e p é o peso. Você DEVE inserir no grafo as arestas na mesma ordem em que elas forem descritas no arquivo.

Os números dos vértices são consecutivos: 0, 1, 2, ..., V-1.

Especificação do arquivo de saída:

Você terá que mostrar todas as informações na lista mostrada na seção acima (na descrição). Dê uma olhada no Exemplo abaixo para ver o formato a seguir. Toda a saída deve ser impressa no arquivo de saída informado na linha de comando.

---

1. Se o grafo não for conexo, haverá mais de uma árvore de busca em largura. Assim, cada vértice pertence a uma única árvore com uma raiz específica.
2. Se o grafo não for conexo, haverá mais de uma árvore de busca em profundidade. Assim, cada vértice pertence a uma única árvore com uma raiz específica.

Exemplo:

Entrada:

7 8
5 6 1
0 2 1
0 1 4
3 5 7
3 4 1
1 3 1
2 3 1
4 6 1

Saída:

BL:
0 1 2 3 4 5 6

Caminhos BL:
0
0 1
0 2
0 1 3
0 1 3 4
0 1 3 5
0 1 3 4 6

BP:
0 1 3 2 4 6 5

Caminhos BP:
0
0 1
0 1 3 2
0 1 3
0 1 3 4
0 1 3 4 6 5
0 1 3 4 6

Componentes Conectados:
C1: 0 1 2 3 4 5 6

Vertices de articulacao:
3

Informações adicionais:

- O código deve ser implementado na linguagem C;
- Assuma que o número máximo de vértices é 100;
- Note que o nome do arquivo ep1_XXXX.c deve ter no lugar do XXXX seu nrUSP (não esqueça de ajustar essa informação também no Makefile).
- Os 6 arquivos devem ser postados individualmente no edisciplinas (não compactá-los!), sem pastas ou subpastas.

Os EPs que não seguirem o padrão de nomenclatura, formatação de saída e nome dos arquivos terão pontuação reduzida.

Qualquer evidência de plágio entre trabalhos, mesmo que parcial, implicará na nota zero no trabalho!