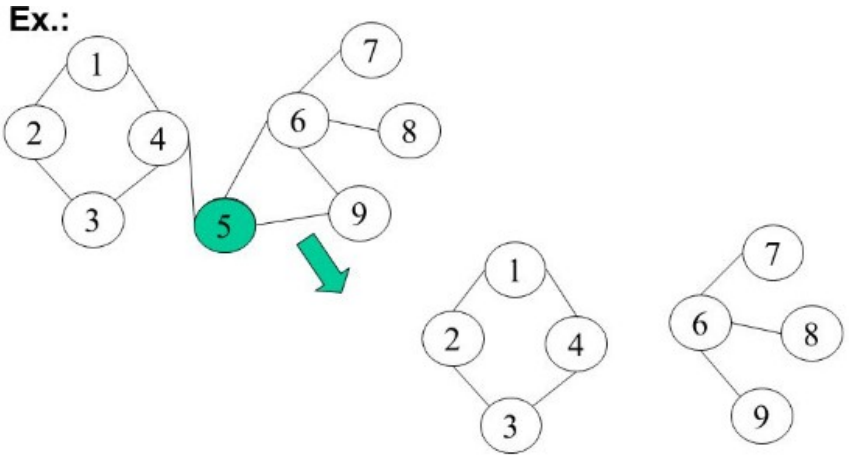
ACH2024 – Algoritmos e Estruturas de Dados II - Profa: Ariane Machado Lima

EXERCÍCIO PROGRAMA 1

Data de entrega: 06 de junho de 2021

Neste EP1, INDIVIDUAL, você deve 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 vértice 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*...)

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

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 (peso inteiro) **não direcionado** de um arquivo cujo nome é entrada.txt e executar alguns algoritmos básicos de grafo:

- Visualização do grafo: impressão do grafo no mesmo formato da entrada

- 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.

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

Especificação da entrada

Você deverá ler o grafo a partir de um arquivo “entrada.txt” (*hard-coded* em seu programa).
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. Cada aresta será sempre descrita de forma que o vértice origem será o de menor número que o vértice alvo (o mesmo vale para a impressão do grafo na saída). Já a lista de arestas não seguirá uma ordem definida no arquivo de entrada. No entanto, você DEVE inseri-las no grafo na mesma ordem em que elas forem descritas, principalmente se você utilizar a implementação por listas de adjacências.
Os números dos vértices são consecutivos: 0, 1, 2, …, V-1.

---

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.

Saída

A primeira informação a ser descrita na saída é o próprio grafo, seguindo o mesmo padrão do arquivo entrada.txt. A diferença é que, no arquivo de saída, as arestas (triplas) devem ser impressas ordenadas pelo vértice origem de forma crescente. Além disso, se houver mais de uma aresta saindo de um mesmo vértice origem, elas devem ser impressas na ordem em que aparecem na estrutura de dados utilizada (matriz ou lista). Lembre-se que inserção em lista de adjacência é sempre no início da lista.
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 "saida.txt" (*hard-coded* em seu programa)

Exemplo

entrada: (arquivo entrada.txt)
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: (arquivo saida.txt)
7 8
0 1 4
0 2 1
1 3 1
2 3 1
3 4 1
3 5 7
4 6 1
5 6 1

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:
4

Os arquivos de entrada e saída de seu EP devem ser “entrada.txt” e “saida.txt”, respectivamente (*hard-coded* em seu programa).

Informações adicionais:

- Você deve fazer um relatório explicando seu algoritmo de identificação de vértices de articulação **didaticamente** (pdf de só uma página).
- O código deve ser implementado na linguagem C.
- Assuma que o número máximo de vértices é 100.
- Vocês poderão optar por usar a implementação de grafos por matriz ou listas de adjacência.[3]
- O trabalho deverá ser postado no edisciplinas como um arquivo compactado, que deverá ser nomeado:
  NomeCompletoTudoJuntoCadaNomeIniciadoComLetraMaiúscula.tar (ou .gz, .rar, etc...)
  Este pacote deve conter 5 arquivos:
  - o arquivo .h contendo as definições e protótipos para grafos (utilizando matriz ou lista de adjacência)

3 Particularmente eu acho que é mais interessante para vocês implementarem usando listas de adjacências, pois além de ser a que mais tem a ver com o EP (já que usa buscas em largura e profundidade), é o tipo de implementação mais indicada na maioria dos casos, e, portanto, com mais chances de vocês precisarem utilizar em breve na vida profissional e/ou acadêmica de vocês. Lembrem-se que se usarem as interfaces sugeridas em aula, muito do EP será independente da implementação por matrizes ou listas de adjacência.

- o arquivo .c contendo a implementação da estrutura de dados de grafos (matriz ou lista de adjacência)
- conectividade.c : o código do EP1 propriamente dito, que usa a API de grafos (dos dois arquivos anteriores)
- Makefile: para compilação do EP. Ele deve gerar um executável chamado **conectividade.exe**
- NomeCompletoTudoJuntoCadaNomeIniciadoComLetraMaiúscula.pdf (contendo o relatório).

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!