UNIVERSIDADE FEDERAL DE VIÇOSA
DEPARTAMENTO DE INFORMÁTICA
INF330 - Teoria e Modelo de Grafos
Segunda prova -- 06/11/2018
Prof. Salles Magalhaes

Aluno:

Matricula:

Obs:
- Nesta prova assuma que o uso de “includes” e de diretivas do tipo “using namespace std;” não seja necessário.
- A não ser que a questão afirme de forma explícita o contrário, assuma que os grafos são simples
- Diferentemente de C++, a ordem com que funções são declaradas não importa (ou seja, se uma função x() chama y() → y() não precisa ser declarada antes de x())

1 (8 pontos) - Implemente a função abaixo (você poderá criar funções auxiliares se necessário). Dada a lista de adjacência (*adj*) de um grafo G (não direcionado), ela deverá armazenar em *compsCompletos* o número de componentes conexos de G que são subgrafos completos (ou seja, você deverá contar um componente conexo <-> o componente inteiro for um grafo completo) e deverá armazenar em *compsComCiclos* o número de componentes conexos de G que contêm ciclos (ou seja, você deverá contar um componente conexo C <-> houver pelo menos um ciclo em C).

void analisa(const vector<vector<int>> &adj, int &compsCompletos, int &compsComCiclos) {

2 (3 pontos) - Considere uma árvore (com raiz) representada por um vetor “pai” (ou seja, pai[v]

representa o pai de cada vértice *v* da árvore, sendo o pai da raiz representado por -1). Implemente uma função que, dado o vetor “pai” (argumento da função), cria uma matriz de adjacência (chame-a de “mat”) e uma lista de adjacência (chame-a de “adj”) para a árvore (a árvore no grafo resultante não deverá ter raiz -- ou seja, deverá ser não direcionada). Obs 1: você não precisa se preocupar em retornar o que foi criado -- basta criar e assumir que algo será feito com as estruturas de dados criadas. Obs 2: você poderá criar funções auxiliares se necessário.

```
void criaListaEMatrix(const vector<int> &pai) {
        //Declare aqui a lista de adjacencia e a matriz de adj



        //Implemente o codigo para preencher a lista de adjacencia e a matriz de adj
```

3 (2 pontos) - Dada uma floresta G representada por uma matriz de incidência, termine a implementação da função abaixo, que deverá retornar true se G for bipartida e false se G não for bipartida (você poderá criar funções auxiliares se necessário).

```
bool verificaFlorestaEhBipartida(vector<vector<bool> > &matrizIncidencia) {
```

4 (7 pontos)- Esta questão foi adaptada de um problema utilizado em uma entrevista de emprego para Engenheiro de Software 3 da Amazon/Seattle. Dada a dependência de compilação de alguns pacotes Java, determine a ordem em que eles devem ser compilados de modo a satisfazer essas restrições. A entrada começará com dois inteiros representando o número de pacotes (P) e o número de dependências (D). A seguir, há D linhas cada uma contendo um par de inteiros (a,b) indicando que o

pacote b depende do pacote a para poder ser compilado.

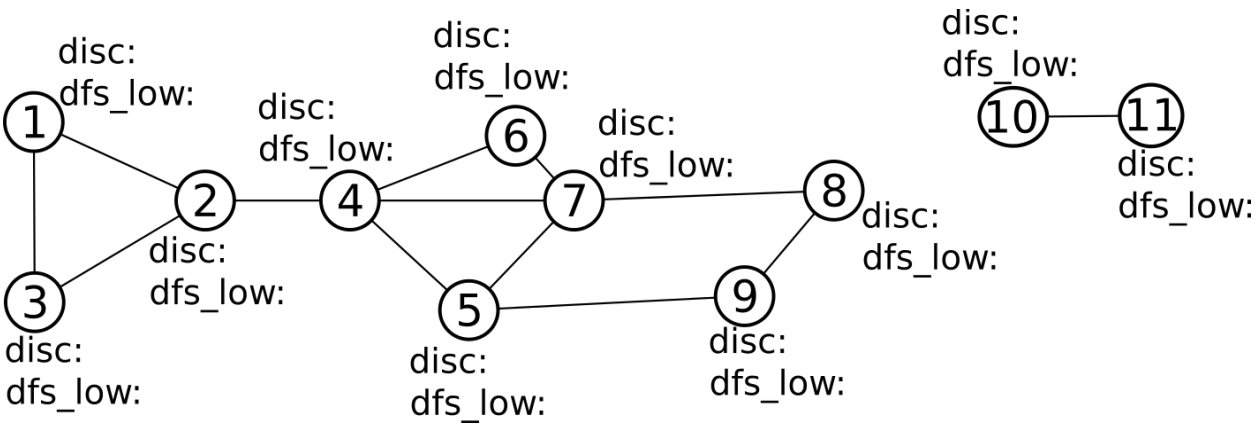
Exemplo de entrada (a explicação entre parênteses não faz parte da entrada):
4 3 (ha 4 pacotes que precisam ser compilados e 3 restrições sobre a ordem de compilacao)
1 2 (o pacote 1 só pode ser compilado após a 2 ter sido compilado)
1 3 (o pacote 1 só pode ser compilado após a 3 ter sido compilado)
2 3 (o pacote 2 só pode ser compilado após a 3 ter sido compilado)

Possivel saida: 4 3 2 1
(se os pacotes forem compilados nessa ordem, as dependências acima serão satisfeitas)

Para facilitar, suponha que a entrada já foi lida e que sua tarefa será apenas implementar uma função para processá-la. Termine a implementação da função abaixo (que, dado o número de pacotes e um vetor contendo as dependências, imprime a saída do problema (dependencias[i].first so pode ser compilado após dependencias[i].second ter sido).

void imprimeOrdem(int pacotes, const vector<pair<int,int> > &dependencias) {

5 (3 pontos) - Dado o grafo abaixo, rotule os vértices usando a DFS de Tarjan (escreva os rótulos de cada vértice em frente a “disc” (também conhecido como dfs_num) e “dfs_low”). Supondo que a busca começará no vértice **8** e que o grafo está armazenado em uma lista de adjacência ordenada (onde os vizinhos de um vértice são sempre processados em ordem crescente de rótulo).
Atenção: confira com bastante atenção seus resultados!! Um erro no início pode se espalhar e reduzir muito sua nota.



Rascunho:

6 (2 pontos) - Sobre buscas DFS e BFS:
a) Qual das duas é mais eficiente? (Justifique, por exemplo, apresentando a ordem de complexidade delas ou algum outro argumento)

b) Apresente um problema que pode ser resolvido facilmente com uma BFS (mas não com uma DFS)

c) Apresente um problema que pode ser resolvido facilmente com uma DFS (mas não com uma BFS)

d) Apresente um problema que pode ser resolvido facilmente com qualquer uma das duas buscas