|  |  |
| --- | --- |
|  | Pontifícia Universidade Católica de Minas Gerais **Bacharelado em Ciência da Computação**  **Algoritmos em Grafos**  **Prof. Alexei Machado** |

**1ª Prova**

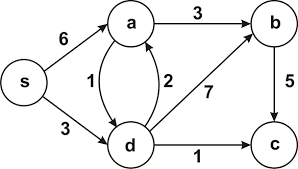
Nome:

1. Seja G um grafo de ordem *n* e seja G’ o grafo que se obtém ao acrescentar a G um novo vértice *v* e uma aresta ligando *v* a *n-1* vértices de G. Quais as características de G para que G’ seja Euleriano? (5 pontos)

**Resposta**:

G deve ter n – 1 vértices de grau ímpar e n – 1 deve ser par.

1. Determine a matriz de adjacências para o grafo abaixo:



**Resposta**:

a b c d s

a 0 3 0 1 0

b 0 0 5 0 0

c 0 0 0 0 0

d 2 7 1 0 0

s 6 0 0 3 0

1. Qual é o número mínimo de arestas necessárias para garantir que um grafo bipartido *n  m* seja conexo? Justifique. (5 pontos)

**Resposta**:

Para n <= m, n arestas são suficientes para conectar 2n vértices do total e isso consome todos os vértices do conjunto N. Ainda, m – n arestas são necessárias para que os vértices restantes de M sejam conectados aos vértices de N. Isso deixa faltando apenas as arestas que criarão os caminhos do grafo conexo que, em quantidade, são n – 1. Total = (n) + (m – n) + (n – 1) = n + m – 1.

Isso é equivalente a criar um grafo bipartido que é uma árvore.

1. Uma fazenda possui um poço artesiano e *n* bebedouros para o gado. Deseja-se construir uma ligação que leve água do poço a todos os bebedouros gastando a menor quantidade de canos. (5 pontos)
2. Modele o problema como um grafo, indicando o que são os vértices, as arestas e os pesos.
3. Proponha uma solução para o problema, indicando o tipo de problema de grafos associado e 2 algoritmos para resolve-lo.

**Respostas**:

**a)** Vértices: bebedouros; Arestas: canos; Pesos: distância do bebedouro ao poço.

**b)** Menor quantidade de canos indica menor quantidade de arestas. Ou seja, estamos diante do problema da árvore geradora mínima. Algoritmos de Prim e Kruskal.

1. Baseado no algoritmo de geração de permutações abaixo, escreva um algoritmo recursivo que determina se 2 grafos com vértices V1 e V2 e com matrizes de adjacências E1 e E2 são isomorfos. (5 pontos)

Permuta (v, i, n)

se i=n então Imprima(v);

senão para k:=1 até n – i +1 faça

Permuta (v, i +1, n)

Rotaciona (v, i, n)

fim para

fim se

Chamada Principal: Permuta(v,1,n)

**Resposta**:

Como só consegui entender o problema implementando-o na mão e pesquisando em vários lugares diferentes, deixarei aqui o código JavaScript que desenvolvi. Caso queira testá-lo, basta copiar tudo, abrir o console do seu navegador e colá-lo lá.

function mostrar(grafo)

{ console.log(JSON.stringify(grafo).replace(/\]\,/g, "],\n")); }

function objetosSaoIguais(matriz1, matriz2)

{ return JSON.stringify(matriz1) == JSON.stringify(matriz2); }

function permutarColunas(grafo, coluna1, coluna2)

{

for (let i = 0; i < grafo.length; i++)

    // Troca os valores das duas colunas um a um verticalmente

    [ grafo[i][coluna1], grafo[i][coluna2] ] = [ grafo[i][coluna2], grafo[i][coluna1] ];

}

function saoIsomorfosR(grafo1, grafo2, start)

{

    let iguais = false;

    if (start == grafo2.length)

    {

        mostrar(grafo2); console.log();

        iguais = objetosSaoIguais(grafo1, grafo2);

    }

    else //

        for (let i = start; !iguais && i < grafo2.length; i++)

        {

            [ grafo2[i], grafo2[start] ] = [ grafo2[start], grafo2[i] ]; // Permuta linhas

            permutarColunas(grafo2, i, start); // Permuta colunas

            // Permuta os vértices a partir do índice start + 1

            iguais = saoIsomorfosR(grafo1, grafo2, start + 1);

            permutarColunas(grafo2, i, start); // Desfaz a permutação das colunas

            [ grafo2[i], grafo2[start] ] = [ grafo2[start], grafo2[i] ];// Desfaz as linhas

        }

    return iguais;

}

function saoIsomorfos(grafo1, grafo2)

{

    console.log('Grafo original:\n'); mostrar(grafo1);

    console.log('\nGrafo a comparar:\n'); mostrar(grafo2);

    console.log('\nIniciando a permutação do grafo acima...\n');

    return saoIsomorfosR(grafo1, grafo2, 0);

}

let grafo1 = [

    [0, 1, 1, 0],

    [1, 0, 0, 1],

    [1, 0, 0, 1],

    [0, 1, 1, 0],

];

let grafo2 = [

    [0, 1, 0, 1],

    [1, 0, 1, 0],

    [0, 1, 0, 1],

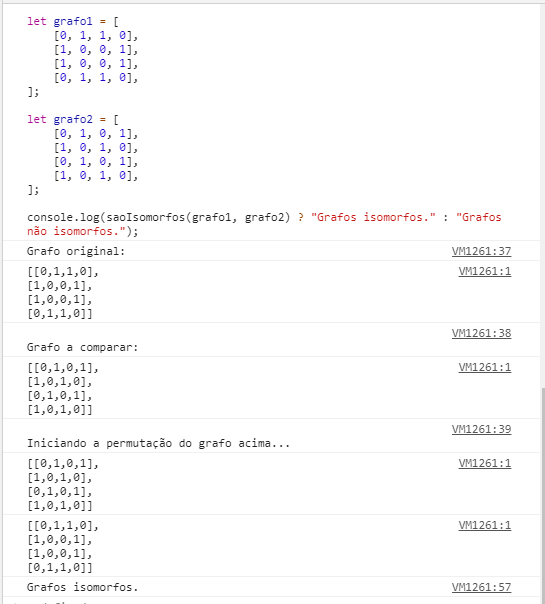
    [1, 0, 1, 0],

];

console.log(

    saoIsomorfos(grafo1, grafo2) ? "Grafos isomorfos." : "Grafos não isomorfos.");

Exemplo do algoritmo rodando:



1. Para o algoritmo de Dijkstra descrito abaixo: (5 pontos)
2. O que significa marcar um vértice como visitado? Justifique.
3. Porque a presença de pesos negativos pode fazer com que o algoritmo falhe? Exemplifique.
4. Inicializações
5. Escolher um vértice não visitado x cuja distância mínima para V0 seja a menor conhecida. Se x for NULO, termine o algoritmo
6. Marcar x como visitado
7. **Para** cada vizinho i não visitado de x

se d(V0,x) + aresta(x,i) < d(V0, i)

(i) d(V0, i) = d(V0, x) + aresta(x,i)

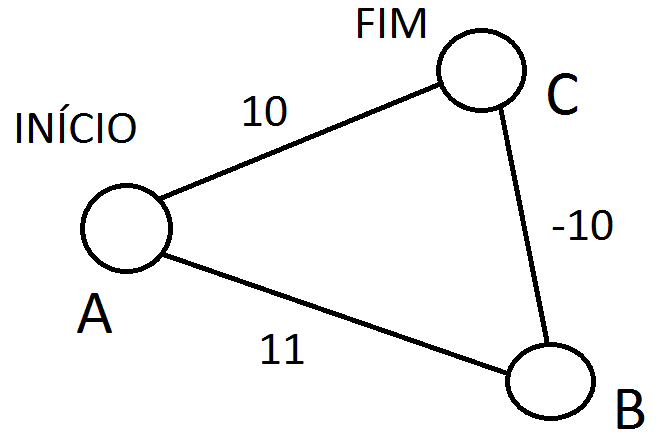
(ii) c(i) = c(x) + i

1. Se existirem vértices não visitados voltar para o passo (2)

**Respostas**:

**a)** Significa não escolhê-lo mais no passo 2, ou seja, depois dessa marcação, na próxima iteração, o algoritmo já terá analisado os caminhos do vértice inicial aos vértices sucessores do que foi marcado.

**b)**

 O caminho direto de A para C é mais custoso do que o que passa por B pois

AC = 10 e AB + BC = 11 + -10 = 1.