Lista de Algoritmos para Implementação do 2º Bimestre – Teoria dos Grafos

As resoluções dos exercícios devem ser implementadas em grupo de 3 alunos e podem utilizar a linguagem C ou a linguagem Python.

Os códigos devem ser comentados e devem rodar em um compilador online ; os links das resoluções devem ser compartilhados em um arquivo PDF enviado pelo blog;

Este arquivo PDF deve ter o enunciado de cada problema e o link com o código com a resolução;

2. *Entrada:* Matriz de adjacências de um grafo simples e ponderado ou grafo direcionado e dois vértices no grafo
Saída: Distância do caminho mínimo entre dois vértices ou uma mensagem de que não existe tal caminho; os vértices no caminho mínimo, se ele existir
{Dica: Você precisará de um modo para denotar quais vértices pertencem a *IN.)*

3. *Entrada:* Matriz de adjacências de um grafo simples, ponderado e conexo
Saída: Arestas (na forma de pares ordenados) da árvore geradora mínima

Outro algoritmo para obtenção de caminhos mínimos a partir de um único vértice de origem para todos os outros vértices no grafo é o *algoritmo de Bellman-Ford.* Ao contrário do algoritmo de Dijkstra, que mantém um conjunto de vértices para os quais o caminho mínimo já foi determinado, independentemente do comprimento desses caminhos (isto é, o número de arestas no caminho); o algoritmo de *Bellman-Ford* realiza uma busca que visa encontrar sucessivamente caminhos de comprimento 1, depois de comprimento 2, depois de comprimento 3 etc, até um máximo de $n - 1$ como comprimento (se existir um caminho, ele não pode ter comprimento maior do que $n - 1$). Uma descrição em pseudocódigo do algoritmo de *Bellman-Ford é* fornecida a seguir (algoritmo *OutroCaminhoMínimo);* quando usamos este algoritmo, a matriz de adjacências A deve conter $A[i, i] = 0$ para todo i.

ALGORITMO *OutroCaminhoMínimo,*

procedure *OutroCaminhoMínimo{A:* matriz n X n; *x:* vértice;
var *d:* vetor de inteiros;
var *s:* vetor de vértices);
{ Algoritmo de Bellman-Ford —*A* é uma matriz de adjacências modificada para um grafo simples, ponderado e conexo; *x é* um vértice no grafo; quando o procedimento termina, os vértices no caminho mínimo de *x* a *y* são *y, s[y], s[s[y]],..., x;* a distância deste caminho é *d[y].}*

var
z,p: vértice; {vértices temporários}
t: vetor de inteiros; {vetor temporário de distâncias criado a cada iteração*}*
i: integer;

```
begin
  {inicia os vetores d e s — estabelece os caminhos mínimos de comprimento 1 a partir de x }
  d[x]:=0;
  for todos os vértices z diferentes de x do
    begin
      d[z] := A[x, z];
      s[z]:=x;
    end;

  {encontra os caminhos mínimos de comprimentos 2, 3, etc.}
  for i: = 2 to n — 1 do
    begin
     t := d; {copia o vetor d corrente em t}
     {altera t a fim de guardar os menores ca\ninhos de comprimento i}
     for todos os vértices z diferentes de x do
        begin
          {encontra o caminho mínimo com uma aresta a mais}
          p := vértice em G para o qual (d[p] + A[p, z]) é mínimo
          t[z]:=d[p] +A[p,z];
          if not (p = z) then
            s[z] := p;
        end;

      d := t; {copia t de volta para d}
    end; {laço for}
end;
```

Nos Exercícios 9 a 12, use o algoritmo *OutroCaminhoMínimo* (algoritmo de Bellman-Ford) para encontrar o caminho mínimo entre o vértice origem e todos os outros vértices. Mostre os sucessivos valores de d e de s.

★•9. Grafo dos Exercícios 1 -4, vértice origem = 2 (compare sua resposta com a do Exercício 1)

10. Grafo dos Exercícios 1-4, vértice origem = 1 (compare sua resposta com a do Exercício 3)

11. Grafo dos Exercícios 5-6, vértice origem = 1 (compare sua resposta com a do Exercício 5)

12. Grafo a seguir, vértice origem = 1 (compare sua resposta com a do Exercício 8)

O algoritmo de Kruskal é outro algoritmo para a obtenção da árvore geradora mínima de um grafo. Enquanto no algoritmo de Prim a árvore "cresce" a partir de um ponto inicial arbitrário, através da inclusão de arestas pequenas adjacentes, o algoritmo de Kruskal inclui arestas na ordem de suas distâncias crescentes, sempre que elas puderem pertencer ao grafo. Os empates são resolvidos arbitrariamente. A única restrição é que uma aresta não seja incluída se a inclusão dela resulta em um ciclo. O algoritmo termina quando todos os vértices tiverem sido incorporados a uma estrutura conexa. Um descrição (muito informal) em pseudocódigo é mostrada a seguir:

```
ALGORITMO  OutroAGM

procedure OutroAGM (A: matriz n X n;
                    var T: coleção de arestas);
{Algoritmo de Kruskal para obtenção de uma árvore geradora mínima; T inicialmente está vazia; ao fim, T
= árvore geradora mínima}

begin
  ordena as arestas de G de forma crescente
  repeat
  if próxima aresta na ordem não cria um ciclo then
    inclui a aresta em T;
  until T é conexo e contém todos os vértices de G;
end;
```

Nos Exercícios 19 a 22 use o algoritmo *OutroAGM* (algoritmo de Kruskal) para encontrar a árvore geradora mínima.

★19. Grafo dos Exercícios 1 -4

20. Grafo do Exercício 16

Nos Exercícios 4 a 6, primeiro escreva uma rotina em que entre as informações referentes a um grafo do usuário e então monte sua representação na forma de uma lista de adjacências; incorpore este procedimento nos programas pedidos.

4. *Entrada:* Informação sobre um grafo (veja instruções acima) e um vértice no grafo
   Saída: Vértices em uma busca em profundidade no grafo, começando no vértice dado

5. *Entrada:* Informação sobre um grafo (veja instruções acima) e um vértice do grafo
   Saída: Vértices em uma busca em largura no grafo, começando no vértice dado

7. Escreva um programa que permita ao usuário entrar uma lista de inteiros, e então construa uma árvore binária de busca (veja Seção 5.4) com esses inteiros como vértices. O usuário pode então entrar o tipo de busca desejada (inordem, preordem ou posordem) e o programa escreve os vértices da árvore na ordem da busca selecionada.