Tópico: **Introdução a Grafos**

1. Considere o tipo abstrato de dados (TAD) definido abaixo para representar grafos:

```
//TAD para grafo G = (V,E), adaptado de:
// Sedgewick, R. Algorithms in C, Part 5, Graph Algorithms, 3a edição, 2002.

typedef struct {int v; int w;} Aresta;

// cria uma aresta (v,w)
Aresta ARESTA(int v,int w);

typedef struct grafo * Grafo;

//inicializa um grafo com |V| vertices
Grafo inicializa(int V);

//Insere a aresta e no grafo g
void grafoInsere(Grafo g, Aresta e);

//Remove a aresta e do grafo g
void grafoRemove(Grafo g, Aresta e);

//Preenche o vetor com as arestas de g
//Retorna |E|
int grafoArestas(Aresta a[], Grafo g);

//imprime um grafo G = (V,E);
void imprimirGrafo(Grafo g);
```

   (a) Crie uma implementação para o TAD grafo que use a representação de matriz adjacência.

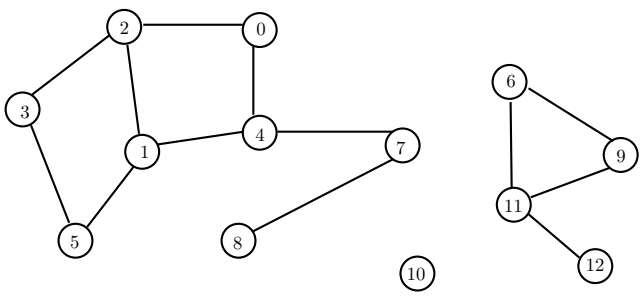
   (b) Crie outra implementação para o mesmo TAD grafo que use a representação de listas de adjacência.

   (c) Implemente uma função `int pertence(Grafo g, Aresta e)` que testa se a aresta `e` pertence ao grafo `g`.

   (d) Implemente a busca em profundidade a partir de um vértice de origem s, tal que imprima a árvore de busca obtida.

   (e) Implemente a busca em largura a partir de um vértice de origem s, tal que imprima a árvore de busca obtida.

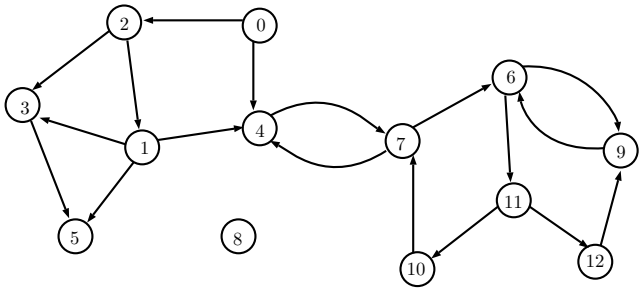
2. Dado o grafo abaixo:



(a) execute a busca em largura a partir do vértice 2, preenchendo as estruturas de dados auxiliares (`cor`$[v]$, $\Pi[v]$, `d`$[v]$).

(b) execute a busca em profundidade a partir do vértice 5, preenchendo as estruturas de dados auxiliares (`cor`$[v]$, $\Pi[v]$, `d`$[v]$, `f`$[v]$).

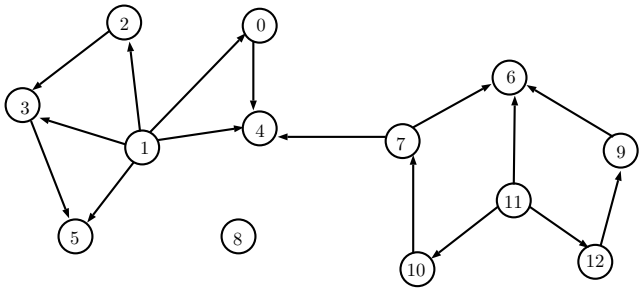
3. Dado o grafo abaixo:



(a) execute a busca em largura a partir do vértice 2, preenchendo as estruturas de dados auxiliares (`cor`$[v]$, $\Pi[v]$, `d`$[v]$).

(b) execute a busca em profundidade a partir do vértice 2, preenchendo as estruturas de dados auxiliares (`cor`$[v]$, $\Pi[v]$, `d`$[v]$, `f`$[v]$).

4. Dado o grafo abaixo, apresente uma ordenação topológica:



5. Dado o grafo abaixo, execute o algoritmo de Dijkstra para encontrar o menor de caminho do vértice 3 para os demais vértices, indicando os valores da distância `d`$[v]$ estimada a cada passo.