Lista de Exercícios - Estruturas de Dados II

Grafos

1. Defina grafo, vértice e aresta.

2. Defina e exemplifique
   a) grafo direcionado.
   b) grafo não direcionado.

3. Defina grau de um vértice e ciclos em um grafo direcionado e em um grafo não direcionado.

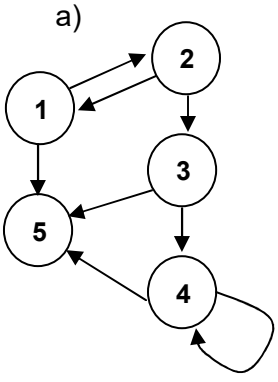
4. Apresente exemplos de problemas que podem ser resolvidos através de modelagem utilizando grafos (pesquisar).

5. Defina as estruturas de dados necessárias para a representação de um dígrafo utilizando Matriz de adjacência.

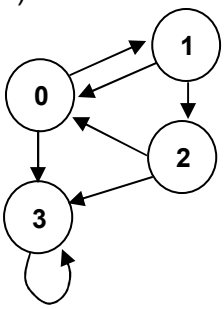
6. Utilizando as estruturas de dados definidas no exercício 5, desenvolva as seguintes rotinas:
   a) Inserção de uma aresta
   b) Remoção de um aresta
   c) Visualização do grafo
   d) Mostra lista de adjacentes do nó x.

7. Apresente as matrizes de adjacência correspondentes aos grafos a seguir:

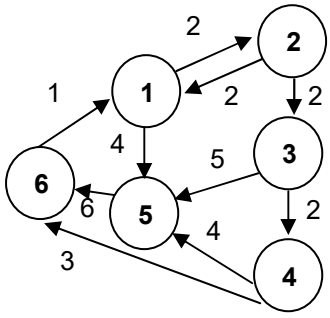
a)



b)



c)



8. Apresente os grafos representados pelas matrizes de adjacências.

a)

	V1	V2	V3	V4	V5
V1	0	1	0	1	0
V2	1	0	1	1	1
V3	0	1	0	0	1
V4	1	1	0	0	1
V5	0	1	1	1	0

b)

	V1	V2	V3	V4	V5
V1	0	0	0	1	0
V2	1	0	1	1	1
V3	0	0	1	0	1
V4	1	0	0	0	1
V5	0	0	1	1	0

c)

	V1	V2	V3	V4
V1	0	1	0	1
V2	1	1	0	1
V3	0	0	1	0
V4	1	0	1	0

d)

	V1	V2	V3	V4	V5	V6
V1	0	1	0	1	1	0
V2	1	0	0	1	0	0
V3	0	0	1	0	0	1
V4	1	0	0	0	0	0
V5	1	0	0	1	1	0
V6	1	0	1	0	0	0

9. Defina as estruturas de dados necessárias para a representação de um grafo utilizando alocação dinâmica.

10. Utilizando as estruturas de dados definidas no exercício 9, desenvolva as seguintes rotinas:
    a) Inserção de uma aresta
    b) Remoção de um aresta
    c) Visualização do grafo
    d) Apresenta lista de adjacentes do grafo (considerando grafo direcionado)

11. Considerando os grafos apresentados no exercício 7, apresente a representação dos mesmos utilizando as estruturas de dados dinâmicas definidas no exercício 9.

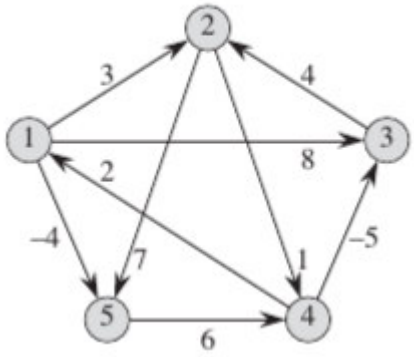
12. Considere as estruturas de dados definidas nos exercícios 5 e 9 para a representação de um grafo, desenvolva rotinas para, dados os vértices X e Y, verifique se Y é adjacente a X, e em caso afirmativo, apresente o peso da aresta.

13. Desenvolva rotinas para implementar os algoritmos de busca em profundidade e em largura.

14. Desenvolva uma rotina para implementar o algoritmo de ordenação topológica.

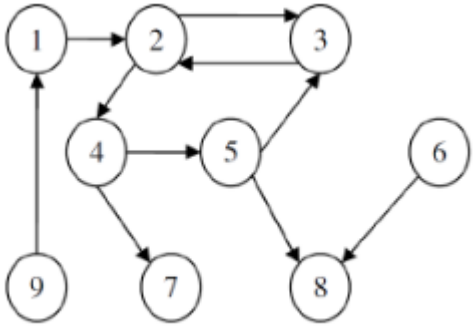
15. Explique o algoritmo de Dijkstra. Exemplifique.

16. Determine os graus de entrada e saída de cada vértice do grafo a seguir.

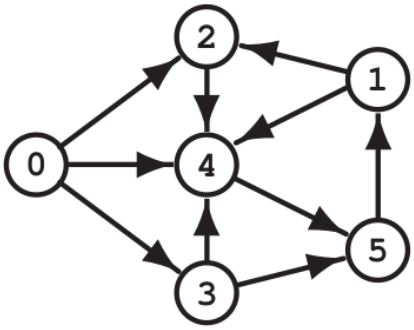


17. Apresente o percurso em profundidade e largura para os grafos a seguir.

vértice inicial 1



vértice inicial 0



18. Resolva os problemas do BeeCrowd:
    1298 - Corrija o Labirinto
    3221 - Faróis

Exemplos aplicações de grafos (notas de aula prof. J. Guimarães UFSCAR)

Make

O programa make do Unix toma como entrada um arquivo texto contendo comandos da forma

Nome : d_1 d_2 … d_n
    Comandos

Nome é o nome de um arquivo que depende dos arquivos d_1 d_2 … d_n.

Make lê este texto e executa Comandos se a data de última atualização de algum arquivo d_i for mais nova que de Nome. Ou seja, se algum d_i for mais novo que Nome, ele executa Comandos, que são comandos para o Unix que possivelmente deverão atualizar arquivo Nome. Ex.:

```
prog :       a.obj  b.obj   c.obj
     ln -o   prog   a.obj   b.obj  c.obj
a.obj : a.c  prog.h
     cc -c   a.c
b.obj : b.c  prog.h
     cc -c   b.c
c.obj : c.c
     cc -c  c .c
```

ln é o linker e cc o compilador. Se, por exemplo, `b.c` for modificado, make irá compilá-lo novamente (Comando "`cc -c b.c`"), porque ele será mais novo que "`b.obj`". Então "`b.obj`" será mais novo que `prog` que será linkado por "`ln -o prog a.obj b.obj c.obj`".

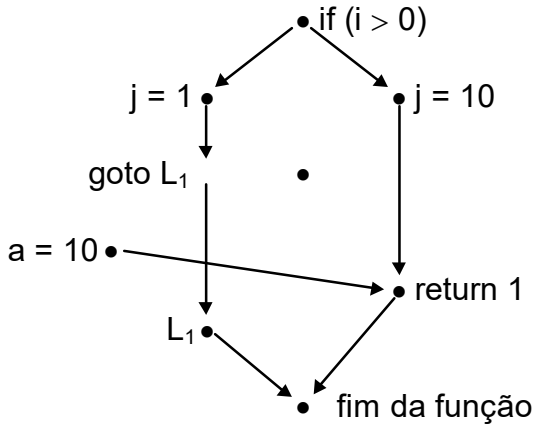
As relações de dependências do make pode ser colocada em forma de um grafo.

Eliminação de Código Morto

Podemos representar um procedimento como um grafo onde as instruções são vértices e existe aresta dirigida de v para w se w é executado após v (ou pode ser executado, no caso de if's e while's). A função

```
void f(
{
  if (i > 0)
  {
    j = 1;
    goto L1;
    a = 10;
  }
  else
    j = 10;
  return 1;
  L1;
}
```

seria transformada no grafo



Para descobrir o código morto, fazemos uma busca a partir da primeira instrução do procedimento marcando todos os vértices visitados. As instruções correspondentes aos vértices não visitados nunca serão executados e podem ser removidos pelo compilador.

Note que com este mesmo grafo pode-se descobrir se a instrução return será executada por todos os caminhos que ligam o vértice inicial ao vértice "fim da função". A resposta para o grafo acima é : não.