Faculdade de Tecnologia e Ciências Sociais Aplicadas – FATECS
Curso de Ciência da Computação

2ª Lista de Exercícios

Teoria de Grafos em Linguagem C

✓ Grafos Eulerianos e Hamiltonianos

↘
Prof. Cleber Pinheiro

NOME:

TURNO: **SEMESTRE:**

PROFESSOR: CLEBER PINHEIRO **DATA:**

DISCIPLINA: TEORIA DE GRAFOS

EXERCÍCIOS DE APRENDIZAGEM
2ª Lista – Linguagem C

QUESTÃO 01

Grafos eulerianos e semi-eulerianos. Crie um código em linguagem C que implemente um exemplo de grafo euleriano contendo quatro vértices, onde será mostrada como resultado via terminal de saída a seguinte mensagem: *"O grafo é euleriano!" ou "O grafo não é euleriano!"*.

Resposta: vide anexo

QUESTÃO 02

Matriz de adjacência em grafos. Crie um código em linguagem C que implemente uma representação da matriz de adjacência do grafo Q_3.

Resposta: vide anexo

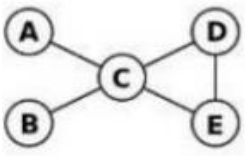
QUESTÃO 03

Matriz de Adjacência em Grafos. Dado um grafo não-direcionado com "**V**" vértices e "**A**" arestas, crie um código em linguagem C que implemente uma representação da matriz de adjacência do grafo dado. Neste caso, o usuário deverá fornecer as seguintes informações de entrada:

1. O tamanho e o número de vértices do grafo;
2. Para a formação da arestas, digite o vértice de partida e o vértice de chegada. Quando terminada a construção do grafo, digite "-1" para encerrar.

Como resultado final, deverá ser mostrada a matriz de incidência.

Exemplo ilustrativo de um grafo não-direcionado e sua representação final via terminal de saída:



	A	B	C	D	E
A	0	0	1	0	0
B	0	0	1	0	0
C	1	1	0	1	1
D	0	0	1	0	1
E	0	0	1	1	0

Resultado da execução

Informe o número de vértice do grafo: 5
OK. Número de vértice do grafo: 5

Matriz Inicial:

M[1][1]= 0	M[1][2]= 0	M[1][3]= 0	M[1][4]= 0	M[1][5]= 0
M[2][1]= 0	M[2][2]= 0	M[2][3]= 0	M[2][4]= 0	M[2][5]= 0
M[3][1]= 0	M[3][2]= 0	M[3][3]= 0	M[3][4]= 0	M[3][5]= 0
M[4][1]= 0	M[4][2]= 0	M[4][3]= 0	M[4][4]= 0	M[4][5]= 0
M[5][1]= 0	M[5][2]= 0	M[5][3]= 0	M[5][4]= 0	M[5][5]= 0

Informe as arestas: "**-1**" para encerrar

Vertice de partida: 1
Vertice de chegada: 3

Vertice de partida: 2
Vertice de chegada: 3

Vertice de partida: 3
Vertice de chegada: 1

Vertice de partida: 3
Vertice de chegada: 2

Vertice de partida: 3
Vertice de chegada: 4

Vertice de partida: 3
Vertice de chegada: 5

Vertice de partida: 4
Vertice de chegada: 3

Vertice de partida: 4
Vertice de chegada: 5

Vertice de partida: 5
Vertice de chegada: 3

Vertice de partida: 5
Vertice de chegada: 4

Vertice de partida: -1

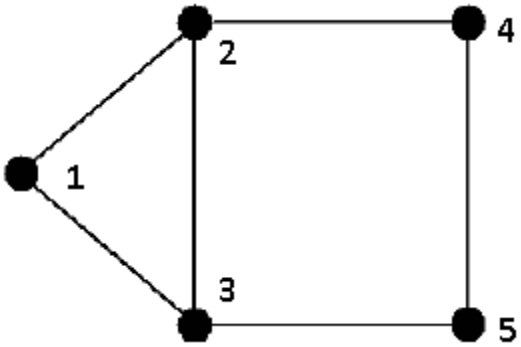
Matriz Final:

M[1][1]= 0	M[1][2]= 0	M[1][3]= 1	M[1][4]= 0	M[1][5]= 0
M[2][1]= 0	M[2][2]= 0	M[2][3]= 1	M[2][4]= 0	M[2][5]= 0
M[3][1]= 1	M[3][2]= 1	M[3][3]= 0	M[3][4]= 1	M[3][5]= 1
M[4][1]= 0	M[4][2]= 0	M[4][3]= 1	M[4][4]= 0	M[4][5]= 1
M[5][1]= 0	M[5][2]= 0	M[5][3]= 1	M[5][4]= 1	M[5][5]= 0

Resposta: vide anexo

QUESTÃO 04

Lista de adjacência em grafos. Considere uma malha aérea de uma região modelada por um grafo, tal que os vértices representem os aeroportos, os quais são conectados por vôos. A figura a seguir mostra um grafo que reproduz os vôos entre os aeroportos 1, 2, 3, 4 e 5. Crie um código em linguagem C que implemente uma representação da lista de adjacência deste grafo não-direcionado.



Resposta: vide anexo

Lista de Adjacência (Voos entre Cidades):
Cidade 1: -> 2 -> 3
Cidade 2: -> 1 -> 3 -> 4
Cidade 3: -> 1 -> 2 -> 5
Cidade 4: -> 2 -> 5
Cidade 5: -> 3 -> 4

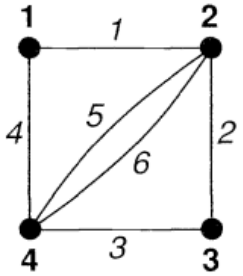
QUESTÃO 05

Lista de adjacência em grafos. Considerando a questão anterior, modifique o código para que o usuário digite as arestas que descrevem os voos (limitando ao máximo de 50 cidades). A lista de adjacência irá ser mostrada no final da execução via terminal de saída.

Resposta: vide anexo

QUESTÃO 06

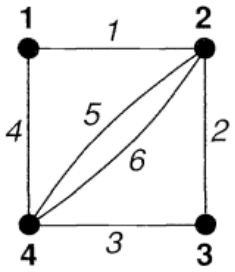
Matrizes de Adjacência e de Incidência. Crie um código em linguagem C que implemente uma representação das matrizes de adjacência do grafo não-direcionado abaixo. para o grafo a seguir:



Resposta: vide anexo

QUESTÃO 07

Matrizes de Incidência. Complete o código abaixo com **42** linhas em linguagem ***C*** (verifique as células incompletas/destacadas com sombreamento na tabela abaixo) que, após o usuário digitar os vértices e as arestas incidentes nos mesmos, o código retornará na saída, como resultado, a matriz de incidência para o grafo a seguir:



Código:

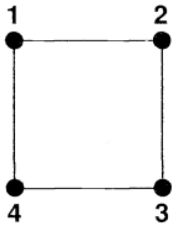
```
1.  #include <stdio.h>
2.  #define NUM_VERTICES 4
3.  #define NUM_ARESTAS 6
4.  int main() {
5.      int vertice, aresta, i, j, nmaxv=0, nmaxa=0;
6.      int grafo[NUM_VERTICES+1][NUM_ARESTAS+1];
7.  for (i=0; i<=NUM_VERTICES; i++) {
8.   for (j=0; j<=NUM_ARESTAS; j++) {
9.    grafo[
10.  }
11. }
12.     printf("\nDigite os vértices e as respectivas arestas associadas do grafo (digite -1 para encerrar):\n");
13.     while (vertice
14.         printf("\nDigite o vértice: ");
15.         scanf("%d", &vertice);
16.         if (
17.             break; // Encerra a entrada quando -1 for inserido
18.         }
19.         printf("\nDigite a aresta associada/incidente neste vértice: ");
```

```
20.         scanf("%d", &aresta);
21.         if (vertice>nmaxv) {
22.             nmaxv = vertice; // Determinação da quantidade máxima de vértices no grafo (ordem da matriz de adjacência)
23.         }
24.         if (aresta>nmaxa) {
25.             nmaxa = aresta; // Determinação da quantidade máxima de vértices no grafo (ordem da matriz de adjacência)
26.         }
27.         if (vertice <= 0 || vertice > NUM_VERTICES || aresta <= 0 || aresta > NUM_ARESTAS) {
28.             printf("Vértices E/OU arestas inválidos. Digite novamente");
29.             continue; // Pula esta iteração e continua pedindo entradas válidas
30.         }
31.         grafo[
32.     }
33.     // Imprime a matriz de incidência
34.     printf("Matriz de Incidência:\n");
35.     for (int i = 1; i <= nmaxv; i++) {
36.         for (int j = 1; j <= nmaxa; j++) {
37.             printf(
38.         }
39.         printf("\n");
40.     }
41.     return 0;
42. }
```

Resposta: vide anexo

QUESTÃO 08

Matriz de adjacência em grafos. Dado um Grafo não direcionado com "V" vértices e "**A**" arestas, crie um Complete o código abaixo com **33** linhas em linguagem ***C*** (verifique as células incompletas/destacadas com sombreamento na tabela abaixo) que, após o usuário digitar os vértices e as arestas incidentes nos mesmos, o código retornará na saída, como resultado, a lista de adjacência para o grafo a seguir:



Código:

```
1.  #include <stdio.h>
2.  #define NUMEROMAXVERTICES 50
3.  int main() {
4.    int nmax, i, j, x, y;
5.    int grafo[NUMEROMAXVERTICES][NUMEROMAXVERTICES] = {0}; // Inicializa a matriz com zeros
6.    printf("\nInforme as arestas/os arcos: (-1) para encerrar. \n");
7.    printf("Neste caso, a especificação da aresta já inclui a vizinhança entre os referidos vértices. Cuidado para que não haja duplicidade na contagem \nInforme as arestas/os arcos: (-1) para encerrar. \n");
8.    printf("\nVértice de origem: ");
9.    scanf("%d",&x);
10.   while (x != -1){
11.    printf("\nVértice de destino: ");
12.    scanf("%d",&y);
13.    grafo[
14.    grafo[
15.     if (x>y) { if
16.                nmax = x; // Determinação da quantidade máxima de vértices no grafo (ordem da matriz de adjacência)
17.          }
18.    nmax=y;}}
```

```
19.    printf("\nVértice de origem: ");
20.    scanf("%d",&x);
21.  }
22.  printf("Lista de Adjacência:\n");
23.  for (int verticeOrigem = 1; verticeOrigem <= nmax;
verticeOrigem++) {
24.   printf("Vértice %d:", verticeOrigem);
25.   for (int verticeDestino = 1; verticeDestino <= nmax;
verticeDestino++) {
26.     if (
27.         printf(" -> %d", verticeDestino);
28.     }
29.   }
30.   printf("\n");
31.  }
32.  return 0;
33. }
```

Resposta: vide anexo

QUESTÃO 09

Grafos hamiltonianos e semi-hamiltonianos. Crie um código em linguagem C que mostre a quantidade e os de ciclos hamiltonianos em grafos completos.

Resposta: vide anexo

QUESTÃO 10

Grafos hamiltonianos e semi-hamiltonianos. O Certificado de Depósito Interbancário (**CDI**) é um tipo de um título emitido pelos bancos para transações entre as instituições financeiras no mercado interbancário. O **CDI** é utilizado como referência para diversas operações financeiras, especialmente em investimentos de renda fixa, servindo como um indicador para a taxa de juros praticada no mercado.

Abaixo está uma série temporal do valor percentual mensal de rentabilidade deste título para o ano de 2.023:

	jan	*fev*	*mar*	*abr*	*mai*	*jun*	*jul*	*ago*	*set*	*out*	*nov*	*dez*
2023	*1,12%*	*0,92%*	*1,17%*	*0,92%*	*1,12%*	*1,07%*	*1,07%*	*1,14%*	*0,97%*	*1,00%*	*0,92%*	*0,89%*

Crie um código em linguagem C que construa um grafo direcionado relacionado a esta série temporal onde os vértices representam o valor do montante aplicado e as arestas sendo os percentuais mensais de rentabilidade do **CDI**. Neste caso, considerando o capital inicial aplicado igual a PV, o montante final será:

$$FV = PV(1+i_1)\times (1+i_1)\times (1+i_1)\times ... \times(1+i_n),$$

onde:
FV é o valor futuro/montante;
PV é o valor presente/capital (antes do acréscimos de rentabilidade);
i_k é a taxa de rentabilidade (acréscimo) na forma decimal.

Neste caso, serão mostrados como resultado via terminal de saída o valor do montante em cada mês até o o final do ano.

Resposta: vide anexo

QUESTÃO 11

Grafos hamiltonianos e semi-hamiltonianos. Em relação à questão anterior, modifique o código para que seja possível calcular o montante de qualquer série temporal relativo à rentabilidade do CDI. Neste caso, serão mostrados como resultado via terminal de saída o valor do montante em cada mês até o o final do ano. O usuário deverá fornecer as seguintes informações de entrada:

1. O tamanho da série temporal (no exemplo, temos 12 meses, correspondendo ao número de vértices do grafo;
2. O valor do capital aplicado inicialmente;
3. Os valores das arestas, sendo estes as taxas de rentabilidade.

Resposta: vide anexo

Anexo

Gabarito Sugerido para as Questões

Questão 01

```c
#include <stdio.h>
#include <stdbool.h>
#define numVertice 4 // Número de vértices no grafo

// Função booleana para verificar se um grafo é euleriano. Após a sua chamada,
retorna VERDADEIRO OU FALSO!
bool Euleriano(int grafo[numVertice][numVertice]) {
    int CONTgrauimpar = 0;
    for (int i = 0; i < numVertice; i++) {
        int grau = 0;
        for (int j = 0; j < numVertice; j++) {
            if (grafo[i][j] == 1)
                grau++;
        }
        if (grau % 2 != 0)
            CONTgrauimpar++;
    }
    return CONTgrauimpar == 0 || CONTgrauimpar == 2;
}

int main() {
    int grafo[numVertice][numVertice] = {
        {0, 1, 1, 1},
        {1, 0, 1, 0},
        {1, 1, 0, 1},
        {1, 0, 1, 0}
    };

    if (Euleriano(grafo))
        printf("O grafo é euleriano.\n");
    else
        printf("O grafo não é euleriano.\n");

    return 0;
}
```

Questão 02

```c
#include <stdio.h>

// Função para contar o número de bits definidos (1s) em um número inteiro
int contarBitsDefinidos(int n) {
    int contagem = 0;
    while (n) {
        contagem += n & 1; // Esta parte da linha utiliza a operação binária AND onde se verifica se o último bit de "n" é 1 ou 0. Caso afirmativo, soma-se 1 à contagem.
        n >>= 1; // executa operação de deslocamento para a direita (right shift) em // um valor inteiro n. Ela é equivalente a dividir n por 2 e atribuir o resultado de // volta a n.

    }
    return contagem;
}

int main() {
    // Defina o valor de k para o grafo Q3
    int k = 3;

    // Calcule o número total de vértices no grafo Q3 (2^k)
    int numVertices = 1<<k;

    // Crie uma matriz de adjacência com base no número de vértices
    int adjacencyMatrix[numVertices][numVertices];

    // Inicialize a matriz de adjacência com zeros
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            adjacencyMatrix[i][j] = 0;
        }
    }

    // Preencha a matriz de adjacência com base nas regras do grafo Q3
    for (int i = 0; i < numVertices; i++) {
        for (int j = i + 1; j < numVertices; j++) {
            // Dois vértices são adjacentes se suas representações binárias diferirem em apenas um bit
            int xorResult = i ^ j;
            if (contarBitsDefinidos(xorResult) == 1) {
                adjacencyMatrix[i][j] = 1;
                adjacencyMatrix[j][i] = 1; // O grafo é não direcionado
            }
        }
    }

    // Imprima a matriz de adjacência
    printf("Matriz de Adjacência do Grafo Q3 (3-cubo):\n");
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            printf("%d ", adjacencyMatrix[i][j]);
        }
        printf("\n");
    }
```

```
    return 0;
}
```

Questão 03

```
#include <stdio.h>
#define NMAX 101
int main() {
 int n, i, j, x, y;
int g[NMAX][NMAX]; // matriz estática 101 x 101
 printf("Informe o número de vértice do grafo: ");
 scanf("%d",&n);
 printf("OK. Número de vértice do grafo: %d \n",n);

for (i=1; i<=n; i++) {
 for (j=1; j<=n; j++) {
  g[i][j]=0;
 }
}
printf("\nMatriz Inicial: \n");
for (i=1; i<=n; i++) {
 for (j=1; j<=n; j++) {
  printf("M[%d][%d]= %d \t",i,j,g[i][j]);
 }
 printf("\n");
}
printf("\nInforme os arcos: (-1) para encerrar\n");
printf("\nVertice de partida: ");
scanf("%d",&x);
while (x != -1){
 printf("\nVertice de chegada: ");
 scanf("%d",&y);
 g[x][y] = 1;
 g[y][x] = 1;
 printf("\nVertice de partida: ");
 scanf("%d",&x);
}

printf("\nMatriz Final: \n");
for (i=1; i<=n; i++) {
 for (j=1; j<=n; j++) {
  printf("M[%d][%d]= %d \t",i,j,g[i][j]);
 }
 printf("\n");
}

return 0;
}
```

Questão 04

```
#include <stdio.h>
#define NUMEROCIDADES 5

int main() {
  int i, j;
  int voos[NUMEROCIDADES][NUMEROCIDADES] = {0}; // Inicializa a matriz com zeros

    // Adicione voos entre cidades (defina as arestas da matriz de adjacência)

    voos[1][2] = 1;
    voos[1][3] = 1;
    voos[2][1] = 1;
    voos[2][3] = 1;
    voos[2][4] = 1;
    voos[3][1] = 1;
    voos[3][2] = 1;
    voos[3][5] = 1;
    voos[4][2] = 1;
    voos[4][5] = 1;
    voos[5][3] = 1;
    voos[5][4] = 1;


 printf("Lista de Adjacência (Voos entre Cidades):\n");
 for (int cidadeOrigem = 1; cidadeOrigem <= NUMEROCIDADES; cidadeOrigem++) {
  printf("Cidade %d:", cidadeOrigem);

  for (int cidadeDestino = 1; cidadeDestino <= NUMEROCIDADES; cidadeDestino++) {
    if (voos[cidadeOrigem][cidadeDestino] == 1) {
        printf(" -> %d", cidadeDestino);
    }
  }
  printf("\n");
 }

 return 0;
}
```

Questão 05

```
#include <stdio.h>
#define NUMEROMAXCIDADES 50

int main() {
  int nmax, i, j, x, y;
  int voos[NUMEROMAXCIDADES][NUMEROMAXCIDADES] = {0}; // Inicializa a matriz com
zeros

    // Adicione voos entre cidades (defina as arestas da matriz de adjacência)


  printf("\nInforme as arestas/os arcos: (-1) para encerrar\n");
  printf("\nCidade de partida: ");
```

```
  scanf("%d",&x);
  while (x != -1){
   printf("\nCidade de chegada: ");
   scanf("%d",&y);
   voos[x][y] = 1;
   voos[y][x] = 1;
   if (x>y) {
              nmax = x; // Determinação da quantidade máxima de cidades no grafo
(ordem da matriz de adjacência)
        }
   nmax=y;
   printf("\nCidade de partida: ");
   scanf("%d",&x);
 }

 printf("Lista de Adjacência (Voos entre Cidades):\n");
 for (int cidadeOrigem = 1; cidadeOrigem <= nmax; cidadeOrigem++) {
  printf("Cidade %d:", cidadeOrigem);

  for (int cidadeDestino = 1; cidadeDestino <= nmax; cidadeDestino++) {
    if (voos[cidadeOrigem][cidadeDestino] == 1) {
        printf(" -> %d", cidadeDestino);
    }
  }
  printf("\n");
 }

 return 0;
}
```

Questão 06

```
#include <stdio.h>
#include <stdlib.h>
#define NMAX 101

int main() {
 int n, i, j, x, y;
 int g[NMAX][NMAX]; // matriz estática 101 x 101

 printf("Informe o tamanho o número de vértices: ");
 scanf("%d",&n);
 printf("OK. Número de vértices: %d \n",n);

for (i=1; i<=n; i++) {
 for (j=1; j<=n; j++) {
  g[i][j]=0;
 }
}

 while (x != -1 && y!=-1){
  printf("\nVertice de origem: ");
  scanf("%d",&x);
      if (x == -1) {
            goto matriz; // Desvia para o encerramento do código quando -1 for
inserido
```

```
  }
  while (x > NMAX || x < -1){
   // Verifica se o vértice é válido
            printf("Erro: vértice inválido. Tente novamente.\n");
            printf("\nVértice de origem: ");
            scanf("%d",&x);
            if (x == -1) {
            goto matriz; // Desvia para o encerramento do código quando -1 for
inserido
  }
  }

  printf("\nVertice de destino: ");
  scanf("%d",&y);
  if (y==-1) {
            goto matriz; // Desvia para o encerramento do código quando -1 for
inserido
  }
  while (y > NMAX || y < -1){
   // Verifica se o vértice é válido
            printf("Erro: vértice inválido. Tente novamente.\n");
            printf("\nVértice de destino: ");
            scanf("%d",&y);
  }

  g[x][y]++;
  g[y][x]++;
 }
 matriz:
    printf("\nMatriz Final: \n");
    for (i=1; i<=n; i++) {
     for (j=1; j<=n; j++) {
      printf("M[%d][%d]= %d \t",i,j,g[i][j]);
     }
     printf("\n");
    }
return 0;
}
```

Questão 07

```
#include <stdio.h>
#define NUM_VERTICES 4
#define NUM_ARESTAS 6

int main() {
    int vertice, aresta, i, j, nmaxv=0, nmaxa=0;
    int grafo[NUM_VERTICES+1][NUM_ARESTAS+1];

for (i=0; i<=NUM_VERTICES; i++) {
 for (j=0; j<=NUM_ARESTAS; j++) {
  grafo[i][j]=0;
 }
}
    printf("\nDigite os vértices e as respectivas arestas associadas do grafo (digite
-1 para encerrar):\n");
```

```
    while (vertice != -1) {
        printf("\nDigite o vértice: ");
        scanf("%d", &vertice);

        if (vertice == -1) {
            break; // Encerra a entrada quando -1 for inserido
        }

        printf("\nDigite a aresta associada/incidente neste vértice: ");
        scanf("%d", &aresta);

        if (vertice>nmaxv) {
             nmaxv = vertice; // Determinação da quantidade máxima de vértices no
grafo (ordem da matriz de adjacência)
        }

        if (aresta>nmaxa) {
             nmaxa = aresta; // Determinação da quantidade máxima de vértices no
grafo (ordem da matriz de adjacência)
        }

        if (vertice <= 0 || vertice > NUM_VERTICES || aresta <= 0 || aresta >
NUM_ARESTAS) {
            printf("Vértices E/OU arestas inválidos. Digite novamente");
            continue; // Pula esta iteração e continua pedindo entradas válidas
        }

        grafo[vertice][aresta] = 1;

    }

    // Imprime a matriz de incidência
    printf("Matriz de Incidência:\n");
    for (int i = 1; i <= nmaxv; i++) {
        for (int j = 1; j <= nmaxa; j++) {
            printf("%d ", grafo[i][j]);
        }
        printf("\n");

    }

    return 0;
}
```

Questão 08

```
#include <stdio.h>
#define NUMEROMAXVERTICES 50

int main() {
  int nmax, i, j, x, y;
  int grafo[NUMEROMAXVERTICES][NUMEROMAXVERTICES] = {0}; // Inicializa a matriz com
zeros
```

```
    // Defina os vértices e as arestas da matriz de adjacência)


  printf("\nInforme as arestas/os arcos: (-1) para encerrar. \n");
  printf("Neste caso, a especificação da aresta já inclui a vizinhança entre os
referidos vértices. Cuidado para que não haja duplicidade na contagem \nInforme as
arestas/os arcos: (-1) para encerrar. \n");

  printf("\nVértice de origem: ");
  scanf("%d",&x);
  while (x != -1){
   printf("\nVértice de destino: ");
   scanf("%d",&y);
   grafo[x][y] = 1;
   grafo[y][x] = 1;
   if (x>y) { if (x>nmax){
              nmax = x; // Determinação da quantidade máxima de vértices no grafo
(ordem da matriz de adjacência)
              }} else {if (y>nmax){
   nmax=y;}}
   printf("\nVértice de origem: ");
   scanf("%d",&x);
 }

 printf("Lista de Adjacência:\n");
 for (int verticeOrigem = 1; verticeOrigem <= nmax; verticeOrigem++) {
  printf("Vértice %d:", verticeOrigem);

  for (int verticeDestino = 1; verticeDestino <= nmax; verticeDestino++) {
    if (grafo[verticeOrigem][verticeDestino] !=0) {
        printf(" -> %d", verticeDestino);
    }
  }
  printf("\n");
 }
 return 0;
}
```

Questão 09

```
#include <stdio.h>
#include <stdbool.h>

// Função para trocar os valores de dois inteiros
void trocar(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Função para imprimir uma permutação de vértices como um ciclo hamiltoniano
void imprimirCiclo(int ciclo[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", ciclo[i]);
    }
    printf("%d\n", ciclo[0]); // Voltar ao ponto de partida
```

```
}

// Função para gerar todas as permutações dos vértices
void gerarPermutacoes(int ciclo[], int inicio, int n, int *contador) {
    if (inicio == n - 1) {
        imprimirCiclo(ciclo, n);
        (*contador)++;
        return;
    }

    for (int i = inicio; i < n; i++) {
        trocar(&ciclo[inicio], &ciclo[i]);
        gerarPermutacoes(ciclo, inicio + 1, n, contador);
        trocar(&ciclo[inicio], &ciclo[i]);
    }
}

// Função principal para contar o número de ciclos hamiltonianos em um grafo completo
void contarCiclosHamiltonianos(int numVertices) {
    int ciclo[numVertices];
    int contador = 0;

    // Inicializa o ciclo com os vértices de 1 a numVertices
    for (int i = 0; i < numVertices; i++) {
        ciclo[i] = i + 1;
    }

    // Gera todas as permutações dos vértices e imprime os ciclos hamiltonianos
    gerarPermutacoes(ciclo, 0, numVertices, &contador);

    // Imprime o número total de ciclos hamiltonianos
    printf("Total de ciclos hamiltonianos: %d\n", contador);
}

int main() {
    int numVertices;

    // Solicita ao usuário o número de vértices do grafo
    printf("Digite o numero de vertices do grafo: ");
    scanf("%d", &numVertices);

    // Chama a função para contar o número de ciclos hamiltonianos
    contarCiclosHamiltonianos(numVertices);

    return 0;
}
```

Questão 10

```c
#include <stdio.h>
#define NUM_VERTICES_MAX 20 // número MÁXIMO de vértices para a construção da matriz
int main() {
    // /Vetor contendo as taxas mensais de rentabilidade do CDI
    double taxas[12] = {0.0112, 0.0092, 0.0117, 0.0092, 0.0112, 0.0107, 0.0107,
0.0114, 0.0097, 0.0100, 0.0092, 0.0089};
    // Array para armazenar o grafo (conexões entre os meses com as taxas de
rentabilidade)
    double grafo[13][13]; // Grafo com 13 vértices e 13 arestas (permeses)
    double vertices[NUM_VERTICES_MAX] ={0};
    double capital_inicial = 1000.00;
    // Inicializando o grafo com 0
    for (int i = 0; i < 13; i++) {
      vertices[i] = 0; // Sem vértices (montante/valor futuro)
        for (int j = 0; j < 13; j++) {
            grafo[i][j] = 0;  // Sem arestas ainda
        }
    }
    vertices[0] = capital_inicial;
    // Preenchendo as arestas e vértices com as taxas de rentabilidade e montantes
    for (int i = 0; i < 12; i++) {
        // Aresta entre os meses consecutivos com a taxa mensal correspondente
        grafo[i][i + 1] = taxas[i];
        vertices[i + 1] = vertices[i] * (1 + taxas[i]);
    }
        // Exibindo os montantes
    printf("\nValor do Montante (por mês):\n");
    for (int i = 0; i < 12; i++) {
        printf("Montante do mês %d: %.4f\n", i + 1, vertices[i+1]);
    }
    return 0;
}
```

Questão 11

```c
#include <stdio.h>
#define NUM_VERTICES_MAX 100 // número MÁXIMO de vértices para a construção da matriz

int main() {
    // /Vetor contendo as taxas mensais de rentabilidade do CDI
    double taxas[NUM_VERTICES_MAX]={0};
    double grafo[NUM_VERTICES_MAX][NUM_VERTICES_MAX]={0}; // Grafo com 13 vértices e
13 arestas (permeses)
    double vertices[NUM_VERTICES_MAX] ={0};
    double capital_inicial;
    double taxa;
    int meses;

    // Array para armazenar o grafo (conexões entre os meses com as taxas de
rentabilidade)
    printf("Digite o número de meses (vértices) para a série temporal de
rentabilidade:");
    scanf("%d", &meses);
```

```
    printf("Digite o capital inicial:");
    scanf("%lf", &capital_inicial);

    // Inicializando o grafo com 0
    for (int i = 0; i < meses; i++) {
      vertices[i] = 0; // Sem vértices (montante/valor futuro)
        for (int j = 0; j < meses; j++) {
            grafo[i][j] = 0;  // Sem arestas ainda

        }
    }
   // Preenchendo o vetor de taxas
    for (int i = 0; i < meses; i++) {
      printf("Digite o valor da taxa %d:", i+1);
      scanf("%lf", &taxa);
      taxas[i] = taxa; // Valor da taxa de rentabilidade
    }

    vertices[0] = capital_inicial;
    // Preenchendo as arestas e vértices com as taxas de rentabilidade e montantes
    for (int i = 0; i < meses; i++) {
        // Aresta entre os meses consecutivos com a taxa mensal correspondente
        grafo[i][i + 1] = taxas[i];
        vertices[i + 1] = vertices[i] * (1 + taxas[i]);
    }

        // Exibindo os montantes
    printf("\nValor do Montante (por mês):\n");
    for (int i = 0; i < meses; i++) {
        printf("Montante do mês %d: %.4f\n", i + 1, vertices[i+1]);
    }

    return 0;
}
```