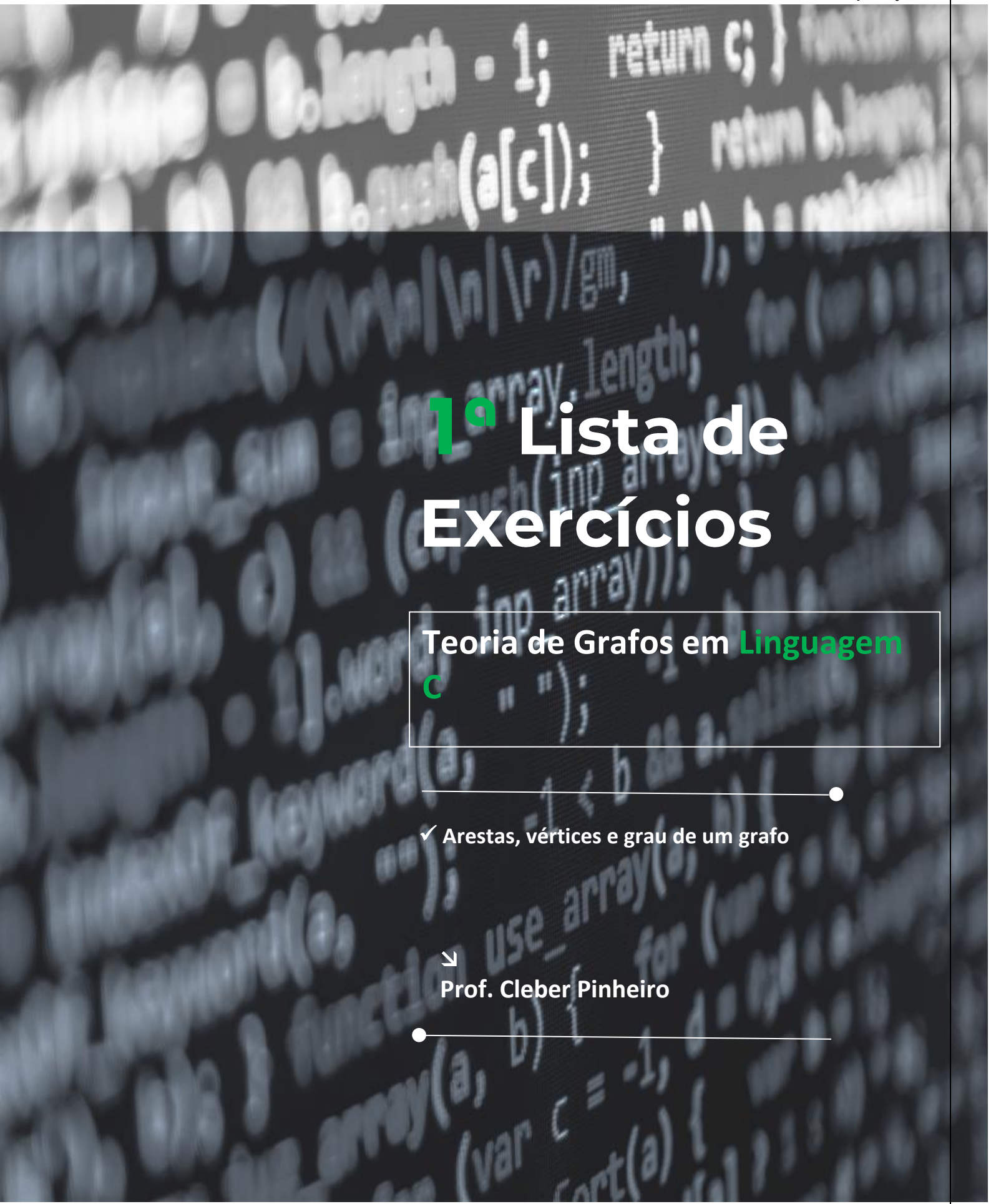
Faculdade de Tecnologia e Ciências Sociais Aplicadas – FATECS
Curso de Ciência da Computação



1ª Lista de Exercícios

Teoria de Grafos em Linguagem C

✓ Arestas, vértices e grau de um grafo

↘
Prof. Cleber Pinheiro

NOME:

TURNO: **SEMESTRE:**

PROFESSOR: CLEBER PINHEIRO **DATA:**

DISCIPLINA: TEORIA DE GRAFOS

EXERCÍCIOS DE APRENDIZAGEM
1ª Lista – Linguagem C

QUESTÃO 01

Arestas, vértices e grau de um grafo. Crie um código em linguagem ***C*** que implemente um grafo simples contendo 3 vértices e 3 arestas, onde será mostrado como resultado via saída o grau de cada vértice, a ordem do grafo, o grau médio, bem como o tamanho do grafo.

Resposta: vide anexo

QUESTÃO 02

Arestas, vértices e grau de um grafo. Escreva um código em linguagem ***C*** que cria um grafo tal que é solicitado ao usuário via entrada o número de vértices e arestas para a construção do grafo. Como resultado, devem ser mostrados via terminal o grau de cada vértice, a ordem do grafo e o tamanho do grafo.

Resposta: vide anexo

QUESTÃO 03

Arestas, vértices e grau de um grafo. Crie um código em linguagem ***C*** um grafo com 3 vértices e 5 arestas, mostrando via saída/terminal a multiplicidade de cada vértice.

Resposta: vide anexo

QUESTÃO 04

Aplicação simples de grafos. Com o intuito de realizar obras com planejamento de infraestrutura rodoviária, o governo de um páis pretende construir ***n*** estradas (todas de mão dupla), sendo que cada estrada liga exatamente 2 das cidades. Há um total de "**N**" cidades. Considere condição que uma das cidades só pode ter uma estrada construída interligando a qualquer outra cidade. Crie um código em linguagem C onde é solicitado o total de cidades "**N**", onde é determinado o menor valor de ***n*** para que, independente de como as estradas sejam construídas, seja possível viajar entre quaisquer 2 cidades (neste caso, as estradas são construídas tal que 20 das mesmas se conectam entre si através uma únicada estrada. Dessa forma, uma viagem pode ser feita passando, possivelmente, por cidades intermediárias, se o condutor/viajante deseja realizar tal opção).

Resposta: vide anexo

QUESTÃO 05

Arestas, vértices e grau de um grafo. Crie um código em linguagem ***C*** que verifica se um grafo de entrada é to tipo estrela, mostrando via saída/terminal uma das seguintes mensagens:

1) O grafo é do tipo estrela;
2) O grafo NÃO é do tipo estrela.

Resposta: vide anexo

QUESTÃO 06

Aplicação simples de grafos. 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 deste título para o ano de 2.023 (considere como data para fechamento de rentabilidade o final do mês):

	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%*

Considere um caso em que um capital é investido em tal título no dia 31 de dezembro de 2.022. A rentabilidade é mensal. Crie um código em linguagem ***C*** que armazene as taxas mensais de rentabilidade durante o ano referente ao título CDI. Posteriormente, crie um grafo direcionado de 13 vértices e 13 arestas, onde cada vértice representa o mês e as arestas, as taxas de rentabilidade (mensais e acumulada no período). O período compreende de dezembro de um ano até dezembro do próximo ano.

Resposta: vide anexo

QUESTÃO 07

Aplicação simples de grafos. Dado o preço de um bem ou serviço, o mesmo pode sofrer alterações durante o tempo. Tais modificações podem ser feitas através de acréscimos sucessivos e/ou simultâneos.

O preço de um bem/seviço com acréscimo sucessivos é dado pela expressão analítica:

$$P = P_0(1+i_1)\times (1+i_2)\times (1+i_3)\times \ldots \times(1+i_n),$$

onde:
P é o preço final;
P_0 é o preço inicial (antes do acréscimo);
i_k é a k-ésima taxa de acréscimo na forma decimal.

Considere que o preço inicial de um bem seja igual a R$ 20,00 e que o mesmo sofra três acréscimos sucessivos relativos às taxas de 1%, 1% e 2%, respectivamente. Em seguida, devido ao cenário momentâneo de deflação, a quarta taxa aplicada é negativa e igual a -3,89254%, aproximadamente. Crie um código em linguagem ***C*** que implemente um grafo cíclico direcionado onde os vértices representam os preços do bem/serviço e, as arestas, as taxas de acréscimos (a construção deve começar do vértice que contenha o preço inicial dado).

Resposta: vide anexo

Anexo

Gabarito sugerido paras as questões
Questão 01

```c
#include <stdio.h>

#define NUM_VERTICES 3
#define NUM_ARESTAS 3

// Função para descrever o grafo
void descreverGrafo(int grafo[NUM_VERTICES][NUM_VERTICES]) {
    int ordem = NUM_VERTICES;
    int tamanho = NUM_ARESTAS;

    printf("Ordem do grafo: %d vértices\n", ordem);
    printf("Tamanho do grafo: %d arestas\n", tamanho);
    printf("Grau de cada vértice:\n");

    int somagrau=0;
    for (int i = 0; i < NUM_VERTICES; i++) {
        int grau = 0;
        for (int j = 0; j < NUM_VERTICES; j++) {
            if (grafo[i][j] == 1 || grafo[j][i] == 1 ) {
                grau++;

            }
        }
        somagrau=somagrau+grau; // Cálculo do somatório dos graus de todos os
vértices conectados
        printf("Vértice %d: %d\n", i, grau);
    }

    float graumedio=0;
    int tamGrafo = 0;
    int tam=0;
    graumedio=somagrau/(float)NUM_VERTICES; // Aqui, deve-se transformar o tipo de
variável antes da divisão

    for (int i = 0; i < NUM_VERTICES; i++) {
        for (int j = 0; j < NUM_VERTICES; j++) {
            if (grafo[i][j] == 1) {
                tam++;
            }
        }
        if (tam > tamGrafo) {
            tamGrafo = tam;
        }
    }
    printf("Tamanho do grafo: %d\n", tamGrafo);
    printf("Grau médio do grafo: %f\n", graumedio);
}

int main() {
    int grafo[NUM_VERTICES][NUM_VERTICES] = {0};

    // Adicionar as arestas ao grafo
```

```
    grafo[0][1] = 1;
    grafo[1][2] = 1;
    grafo[2][0] = 1;


    descreverGrafo(grafo);

    return 0;
}
```

Questão 02

```
#include <stdio.h>
void descreverGrafo(int grafo[][100], int ordem, int tamanho) {
    printf("Ordem do grafo: %d vértices\n", ordem);
    printf("Tamanho do grafo: %d arestas\n", tamanho);
    printf("Grau de cada vértice:\n");
    for (int i = 1; i <= ordem; i++) {
        int grau=0;
        for (int j = 1; j <= ordem; j++) {
            if (grafo[i][j] == 1 || grafo[j][i] == 1 ) {
                grau++;
            }
        }
        printf("Vértice %d: %d\n", i, grau);
    }
    int tamGrafo = 0;
    int tam = 0;
    for (int i = 1; i <= ordem; i++) {
        for (int j = 1; j <= ordem; j++) {
            if (grafo[i][j] == 1) {
                tam++;
            }
        }
        if (tam > tamGrafo) {
            tamGrafo = tam;
        }
    }
    printf("Tamanho do grafo: %d\n", tamGrafo);
}
int main() {
    int ordem, tamanho;
    printf("Digite o número de vértices: ");
    scanf("%d", &ordem);
    printf("Digite o número de arestas: ");
    scanf("%d", &tamanho);
    int grafo[100][100] = {0};
    printf("Digite as arestas (pares de vértices):\n");
    for (int i = 1; i <= tamanho; i++) {
        int u, v;
        printf("Digite o vértice de origem/partida: ");
        scanf("%d", &u);
        printf("Digite o vértice de destino/chegada: ");
        scanf("%d", &v);
        grafo[u][v] = 1;
    }
```

```
    descreverGrafo(grafo, ordem, tamanho);
    return 0;
}
```

Questão 03

```
#include <stdio.h>

#define NUM_VERTICES 3
#define NUM_ARESTAS 5

int main() {
    int grafo[NUM_VERTICES][NUM_VERTICES] = {0};
    int multiplicidade[NUM_VERTICES] = {0};

    // Adicionar as arestas ao grafo
    grafo[0][1] = 1;
    grafo[1][0] = 1;
    grafo[1][2] = 1;
    grafo[2][0] = 1;
    grafo[0][2] = 1;


    // Calcular a multiplicidade de cada vértice
    for (int i = 0; i < NUM_VERTICES; i++) {
        for (int j = 0; j < NUM_VERTICES; j++) {
            if (grafo[i][j] == 1) {
                multiplicidade[i]++;
            }
        }
    }

    // Mostrar a multiplicidade de cada vértice
    printf("Multiplicidade de cada vértice:\n");
    for (int i = 0; i < NUM_VERTICES; i++) {
        printf("Vértice %d: %d\n", i, multiplicidade[i]);
    }

    return 0;
}
```

Questão 04

```
#include <stdio.h>

int num;
double fatorial(int n){
    double fat;
    if ( n <= 1 )
        return (1);
    else{
       return n * fatorial(n - 1);
    }
}
```

```
int main() {
    int numeroCidades;
    int totaldeEstradas;

    printf("Digite o número de cidades: ");
    scanf("%d", &numeroCidades);

    if (numeroCidades < 1) {
        printf("Número de cidades inválido.\n");
        return 1;
    }

    // Se tivermos apenas uma cidade, não é necessário construir estradas.
    if (numeroCidades == 1) {
        totaldeEstradas = 0;
    } else {

        totaldeEstradas=fatorial(numeroCidades-1)/(2*fatorial(numeroCidades-3));

        }

    totaldeEstradas+=1;
    printf("O número total de estradas a serem construídas é: %d\n",
totaldeEstradas);
    return 0;
}
```

Outra forma equivalente de implementação consiste no uso de laços/loop's. Veja o código abaixo:

```
#include <stdio.h>

int main() {
    int n; // Número de cidades
    printf("Digite o número de cidades: ");
    scanf("%d", &n);


   double fatorial1 = 1.0;
   double fatorial2 = 1.0;
   double numestradas=0;
   int numestradasint=0;

    for (int i = 1; i <= (n-1); i++) {
        fatorial1 *= i; // Calcula (n-1)!
    }

    for (int i = 1; i <= (n-1-2); i++) {
        fatorial2 *= i; // Calcula (n-1-2)!
    }

    numestradas= fatorial1/(2*fatorial2);
    numestradasint=(int)numestradas+1; //uso da técnica "casting" na conversão para
valor inteiro

    printf("O número de estradas é: %d\n", numestradasint);
```

```
    return 0;
}
```

Questão 05

```
#include <stdio.h>
#define MAX_VERTICES 100  // Número máximo de vértices suportado

int main() {
    int numVertices, numArestas;  // Variáveis para armazenar o número de vértices e
arestas
    int i, j, v1, v2;            // Variáveis auxiliares para iterações e leitura de
vértices

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

    // Declara a matriz de adjacência e a inicializa com zeros
    int grafo[MAX_VERTICES][MAX_VERTICES];
    for (i = 0; i < numVertices; i++) {         // Para cada linha (vértice)
        for (j = 0; j < numVertices; j++) {     // Para cada coluna (vértice)
            grafo[i][j] = 0;                 // Inicializa a posição com 0 (nenhuma
aresta)
        }
    }

    // Leitura dos pares de vértices para definir as arestas
    printf("Digite os pares de vértices conectados (digite -1 para encerrar):\n");
    numArestas = 0;  // Inicializa o contador de arestas
    while (1) {
        // Lê o vértice de partida
        printf("Vértice de partida: ");
        scanf("%d", &v1);
        if (v1 == -1) break;  // Se for -1, encerra a entrada

        // Lê o vértice de chegada
        printf("Vértice de chegada: ");
        scanf("%d", &v2);
        if (v2 == -1) break;  // Se for -1, encerra a entrada

        // Verifica se os vértices informados são válidos (entre 1 e numVertices)
        if (v1 < 1 || v1 > numVertices || v2 < 1 || v2 > numVertices) {
            printf("Vértices inválidos! Digite valores entre 1 e %d.\n",
numVertices);
            continue;  // Se inválido, volta ao início do loop para nova entrada
        }

        // Atualiza a matriz de adjacência para um grafo não-direcionado:
        // Subtrai 1 dos valores para converter de 1..n para índices 0..n-1.
        grafo[v1 - 1][v2 - 1] = 1;
        grafo[v2 - 1][v1 - 1] = 1;

        numArestas++;  // Incrementa o número de arestas
    }
```

```
    // Calcula o grau de cada vértice (soma dos valores na linha correspondente)
    int grau[MAX_VERTICES];
    for (i = 0; i < numVertices; i++) {
        grau[i] = 0;  // Inicializa o grau do vértice i com 0
        for (j = 0; j < numVertices; j++) {
            grau[i] += grafo[i][j];  // Soma o valor da célula (0 ou 1) para obter o
grau
        }
    }

    // Verifica quantos vértices possuem grau igual a (numVertices-1) e quantos têm
grau igual a 1
    int numCentrais = 0;       // Contador para vértices centrais (grau =
numVertices-1)
    int verticeCentral = -1;   // Armazena o índice do vértice central (se houver)
    int numExternos = 0;       // Contador para os demais vértices (externos) que
devem ter grau 1
    for (i = 0; i < numVertices; i++) {
        if (grau[i] == numVertices - 1) { // Se o grau é igual a numVertices-1, é
potencial vértice central
            numCentrais++;
            verticeCentral = i;  // Guarda o índice do vértice central
        } else if (grau[i] == 1) {  // Se o grau é 1, é um vértice externo
            numExternos++;
        }
    }

    // Em um grafo estrela, deve haver exatamente 1 vértice central e (numVertices -
1) vértices externos
    if (numCentrais == 1 && numExternos == numVertices - 1)
        printf("O grafo é uma ESTRELA. Vértice central: %d\n", verticeCentral + 1);
    else
        printf("O grafo NÃO é uma ESTRELA.\n");

    return 0;
}
```

Questão 06

```
#include <stdio.h>

// Função para calcular a rentabilidade acumulada ao longo do período
double taxaacumulada(double taxas[], int mes) {
    double acumulada = 1.0;
    for (int i = 0; i <= mes; i++) {
        acumulada *= (1 + taxas[i]);
    }
    return acumulada - 1;
}

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 (meses)

    // Inicializando o grafo com 0
    for (int i = 0; i < 13; i++) {
        for (int j = 0; j < 13; j++) {
            grafo[i][j] = 0;  // Sem arestas ainda
        }
    }

    // Preenchendo as arestas com as taxas de rentabilidade
    for (int i = 0; i < 12; i++) {
        // Aresta entre os meses consecutivos com a taxa mensal correspondente
        grafo[i][i + 1] = taxas[i];
    }

    // Calculando a rentabilidade acumulada e conectando o mês de dezembro anterior
até o mês de dezembro seguinte
    for (int i = 0; i < 12; i++) {
        grafo[0][i + 1] = taxaacumulada(taxas, i); // Arestas para as taxas
acumuladas
    }


    // Exibindo as arestas do grafo (taxas de rentabilidade)
    printf("\nArestas (taxas de rentabilidade entre os meses):\n");
    for (int i = 0; i < 12; i++) {
        printf("Taxa de %d para %d: %.4f%%\n", i + 1, i + 2, taxas[i] * 100);
    }

    // Exibindo as arestas acumuladas
    printf("\nRentabilidade acumulada (de dezembro a cada mês):\n");
    for (int i = 0; i < 12; i++) {
        printf("Acumulada de dezembro para %d: %.4f%%\n", i + 2, grafo[0][i + 1] *
100);
    }

    return 0;
}
```

Questão 07

```
#include <stdio.h>
#define NUM_VERTICES_MAX 10 // número MÁXIMO de vértices para a construção da matriz

int main() {
    int n = 4; // número de acréscimos sucessivos
```

```
    double preco_inicial = 20.0;
    double taxas[] = {0.01, 0.01, 0.02, -0.0389254}; // taxas de acréscimos
sucessivos
    double precos[NUM_VERTICES_MAX] ={0}; // matriz para armazenar os preços

    // Inicializa o preço inicial no primeiro vértice
    precos[1] = preco_inicial;

    // Calcula os preços após os acréscimos sucessivos
    for (int i = 1; i <= n; i++) {
        precos[i + 1] = precos[i] * (1 + taxas[i - 1]);
    }

    // Imprime o grafo cíclico
    printf("Vértices (Preços):\n");
    for (int i = 1; i <= n + 1; i++) {
        printf("Vértice %d: Preço = %.2f\n", i, precos[i]);
    }

    printf("\nArestas (Taxas):\n");
    for (int i = 1; i <= n; i++) {
        printf("Aresta do Vértice %d para Vértice %d: Taxa = %.5f%%\n", i, i + 1,
taxas[i - 1] * 100);
    }

    return 0;
}
```