

Aula Prática 8

Prazo de entrega: conferir no Moodle

Exercício 1

Neste exercício você deve fazer um programa para encontrar sequências de números iguais consecutivos, tanto na horizontal quanto na vertical, em uma matriz $m \times n$. Depois, você deve substituir esses números por zeros e colocá-los nas primeiras linhas da matriz. Todo o programa será implementado a partir das questões a seguir.

1.1) Faça um programa para preencher uma matriz $m \times n$ com números aleatórios entre **1** e **k**. Os valores de **m**, **n** e **k** devem ser lidos do teclado. Como ainda não aprendemos alocação dinâmica de memória, crie uma matriz estaticamente com os limites superiores de **m** e **n**. Considere que **m** e **n** não podem ser maiores que **100**. Não permita que o usuário entre com valores inválidos para **m**, **n** e **k**.

Exemplo de uma matriz para $m=5$, $n=4$ e $k=3$:

3	3	3	2
3	2	2	3
1	1	1	1
2	1	2	1
2	3	3	1

1.2) Procure por sequências com pelo menos três números consecutivos iguais tanto nas linhas quanto nas colunas da matriz. Substitua todos os números que estão nessas sequências por **0**.

Depois de executar este procedimento na matriz exemplo do item anterior, ela deverá ficar assim:

0	0	0	2
3	2	2	3
0	0	0	0
2	1	2	0
2	3	3	0

1.3) Imprima na tela o número de zeros que a matriz possui depois do passo **1.2**.

Para a matriz do exemplo anterior, o seu código deve imprimir: 9

1.4) Altere a matriz colocando todos os zeros nas primeiras linhas das suas respectivas colunas. Preserve a ordem dos outros números dentro da coluna. Imprima a matriz final.

Para a matriz do exemplo anterior, o seu código deve imprimir a seguinte matriz:

0	0	0	0
0	0	0	0
3	2	2	0
2	1	2	2
2	3	3	3

1.5) Repita os procedimentos descritos nos itens 1.2, 1.3 e 1.4 até que a matriz final não tenha sequências de tamanho maior ou igual a 3 de números consecutivos maiores que zero.

Para a matriz do item anterior, o seu programa deve realizar as seguintes operações:

- a) **Encontrar sequências de tamanho maior ou igual a três de números maiores que zero e substituir os números por zeros:**

0	0	0	0
0	0	0	0
3	2	2	0
2	1	2	2
2	0	0	0

- b) **Colocar os zeros no topo da matriz:**

0	0	0	0
0	0	0	0
3	0	0	0
2	2	2	0
2	1	2	2

- c) **Encontrar sequências de tamanho maior ou igual a três de números maiores que zero e substituir os números por zeros:**

0	0	0	0
0	0	0	0
3	0	0	0
0	0	0	0
2	1	2	2

d) Colocar os zeros no topo da matriz:

0	0	0	0
0	0	0	0
0	0	0	0
3	0	0	0
2	1	2	2

e) Não há mais sequências de tamanho maior que 3 de números maiores que zero. Imprima a matriz final e termine o programa.

Exercício 2

Forma de Entrega: Nesta prática você vai implementar um simulador de redes sociais na linguagem C. Este simulador deve ser compilado em um módulo de nome `redesocial`, que consiste de dois arquivos:

`redesocial.c`, que contém o código das funções e variáveis globais usados pelo simulador, ou seja, não tem o procedimento `main`;

`redesocial.h`, que contém o cabeçalho das funções e as definições (este arquivo é disponibilizado pelo professor -- ver final deste documento);

Somente esses dois arquivos devem ser submetidos! Esse procedimento permite que a correção do exercício seja feita de forma automática. O professor desenvolveu um programa que usa e testa todas as funções do módulo `redesocial`. Assim, se o módulo contiver funções com nomes diferentes daqueles propostos nos exercícios ou o módulo não for entregue, não será possível avaliar o exercício. **Importante:** no final deste documento há a implementação do arquivo `redesocial.h`, do arquivo `pratica8.c`, que contém o procedimento `main`, além de um protótipo do arquivo `redesocial.c`.

Problema

Uma rede social de amizades pode ser representada por um grafo $G(V, E)$ em que V é o conjunto de nós e E o conjunto de arestas do mesmo. Cada um dos nós n_0, n_1, n_2, \dots representa uma pessoa e, caso duas pessoas n_i e n_j sejam amigas, existe uma aresta $(i, j) \in E$. Uma das maneiras usuais para se representar um grafo é através de uma matriz de adjacência $n \times n$ de n colunas e n linhas. Cada linha (ou coluna) i contém as relações da pessoa n_i . Considere a matriz de adjacência abaixo: $G(V, E)$ em que V é o conjunto de nós e E o conjunto de arestas do mesmo. Cada um dos nós n_0, n_1, n_2, \dots representa uma pessoa e, caso duas pessoas n_i e n_j sejam amigas, existe uma aresta $(i, j) \in E$. Uma das maneiras usuais para se representar um grafo é através de uma matriz de adjacência $n \times n$ de n colunas e n linhas. Cada linha (ou coluna) i contém as relações da pessoa n_i . Considere a matriz de adjacência abaixo:

id	n_0	n_1	n_2	n_3	n_4
n_0	0	1	1	0	1
n_1	1	0	0	1	0
n_2	1	0	0	0	0
n_3	0	1	0	0	1
n_4	1	0	0	1	0

Esta matriz representa uma rede social entre 5 pessoas: n_0, n_1, n_2, n_3 e n_4 . Além disso, quando a posição (i, j) da matriz é 1, então as pessoas n_i e n_j são amigas entre si. Caso a posição (i, j) da matriz é 0, então n_i e n_j não são amigas. Observe que a pessoa n_0 é amiga das pessoas n_1, n_2 e n_4 , mas não é amiga da pessoa n_3 . **Importante:** a relação de amizade é simétrica: se n_i é amigo de n_j , então n_j é, necessariamente, amigo de n_i . Além disso, em redes sociais de amizade, não existe aresta entre a mesma pessoa, ou seja, não existem arestas do tipo (i, i) . Além disso, quando a posição (i, j) da matriz é 1, então as pessoas n_i e n_j são amigas entre si. Caso a posição (i, j) da matriz é 0, então n_i e n_j não são amigas. Observe que a pessoa n_0 é amiga das pessoas n_1, n_2 e n_4 , mas não é amiga da pessoa n_3 . Importante: a relação de amizade é simétrica: se n_i é amigo de n_j , então n_j é, necessariamente, amigo de n_i . Além disso, em redes sociais de amizade, não existe aresta entre a mesma pessoa, ou seja, não existem arestas do tipo (i, i) .

Nesta prática, considere que você vai implementar um simulador de redes sociais de amizade usando uma matriz de adjacência. O número de pessoas da rede social é definido na constante `NUM_PESSOAS` do arquivo `redesocial.h`. A matriz de adjacência é a variável global `M[NUM_PESSOAS][NUM_PESSOAS]`, declarada no arquivo `redesocial.c`. Uma variável global tem um escopo global, ou seja, pode ser usada em qualquer parte do arquivo em que ela foi declarada sem a necessidade de passá-la como parâmetro. Neste exercício, considere que as pessoas da rede social podem ser identificadas pelos inteiros $0, 1, 2, \dots, \text{NUM_PESSOAS}-1$.

Questões

Todos os exercícios a seguir devem ser implementados no arquivo `redesocial.c`.

2.1 Implementar um procedimento para inicializar a matriz de adjacência que gerencia a rede social. Inicialmente, ninguém é amigo de ninguém, ou seja, todas as posições da matriz são zeradas. Protótipo:

```
void inicializar_rede();
```

2.2 Implementar um procedimento para marcar duas pessoas como amigas na matriz de adjacência. Protótipo:

```
void adicionar_amizade(int i, int j);
```

Observação: Lembre que a relação de amizade é simétrica!

2.3 Implementar uma função que retorna um número aleatório de tipo ponto flutuante entre 0 e 1. **Dica:** o maior número aleatório gerado pela função `rand()` é definido pela constante `RAND_MAX` da biblioteca `stdlib.h`. Protótipo:

```
float random_float();
```

2.4 Implementar um procedimento para criar uma rede social aleatória a partir de um único parâmetro $P \in [0, 1]$. **Para cada par de pessoas** (i, j) , este procedimento deve gerar um número aleatório de tipo ponto flutuante r entre 0 e 1 (ex: 0.2345). Caso r seja menor que P , então deve-se criar uma amizade entre as pessoas n_i e n_j . Exemplo: se $P = 0.8$, para o par de pessoas n_0 e n_1 , se o número r gerado for 0.5412, então você deve criar uma amizade entre essas pessoas. Você deve repetir esse processo para todos os pares de pessoas.

Protótipo: $P \in [0, 1]$. Para cada par de pessoas (i, j) , este procedimento deve gerar um número aleatório de tipo ponto flutuante r entre 0 e 1 (ex: 0.2345). Caso r seja menor que P , então deve-se criar uma amizade entre as pessoas n_i e n_j . Exemplo: se $P = 0.8$, para o par de pessoas n_0 e n_1 , se o número r gerado for 0.5412, então você deve criar uma amizade entre essas pessoas. Você deve repetir esse processo para todos os pares de pessoas.

Protótipo:

```
void popularRedeSocialAleatoriamente(float P);
```

Observações: Lembre que a relação de amizade é simétrica, ou seja, se você testou o par (i, j) então você não deve testar o par (j, i) . Lembre também que uma pessoa não pode ser amiga dela mesma. (i, j) então você não deve testar o par (j, i) . Lembre também que uma pessoa não pode ser amiga dela mesma.

2.5 Implementar um procedimento para imprimir a matriz de adjacência de uma rede social. Protótipo:

```
void imprimirRedeSocial();
```

2.6 Implementar uma função para retornar o número de amigos em comum que duas pessoas têm. Essa função deve também imprimir os identificadores dos amigos em comum. Protótipo:

```
int numAmigosEmComum(int v, int u);
```

2.7 DESAFIO PARA OS FORTES: Implementar uma função para calcular o coeficiente de aglomeração de uma pessoa. Protótipo:

```
float coeficienteAglomeracao(int v);
```

O coeficiente de aglomeração de um nó i em um grafo é a probabilidade de dois amigos de i serem também amigos entre si. Ele é calculado da seguinte maneira: i em um grafo é a probabilidade de dois amigos de i serem também amigos entre si. Ele é calculado da seguinte maneira:

- Conte o número n de amigos de i . n de amigos de i .
- Crie um contador $cont$ e o inicialize com 0. $cont$ e o inicialize com 0.
- Para cada amigo u de i , conte quantos amigos v de i também é amigo de u , lembrando que $u \neq v$. Adicione essa contagem à $cont$. u de i , conte quantos amigos v de i também é amigo de u , lembrando que $u \neq v$. Adicione essa contagem à $cont$.
- O coeficiente de aglomeração é o quociente da divisão entre $cont$ e o número máximo possível de amizades entre os n amigos de i , dado por $n \times (n - 1) / 2$. $cont$ e o número máximo possível de amizades entre os n amigos de i , dado por $n \times (n - 1) / 2$.

Observação: o coeficiente de aglomeração deve ser um número entre 0 e 1.

```
/*----- redesocial.h -----*/  
  
#define NUM_PESSOAS 7  
  
void inicializar_rede();  
  
void adicionar_amizade(int i, int j);  
  
float random_float();  
  
void popularRedeSocialAleatoriamente(float P);  
  
void imprimirRedeSocial();  
  
int numAmigosEmComum(int v, int u);
```



```
/*----- redesocial.c (INCOMPLETO) -----*/

#include <stdio.h>

#include <stdlib.h>

#include "redesocial.h"

int M[NUM_PESSOAS][NUM_PESSOAS];

void inicializar_rede() {

}

void adicionar_amizade(int i, int j) {

}

float random_float() {

    return 0.0;

}

void popularRedeSocialAleatoriamente(float P) {

}

void imprimirRedeSocial() {

}

int numAmigosEmComum(int v, int u) {

    return 0;

}
```

```
/*----- pratica8.c (main) -----*/  
  
#include <stdio.h>  
  
#include "redesocial.h"  
  
  
void main() {  
  
    popularRedeSocialAleatoriamente(0.6);  
  
    imprimirRedeSocial();  
  
    int n = numAmigosEmComum(2,4);  
  
    printf("\nnúmero de amigos em comum entre 2 e 4: %d", n);  
    //gabarito: 2  
  
    //se voce eh forte, remova o comentario da linha abaixo  
  
    //printf("coef. de aglomeracao da pessoa 2 eh: %.2f",  
    coeficienteAglomeracao(2));  
  
    //gabarito: 0.67  
  
}
```