UNIVERSIDADE FEDERAL DO PIAUÍ
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS - CSHNB

Curso: Sistemas de Informação
Disciplina: Estruturas de dados 2
Docente: Juliana Oliveira de Carvalho
Discente: Marcos André Leal Silva

UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI

Relatório técnico

Picos-PI
04 de fevereiro de 2024

Resumo: Este projeto, tem como objetivo demonstrar a prática e implementação de duas Estruturas de Dados: Grafos e hash. Essas estruturas foram utilizadas para a resolução de alguns problemas práticos proposto no projeto através da Linguagem de programação C. O mesmo abordará as etapas de cada questão e apresentará suas respectivas implementações, oferecendo uma visão abrangente das aplicações dessas estruturas de dados. Além do mais, serão feitos experimentos para verificar os tempos de desempenho tanto das questões que envolvem grafos quanto das questões que envolvem funções hash.

Introdução: As estruturas de dados são um conceito muito importante para a programação e ciência de dados. Tratam-se de maneiras de agregar e organizar dados na memória de um computador ou dispositivo, de forma que façam sentido e proporcionem um bom desempenho ao serem processados. Nesse sentido, estamos considerando dados como sendo blocos de programação que representam algo e têm a função de resolver problemas computacionais. Para isso, eles devem ter a possibilidade de ser simbolizados, armazenados e manipulados.

A teoria dos grafos ou de grafos é um ramo da matemática que estuda as relações entre os objetos de um determinado conjunto. Para tal são utilizadas estruturas chamadas de grafos, $G(V,E)$ onde V é um conjunto não vazio de objetos denominados vértices (ou nós) e E (do inglês edges - arestas) é um subconjunto de pares não ordenados de V.

Uma função hash é um algoritmo que mapeia dados de comprimento variável para dados de comprimento fixo. Os valores retornados por uma função hash são chamados valores hash, códigos hash, somas hash (hash sums), checksums ou simplesmente hashes. Um uso é uma estrutura de dados chamada de tabela hash, amplamente usada em software de computador para consulta de dados rápida.

O objetivo desse projeto é através das estruturas de dados “Grafos” e “Tabela Hash”, criar três aplicações capazes de:

1. Aplicar um algoritmo de caminho mínimo para resolver o problema da Torre de Hanói com 4 discos a partir da combinação de modelagem de problemas com grafos, implementação de uma matriz de adjacência, aplicação de um algoritmo clássico de caminho mínimo (Ford-Moore-Bellman) e a análise do tempo de execução.
2. Encontrar o caminho mais confiável entre dois vértices em um grafo orientado. Neste contexto, a confiabilidade de uma aresta é representada por um valor no intervalo de 0 a 1, indicando a probabilidade de que o canal de comunicação associado não falhe. A busca pelo caminho mais confiável envolve encontrar a rota que maximize a probabilidade total ao longo das arestas entre dois vértices específicos.
3. Desenvolver dois programas que implementem diferentes funções de hashing e tratamento de colisões para organizar uma base de dados de 1000 funcionários. A organização é feita utilizando uma tabela de hash para localizar os dados dos funcionários com base em seus números de matrícula. A questão propõe duas funções de hashing distintas e pede para comparar o desempenho de ambas, tanto em termos de eficiência quanto no número de colisões geradas.

Para a implementação dos projetos, foi utilizado a linguagem de programação C, com seu paradigma de programação imperativo para a criação do código fonte. Através da ferramenta “Visual Studio Code (VS Code)”, esses programas foram criados, testados e executados.

O primeiro programa apresenta duas estruturas (Vertice e Grafo) para representar a implementação do grafo. “Vertice” representa um vértice no grafo. Cada vértice corresponde a uma configuração do jogo da Torre de Hanói (discos[4]: Array que armazena a configuração dos discos nos pinos), armazenando a disposição dos discos nos pinos. “Grafo” contém informações sobre o grafo que representa as configurações do jogo da Torre de Hanói (vertices: Ponteiro para um array de structs Vertice, representando os vértices do grafo; arestas: Matriz de adjacência representando as arestas do grafo. Cada entrada indica se há uma ligação entre dois vértices).

O segundo programa apresenta três estruturas (Aresta, Graco e FilaPrioridade) para representar a implementação do grafo. “Aresta” representa uma aresta no grafo, contendo informações sobre o vértice destino e sua confiabilidade (vertice: Número do vértice de destino; confiabilidade: Valor representando a confiabilidade da aresta). “Grafo” contém informações sobre o grafo, como as arestas que ligam os vértices (arestas: Ponteiro para um array de structs Aresta, representando as arestas conectadas a um vértice; num_arestas: Número total de arestas conectadas ao vértice). “FilaPrioridade” utilizada para implementar uma fila de prioridade durante o algoritmo de Dijkstra, mantendo informações sobre a confiabilidade e o vértice (confiabilidade: Valor representando a confiabilidade do vértice; vertice: Número do vértice).

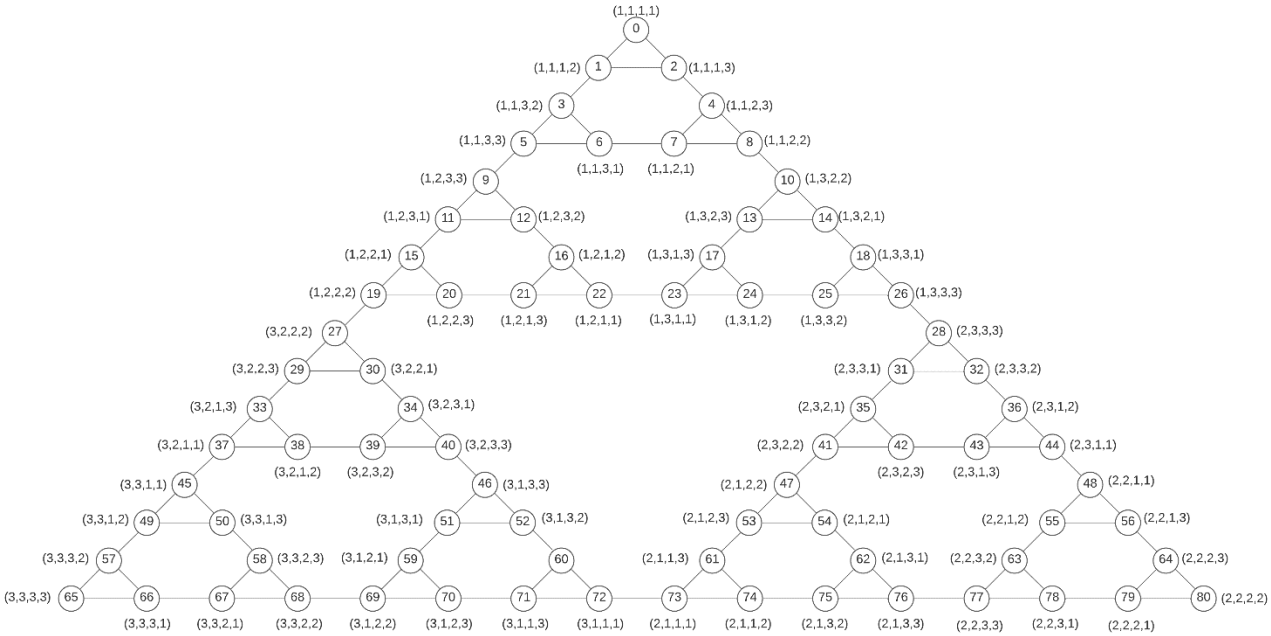
O terceiro programa apresenta uma estrutura (Func) para representar as informações sobre os funcionários (matricula: Uma string de até 6 caracteres que representa a matrícula do funcionário; nome: Uma string de até 49 caracteres que armazena o nome do funcionário; funcao: Uma string de até 19 caracteres que contém a função do funcionário; salario: Um valor em ponto flutuante que armazena o salário do funcionário). funcaoHashStringRotacao (char str): Realiza uma transformação na string de matrícula, efetua cálculos com alguns dígitos e retorna um índice para a tabela de tamanho 101 (MAX101).funcaoHashStringFoldShift (char str): Converte parte da string de matrícula em dois inteiros, soma-os e retorna um índice para a tabela de tamanho 150 (MAX150).

Seções específicas:

Informações técnicas: Para o desenvolvimento e testes deste projeto foi utilizado um notebook com um processador 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz, doze gigas de memória RAM ddr4, sistema operacional com a arquitetura de 64 bits, Windows 11 Home.

Os códigos foram feitos na IDE e editor de texto Visual Studio Code e compilados pelo compilador gcc (MinGW.org GCC Build-2) 9.2.0.

Questão 1:



O programa em questão é um jogo da Torre de Hanói implementado em C, que utiliza um grafo representado por uma matriz de adjacência para armazenar as configurações possíveis do jogo e as transições entre elas.

1. Estrutura do Grafo:
- O grafo é representado pela estrutura Grafo, que contém um vetor de vértices (vertices) e uma matriz de adjacência (arestas).
- Cada vértice é representado pela estrutura Vertice, que armazena as configurações dos discos nos pinos.

2. Criação do Grafo:
- A função Criar_Grafo aloca dinamicamente a memória necessária para o grafo, seus vértices e a matriz de adjacência.

3. Leitura do Grafo a partir de um Arquivo:

- A função Inserir_vertices_e_arestas lê as informações do arquivo torre_hanoi.txt para preencher os valores dos vértices e a matriz de adjacência.

4. Impressões:
- A função Imprimir_Matriz_adjacente imprime a matriz de adjacência.
- A função Imprimir_conteudo_do_vertice imprime as configurações de um vértice específico.

5. Algoritmo de Bellman-Ford:
- A função bellmanFord implementa o algoritmo de Bellman-Ford para encontrar o menor caminho em um grafo direcionado com arestas ponderadas.
- Verifica a existência de ciclos negativos no grafo.

6. Jogo da Torre de Hanói:
- As funções Vertices_adjacentes, Verificar_Vitoria, Verificar_Movimento_Valido e jogar implementam a lógica do jogo.

- O jogador pode escolher iniciar o jogo a partir de um vértice específico ou do vértice inicial.

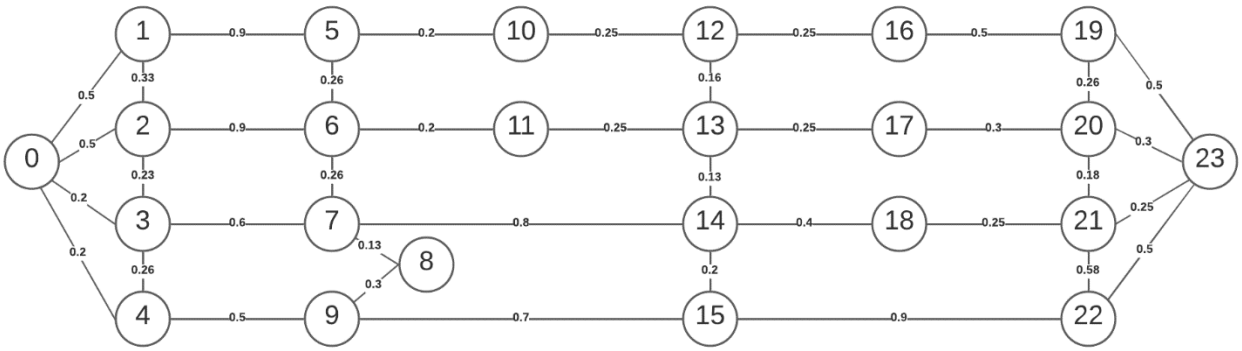
7. Menu Interativo:
- O programa apresenta um menu interativo que permite ao usuário escolher diferentes funcionalidades, como imprimir valores dos vértices, matriz adjacente, jogar, etc.

8. Liberação de Memória:
- A função Liberar_Grafo é responsável por liberar a memória alocada dinamicamente para o grafo.

9. Tempo de Execução:
- O programa utiliza a função QueryPerformanceCounter para medir o tempo de execução do algoritmo de Bellman-Ford.

10. Encerramento do Programa:
- O programa continua em execução até que o usuário escolha a opção "0" no menu, indicando a saída.

Questão 2:



O programa em questão implementa o algoritmo de Dijkstra para encontrar o caminho mais confiável entre dois vértices em um grafo ponderado.

1. Estruturas de Dados:
- Utiliza as estruturas Aresta, Grafo, e FilaPrioridade para representar arestas, o grafo e a fila de prioridade, respectivamente.

2. Alocação Dinâmica:
- Usa alocação dinâmica de memória para criar o grafo e suas arestas, garantindo flexibilidade em relação ao número de vértices e arestas.

3. Leitura de Arquivo:
- Lê as informações do grafo a partir de um arquivo (grafo.txt) que contém o número de vértices, o número de arestas e as informações de cada aresta.

4. Preenchimento do Grafo:
- A função Ler_Arestas_Do_Arquivo lê as arestas do arquivo e as insere no grafo utilizando a função Inserir_Arestas.

5. Algoritmo de Dijkstra:
- A função dijkstra implementa o algoritmo de Dijkstra para encontrar o caminho mais confiável entre dois vértices em um grafo ponderado.

- Usa uma fila de prioridade para otimizar a seleção do próximo vértice a ser explorado.

6. Impressão do Resultado:
- Imprime o caminho mais confiável entre os vértices de origem e destino, bem como a confiabilidade desse caminho.

7. Liberação de Memória:
- A função Liberar_Dijkstra_Memoria é responsável por liberar a memória alocada durante o algoritmo de Dijkstra.

8. Verificação de Vértices Válidos:
- Verifica se os vértices de origem e destino fornecidos pelo usuário são válidos, ou seja, estão dentro dos limites do grafo.

9. Entrada do Usuário:
- Solicita que o usuário insira os vértices de origem e destino para calcular o caminho mais confiável entre eles.

10. Liberação de Memória ao Final:
- A função Liberar_Grafo é utilizada para liberar a memória alocada para o grafo ao final do programa.

Questão 3:

O programa apresenta um sistema de gerenciamento de funcionários com duas tabelas hash diferentes: uma com 101 posições e outra com 150 posições.

1. Struct Func:
- Representa a estrutura de um funcionário com os campos matrícula, nome, função e salário.

2. Função lerFunc():
- Solicita ao usuário a entrada dos dados de um funcionário e retorna uma estrutura Func com essas informações.

3. Funções de Hash:
- colisaoRotacao: Lida com colisões usando o método de rotação.
- funcaoHashStringRotacao: Gera o valor hash utilizando o método de rotação.
- colisaoFoldShift: Lida com colisões usando o método de fold-shift.
- funcaoHashStringFoldShift: Gera o valor hash utilizando o método de fold-shift.

4. Funções inserirTabelaRotacao e inserirTabelaFoldShift:
- Inserem um funcionário na tabela de acordo com o método de hash escolhido. Contabilizam o número de colisões ocorridas.

5. Função imprimirFunc:
- Imprime as informações dos funcionários armazenados em uma tabela.

6. Menu de Opções:
- Permite escolher o tamanho da tabela (101 ou 150).
- Permite escolher o método de hash (rotação ou fold-shift).
- Oferece opções para inserir funcionários, imprimir a tabela e visualizar o número total de colisões.

7. Laços de Controle de Fluxo:
- Utiliza loops while para manter o programa em execução até a escolha da opção "0" para sair.

8. Alocação Dinâmica de Memória:
- Utiliza malloc para alocar memória para as tabelas de funcionários.

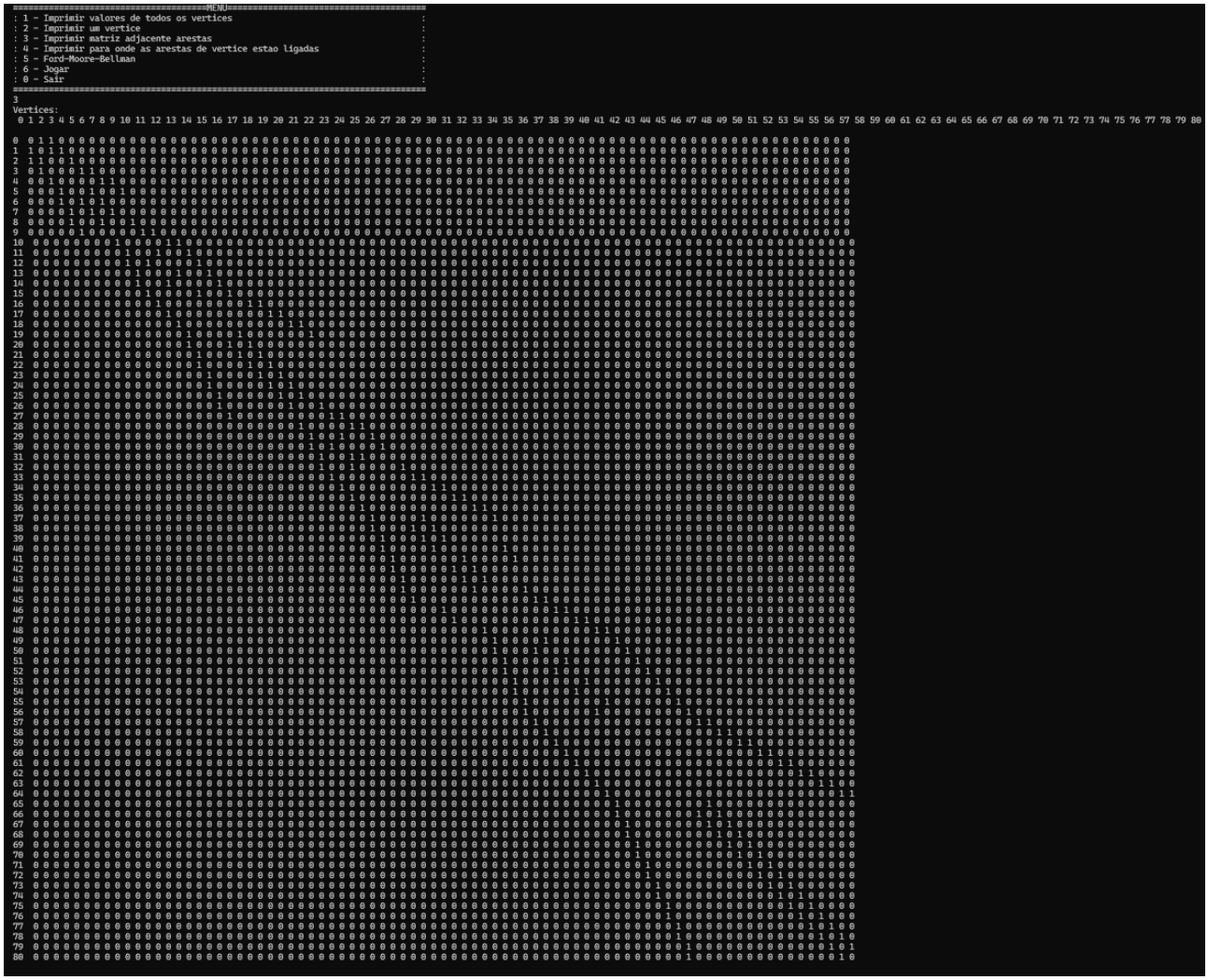
9. Liberação de Memória:
- Utiliza free para liberar a memória alocada após a conclusão do programa.

Resultados da execução do programa:

Questão 1:

```
=====================================MENU=====================================
: 1 - Imprimir valores de todos os vertices                                   :
: 2 - Imprimir um vertice                                                     :
: 3 - Imprimir matriz adjacente arestas                                       :
: 4 - Imprimir para onde as arestas de vertice estao ligadas                  :
: 5 - Ford-Moore-Bellman                                                      :
: 6 - Jogar                                                                   :
: 0 - Sair                                                                    :
==============================================================================
1
0: (1, 1, 1, 1)
1: (1, 1, 1, 2)
2: (1, 1, 1, 3)
3: (1, 1, 3, 2)
4: (1, 1, 2, 3)
5: (1, 1, 3, 3)
6: (1, 1, 3, 1)
7: (1, 1, 2, 1)
8: (1, 1, 2, 2)
9: (1, 2, 3, 3)
10: (1, 3, 2, 2)
11: (1, 2, 3, 1)
12: (1, 2, 3, 2)
13: (1, 3, 2, 3)
14: (1, 3, 2, 1)
15: (1, 2, 2, 1)
16: (1, 2, 1, 2)
17: (1, 3, 1, 3)
18: (1, 3, 3, 1)
19: (1, 2, 2, 2)
20: (1, 2, 2, 3)
21: (1, 2, 1, 3)
22: (1, 2, 1, 1)
23: (1, 3, 1, 1)
24: (1, 3, 1, 2)
25: (1, 3, 3, 2)
26: (1, 3, 3, 3)
27: (3, 2, 2, 2)
28: (2, 3, 3, 3)
29: (3, 2, 2, 3)
30: (3, 2, 2, 1)
31: (2, 3, 3, 1)
32: (2, 3, 3, 2)
33: (3, 2, 1, 3)
34: (3, 2, 3, 1)
35: (2, 3, 2, 1)
36: (2, 3, 1, 2)
37: (3, 2, 1, 1)
38: (3, 2, 1, 2)
39: (3, 2, 3, 2)
40: (3, 2, 3, 3)
41: (2, 3, 2, 2)
42: (2, 3, 2, 3)
43: (2, 3, 1, 3)
44: (2, 3, 1, 1)
45: (3, 3, 1, 1)
46: (3, 1, 3, 3)
47: (2, 1, 2, 2)
48: (2, 2, 1, 1)
49: (3, 3, 1, 2)
50: (3, 3, 1, 3)
51: (3, 1, 3, 1)
52: (3, 1, 3, 2)
53: (2, 1, 2, 3)
54: (2, 1, 2, 1)
55: (2, 2, 1, 2)
56: (2, 2, 1, 3)
57: (3, 3, 3, 2)
58: (3, 3, 2, 3)
59: (3, 1, 2, 1)
60: (3, 1, 1, 2)
61: (2, 1, 1, 3)
62: (2, 1, 3, 1)
63: (2, 2, 3, 2)
64: (2, 2, 2, 3)
65: (3, 3, 3, 3)
66: (3, 3, 3, 1)
67: (3, 3, 2, 1)
68: (3, 3, 2, 2)
69: (3, 1, 2, 2)
70: (3, 1, 2, 3)
71: (3, 1, 1, 3)
72: (3, 1, 1, 1)
73: (2, 1, 1, 1)
74: (2, 1, 1, 2)
75: (2, 1, 3, 2)
76: (2, 1, 3, 3)
77: (2, 2, 3, 3)
78: (2, 2, 3, 1)
79: (2, 2, 2, 1)
80: (2, 2, 2, 2)
=====================================MENU=====================================
```

```
=========================================MENU=========================================
: 1 - Imprimir valores de todos os vertices                                         :
: 2 - Imprimir um vertice                                                           :
: 3 - Imprimir matriz adjacente arestas                                             :
: 4 - Imprimir para onde as arestas de vertice estao ligadas                        :
: 5 - Ford-Moore-Bellman                                                            :
: 6 - Jogar                                                                         :
: 0 - Sair                                                                          :
======================================================================================
2
Digite o indice do vertice: 67
67: (3, 3, 2, 1)
```




```
=========================================MENU=========================================
: 1 - Imprimir valores de todos os vertices                                         :
: 2 - Imprimir um vertice                                                           :
: 3 - Imprimir matriz adjacente arestas                                             :
: 4 - Imprimir para onde as arestas de vertice estao ligadas                        :
: 5 - Ford-Moore-Bellman                                                            :
: 6 - Jogar                                                                         :
: 0 - Sair                                                                          :
======================================================================================
4
Digite o indice do vertice: 4
2: (1, 1, 1, 3)
7: (1, 1, 2, 1)
8: (1, 1, 2, 2)
```

```
=====================================MENU=======================================
: 1 - Imprimir valores de todos os vertices                                     :
: 2 - Imprimir um vertice                                                       :
: 3 - Imprimir matriz adjacente arestas                                         :
: 4 - Imprimir para onde as arestas de vertice estao ligadas                    :
: 5 - Ford-Moore-Bellman                                                        :
: 6 - Jogar                                                                     :
: 0 - Sair                                                                      :
================================================================================
5
Ford-Moore-Bellman
Digite o vertice de origem: 0
Digite o vertice de destino: 1
Tempo decorrido: 1.75380 ms

Caminho mais curto: 0 -> 1
Distancia: 1
-------------------------------------MENU---------------------------------------
```

```
=====================================MENU=======================================
: 1 - Imprimir valores de todos os vertices                                     :
: 2 - Imprimir um vertice                                                       :
: 3 - Imprimir matriz adjacente arestas                                         :
: 4 - Imprimir para onde as arestas de vertice estao ligadas                    :
: 5 - Ford-Moore-Bellman                                                        :
: 6 - Jogar                                                                     :
: 0 - Sair                                                                      :
================================================================================
5
Ford-Moore-Bellman
Digite o vertice de origem: 0
Digite o vertice de destino: 80
Tempo decorrido: 1.78510 ms

Caminho mais curto: 0 -> 2 -> 4 -> 8 -> 10 -> 14 -> 18 -> 26 -> 28 -> 32 -> 36 -> 44 -> 48 -> 56 -> 64 -> 80
Distancia: 15
```

```
=====================================MENU=======================================
: 1 - Imprimir valores de todos os vertices                                     :
: 2 - Imprimir um vertice                                                       :
: 3 - Imprimir matriz adjacente arestas                                         :
: 4 - Imprimir para onde as arestas de vertice estao ligadas                    :
: 5 - Ford-Moore-Bellman                                                        :
: 6 - Jogar                                                                     :
: 0 - Sair                                                                      :
================================================================================
5
Ford-Moore-Bellman
Digite o vertice de origem: 1
Digite o vertice de destino: 45
Tempo decorrido: 1.76440 ms

Caminho mais curto: 1 -> 3 -> 5 -> 9 -> 11 -> 15 -> 19 -> 27 -> 29 -> 33 -> 37 -> 45
Distancia: 11
```

```
=========================================MENU=============================================
: 1 - Imprimir valores de todos os vertices                                             :
: 2 - Imprimir um vertice                                                               :
: 3 - Imprimir matriz adjacente arestas                                                 :
: 4 - Imprimir para onde as arestas de vertice estao ligadas                            :
: 5 - Ford-Moore-Bellman                                                                :
: 6 - Jogar                                                                             :
: 0 - Sair                                                                              :
==========================================================================================
5
Ford-Moore-Bellman
Digite o vertice de origem: 1
Digite o vertice de destino: 5
Tempo decorrido: 1.77790 ms

Caminho mais curto: 1 -> 3 -> 5
Distancia: 2
```

```
=====================================MENU=====================================
: 1 - Imprimir valores de todos os vertices                                 :
: 2 - Imprimir um vertice                                                   :
: 3 - Imprimir matriz adjacente arestas                                     :
: 4 - Imprimir para onde as arestas de vertice estao ligadas                :
: 5 - Ford-Moore-Bellman                                                    :
: 6 - Jogar                                                                 :
: 0 - Sair                                                                  :
==============================================================================
5
Ford-Moore-Bellman
Digite o vertice de origem: 27
Digite o vertice de destino: 65
Tempo decorrido: 1.76160 ms

Caminho mais curto: 27 -> 29 -> 33 -> 37 -> 45 -> 49 -> 57 -> 65
Distancia: 7
```

```
=====================================MENU=====================================
: 1 - Imprimir valores de todos os vertices                                 :
: 2 - Imprimir um vertice                                                   :
: 3 - Imprimir matriz adjacente arestas                                     :
: 4 - Imprimir para onde as arestas de vertice estao ligadas                :
: 5 - Ford-Moore-Bellman                                                    :
: 6 - Jogar                                                                 :
: 0 - Sair                                                                  :
==============================================================================
5
Ford-Moore-Bellman
Digite o vertice de origem: 56
Digite o vertice de destino: 34
Tempo decorrido: 1.81540 ms

Caminho mais curto: 56 -> 55 -> 63 -> 77 -> 76 -> 75 -> 74 -> 73 -> 72 -> 60 -> 52 -> 46 -> 40 -> 34
Distancia: 13
```

```
=====================================MENU=====================================
: 1 - Imprimir valores de todos os vertices                                 :
: 2 - Imprimir um vertice                                                   :
: 3 - Imprimir matriz adjacente arestas                                     :
: 4 - Imprimir para onde as arestas de vertice estao ligadas                :
: 5 - Ford-Moore-Bellman                                                    :
: 6 - Jogar                                                                 :
: 0 - Sair                                                                  :
==============================================================================
5
Ford-Moore-Bellman
Digite o vertice de origem: 46
Digite o vertice de destino: 80
Tempo decorrido: 1.76600 ms

Caminho mais curto: 46 -> 52 -> 60 -> 72 -> 73 -> 74 -> 75 -> 76 -> 77 -> 78 -> 79 -> 80
Distancia: 11
```

```
=================================MENU=================================
: 1 - exprimir valores de todos os vertices                          :
: 2 - exprimir um vertice                                            :
: 3 - exprimir vertice adjacente aresta                              :
: 4 - exprimir para onde as arestas de vertice estao ligadas         :
: 5 - zero-more-matisse                                              :
: 6 - jogar                                                          :
: 0 - sair                                                           :
======================================================================
0

Deseja comecar de algum vertice? 1 - sim, 0 - nao: 0

Vertice atual e corrente.
 0: (1, 1, 1, 1)
Possibilidades de para onde pode ir.
1: (1, 1, 1, 0)
2: (1, 1, 0, 0)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 2

Vertice atual e corrente.
 2: (1, 1, 0, 0)
Possibilidades de para onde pode ir.
3: (1, 1, 1, 1)
1: (1, 1, 1, 0)
4: (1, 1, 0, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 4

Vertice atual e corrente.
 4: (1, 1, 0, 1)
Possibilidades de para onde pode ir.
6: (1, 1, 1, 1)
7: (1, 1, 0, 1)
8: (1, 1, 0, 0)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 8

Vertice atual e corrente.
 8: (1, 1, 0, 0)
Possibilidades de para onde pode ir.
9: (1, 1, 0, 1)
7: (1, 1, 0, 1)
10: (1, 0, 0, 0)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 10

Vertice atual e corrente.
 10: (1, 0, 0, 0)
Possibilidades de para onde pode ir.
9: (1, 1, 0, 0)
12: (1, 0, 0, 1)
14: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 14

Vertice atual e corrente.
 14: (1, 0, 1, 1)
Possibilidades de para onde pode ir.
12: (1, 0, 0, 1)
15: (1, 0, 1, 0)
13: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 1
Todos os vertices ja foram visitados ou nao foi informado um vertice que nao tem ligacao com o vertice atual.
Digite novamente o vertice que deseja ir: 15

Vertice atual e corrente.
 15: (1, 0, 1, 1)
Possibilidades de para onde pode ir.
14: (1, 0, 1, 1)
13: (1, 0, 1, 0)
11: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 13

Vertice atual e corrente.
 13: (1, 0, 1, 0)
Possibilidades de para onde pode ir.
14: (1, 0, 1, 1)
15: (1, 0, 1, 0)
11: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 11

Vertice atual e corrente.
 11: (1, 0, 1, 1)
Possibilidades de para onde pode ir.
13: (1, 0, 1, 0)
14: (1, 0, 1, 1)
10: (1, 0, 0, 0)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 10

Vertice atual e corrente.
 10: (1, 0, 0, 0)
Possibilidades de para onde pode ir.
11: (1, 0, 1, 1)
9: (1, 1, 0, 0)
12: (1, 0, 0, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 12

Vertice atual e corrente.
 12: (1, 0, 0, 1)
Possibilidades de para onde pode ir.
10: (1, 0, 0, 0)
14: (1, 0, 1, 1)
15: (1, 0, 1, 0)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 15

Vertice atual e corrente.
 15: (1, 0, 1, 0)
Possibilidades de para onde pode ir.
14: (1, 0, 1, 1)
13: (1, 0, 1, 0)
11: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 14

Vertice atual e corrente.
 14: (1, 0, 1, 1)
Possibilidades de para onde pode ir.
15: (1, 0, 1, 0)
13: (1, 0, 1, 0)
11: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 13

Vertice atual e corrente.
 13: (1, 0, 1, 0)
Possibilidades de para onde pode ir.
14: (1, 0, 1, 1)
15: (1, 0, 1, 0)
11: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 11

Vertice atual e corrente.
 11: (1, 0, 1, 1)
Possibilidades de para onde pode ir.
13: (1, 0, 1, 0)
14: (1, 0, 1, 1)
10: (1, 0, 0, 0)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 10

Vertice atual e corrente.
 10: (1, 0, 0, 0)
Possibilidades de para onde pode ir.
11: (1, 0, 1, 1)
9: (1, 1, 0, 0)
12: (1, 0, 0, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 12

Vertice atual e corrente.
 12: (1, 0, 0, 1)
Possibilidades de para onde pode ir.
10: (1, 0, 0, 0)
14: (1, 0, 1, 1)
15: (1, 0, 1, 0)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 15

Vertice atual e corrente.
 15: (1, 0, 1, 0)
Possibilidades de para onde pode ir.
14: (1, 0, 1, 1)
13: (1, 0, 1, 0)
11: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 14

Vertice atual e corrente.
 14: (1, 0, 1, 1)
Possibilidades de para onde pode ir.
15: (1, 0, 1, 0)
13: (1, 0, 1, 0)
11: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 13

Vertice atual e corrente.
 13: (1, 0, 1, 0)
Possibilidades de para onde pode ir.
14: (1, 0, 1, 1)
15: (1, 0, 1, 0)
11: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 11

Vertice atual e corrente.
 11: (1, 0, 1, 1)
Possibilidades de para onde pode ir.
13: (1, 0, 1, 0)
14: (1, 0, 1, 1)
10: (1, 0, 0, 0)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 10

Vertice atual e corrente.
 10: (1, 0, 0, 0)
Possibilidades de para onde pode ir.
11: (1, 0, 1, 1)
9: (1, 1, 0, 0)
12: (1, 0, 0, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 12

Vertice atual e corrente.
 12: (1, 0, 0, 1)
Possibilidades de para onde pode ir.
10: (1, 0, 0, 0)
14: (1, 0, 1, 1)
15: (1, 0, 1, 0)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 15

Vertice atual e corrente.
 15: (1, 0, 1, 0)
Possibilidades de para onde pode ir.
14: (1, 0, 1, 1)
13: (1, 0, 1, 0)
11: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 14

Vertice atual e corrente.
 14: (1, 0, 1, 1)
Possibilidades de para onde pode ir.
15: (1, 0, 1, 0)
13: (1, 0, 1, 0)
11: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 13

Vertice atual e corrente.
 13: (1, 0, 1, 0)
Possibilidades de para onde pode ir.
14: (1, 0, 1, 1)
15: (1, 0, 1, 0)
11: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 11

Vertice atual e corrente.
 11: (1, 0, 1, 1)
Possibilidades de para onde pode ir.
13: (1, 0, 1, 0)
14: (1, 0, 1, 1)
10: (1, 0, 0, 0)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 10

Vertice atual e corrente.
 10: (1, 0, 0, 0)
Possibilidades de para onde pode ir.
11: (1, 0, 1, 1)
9: (1, 1, 0, 0)
12: (1, 0, 0, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 12

Vertice atual e corrente.
 12: (1, 0, 0, 1)
Possibilidades de para onde pode ir.
10: (1, 0, 0, 0)
14: (1, 0, 1, 1)
15: (1, 0, 1, 0)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 15

Vertice atual e corrente.
 15: (1, 0, 1, 0)
Possibilidades de para onde pode ir.
14: (1, 0, 1, 1)
13: (1, 0, 1, 0)
11: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 14

Vertice atual e corrente.
 14: (1, 0, 1, 1)
Possibilidades de para onde pode ir.
15: (1, 0, 1, 0)
13: (1, 0, 1, 0)
11: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 13

Vertice atual e corrente.
 13: (1, 0, 1, 0)
Possibilidades de para onde pode ir.
14: (1, 0, 1, 1)
15: (1, 0, 1, 0)
11: (1, 0, 1, 1)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 11

Vertice atual e corrente.
 11: (1, 0, 1, 1)
Possibilidades de para onde pode ir.
13: (1, 0, 1, 0)
14: (1, 0, 1, 1)
10: (1, 0, 0, 0)
Deseja continuar jogando? 1 - sim, 0 - nao: 1

Digite o indice do vertice que deseja ir: 10
Nova versao em 15 exibentro
======================================================================
```

Questão 2:

```
Digite o vertice de origem e o vertice de destino: 0 23
O caminho mais confiavel entre 0 e 23 tem confiabilidade 0.0315
Caminho: 0 -> 4 -> 9 -> 15 -> 22 -> 23
```

```
Digite o vertice de origem e o vertice de destino: 15 22
O caminho mais confiavel entre 15 e 22 tem confiabilidade 0.9000
Caminho: 15 -> 22
```

```
Digite o vertice de origem e o vertice de destino: 8 23
O caminho mais confiavel entre 8 e 23 tem confiabilidade 0.0945
Caminho: 8 -> 9 -> 15 -> 22 -> 23
```

```
Digite o vertice de origem e o vertice de destino: 5 12
O caminho mais confiavel entre 5 e 12 tem confiabilidade 0.0500
Caminho: 5 -> 10 -> 12
```

```
Digite o vertice de origem e o vertice de destino: 0 7
O caminho mais confiavel entre 0 e 7 tem confiabilidade 0.1200
Caminho: 0 -> 3 -> 7
```

```
Digite o vertice de origem e o vertice de destino: 9 22
O caminho mais confiavel entre 9 e 22 tem confiabilidade 0.6300
Caminho: 9 -> 15 -> 22
```

```
Digite o vertice de origem e o vertice de destino: 9 23
O caminho mais confiavel entre 9 e 23 tem confiabilidade 0.3150
Caminho: 9 -> 15 -> 22 -> 23
```

Questão 3.1:

```
116: 112558
117: 118551
118: 113459
119: 120789
120: 118070
121: 126950
122: 128480
123: 106087
124: 105102
125: 130396
126: 109220
127: 119786
128: 109982
129: 131835
130: 114516
131: 110509
132: 103044
133: 127995
134: 123160
135: 114209
136: 129161
137: 117207
138: 125140
139: 110743
140: 103672
141: 123142
142: 112077
143: 108740
144: 102270
145: 129421
146: 109674
147: 122339
148: 104996
149: 131843

Colisoes Rotacao 101: 3137 || Num inseridos: 1000
Colisoes Fold Shift 101: 485 || Num inseridos: 1000
Colisoes Rotacao 150: 7544 || Num inseridos: 1000
Colisoes Fold Shift 150: 1163 || Num inseridos: 1000
```

```
116: 130771
117: 129897
118: 123323
119: 119742
120: 124514
121: 115301
122: 100476
123: 109976
124: 123313
125: 105322
126: 111654
127: 117830
128: 119936
129: 115637
130: 107236
131: 119799
132: 131463
133: 107939
134: 119560
135: 103701
136: 111994
137: 103441
138: 119494
139: 123542
140: 130381
141: 124295
142: 106173
143: 121513
144: 130975
145: 127166
146: 112372
147: 125160
148: 115486
149: 109013

Colisoes Rotacao 101: 3221 || Num inseridos: 1000
Colisoes Fold Shift 101: 507 || Num inseridos: 1000
Colisoes Rotacao 150: 7352 || Num inseridos: 1000
Colisoes Fold Shift 150: 844 || Num inseridos: 1000
```

```
116: 129546
117: 105796
118: 114760
119: 111471
120: 124976
121: 122015
122: 122689
123: 101657
124: 112114
125: 102105
126: 129357
127: 123498
128: 131746
129: 117753
130: 103760
131: 118804
132: 107235
133: 124574
134: 123713
135: 130392
136: 100183
137: 128558
138: 103768
139: 110245
140: 119851
141: 112519
142: 124081
143: 120298
144: 119870
145: 109942
146: 110909
147: 102009
148: 115080
149: 105039

Colisoes Rotacao 101: 3129 || Num inseridos: 1000
Colisoes Fold Shift 101: 464 || Num inseridos: 1000
Colisoes Rotacao 150: 7431 || Num inseridos: 1000
Colisoes Fold Shift 150: 1157 || Num inseridos: 1000
```

```
116: 126959
117: 119016
118: 126543
119: 115738
120: 101557
121: 131563
122: 118793
123: 118819
124: 104862
125: 124970
126: 127855
127: 111691
128: 124606
129: 108941
130: 129573
131: 128655
132: 124211
133: 101267
134: 127865
135: 106395
136: 102671
137: 107753
138: 125789
139: 106681
140: 108336
141: 108438
142: 100597
143: 106798
144: 131748
145: 123531
146: 114417
147: 112860
148: 131160
149: 102987

Colisoes Rotacao 101: 3298 || Num inseridos: 1000
Colisoes Fold Shift 101: 831 || Num inseridos: 1000
Colisoes Rotacao 150: 6872 || Num inseridos: 1000
Colisoes Fold Shift 150: 1470 || Num inseridos: 1000
```

```
128: 129076
129: 129502
130: 124696
131: 121829
132: 101862
133: 122704
134: 112336
135: 103236
136: 129417
137: 118672
138: 104245
139: 129855
140: 125090
141: 132602
142: 105705
143: 110082
144: 108281
145: 114485
146: 124537
147: 115175
148: 105721
149: 120393

Colisoes Rotacao 101: 3015 || Num inseridos: 1000
Colisoes Fold Shift 101: 524 || Num inseridos: 1000
Colisoes Rotacao 150: 7325 || Num inseridos: 1000
Colisoes Fold Shift 150: 1141 || Num inseridos: 1000
```

```
116: 119615
117: 102689
118: 101485
119: 132312
120: 118423
121: 115701
122: 105645
123: 116151
124: 126060
125: 120063
126: 108656
127: 116182
128: 113361
129: 109963
130: 102779
131: 116477
132: 109117
133: 103399
134: 107407
135: 107455
136: 101778
137: 101212
138: 112175
139: 119696
140: 106770
141: 106839
142: 109093
143: 121301
144: 130290
145: 113309
146: 105982
147: 124631
148: 111646
149: 110284

Colisoes Rotacao 101: 3233 || Num inseridos: 1000
Colisoes Fold Shift 101: 342 || Num inseridos: 1000
Colisoes Rotacao 150: 6649 || Num inseridos: 1000
Colisoes Fold Shift 150: 1380 || Num inseridos: 1000
```

A partir da questão 1, foi observado que o número de movimentos mínimos para vencer o jogo da torre de hanoi com 4 discos (começando no vértice 0 até o vértice 65 ou 80) são de 15 movimentos. A questão 2 foi observado que o nível de confiabilidade pode variar bastante de um vértice até outro, e que o nível de confiabilidade de uma extremidade a outra do grafo (vértice 0 até vértice 23) foi de 0.0315 de acordo com o caminho percorrido, que pode ser analisada como de baixa confiabilidade, valores mais próximos de 1 são mais confiáveis. Por fim, a questão 3.1 (variação da questão 3 só que inserindo somente números de matriculas aleatórias) foi perceptível analisar que o método fold shift foi mais eficiente para tratar as colisões, pois a partir dele, outras colisões foram evitadas, aumentando seu desempenho em relação ao de rotação (Tanto no vetor de 100 posições quanto no de 150 posições).

Conclusão: Os programas forneceram uma estrutura básica para manipulação de grafos e de tabela hash. Além do mais, os experimentos de verificação de tempo, se mostraram ser eficientes perante a máquina utilizada para testar os mesmos. Com tudo que já foi dito, podemos concluir que os programas se mostram eficiente nos resultados de saída exibidos. Todas demais informações já foram citadas, explanadas e detalhadas nos tópicos acima. Sendo assim, os programas conseguiram responder as problemáticas dos projetos e se mostram sem erros ou insuficiências. Com estes experimentos, foi possível desenvolver e manipular grafos e tabela hash através da Linguagem C, apesar de ser um projeto simples, a boa interpretação do problema é crucial para uma ótima aplicação.

Apêndice:

Questão 1:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>

#define TAM 81
#define INF 999999

/*Prototipos*/
typedef struct vertice Vertice;
typedef struct grafo Grafo;

/*Structs*/
typedef struct vertice{

    int discos[4];
}Vertice;

typedef struct grafo {
    Vertice *vertices;
    int **arestas;

}Grafo;

/*Declaração das funções*/
```

```c
Grafo *Criar_Grafo(int n);
void menu();

/*Impressões*/
void Imprimir_Matriz_adjacente(Grafo *G, int n);
void Imprimir_Menor_Caminho_Bellman(int vertices, int destino, int
verticeAnterior[], int distancia[]);
void Imprimir_conteudo_do_vertice(Grafo *G, int linha);

void Inserir_vertices_e_arestas(Grafo *G);

void bellmanFord(int **grafo, int vertices, int origem, int destino);

void Liberar_Grafo(Grafo **G, int qtdVertice);

/*Jogo*/
void Vertices_adjacentes(int *vetorVertice, int *vetorResposta, int *cont);
int Verificar_Vitoria(Grafo *G, int vertice, int pinoDesejado);
int Verificar_Movimento_Valido(int *vetorResposta, int vertice);
void jogar(Grafo *G, int vertice);


Grafo *Criar_Grafo(int n){
    // n é o numero de vertices ou configuracoes possiveis
    Grafo *G;

    G = (Grafo *) malloc(sizeof(Grafo));

    // Alocação de memoria para o vetor de vertices
    G->vertices = (Vertice *) malloc(n * sizeof(Vertice));

    // Alocação de memoria para a matriz de arestas
    G->arestas = (int **) malloc(n * sizeof(int *));
    for(int i = 0; i < n; i++){
        G->arestas[i] = (int *) calloc(n, sizeof(int));
    }

    return G;
}

void Imprimir_Matriz_adjacente(Grafo *G, int n){
    printf("Vertices: \n");
    for(int i = 0; i < n; i++){
        printf(" %d", i);
    }
    printf("\n\n");
    for(int i = 0; i < n; i++){
        printf("%d  ", i);
        for(int j = 0; j < n; j++){
            printf("%d ", G->arestas[i][j]);
```

```
        }
        printf("\n");
    }
    printf("\n");
}

void Inserir_vertices_e_arestas(Grafo *G){
    FILE *verticesEarestas;

    verticesEarestas = fopen("../torre_hanoi.txt", "r");
    if(verticesEarestas == NULL){
        printf("Error ao localizar o arquivo desejado!!");
    }else{

        int verticeAtual = 0, vermelho, amarelo, verde, rosa, aresta1, aresta2,
aresta3;

        // Inserir os valores dos vertices no grafo a partir do arquivo ../
        while (fscanf(verticesEarestas, "%d %d %d %d %d %d %d %d", &verticeAtual,
&vermelho, &amarelo, &verde, &rosa, &aresta1, &aresta2, &aresta3) != EOF){

        //preenche os pinos dos vertices
        G->vertices[verticeAtual].discos[0] = vermelho;
        G->vertices[verticeAtual].discos[1] = amarelo;
        G->vertices[verticeAtual].discos[2] = verde;
        G->vertices[verticeAtual].discos[3] = rosa;

        // marca a ligação das arestas na matriz adjacente
        G->arestas[verticeAtual][aresta1] = 1;
        G->arestas[verticeAtual][aresta2] = 1;
        if(aresta3 != 9999)
            G->arestas[verticeAtual][aresta3] = 1;
    }
    }

}

void Imprimir_conteudo_do_vertice(Grafo *G, int linha){
    printf("%d: (%d, %d, %d, %d)\n", linha, G->vertices[linha].discos[0], G-
>vertices[linha].discos[1], G->vertices[linha].discos[2], G-
>vertices[linha].discos[3]);
}

void bellmanFord(int **grafo, int vertices, int origem, int destino) {
    LARGE_INTEGER start_time, end_time, frequency;
    double elapsed_time_ms;
    int possuiCicloNegativo = 0;
    int *distancia, *verticeAnterior;
    distancia = (int*)malloc(vertices * sizeof(int));
    verticeAnterior = (int*)malloc(vertices * sizeof(int));
```

```
    // Os arrays de distância e vértices anteriores são inicializados com valores
que indicam a
    // ausência de ligação(infinito) ou que a posição do vetor ainda não foi
utilizada(-1)
    for (int i = 0; i < vertices; i++) {
        distancia[i] = INF;
        verticeAnterior[i] = -1;
    }
    // A distância do vértice de origem para ele mesmo é zero
    distancia[origem] = 0;
    QueryPerformanceFrequency(&frequency);
    QueryPerformanceCounter(&start_time);
    // Encontra o caminho mais curto partindo do vértice de origem para todos os
demais vértices
    for (int qtdRelaxamentos = 0; qtdRelaxamentos < vertices - 1;
qtdRelaxamentos++) {
        for (int i = 0; i < vertices; i++) {
            for (int j = 0; j < vertices; j++) {
                if (grafo[i][j] && distancia[i] != INF && distancia[i] +
grafo[i][j] < distancia[j]) {
                    distancia[j] = distancia[i] + grafo[i][j];
                    verticeAnterior[j] = i;
                }
            }
        }
    }

    // Checa se existe um ciclo negativo no grafo,
    // caso exista não é possível encontrar o menor caminho no grafo devido ás
limitações do algoritmo
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            if (grafo[i][j] && distancia[i] != INF && distancia[i] + grafo[i][j] <
distancia[j]) {
                printf("O grafo contem um ciclo negativo.\n");
                possuiCicloNegativo = 1;
                j = vertices;
                i = j;
            }
        }
    }
    QueryPerformanceCounter(&end_time);
    elapsed_time_ms = ((double)(end_time.QuadPart - start_time.QuadPart) * 1000.0)
/ frequency.QuadPart;
    printf("Tempo decorrido: %.5f ms\n", elapsed_time_ms);
    // caso não exista um ciclo negativo, imprime o menor caminho
    if(!possuiCicloNegativo)
        Imprimir_Menor_Caminho_Bellman(vertices,destino,verticeAnterior,distancia);
    free(distancia);
    free(verticeAnterior);
}
```

```c
void Imprimir_Menor_Caminho_Bellman(int vertices, int destino, int
verticeAnterior[], int distancia[]){
    // Mostrar o caminho mais curto para o vértice de destino como também a
distância
    printf("\nCaminho mais curto: ");
    int comprimentocaminho = 0;
    int *caminhovertices; caminhovertices = (int*)malloc(vertices * sizeof(int));

    int verticeatual = destino;

    while (verticeatual != -1) {
        caminhovertices[comprimentocaminho++] = verticeatual;
        verticeatual = verticeAnterior[verticeatual];
    }

    for (int i = comprimentocaminho - 1; i >= 0; i--) {
        printf("%d ", caminhovertices[i]);
        if (i > 0)
            printf("-> ");
    }
    printf("\nDistancia: %d\n", distancia[destino]);
    free(caminhovertices);
}

void Liberar_Grafo(Grafo **G, int qtdVertice){
    for (int i = 0; i < qtdVertice; i++){
        free((*G)->arestas[i]);
        (*G)->arestas[i] = NULL;
    }
    free((*G)->vertices);
    (*G)->vertices = NULL;

    free(*G);
    *G = NULL;
}

// jogo
// verifica resposta informa para quais arestas um vertice pode ir
void Vertices_adjacentes(int *vetorVertice, int *vetorResposta, int *cont){
    *cont = 0;

    for(int i = 0; i < 81; i++){
        if(vetorVertice[i] == 1){
            if(*cont == 0)
                vetorResposta[0] = i;
            if(*cont == 1)
                vetorResposta[1] = i;
            if(*cont == 2)
                vetorResposta[2] = i;
            *cont += 1;
```

```
        }
    }
}

// verificar se todos os pinos estão no 3 pino, retorna 4 caso estejam.
int Verificar_Vitoria(Grafo *G, int vertice, int pinoDesejado){
    int incremento = 0;
    for (int i = 0; i < 4; i++){
        if(G->vertices[vertice].discos[i] == pinoDesejado)
        {
            incremento += 1;
        }
    }
    return incremento; // Retorna 1 se todos os discos estiverem no pino desejado,
0 caso contrário.
}

// verifica se o vertice informado é uma opção para onde ir
int Verificar_Movimento_Valido(int *vetorResposta, int vertice){
    int cont = 1;
    if (vertice == vetorResposta[0])
        cont = 0;
    if (vertice == vetorResposta[1])
        cont = 0;
    if (vertice == vetorResposta[2])
        cont = 0;

    return cont;
}

// Essa função permite que o usuário jogue o jogo
void jogar(Grafo *G, int vertice){
    int numMovimentos = 0;
    int vetorResposta[3], cont, flagParada = 1;

    do{
         // Esse while continuar até que o usuário ganhe ou desista
        printf("\nVertice atual e conteudo.\n ");
        Imprimir_conteudo_do_vertice(G, vertice);
        printf("Possibilidades de para onde pode ir: \n");

        // zera o vetor de resposta
        memset(vetorResposta, 0, sizeof(vetorResposta));

        // verifica resposta informa para quais arestas um vertice pode ir
        Vertices_adjacentes(G->arestas[vertice], vetorResposta, &cont);

        Imprimir_conteudo_do_vertice(G, vetorResposta[0]);
        Imprimir_conteudo_do_vertice(G, vetorResposta[1]);

        if(cont == 3){
```

```
            Imprimir_conteudo_do_vertice(G, vetorResposta[2]);
        }

        printf("Deseja continuar jogando? 1 - sim, 0 - nao: ");
        scanf("%d", &flagParada);

        // entra nesse if se o usuário deseja continuar jogando, se não entrar isso
significa que ele desistiu.
        if(flagParada == 1){

            printf("\nDigite o indice do vertice que deseja ir: ");
            scanf("%d", &vertice);

            while ((vertice < 0 || vertice > 80) ||
(Verificar_Movimento_Valido(vetorResposta, vertice))){
                printf("Indice de vertice nao existe ou foi informado um vertice
que nao tem ligacao com o vertice atual.\n");
                printf("Digite novamente o vertice que deseja ir: ");
                scanf("%d", &vertice);
            }
            numMovimentos++;
        }else if(flagParada == 0){
            printf("Voce desistiu em %d movimentos\n", numMovimentos);
        }else{
            printf("Opcao invalida\n");
        }
    }while((Verificar_Vitoria(G, vertice, 1) != 4) && (Verificar_Vitoria(G,
vertice, 2) != 4) && (Verificar_Vitoria(G, vertice, 3) != 4) && (flagParada != 0));


    if(flagParada == 1)
        printf("Voce venceu em %d movimentos\n", numMovimentos);
}

void menu()
{
    printf("=====================================MENU=============================
==========\n");
    printf(": 1 - Imprimir valores de todos os
vertices                                    :\n");
    printf(": 2 - Imprimir um
vertice                                                  :\n");
    printf(": 3 - Imprimir matriz adjacente
arestas                                          :\n");
    printf(": 4 - Imprimir para onde as arestas de vertice estao
ligadas                    :\n");
    printf(": 5 - Ford-Moore-
Bellman                                                   :\n");
    printf(": 6 -
Jogar                                                                  :\n");
```

```
    printf(": 0 -
Sair                                                                    :\n");
    printf("================================================================================
==========\n");
}

int main(){
    Grafo *G;
    int op, vertice, opJogo, cont, vetorResposta[3], origem, destino;

    G = Criar_Grafo(81);

    Inserir_vertices_e_arestas(G);

    do{
        menu();
        scanf("%d", &op);

        switch (op){
        case 0:
            printf("Saindo...\n");
            break;
        case 1:
            for(int i = 0; i < TAM; i++)
                Imprimir_conteudo_do_vertice(G, i); // imprimir valores dos
vertices
            break;
        case 2:
            printf("Digite o indice do vertice: ");
            scanf("%d", &vertice);
            if(vertice >= 0 && vertice <= 80)
                Imprimir_conteudo_do_vertice(G, vertice);
            else
                printf("Indice de vertice nao existe.\n");
            break;
        case 3:
            Imprimir_Matriz_adjacente(G, TAM); //imprimir arestas
            break;
        case 4:
            cont = 0;
            printf("Digite o indice do vertice: ");
            scanf("%d", &vertice);

            if(vertice >= 0 && vertice <= 80){
                Vertices_adjacentes(G->arestas[vertice], vetorResposta, &cont);

                Imprimir_conteudo_do_vertice(G, vetorResposta[0]);
                Imprimir_conteudo_do_vertice(G, vetorResposta[1]);

                if(cont == 3){
                    Imprimir_conteudo_do_vertice(G, vetorResposta[2]);
```

```
                }
                memset(vetorResposta, 0, sizeof(vetorResposta));

            }else
                printf("Indice de vertice nao existe.\n");
            break;
        case 5:
            printf("Ford-Moore-Bellman\n");
            printf("Digite o vertice de origem: ");
            scanf("%d", &origem);
            printf("Digite o vertice de destino: ");
            scanf("%d", &destino);

            if((origem >= 0 && origem <= 80) && (destino >= 0 && destino <= 80) )
                bellmanFord(G->arestas, TAM, origem, destino);
            else
                printf("Verifique se os vertices informados existem\n");
            break;
        case 6:
            printf("\nDeseja comecar de algum vertice? 1 - sim, 2 - nao: ");
            scanf("%d", &opJogo);

            if(opJogo == 1){
                printf("\nDigite o indice do vertice: ");
                scanf("%d", &vertice);

                if(vertice >= 0 && vertice <= 80)
                    jogar(G, vertice);
                else
                    printf("\nIndice de vertice nao existe.\n");
            }else if(opJogo == 2)
                jogar(G, 0);
            else
                printf("\nOpcao invalida\n");
            break;
        default:
            printf("Opcao invalida.\n");
            break;
        }
    }while (op != 0);
    Liberar_Grafo(&G, 81);
    return 0;
}
```

Questão 2:

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

```c
#define MAX_VERTICES 1000

/*Prototipos*/
typedef struct aresta Aresta;
typedef struct grafo Grafo;
typedef struct filaprioridade FilaPrioridade;

/*Structs*/
typedef struct aresta{
    int vertice;
    double confiabilidade;
} Aresta;

typedef struct grafo{
    Aresta *arestas;
    int num_arestas;
} Grafo;

typedef struct filaprioridade{
    double confiabilidade;
    int vertice;
} FilaPrioridade;

/*Declaração das funções*/
void Liberar_Grafo(Grafo *grafo, int num_vertices);
Grafo* Criar_Grafo(int num_vertices);
void Imprimir_Caminho_Mais_Confiavel(const int *predecessores, int origem, int
destino);
void Liberar_Dijkstra_Memoria(bool *visitados, double *confiabilidade,
FilaPrioridade *fp, int *predecessores);
void dijkstra(const Grafo *grafo, int origem, int destino, int num_vertices);
void Inserir_Arestas(Grafo *grafo, int u, int v, double confiabilidade);
void Ler_Arestas_Do_Arquivo(FILE *arquivo, Grafo *grafo, int num_arestas);


// Função para liberar a memória alocada para o grafo
void Liberar_Grafo(Grafo *grafo, int num_vertices) {
    for (int i = 0; i < num_vertices; ++i) {
        free(grafo[i].arestas);
    }
    free(grafo);
}

// Função para preencher o grafo a partir de um arquivo
Grafo* Criar_Grafo(int num_vertices) {
    Grafo *grafo = (Grafo *)malloc(num_vertices * sizeof(Grafo));
    for (int i = 0; i < num_vertices; ++i) {
        grafo[i].num_arestas = 0;
        grafo[i].arestas = NULL;
    }
```

```
    return grafo;
}

void Ler_Arestas_Do_Arquivo(FILE *arquivo, Grafo *grafo, int num_arestas) {

    for (int i = 0; i < num_arestas; ++i) {
        int u, v;
        double confiabilidade;
        fscanf(arquivo, "%d %d %lf", &u, &v, &confiabilidade);
        Inserir_Arestas(grafo, u, v, confiabilidade);
    }
}

void Inserir_Arestas(Grafo *grafo, int u, int v, double confiabilidade) {
    grafo[u].arestas = (Aresta *)realloc(grafo[u].arestas, (grafo[u].num_arestas +
1) * sizeof(Aresta));
    grafo[u].arestas[grafo[u].num_arestas++] = (Aresta){v, confiabilidade};
}

// Função para imprimir o caminho mais confiável
void Imprimir_Caminho_Mais_Confiavel(const int *predecessores, int origem, int
destino) {
    if (destino == origem) {
        printf("%d", origem);
    } else if (predecessores[destino] == -1) {
        printf("Caminho inexistente entre %d e %d\n", origem, destino);
    } else {
        Imprimir_Caminho_Mais_Confiavel(predecessores, origem,
predecessores[destino]);
        printf(" -> %d", destino);
    }
}

// Função para liberar a memória alocada durante o algoritmo de Dijkstra
void Liberar_Dijkstra_Memoria(bool *visitados, double *confiabilidade,
FilaPrioridade *fp, int *predecessores) {
    free(visitados);
    free(confiabilidade);
    free(fp);
    free(predecessores);
}

// Função principal do algoritmo de Dijkstra
void dijkstra(const Grafo *grafo, int origem, int destino, int num_vertices) {
    bool *visitados = (bool *)malloc(num_vertices * sizeof(bool));
    double *confiabilidade = (double *)malloc(num_vertices * sizeof(double));
    int *predecessores = (int *)malloc(num_vertices * sizeof(int));

    for (int i = 0; i < num_vertices; ++i) {
        visitados[i] = false;
        confiabilidade[i] = 0.0;
```

```
        predecessores[i] = -1;
    }
    confiabilidade[origem] = 1.0;

    FilaPrioridade *fp = (FilaPrioridade *)malloc(num_vertices *
sizeof(FilaPrioridade));
    int tamanho_fp = 0;
    fp[tamanho_fp++] = (FilaPrioridade){1.0, origem};

    while (tamanho_fp > 0) {
        FilaPrioridade min_confiab = fp[0];
        int min_idx = 0;
        for (int i = 1; i < tamanho_fp; ++i) {
            if (fp[i].confiabilidade > min_confiab.confiabilidade) {
                min_confiab = fp[i];
                min_idx = i;
            }
        }

        fp[min_idx] = fp[--tamanho_fp];
        visitados[min_confiab.vertice] = true;

        if (min_confiab.vertice == destino) {
            printf("O caminho mais confiavel entre %d e %d tem confiabilidade
%.4lf\n", origem, destino, min_confiab.confiabilidade);
            printf("Caminho: ");
            Imprimir_Caminho_Mais_Confiavel(predecessores, origem, destino);
            printf("\n");

            Liberar_Dijkstra_Memoria(visitados, confiabilidade, fp, predecessores);
            return;
        }

        for (int i = 0; i < grafo[min_confiab.vertice].num_arestas; ++i) {
            Aresta aresta = grafo[min_confiab.vertice].arestas[i];
            if (!visitados[aresta.vertice]) {
                double nova_confiab = min_confiab.confiabilidade *
aresta.confiabilidade;
                if (nova_confiab > confiabilidade[aresta.vertice]) {
                    confiabilidade[aresta.vertice] = nova_confiab;
                    predecessores[aresta.vertice] = min_confiab.vertice;
                    fp[tamanho_fp++] = (FilaPrioridade){nova_confiab,
aresta.vertice};
                }
            }
        }
    }

    printf("Nao ha caminho confiavel entre %d e %d\n", origem, destino);
    Liberar_Dijkstra_Memoria(visitados, confiabilidade, fp, predecessores);
}
```

```c
int main() {
    FILE *arquivo = fopen("../Grafo.txt", "r");
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo.\n");
        return 1;
    }

    int num_vertices, num_arestas;
    fscanf(arquivo, "%d %d", &num_vertices, &num_arestas);

    Grafo *grafo = Criar_Grafo(num_vertices);
    Ler_Arestas_Do_Arquivo(arquivo, grafo, num_arestas * 2);

    int origem, destino;
    printf("Digite o vertice de origem e o vertice de destino: ");
    scanf("%d %d", &origem, &destino);

    if (origem < 0 || origem >= num_vertices || destino < 0 || destino >=
num_vertices) {
        printf("Vertices invalidos. Certifique-se de que os vertices inseridos
existem no grafo.\n");
    } else {
        dijkstra(grafo, origem, destino, num_vertices);
    }

    Liberar_Grafo(grafo, num_vertices);

    fclose(arquivo);

    return 0;
}
```

Questão 3:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX101 101
#define MAX150 150

/* Prototypes */
typedef struct func Func;
void imprimirFunc(Func *tabela, int MAX);
void inicializarTabela(Func *f, int MAX);

int colisaoRotacao(int chave, char primeiroDigito, int MAX);
int funcaoHashStringRotacao(char *str, int MAX);
```

```c
void inserirTabelaRotacao(Func *tabela, int *numColisoes, int *countTabela, int
MAX);

int colisaoFoldShift(int chave, int MAX);
int funcaoHashStringFoldShift(char *str, int MAX);
void inserirTabelaFoldShift(Func *tabela, int *numColisoes, int *countTabela, int
MAX);

/* Struct */

typedef struct func
{
    char matricula[7];
    char nome[50];
    char funcao[20];
    float salario;
} Func;

/* Imprimir */

void imprimirFunc(Func *tabela, int MAX)
{
    for (int i = 0; i < MAX; i++)
    {
        if (tabela[i].matricula[0] != '\0')
        {
            printf("Matricula: %s\n", tabela[i].matricula);
            printf("Nome: %s\n", tabela[i].nome);
            printf("Funcao: %s\n", tabela[i].funcao);
            printf("Salario: %.2f\n", tabela[i].salario);
            printf("\n");
        }
    }
}

/* Leitura dos dados */

Func lerFunc()
{
    Func f;
    while (strlen(f.matricula) < 6 || strlen(f.matricula) > 6)
    {
        printf("Informe a matricula do funcionario: ");
        scanf(" %[^\n]", f.matricula);
    }
    printf("Informe o nome do funcionario: ");
    scanf(" %[^\n]", f.nome);
    printf("Informe a funcao do funcionario: ");
    scanf(" %[^\n]", f.funcao);
    printf("Informe o salario do funcionario: ");
    scanf("%f", &f.salario);
```

```c
    return f;
}

/* Funcoes Hash */

void inicializarTabela(Func *f, int MAX)
{
    for (int i = 0; i < MAX; i++)
    {
        strcpy(f[i].matricula, "\0");
    }
}

/*Função hash rotação*/

int colisaoRotacao(int chave, char primeiroDigito, int MAX)
{
    return (chave + (primeiroDigito - '0'))% MAX;
}

int funcaoHashStringRotacao(char *str, int MAX)
{
    char chave[7];
    sprintf(chave, "%c%c%s", str[4], str[5], str);
    int d2 = chave[1] - '0';
    int d4 = chave[3] - '0';
    int d6 = chave[5] - '0';
    int valor_hash = d2 * 100 + d4 * 10 + d6;
    int posicao = valor_hash % MAX;
    return posicao;
}

void inserirTabelaRotacao(Func *tabela, int *numColisoes, int *countTabela, int
MAX)
{
    Func f = lerFunc();
    int indice = funcaoHashStringRotacao(f.matricula, MAX);
    if ((*countTabela) < MAX)
    {
        if (tabela[indice].matricula[0] != '\0')
        {
            (*numColisoes)++;
            int novo = colisaoRotacao(indice, f.matricula[0], MAX);
            printf("\n\n\nnovo:%d\n\n\n", novo);
            while (tabela[novo].matricula[0] != '\0')
            {
                (*numColisoes)++;
                novo = colisaoRotacao(novo, f.matricula[0], MAX);

            }
```

```
            tabela[novo] = f;
            (*countTabela)++;
        }
        else
        {
            (*countTabela)++;
            tabela[indice] = f;
        }
    }else
    {
        (*countTabela)++;
        tabela[indice] = f;
    }


}

/*Função hash fold shift*/

int colisaoFoldShift(int chave, int MAX)
{
    return (chave + 7) % MAX;
}

int funcaoHashStringFoldShift(char *str, int MAX)
{
    int tam = strlen(str);
    unsigned int hash = 0;

    unsigned int aux1 = 0;
    unsigned int aux2 = 0;

    for (int i = 0; i < tam; i++)
    {
        if (i == 0 || i == 2 || i == 5)
            aux1 = aux1 * 10 + (str[i] - '0');
        else
            aux2 = aux2 * 10 + (str[i] - '0');
    }

    hash = (aux1 + aux2) % MAX;

    return hash;
}

void inserirTabelaFoldShift(Func *tabela, int *numColisoes, int *countTabela, int
MAX)
{
    Func f = lerFunc();
    int indice = funcaoHashStringFoldShift(f.matricula, MAX);
```

```
    if ((*countTabela) < MAX)
    {
        if (tabela[indice].matricula[0] != '\0')
        {
            (*numColisoes)++;
            int novo = colisaoFoldShift(indice, MAX);

            while (tabela[novo].matricula[0] != '\0')
            {
                (*numColisoes)++;
                novo = colisaoFoldShift(novo, MAX);

            }
            tabela[novo] = f;
            (*countTabela)++;
        }
        else
        {
            (*countTabela)++;
            tabela[indice] = f;
        }
    }else
    {
        (*countTabela)++;
        tabela[indice] = f;
    }
}

void menu_tabela()
{
    printf("========== MENU ==========\n");
    printf("1 - Tabela de 101 posicoes\n");
    printf("2 - Tabela de 150 posicoes\n");
    printf("0 - Sair                  \n");
    printf("==========================\n");
}

void menu_hash()
{
    printf("========== MENU ==========\n");
    printf("1 - Hash rotacao          \n");
    printf("2 - Hash fold shift       \n");
    printf("0 - Sair                  \n");
    printf("==========================\n");
}
void menu_opcoes()
{
    printf("========== MENU ==========\n");
    printf("1 - Inserir funcionarios  \n");
    printf("2 - Imprimir funcionarios \n");
    printf("3 - Colisoes totais       \n");
```

```
    printf("0 - Voltar                \n");
    printf("==========================\n");
}

int main()
{
    Func *tabela101 = (Func *)malloc(MAX101 * sizeof(Func));
    Func *tabela150 = (Func *)malloc(MAX150 * sizeof(Func));
    int numColisoes101 = 0;
    int numColisoes150 = 0;
    int countTabela101 = 0;
    int countTabela150 = 0;
    int op = -1;

    while(op != 0)
    {
        menu_tabela();
        scanf("%d", &op);
        switch (op)
        {
        case 1:
            {
                menu_hash();
                scanf("%d", &op);
                switch (op)
                {
                case 1:
                    inicializarTabela(tabela101, MAX101);
                    op = -1;
                    while (op != 0)
                    {
                        op = -1;
                        menu_opcoes();
                        scanf("%d", &op);
                        switch (op)
                        {
                        case 1:
                            inserirTabelaRotacao(tabela101, &numColisoes101,
&countTabela101, MAX101);
                            break;
                        case 2:
                            imprimirFunc(tabela101, MAX101);
                            break;
                        case 3:
                            printf("Numero total de colisoes: %d\n",
numColisoes101);
                            break;
                        default:
                            break;
                        }
                    }
```

```
                    numColisoes101 = 0;
                    countTabela101 = 0;
                    op = -1;
                    break;
                case 2:
                    inicializarTabela(tabela101, MAX101);
                    while (op != 0)
                    {
                        op = -1;
                        menu_opcoes();
                        scanf("%d", &op);
                        switch (op)
                        {
                        case 1:
                            inserirTabelaFoldShift(tabela101, &numColisoes101,
&countTabela101, MAX101);
                            break;
                        case 2:
                            imprimirFunc(tabela101, MAX101);
                            break;
                        case 3:
                            printf("Numero total de colisoes: %d\n",
numColisoes101);
                            break;
                        default:
                            break;
                        }
                    }
                    numColisoes101 = 0;
                    countTabela101 = 0;
                    op = -1;
                    break;
                default:
                    break;
                }
            }
            op = -1;
            break;
        case 2:
            {
                menu_hash();
                scanf("%d", &op);
                switch (op)
                {
                case 1:
                    inicializarTabela(tabela150, MAX150);
                    op = -1;
                    while (op != 0)
                    {
                        op = -1;
                        menu_opcoes();
```

```
                        scanf("%d", &op);
                        switch (op)
                        {
                        case 1:
                            inserirTabelaRotacao(tabela150, &numColisoes150,
&countTabela150, MAX150);
                            break;
                        case 2:
                            imprimirFunc(tabela150, MAX150);
                            break;
                        case 3:
                            printf("Numero total de colisoes: %d\n",
numColisoes150);
                            break;
                        default:
                            break;
                        }
                    }
                    numColisoes150 = 0;
                    countTabela150 = 0;
                    op = -1;
                    break;
                case 2:
                    inicializarTabela(tabela150, MAX150);
                    while (op != 0)
                    {
                        op = -1;
                        menu_opcoes();
                        scanf("%d", &op);
                        switch (op)
                        {
                        case 1:
                            inserirTabelaFoldShift(tabela150, &numColisoes150,
&countTabela150, MAX150);
                            break;
                        case 2:
                            imprimirFunc(tabela150, MAX150);
                            break;
                        case 3:
                            printf("Numero total de colisoes: %d\n",
numColisoes150);
                            break;
                        default:
                            break;
                        }
                    }
                    numColisoes150 = 0;
                    countTabela150 = 0;
                    op = -1;
                    break;
                default:
```

```
                    break;
                }
            }

        default:
            break;
        }
    }

    free(tabela101);
    free(tabela150);

    return 0;
}
```

Questão 3.1:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#define MAX101 101
#define MAX150 150

typedef struct func {
    char matricula[7];
} Func;

char* matricula_aleatoria() {
    char* matricula = malloc(7 * sizeof(char));

    for (int i = 0; i < 6; i++) {
        matricula[i] = '0' + rand() % 10;
    }
    matricula[6] = '\0';

    return matricula;
}

void inicializarTabela(Func *f, int MAX) {
    for (int i = 0; i < MAX; i++) {
        strcpy(f[i].matricula, "\0");
    }
}

int colisaoRotacao(int chave, char primeiroDigito, int MAX) {
    return (chave + (primeiroDigito - '0')) % MAX;
}
```

```c
int funcaoHashStringRotacao(char *str) {
    char chave[7];
    sprintf(chave, "%c%c%s", str[4], str[5], str);
    int d2 = chave[1] - '0';
    int d4 = chave[3] - '0';
    int d6 = chave[5] - '0';
    int valor_hash = d2 * 100 + d4 * 10 + d6;
    return valor_hash % MAX101;
}

void inserirTabelaRotacao(Func *tabela, int *numColisoes, int *countTabela, int
MAX, int num) {
    Func f;
    snprintf(f.matricula, sizeof(f.matricula), "%d", num);
    int indice = funcaoHashStringRotacao(f.matricula);

    if ((*countTabela) < MAX) {
        if (tabela[indice].matricula[0] != '\0') {
            (*numColisoes)++;
            int novo = colisaoRotacao(indice, f.matricula[0], MAX);

            while (tabela[novo].matricula[0] != '\0') {
                (*numColisoes)++;
                novo = colisaoRotacao(novo, f.matricula[0], MAX);
            }
            tabela[novo] = f;
            (*countTabela)++;
        } else {
            (*countTabela)++;
            tabela[indice] = f;
        }
    } else {
        (*countTabela)++;
        tabela[indice] = f;
    }
}

int colisaoFoldShift(int chave, int MAX) {
    return (chave + 7) % MAX;
}

int funcaoHashStringFoldShift(char *str) {
    int tam = strlen(str);
    unsigned int hash = 0;

    unsigned int aux1 = 0;
    unsigned int aux2 = 0;

    for (int i = 0; i < tam; i++) {
        if (i == 0 || i == 2 || i == 5)
            aux1 = aux1 * 10 + (str[i] - '0');
```

```
        else
            aux2 = aux2 * 10 + (str[i] - '0');
    }

    hash = (aux1 + aux2) % MAX101;

    return hash;
}

void inserirTabelaFoldShift(Func *tabela, int *numColisoes, int *countTabela, int
MAX, int num) {
    Func f;
    snprintf(f.matricula, sizeof(f.matricula), "%d", num);
    int indice = funcaoHashStringFoldShift(f.matricula);

    if ((*countTabela) < MAX) {
        if (tabela[indice].matricula[0] != '\0') {
            (*numColisoes)++;
            int novo = colisaoFoldShift(indice, MAX);

            while (tabela[novo].matricula[0] != '\0') {
                (*numColisoes)++;
                novo = colisaoFoldShift(novo, MAX);
            }
            tabela[novo] = f;
            (*countTabela)++;
        } else {
            (*countTabela)++;
            tabela[indice] = f;
        }
    } else {
        (*countTabela)++;
        tabela[indice] = f;
    }
}

void imprimirTabela(Func *tabela, int MAX, const char *tableLabel) {
    printf("\n%s:\n", tableLabel);
    for (int i = 0; i < MAX; i++) {
        printf("%d: %s\n", i, tabela[i].matricula);
    }
}

int main() {
    Func *tabela101 = (Func *)malloc(MAX101 * sizeof(Func));
    Func *tabela150 = (Func *)malloc(MAX150 * sizeof(Func));

    int numColisoesRotacao101 = 0;
    int numColisoesFoldShift101 = 0;
    int numColisoesRotacao150 = 0;
    int numColisoesFoldShift150 = 0;
```

```c
    int countTabelaRotacao101 = 0;
    int countTabelaRotacao150 = 0;
    int countTabelaFoldShift101 = 0;
    int countTabelaFoldShift150 = 0;

    srand(time(NULL));

    inicializarTabela(tabela101, MAX101);
    inicializarTabela(tabela150, MAX150);

    int i = 0;

    while (i < 1000) {
        int num = rand() % 900000 + 100000;
        inserirTabelaRotacao(tabela101, &numColisoesRotacao101,
&countTabelaRotacao101, MAX101, num);
        inserirTabelaRotacao(tabela150, &numColisoesRotacao150,
&countTabelaRotacao150, MAX150, num);
        i++;
        // printf("\n\n\ni : %d\n\n\n", i);
    }

    imprimirTabela(tabela101, MAX101, "Table 101 (Rotacao)");
    imprimirTabela(tabela150, MAX150, "Table 150 (Rotacao)");
    inicializarTabela(tabela101, MAX101);
    inicializarTabela(tabela150, MAX150);
    i = 0;
    while (i < 1000) {
        int num = rand() % 900000 + 100000;
        inserirTabelaFoldShift(tabela101, &numColisoesFoldShift101,
&countTabelaFoldShift101, MAX101, num);
        inserirTabelaFoldShift(tabela150, &numColisoesFoldShift150,
&countTabelaFoldShift150, MAX150, num);
        i++;
        // printf("\n\n\naaaa : %d\n\n\n", i);
    }

    imprimirTabela(tabela101, MAX101, "Table 101 (FoldShift)");
    imprimirTabela(tabela150, MAX150, "Table 150 (FoldShift)");

    printf("\nColisoes Rotacao 101: %d || Num inseridos: %d\n",
numColisoesRotacao101, countTabelaRotacao101);
    printf("Colisoes Fold Shift 101: %d || Num inseridos: %d\n",
numColisoesFoldShift101, countTabelaFoldShift101);
    printf("Colisoes Rotacao 150: %d || Num inseridos: %d\n",
numColisoesRotacao150, countTabelaRotacao150);
    printf("Colisoes Fold Shift 150: %d || Num inseridos: %d\n\n",
numColisoesFoldShift150, countTabelaFoldShift150);

    free(tabela101);
    free(tabela150);
```

```
    return 0;
}
```