
UNIVERSIDADE FEDERAL DE MATO GROSSO (UFMT)
INSTITUTO DE CIÊNCIAS EXATAS DA TERRA (ICET)
ESTRUTURA DE DADOS L

JOÃO PAULO ALVES CAMPOS

# ANALISE DO DATASET SOBRE O COVID-19

BARRA DO GARÇAS - MT
2025

UNIVERSIDADE FEDERAL DE MATO GROSSO (UFMT)
JOÃO PAULO ALVES CAMPOS

# ANALISE DO DATASET SOBRE O COVID-19

Trabalho apresentado como requisito à obtenção de uma das notas parciais da disciplina de Estrutura de dados l do curso de Ciência da Computação da Universidade Federal de Mato Grosso.

Ivairton M. dos Santos

BARRA DO GARÇAS - MT
2025

# Sumário

# 1 Introdução

O presente relatório descreve o desenvolvimento de um sistema de análise de dados relacionados à pandemia de COVID-19, utilizando a linguagem de programação C. O sistema foi projetado com o objetivo de carregar, organizar e exibir informações do dataset de COVID-19 de maneira eficiente, permitindo ao usuário visualizar dados de diferentes países ou regiões, com foco no número de casos, mortes e recuperados.

A estrutura do sistema é baseada em duas estruturas de dados principais: listas encadeadas e heaps. As listas encadeadas são usadas para armazenar os dados extraídos do arquivo CSV, garantindo que a alocação de memória seja dinâmica e eficiente. Já os heaps são empregados para organizar esses dados de acordo com uma chave específica, que pode ser o número de casos, mortes ou recuperados, com o objetivo de fornecer uma visualização ordenada dos registros mais relevantes.

O processo de carregamento dos dados foi desenvolvido para ler um arquivo CSV contendo informações sobre o COVID-19 e armazenar cada linha em uma lista encadeada, permitindo a fácil manipulação e acesso aos dados. Após a carga dos dados, o sistema oferece ao usuário um menu de opções para escolher qual tipo de informação deseja visualizar. Com base na escolha, os dados são organizados em um heap, que permite a exibição dos registros de maneira eficiente, priorizando os valores mais significativos de acordo com a chave selecionada.

A modularização do código foi uma prioridade, com funções específicas para manipulação de memória, ordenação de dados e exibição das informações, a fim de garantir que o sistema fosse tanto eficiente quanto fácil de manter

A seguir, serão discutidos os detalhes do código e os resultados esperados com base na implementação das funcionalidades descritas.

# 2 Objetivos

Desenvolver um sistema em C para carregar, manipular e exibir dados de COVID-19 utilizando listas encadeadas e heaps. O sistema permite ao usuário visualizar informações sobre o número de casos, mortes ou recuperados de acordo com a chave selecionada.

# 3 Estrutura de dados

O sistema foi projetado com o uso de três estruturas principais para armazenar e manipular os dados de COVID-19:

## 3.1 Lista Encadeada:

Utilizada para armazenar as informações de cada país. Cada nó da lista contém os dados de COVID-19 como número de casos, mortes, recuperados e taxas relacionadas. Isso permite que os dados sejam armazenados dinamicamente, facilitando a inserção e remoção de registros.

## 3.2 Heap (Máximo):

Uma estrutura de dados que garante que o maior valor (de acordo com a chave selecionada) esteja sempre na raiz. O heap foi implementado para ordenar os dados conforme a chave escolhida pelo usuário (casos, mortes ou recuperados), permitindo uma visualização eficiente dos dados mais relevantes.

## 3.3 Estrutura data:

Contém os dados de cada país ou região, como nome do país, número de casos, mortes, recuperados, e outras métricas associadas, como taxas de mortes e recuperação por 100 casos e 100 recuperados.

# 4 Carregamento de dados

O arquivo CSV contendo os dados de COVID-19 foi carregado através da função carregarcsv. Esta função abre o arquivo, lê cada linha e divide os dados usando a função strtok. Cada linha do arquivo é processada e seus dados são armazenados em uma lista encadeada. A função duplicar_string foi utilizada para alocar memória para as strings de forma segura, evitando vazamentos de memória.

```
lista* carregarcsv(const char* csv_file) {
    FILE* arquivocsv = fopen(csv_file, "r");
    char linha[TAMANHO_LINHA];
    lista* cabeca = NULL;
    lista* atual = NULL;

    // Leitura e processamento dos dados do arquivo CSV
    while (fgets(linha, sizeof(linha), arquivocsv)) {
        lista* novo = aloca_no();
        char* token;
        // Processamento dos campos do CSV
        token = strtok(linha, ",");
        novo->info.pais = duplicar_string(token);
        // Continuacao do processamento dos dados...
    }
    fclose(arquivocsv);
    return cabeca;
}
```

A função duplicar_string aloca memória para armazenar as strings de forma segura:

```
char* duplicar_string(const char* str) {
    size_t len = strlen(str) + 1;
    char* nova_str = (char*) malloc(len);
    return memcpy(nova_str, str, len);
}
```

# 5 Manipulação de dados com o Heap

O sistema utiliza um heap para ordenar os dados de acordo com o tipo escolhido pelo usuário (casos, mortes ou recuperados). A função comparar é responsável por comparar os valores com base no tipo selecionado. O heap é criado com a função criar_heap e os elementos são inseridos no heap com a função inserir_heap. O heap é mantido em ordem através da função heapify.

```
int comparar(data a, data b, int tipo) {
    // Compara os dados de acordo com o tipo: casos, mortes ou
       recuperados
    switch(tipo) {
        case 1: return a.casos < b.casos ? -1 : (a.casos > b.casos ?
           1 : 0);
        case 2: return a.mortes < b.mortes ? -1 : (a.mortes > b.
           mortes ? 1 : 0);
        case 3: return a.recuperados < b.recuperados ? -1 : (a.
           recuperados > b.recuperados ? 1 : 0);
        default: return 0;
    }
}
```

A função inserir_heap insere um novo elemento no heap e realiza o ajuste necessário para manter a propriedade do heap:

```
void inserir_heap(heap* h, data elemento) {
    if (h->tamanho >= h->capacidade) {
        h->capacidade *= 2;
        h->elementos = (data*) realloc(h->elementos, h->capacidade *
           sizeof(data));
    }
    h->elementos[h->tamanho] = elemento;
    // Ajuste para manter a propriedade de heap (up-heapify)
}
```

A função extrair_raiz extrai o maior valor (raiz do heap) para exibição:

```
data extrair_raiz(heap* h) {
    data raiz = h->elementos[0];
    h->elementos[0] = h->elementos[h->tamanho - 1];
    h->tamanho--;
    heapify(h, 0); // Ajusta o heap apos a remocao da raiz
    return raiz;
}
```

# 6 Exibição dos Dados

Após construir o heap, o sistema exibe os registros conforme a escolha do usuário. A função exibir_dados imprime as informações relevantes de cada país, como o número de casos, mortes e recuperados, e as taxas associadas, com base na chave escolhida.

```
void exibir_dados(data d, int tipo) {
    printf("\nPais/Regiao: %%s\n", d.pais);
    if (tipo == 1) printf("Casos: %d (Selecionado)\n", d.casos);
    else if (tipo == 2) printf("Mortes: %d (Selecionado)\n", d.mortes
       );
    else if (tipo == 3) printf("Recuperados: %d (Selecionado)\n", d.
       recuperados);
}
```

# 7 Interface de Usuário

A interface do usuário é interativa e simples. O sistema apresenta um menu com as opções de visualização de dados (casos, mortes ou recuperados) e permite ao usuário escolher quantos registros deseja ver. A cada escolha, o sistema exibe os dados mais relevantes utilizando o heap.

```
while (1) {
    printf("1. Visualizar numero de casos\n");
    printf("2. Visualizar numero de mortes\n");
    printf("3. Visualizar numero de recuperados\n");
    printf("4. Sair\n");
    scanf("%d", &opcao);

    // Processamento da escolha do usuario
}
```

# 8 Gerenciamento de Memória

O código é projetado para evitar vazamentos de memória. Funções como liberar_lista e liberar_data garantem que toda a memória alocada seja liberada corretamente após o uso.

```
void liberar_lista(lista* l) {
    while (l) {
        lista* temp = l;
        l = l->prox;
        liberar_data(&temp->info); // Libera os dados de cada no
        free(temp); // Libera o no
    }
}
```

Durante o desenvolvimento, utilizei a ferramenta Valgrind para realizar a análise de memória do programa. Os resultados indicaram que não houve vazamentos de memória e que toda a memória alocada foi corretamente liberada.

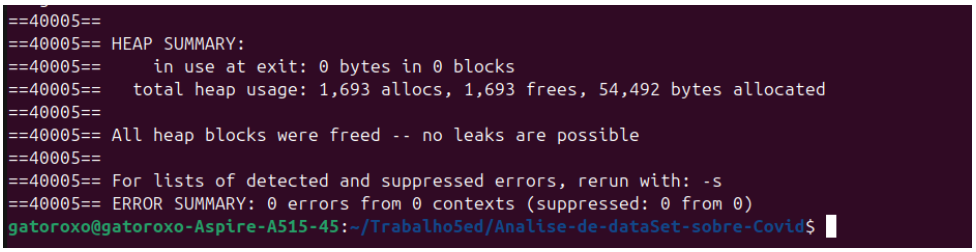


Figura 1: Mensagem após o uso do Valgrind

Os resultados mostraram que o programa não apresentou vazamentos de memória, e o total de memória utilizada foi 54 KBs, o que é considerado um uso eficiente de recursos. Esse cuidado no gerenciamento de memória garante que o sistema seja estável e escalável, evitando problemas de desempenho em caso de grandes volumes de dados.

# 9 Conclusão

O sistema foi desenvolvido com sucesso, utilizando as estruturas de dados lista e heap para armazenar, organizar e exibir os dados de COVID-19 de maneira eficiente. O código é modular e otimizado para oferecer uma interface simples ao usuário, com opções para visualizar dados específicos de maneira ordenada. O uso de listas encadeadas e heaps garante que o sistema seja eficiente, mesmo com grandes volumes de dados.

O código foi estruturado de maneira que facilita a manutenção, com funções bem definidas para cada operação. Além disso, a gestão de memória foi cuidadosamente implementada para evitar vazamentos.

# Referências


- chatgpt.com
- Estudos realizados em sala de aula.