Relatório do projeto

Objetivo do projeto

O objetivo do nosso projeto foi tornar possível uma correlação dos fatores de saúde de cada estado do Brasil durante a pandemia com o número de casos de covid-19 de cada estado. Sendo assim, são analisados: número de casos de covid-19, nível de sedentarismo (o quanto de exercícios físicos foi realizado), qualidade do sono, qualidade da alimentação e saúde mental de determinado estado do Brasil.

Os dados vêm de um arquivo csv (arquivo de texto que representa uma tabela).
Visto que ler dados de um arquivo csv se torna bastante difícil, objetivamos construir funções de manipulação destes dados do arquivo csv, para que assim a correlação dos dados seja feita de uma melhor forma.
O objetivo é que o usuário visualize melhor as informações contidas no arquivo csv. E assim, possa realizar operações de: ordenação (da forma como o usuário preferir), busca de determinados dados neste arquivo e através desta ordenação e busca de dados ser possível gerar um novo arquivo csv para o resultados retornados.

O time

O time é composto por dois alunos da disciplina Algoritmo e estrutura de dados:

- Daniel Alencar Penha Carvalho
- Talissom Cardoso

A ideia

Inicialmente, tivemos que discutir bem a ideia do projeto, para isto utilizamos o site **whimsical**, no qual foi possível realizar tabelas, artes e outras ferramentas para tornar a ideia do projeto a mais clara possível. Segue o link da ideia do projeto:
https://whimsical.com/covid-x9-PfwdUHtnwDLUUJwHAKQGX3@VsSo8s35VZFteL4SKHjhEQ

TAD CSV

Os dados viriam em formato csv (arquivo de texto), para colocarmos estas informações dentro do código criamos um TAD bem semelhante ao TAD LISTA (definido na aula 6) o qual contempla a seguinte configuração:

```
typedef struct{
   int row;
   int column;
   char ***array;
} CSV;
```

```

```

Ou seja:

- row: indica o número de linhas que o arquivo csv possui
- column: indica o número de colunas que o arquivo csv possui
- array: é uma matriz de strings que serve para armazenar o conteúdo do arquivo csv

As partes relacionadas somente a manipulação do TAD CSV foram feitas dentro da pasta csv lib. Para a manipulação foi feito algumas funções.

Funções do TAD CSV

```
void setRowsAndColumns(FILE *file, CSV *csv);
```

Como o nome sugere, é pego do arquivo csv a quantidade de linhas e colunas das informações que o contém e passado estes valores para as variáveis row e column do TAD CSV respectivamente.

```
void readCSV(FILE *file, CSV *csv);
```

É feita a leitura do arquivo csv e passado as informações para array (em forma de matriz de strings). Todas as informações do arquivo ficam disponíveis no TAD CSV.

```
void printTitle(CSV csv);
```

Exibe somente a primeira linha de nossa matriz de strings em array, a intenção é que os nomes das colunas de nossa tabela possam ser enxergados a partir desta função.

```
void printCSV(CSV csv);
```

A função exibe no console todas as informações contidas no csv em forma de tabela.

TAD BINARY_TREE

Como mencionado, o objetivo é que o usuário visualize melhor as informações contidas no arquivo csv (pois a leitura de dados em um arquivo csv é difícil).

Sendo assim, acabamos optando por usar outro Tipo Abstrato de Dados para funções como ordenação e busca de informações específicas, por exemplo. Para isto, decidimos usar:
TAD BINARY_TREE: nada mais do que a árvore binária de busca.

```
typedef struct {
   int sedentarismo;
   int qualidadeDoSono;
   int qualidadeDaAlimentacao;
   int estadoPsicologico;
} HEALTH_INFORMATION;

typedef struct {
   char estado[30];
   int casos;
   HEALTH_INFORMATION saude;
} VALUES;

typedef struct node {
   struct node *left;
   struct node *right;
   struct node *father;
   VALUES informations;
} NODE;

typedef NODE *BINARY_TREE;
```

É basicamente o mesmo tipo de estrutura apresentado nas aulas, exceto por:

1. Estrutura HEALTH_INFORMATION para guardar informações referentes a saúde:
    - sedentarismo: int (representando a porcentagem) referente ao nível de atividade física de determinado estado do Brasil.
    - qualidadeDoSono: (representando a porcentagem) referente a qualidade do sono de determinado estado do Brasil.
    - qualidadeDaAlimentacao: (representando a porcentagem) referente a qualidade da alimentação de determinado estado do Brasil.
    - estadoPsicologico: int (representando a porcentagem) referente a saúde mental de determinado estado do Brasil.

2. Estrutura VALUES para guardar informações gerais do estado especificado:
    - estado: string para armazenar o nome do estado.
    - casos: int informando a quantidade de casos de covid-19 que o estado possuía quando foram pegos os dados.
    - saude: HEALTH_INFORMATION para informações gerais de saúde.

3. Finalmente a estrutura NODE que além de conter os ponteiros para NODE: right, left e father. Também contém:
    - informations: VALUES contendo informações gerais do estado especificado.
4. E finalmente definimos a BINARY_TREE que nada mais é do que um ponteiro para NODE.

Funções do TAD BINARY_TREE

```
void makeTree(BINARY_TREE *tree, VALUES values);
```

Basicamente cria o nó raiz na nossa árvore tree, colocando values neste novo nó.

```
void setLeft(BINARY_TREE tree, VALUES values);
void setRight(BINARY_TREE tree, VALUES values);
```

setLeft e setRight armazenam na árvore o filho a esquerda e o filho a direita respectivamente.

```
VALUES valueOfNode(BINARY_TREE tree);
```

Retorna a variável informations (do tipo VALUES) do NODE que tree está apontando.

```
BINARY_TREE left(BINARY_TREE tree);
BINARY_TREE right(BINARY_TREE tree);
BINARY_TREE father(BINARY_TREE tree);
BINARY_TREE brother(BINARY_TREE tree);
```

Retornam respectivamente a referência para: o filho esquerdo, o filho direito, o pai e o irmão do NODE que tree está apontando.
O irmão se refere ao outro filho do seu pai.

```
int isLeft(BINARY_TREE tree);
int isRight(BINARY_TREE tree);
```

Verificam respectivamente se o NODE em que tree faz referência é o filho esquerdo e se é o filho direito (retornado 1 caso seja e retornando 0 caso não).

```
void insertElement(BINARY_TREE *tree, VALUES values, int organizate);
```

Basicamente, insere um novo nó na nossa árvore, colocando values neste novo nó.
O novo aqui é o parâmetro organizate, que basicamente nos fala como devemos organizar os dados em nossa árvore binária:

- organizate == 1: os dados devem ser organizados de forma a facilitar a busca e a ordenação do número de casos de cada estado.
- organizate == 2: os dados devem ser organizados de forma a facilitar a busca e a ordenação dos níveis de atividade física de cada estado.
- organizate == 3: os dados devem ser organizados de forma a facilitar a busca e a ordenação da qualidade do sono de cada estado.
- organizate == 4: os dados devem ser organizados de forma a facilitar a busca e a ordenação da qualidade da alimentação de cada estado.
- organizate == 5: os dados devem ser organizados de forma a facilitar a busca e a ordenação dos índices de saúde mental de cada estado.

```
// Raiz - Esquerda - Direita
void preOrderRoute(BINARY_TREE tree);
// Esquerda - Direita - Raíz
void postOrderRoute(BINARY_TREE tree);
// Esquerda - Raiz - Direita
void ascendingOrder(BINARY_TREE tree);
// Direita - Raiz - Esquerda
void descendingOrder(BINARY_TREE tree);
```

O comentário acima do protótipo nos dá uma ideia do que cada uma realiza:

- preOrderRoute(): Percorre os nós da árvore em pré-ordem e exibe na tela os dados na ordem em que foram percorridos.
- postOrderRoute(): Percorre os nós da árvore em pós-ordem e exibe na tela os dados na ordem em que foram percorridos.
- ascendingOrder(): Percorre os nós da árvore em in-ordem e exibe na tela os dados na ordem em que foram percorridos.
- descendingOrder(): Percorre os nós da árvore (sempre na ordem: Direita -> Raiz -> Esquerda) e exibe na tela os dados na ordem em que foram percorridos.

```
```

```
void generateTree(BINARY_TREE *tree, CSV *csv, int organizate);
```

Basicamente gera uma árvore binária a partir dos dados extraídos do arquivo csv em que os nós serão colocados da forma como organizate está configurado.

```
VALUES *fromCSVToValue(CSV *csv, int line);
```

Transforma os dados de alguma linha do csv para o tipo VALUES.

```
int putOnTheSide(VALUES values1, VALUES values2, int organizate);
```

É a função que nos diz em qual lado devemos colocar o nó na árvore (do lado esquerdo ou do lado direito):
Compara values1 com values2 de acordo com o valor de organizate:

- organizate == 1: compara o número de casos contidos em values1 e values2.
- organizate == 2: compara o valor da variável sedentarismo contidos em values1 e values2.
- organizate == 3: compara o valor da variável qualidadeDoSono contidos em values1 e values2.
- organizate == 4: compara o valor da variável qualidadeDaAlimentacao contidos em values1 e values2.
- organizate == 5: compara o valor da variável estadoPsicologico contidos em values1 e values2.

```
FILE *generateFromTreeToCSVFile(BINARY_TREE *tree, char *nameOfFile, int
order, CSV csv);
```

Como o nome sugere, gera um novo arquivo csv a partir de uma árvore binária:

- tree: referência para a árvore binária.
- nameOfFile: string que nos diz o nome do arquivo csv que será gerado.
- order: int que nos diz como o arquivo será organizado:
    - Order == 1: dados organizados através do percurso Esquerda - Raiz - Direita.
    - Order == 2: dados organizados através do percurso Direita - Raiz - Esquerda.
- csv: Para pegarmos as informações de título de cada coluna da tabela (que somente estão no csv).

```
void printInFile(BINARY_TREE tree, FILE *file, int order);
```

Função auxiliar da função generateFromTreeToCSVFile().

Ela deve colocar os dados presentes na árvore binária tree no arquivo csv apontado por file

- tree: referência para a árvore binária.
- file: FILE (referência para um arquivo) que aponta para o arquivo em que os dados devem ser escritos.
- order: int que nos diz como o arquivo será organizado:
    - Order == 1: dados organizados através do percurso Esquerda - Raiz - Direita.
    - Order == 2: dados organizados através do percurso Direita - Raiz - Esquerda.

```
FILE *generateFromCsvToCSVFile(CSV csv, char *nameOfFile);
```

Faz algo bem parecido com a função generateFromTreeToCsvFile(), exceto que ao invés de pegar os dados da árvore binária, é gerado o arquivo csv a partir dos dados do csv (TAD CSV).

```
BINARY_TREE *buscaNaArvore(BINARY_TREE *tree, int quantidade, int
escolha);
```

Como o nome sugere, faz uma busca na árvore, no qual:

- tree: é uma referência para a árvore que vamos buscar os dados.
- escolha: int que nos diz qual o campo que se deve fazer a busca:
    - Escolha == 1: realizar a busca pelo número de casos.
    - Escolha == 2: realizar a busca pelo nível de sedentarismo.
    - Escolha == 3: realizar a busca pela qualidade do sono.
    - Escolha == 4: realizar a busca pela qualidade da alimentação.
    - Escolha == 5: realizar a busca pela saúde mental.
- quantidade: int que nos diz a quantidade que deve ser buscada na árvore a partir do campo especificado pela variável escolha.

Bibliotecas necessárias

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

Bibliotecas necessárias para trabalhar com as funções adequadamente (estas bibliotecas estão importadas em um arquivo separado chamado common.h).

Algumas configurações

```
#define NONE 0
#define CASOS 1
#define SEDENTARISMO 2
#define SONO 3
#define ALIMENTACAO 4
#define PSICOLOGICO 5
```

Foi definido porque estávamos usando muito os números 0,1,2,3,4 e 5 em nosso programa (representando o modo de como os dados serão ordenados).

O programa

O programa completo está na main() do arquivo Árvore binária.c

```
  FILE *file;
  CSV csv;
  BINARY_TREE tree;
  BINARY_TREE *busca;
  char optionUser[30];
  int choice;
  int ordenacao;
  int teveABusca = NONE;
  int treeOrdering = NONE;
```

Algumas variáveis (que serão necessárias) dentro da main().

```
  file = fopen("amostra de dados.csv", "r");
  setRowsAndColumns(file, &csv);
  readCSV(file, &csv);
  printCSV(csv);
```

```
   fclose(file);
   printf("\n(Digite help para saber mais)\n");
```

- Nesta parte do programa é aberto o arquivo ‘amostra de dados.csv’ (para leitura) que está no mesmo diretório que o nosso programa.
- Dentro do csv é colocado a quantidade de linhas e colunas que o arquivo csv possui
- É feita a leitura de todos os dados do arquivo ‘amostra de dados.csv’ e colocado estes dados em nosso csv (do TAD CSV)
- É feita a exibição (em forma de tabela) de todos os dados contidos no csv.
- Fechado o arquivo ‘amostra de dados.csv’
- Exibido na tela a mensagem de sugestão ao usuário

```
  do {
       printf(">>> ");
       setbuf(stdin, NULL);
       scanf("%[^\n]s", optionUser);
```

- Automaticamente entra-se em um do...while().
- Pede-se ao usuário que digite uma string. Dependendo da string que o usuário digite, será executado determinadas funções para o usuário.

```
       if(strcmp(optionUser, "ordenate") == 0) {

           printf("Ordernar por...\n[ 1 ] Quantidade de casos\n[ 2 ] Nível
de sedentarismo\n[ 3 ] Qualidade do sono\n[ 4 ] Qualidade da
alimentação\n[ 5 ] Qualidade psicológica\n");
           printf("Escolha uma opção: ");
           scanf("%d", &choice);

           if(treeOrdering != choice) {
               generateTree(&tree, &csv, choice);
               treeOrdering = choice;
           }
           teveABusca = 0;

           printf("Tipo de ordenação...\n[ 1 ] Crescente\n[ 2 ]
Decrescente\nEscolha a opção: ");
           scanf("%d", &ordenacao);
```

```
            printTitle(csv);
            if(ordenacao == 1) {
                ascendingOrder(tree);
            } else {
                descendingOrder(tree);
            }
        }
```

- Se o que o usuário digitou foi a palavra ‘ordenate’ automaticamente entrará dentro da opção de ordenação dos dados da tabela exibida anteriormente.
- O usuário pode escolher qual coluna da tabela que irá ser ordenada e qual tipo de ordenação dos dados (crescente ou decrescente).
- A partir da escolha, será gerada uma árvore de acordo com as especificações e automaticamente a tabela é exibida na tela com as mudanças.

```
        else if(strcmp(optionUser, "generate csv file") == 0) {
            char nameOfFile[32];
            printf(">>> Nome do arquivo: ");
            setbuf(stdin, NULL);
            scanf("%31[^\n]s", nameOfFile);

            if(teveABusca) {
                if(generateFromTreeToCSVFile(busca, nameOfFile, 1, csv)) {
                    printf("File created successfully!!!\n");
                }
            } else {
                if(treeOrdering > 0) {
                    if(generateFromTreeToCSVFile(&tree, nameOfFile,
ordenacao, csv)) {
                        printf("File created successfully!!!\n");
                    }
                } else {
                    if(generateFromCsvToCSVFile(csv, nameOfFile)) {
                        printf("File created successfully!!!\n");
                    }
                }
            }
        }
```

- Se o que o usuário digitou foi a frase ‘generate csv file’ automaticamente entrará dentro da opção de gerar novo arquivo csv (o qual gerará o novo arquivo a partir da última tabela exibida no programa).
- É pedido ao usuário que digite o nome do novo arquivo. Este nome não pode ultrapassar 30 caracteres (se ultrapassar, o arquivo não é gerado).

```
        else if(strcmp(optionUser, "search") == 0) {
            int quantidade;

            printf("Procurar com base em...\n[ 1 ] Quantidade de casos\n[ 2
] Nível de sedentarismo\n[ 3 ] Qualidade do sono\n[ 4 ] Qualidade da
alimentação\n[ 5 ] Qualidade psicológica\n");

            printf("Escolha uma opção: ");
            scanf("%d", &choice);

            if(treeOrdering != choice) {
                generateTree(&tree, &csv, choice);
                treeOrdering = choice;
            }
            printf("Quantidade: ");
            scanf("%d", &quantidade);

            busca = buscaNaArvore(&tree, quantidade, choice);
            teveABusca = 1;

            printTitle(csv);
            ascendingOrder(*busca);

        }
```

- Se o que o usuário digitou foi a palavra ‘search’ automaticamente entrará dentro da opção de busca de dados.

```
        else if(strcmp(optionUser, "help") == 0) {
            printf("\nFunções:\n- ordenate (faz a ordenação crescente ou
decrescente dos dados a partir de um campo especificado)\n- generate csv
file (gera arquivo csv a partir da última tabela amostrada no programa)\n-
search (faz a busca de um determinado valor dentro dos dados a partir de
um campo especificado)\n- exit (finaliza o programa)\n- help (amostra
todas as funcionalidades do programa)\n\n");
```

```
        }
```

- Se o que o usuário digitou foi a palavra ‘help’ automaticamente será exibido na tela todas as funcionalidades do programa para o usuário

```
} while(strcmp(optionUser, "exit"));
```

- Final do do...while().
- Basicamente, se o usuário digitar a palavra ‘exit’ o programa se encerra.

Contribuição da dupla

Para isto, podemos dividir as contribuições em algumas partes:

- A ideia do projeto
- A criação das funções
- Testes realizados no programa
- Criação do relatório

A ideia do projeto

Para encontrar a ideia do projeto debatemos bastante nesta parte. Utilizamos de chamadas pelo discord e criação da ideia pelo site **Whimsical**.
Cada um mostrou suas ideias e fez sua parte.

A criação das funções

```
void setRowsAndColumns(FILE *file, CSV *csv);
```

Esta função foi feita por ambos.

```
void readCSV(FILE *file, CSV *csv);
```

Esta função foi feita por ambos.

```
void printCSV(CSV csv);
```

Esta função foi feita por ambos. Porém Talissom contribuiu mais.

```
void printTitle(CSV csv);
```

Esta função foi feita por ambos. Porém Talissom contribuiu mais.

```
void makeTree(BINARY_TREE *tree, VALUES values);
```

Esta função foi feita por Daniel.

```
void setLeft(BINARY_TREE tree, VALUES values);
```

Esta função foi feita por Daniel.

```
void setRight(BINARY_TREE tree, VALUES values);
```

Esta função foi feita por Daniel.

```
VALUES valueOfNode(BINARY_TREE tree);
```

Esta função foi feita por Daniel.

```
BINARY_TREE left(BINARY_TREE tree);
```

Esta função foi feita por Daniel.

```
BINARY_TREE right(BINARY_TREE tree);
```

Esta função foi feita por Daniel.

```
BINARY_TREE father(BINARY_TREE tree);
```

Esta função foi feita por Daniel.

```
BINARY_TREE brother(BINARY_TREE tree);
```

Esta função foi feita por Daniel.

```
int isLeft(BINARY_TREE tree);
```

Esta função foi feita por Daniel.

```
int isRight(BINARY_TREE tree);
```

Esta função foi feita por Daniel.

```
void insertElement(BINARY_TREE *tree, VALUES values, int organizate);
```

Esta função foi feita por Daniel.

```
// Raiz - Esquerda - Direita
void preOrderRoute(BINARY_TREE tree);
// Esquerda - Direita - Raíz
void postOrderRoute(BINARY_TREE tree);
// Esquerda - Raiz - Direita
void ascendingOrder(BINARY_TREE tree);
// Direita - Raiz - Esquerda
void descendingOrder(BINARY_TREE tree);
```

As funções acima foram feitas por ambos.

```
void generateTree(BINARY_TREE *tree, CSV *csv, int organizate);
```

Esta função foi feita por Daniel.

```
VALUES *fromCSVToValue(CSV *csv, int line);
```

Esta função foi feita por Daniel.

```
int putOnTheSide(VALUES values1, VALUES values2, int organizate);
```

Esta função foi feita por ambos.

```
FILE *generateFromTreeToCSVFile(BINARY_TREE *tree, char *nameOfFile, int order, CSV csv);
```

Esta função foi feita por Talissom.

```
FILE *generateFromCsvToCSVFile(CSV csv, char *nameOfFile);
```

Esta função foi feita por Talissom.

```
void printInFile(BINARY_TREE tree, FILE *file, int order);
```

Esta função foi feita por Talissom.

```
BINARY_TREE *buscaNaArvore(BINARY_TREE *tree, int quantidade, int escolha);
```

Esta função foi feita por Daniel.

Testes realizados no programa
Para a realização de testes no programa (para saber se o mesmo está funcionando bem), usamos de chamada no discord e reuniões no google meet para testarmos e resolvermos os erros. Também, testamos e consertamos erros separadamente.
Utilizamos de um software de versionamento de código (neste caso, o git) para juntarmos os códigos que eram feitos separadamente.

Criação do relatório
Para a realização deste relatório, Daniel Alencar fez todo o relatório.
No dia que está sendo feito (11/12/2020) Talissom Cardoso está doente, por isso não foi possível que o mesmo contribuísse.

Conclusões
Acredito que foi de grande valia tudo que foi possível montar utilizando AED para tornar o código mais eficiente e utilizando menos processamento do computador.