ANÁLISE COMPARATIVA DE DESEMPENHO DA FDE-PRIORIDADE COM E SEM REFERENCIAL MÓVEL.

Conforme o enunciado, para esse exercício foi implementada uma fila **com** referencial móvel a partir de um arquivo de fila **sem** referencial móvel disponível no Moodle da disciplina: arquivo **`FDEdePrioridade_semPontFunc_V1.zip`**. Também foi utilizado o arquivo **`dataset_V1.csv`** para obter 10.000 registros de dados. Cada registro correspondendo a uma linha do arquivo, o valor do atributo "ranking" corresponde à prioridade do registro.

3.1. e 3.2. Implementação da FDE_refMovel de prioridade e do Menu

Para a compilação utilizamos o gcc em um ambiente linux(Arch), o comando utilizado foi: **`gcc main.c FilaDePrioridade.c FilaDePrioridade.h`** e **`a.out dataset_V1.csv`** para poder ler o arquivo. As bibliotecas usadas foram **`<stdio.h>`** e **`<stdlib.h>`**. O arquivo **`FilaDePrioridade.c`** contém as funções tanto com referencial móvel quanto sem referencial móvel.

Inicialmente, é feita a leitura do arquivo em CSV contendo os dados.

No arquivo **`FilaDePrioridade.c`**, o referencial móvel é alocado dinamicamente no código e é feita a inicialização das estruturas de cauda, frente, referencial e tamanho.

```
RefMovel *cria(){
    RefMovel *desc = (RefMovel*) malloc(sizeof(RefMovel));
    if(desc != NULL) {
        desc->cauda = NULL;
        desc->frente = NULL;
        desc->referencial = NULL;
        desc->tam= 0;
    }
    return desc;
}
```

Na Opção 1 do Menu, é possível INSERIR um elemento na fila com base no ranking, os seguintes dados são solicitados:

```
    int matricula;
    int ranking;
    char nome[50];
    char curso[50];
```

A inserção ocorre chamando a função: **`int insere(Pessoa *pessoa, RefMovel *fila, long *cont);`**

Devido ao referencial móvel, a escolha do percurso para inserção depende de uma medida de distância calculada/estimada com base na diferença entre a prioridade do novo item e o item de cauda (ou frente).

Caso 1 - Se **`ranking(novo) ≤ ranking(cauda)`**, então o novo elemento será inserido convencionalmente como nova cauda

Caso 2 - Se **`ranking(frente) > ranking(novo)`**, então o novo elemento será inserido como novo item de frente

Caso 3 - Se **`ranking(cauda) < ranking(novo) ≤ ranking(refMovel)`**, então a posição do novo elemento estará entre cauda e refMovel

Δa = | ranking(cauda) – ranking(novo) |

Δb = | ranking(refMovel) – ranking(novo) |

Se Δa ≤ Δb : localize a posição do novo elemento pela cauda

Caso 4 - Se **`ranking(refMovel) < ranking(novo) ≤ ranking(frente)`**, então a posição do novo elemento estará entre refMovel e frente

Δc = | ranking(frente) – ranking(novo) |

Δd = | ranking(refMovel) – ranking(novo) |

Se Δc ≤ Δd : localize a posição do novo elemento pela frente

Na Opção 2 do Menu, é possível REMOVER um elemento na fila. É feito um teste se a fila está vazia então a remoção do elemento é feita com auxílio de uma variável auxiliar. No final o tamanho é decrementado em 1.

```
Pessoa * remover(RefMovel *desc){
    NoFila *aux = desc->cauda;
    if(testaVazia(desc)){
        Pessoa *p = NULL;
        return p;
    }
    Pessoa *temp = (Pessoa*) malloc(sizeof(Pessoa));
    memcpy(temp, &(desc->frente->pessoa), sizeof(Pessoa));
    if(aux == desc->frente){
        free(desc->frente);
        desc->frente = desc->cauda = desc->referencial = NULL;
    }else{
        desc->frente = desc->frente->menor;
        free(desc->frente->maior);
        desc->frente->maior = NULL;
    }
    desc->tam -= 1;
    return temp;
}
```

Na Opção 3 do Menu, é possível BUSCAR O ELEMENTO DA FRENTE da fila.

```
int buscaFrente(RefMovel *fila, Pessoa *pessoa){
    if(testaVazia(fila)){
        return 0;
    }
    *pessoa = fila->frente->pessoa;
    return 1;
}
```

Na Opção 4 do Menu, é possível BUSCAR O ELEMENTO DA CAUDA da fila.

```
int buscaCauda(RefMovel *fila, Pessoa *pessoa){
    if(testaVazia(fila)){
        return 0;
    }
    *pessoa = fila->cauda->pessoa;
    return 1;
}
```

Na Opção 5 do Menu, é retornado o TAMANHO da fila.

```
int tamanhoFila(RefMovel *fila){
    return fila->tam;
}
```

Na Opção 6 do Menu, a fila é REINICIADA. Primeiro é feito o teste se a fila está vazia, então a fila é reiniciada.

```
int reinicia(RefMovel *fila){
    if(testaVazia(fila)){
        return 0;
    }
    NoFila *atual = fila->frente;
    while (atual != NULL){
        atual = atual->menor;
        free(fila->frente);
        fila->frente = atual;
    }
    fila->cauda = fila->frente=fila->referencial = NULL;
    fila->tam = 0;
    return 1;
}
```

Na Opção 7 do Menu, a fila é DESTRUÍDA. Antes de dar um **`free`** na fila ela é reiniciada.

```
RefMovel *destroi(RefMovel *fila){
    reinicia(fila);
    free(fila);
    return NULL;
}
```

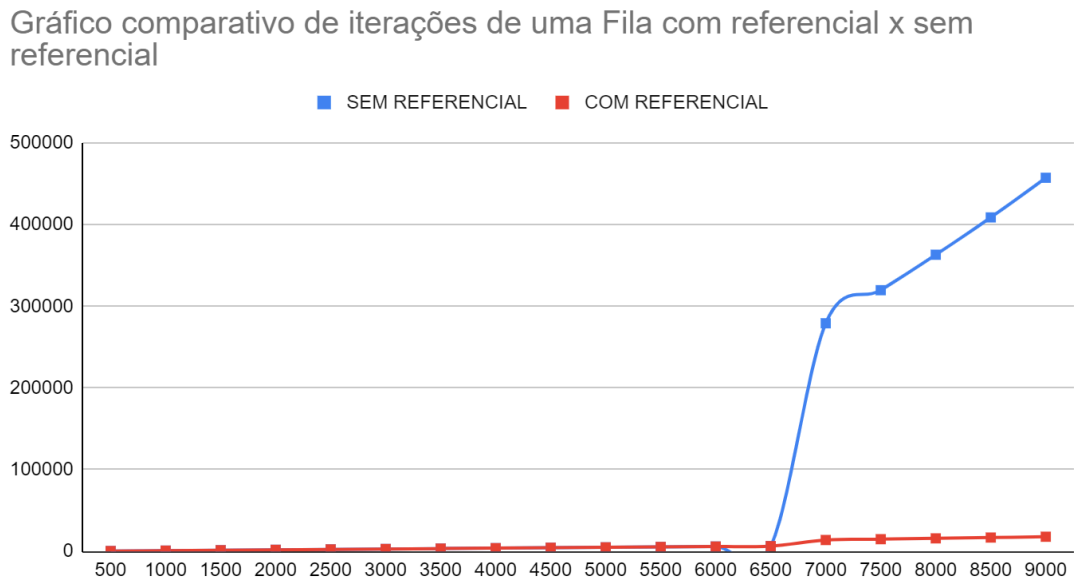
Na Opção 8 do Menu, é feito o teste de desempenho da fila mais detalhado a seguir. Inicialmente é feita a seleção aleatória de um número de linhas

3.3. Teste desempenho - casos 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000

a) Cálculo do número médio de iterações de laço para a inserção de todos os itens na base

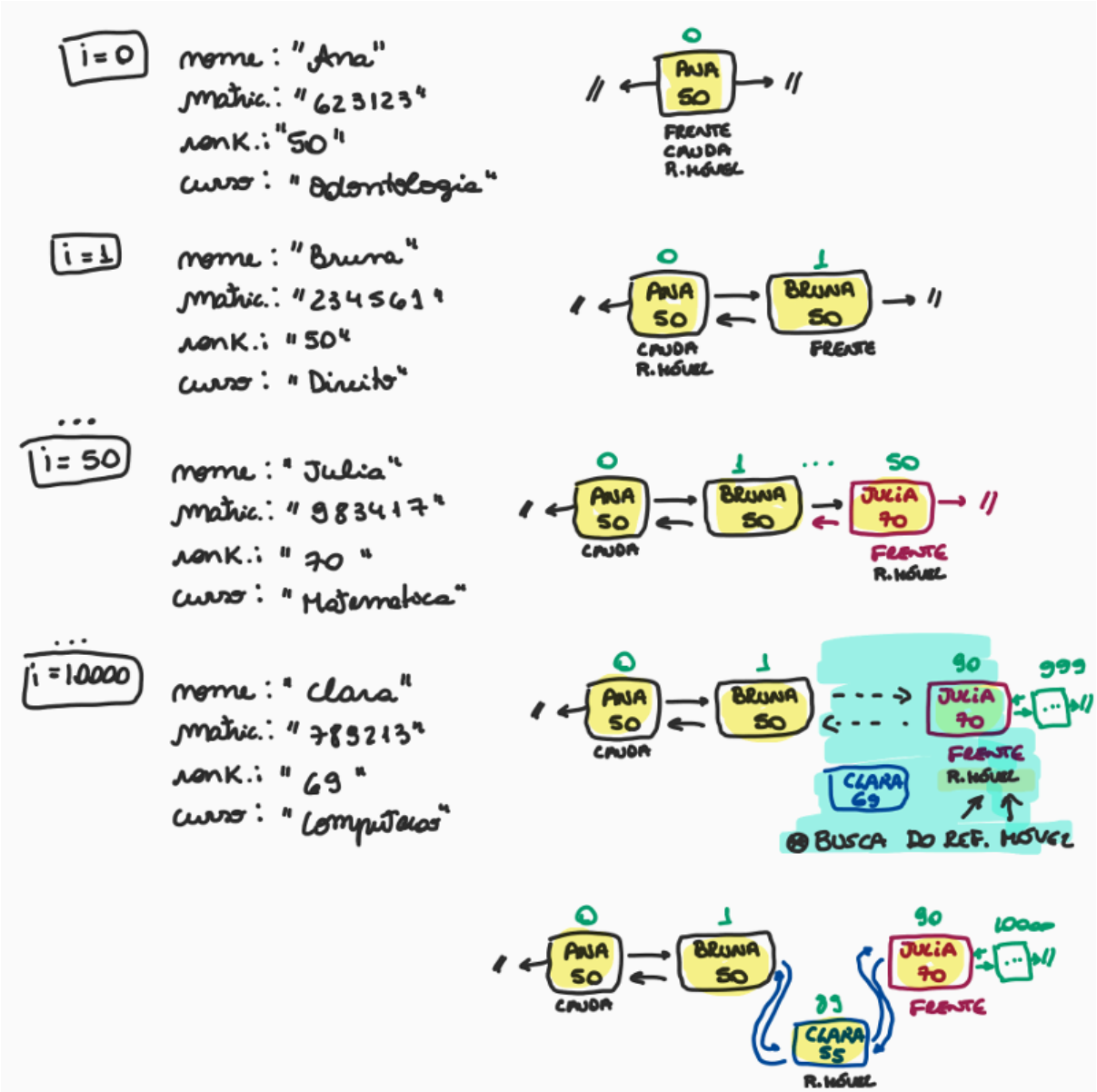
O cálculo do número médio de iterações necessárias para inserir elementos em cada fila e comparar os resultados. O vetor **`numDados`** receberá diferentes tamanhos de entradas para o experimento. A função **`lerCSV`** irá ler um arquivo CSV contendo dados de pessoas, selecionando aleatoriamente uma quantidade especificada de linhas. Então é feita a inserção de pessoas em cada fila (com e sem referencial) e é contado o número de iterações necessárias para cada inserção. A variável **`numRepA`** irá acumular o número de interações para a fila com referencial móvel e a variável **`numRepB`** para a fila sem referencial móvel. As filas são reiniciadas a cada conjunto de inserção para o próximo tamanho de entrada.

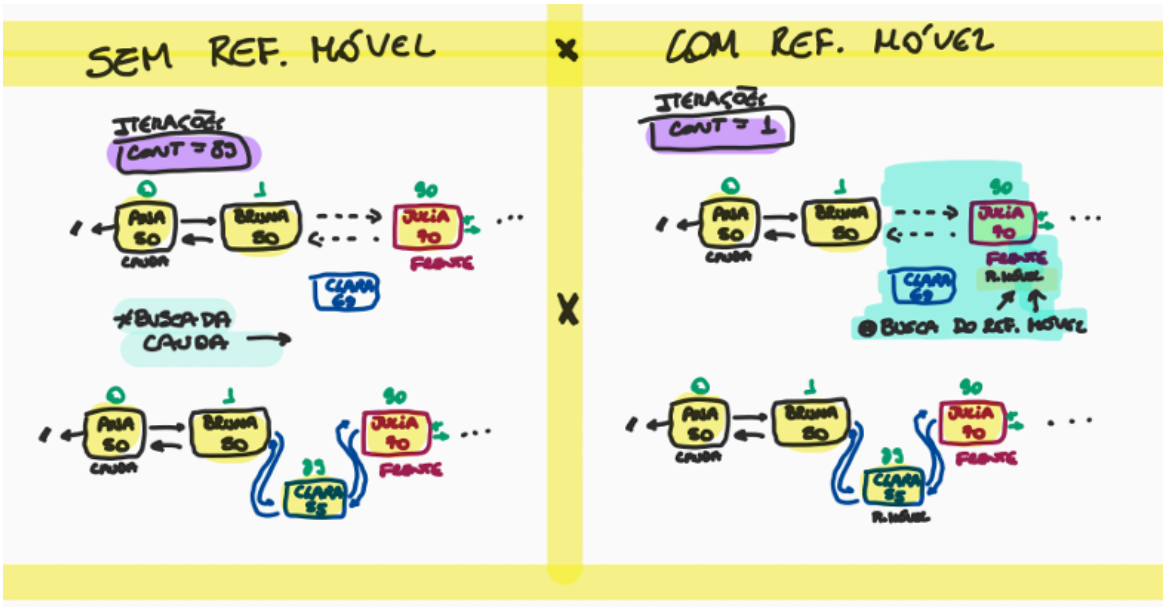
b) Gráficos comparando as duas estratégias de implementação. A partir dos gráficos, qual é a melhor estratégia dentre as duas e porque?



Quando analisamos o gráfico percebemos que as filas são bem parecidas até 6500 interações, após isso vemos que a fila sem o referencial móvel sobe muito a quantidade de interações necessárias, enquanto a fila com o referencial móvel sobe menos. Desta forma concluímos que a fila **com o referencial móvel é a mais eficiente para casos com muitas inserções**.

c) Teste de mesa e vantagem da implementação com referencial móvel sobre o caso convencional


i = 0
nome: "Ana"
matric.: "623123"
rank.: "50"
curso: "Odontologia"
ANA 50
FRENTE CAUDA R. MÓVEL
i = 1
nome: "Bruna"
matric.: "234561"
rank.: "50"
curso: "Direito"
ANA 50
BRUNA 50
CAUDA R. MÓVEL
FRENTE
i = 50
nome: "Julia"
matric.: "983417"
rank.: "70"
curso: "Matemática"
ANA 50
BRUNA 50
JULIA 70
CAUDA
FRENTE R. MÓVEL
i = 10000
nome: "clara"
matric.: "789213"
rank.: "69"
curso: "Computação"
ANA 50
BRUNA 50
JULIA 70
CLARA 69
FRENTE R. MÓVEL
BUSCA DO REF. MÓVEL
CLARA 55
CAUDA
FRENTE
R. MÓVEL


SEM REF. MÓVEL
X
COM REF. MÓVEL
ITERAÇÕES
CONT = 83
CONT = 1
BUSCA DA CAUDA
BUSCA DO REF. MÓVEL
ANA 50
BRUNA 50
JULIA 70
CLARA 69
CLARA 55
CAUDA
FRENTE
R. MÓVEL