Relatório da Tarefa 1B

Alunos: Gustavo de Souza; Felipe Jojima.

Parte 1 da tarefa (análise do arquivo csv utilizando fila de prioridade com e sem ref móvel):

O desenvolvimento foi feito em ambiente Linux utilizando a plataforma Vscode, porém também efetuamos testes utilizando a seguinte linha de comando para o teste do codigo fila.c:

gcc -Wall arq.c fila.c -Larq.h && ./a.out dataset_v1.csv

Antes de partimos para as análises do desempenho de cada pilha e respondermos as perguntas necessárias vamos fazer uma análise das partes principais dos arquivos que desenvolvemos e analisarmos o código testesFuncFDEcomRefMovel.c para compreender melhor o funcionamento do referencial móvel, começando pelo arq.h:

```c
typedef struct infoDados
{
    int matricula;
    char curso[50];
    int ranking;
    char nome[30];
} info;

typedef struct noFila
{
    info dados;
    struct noFila *atras;
    struct noFila *frente;
} noF;

typedef struct descFDEsReferencialMovel
{
    noF *primeiro;
    noF *ultimo;
    int tamInfo;
} descFDEsRef;

typedef struct descFDEcRefMovel{
    noF *primeiro;
    noF *ultimo;
    noF *refMovel;
    int tamInfo;
}descFDEcRef;
```

Além das estruturas que já estão disponíveis no moodle fornecidas pelo professor de uma FDE de prioridade, desenvolvemos a struct descFDEcRef a qual possui um nó que é o referencial móvel que utilizaremos ao longo do código.

Partindo para o arquivo arq.c daremos ênfase na análise das funções que demandam mais trabalho e complexidade, dividimos as funções em tipo A e B, onde A são referentes a FDE sem refMovel e as do tipo B são de FDE com refMovel. A primeira a ser analisada será insereA (referente a FDE sem refMovel):

```c
int insereA(info *novo, descFDEsRef *p, int (*compara)(info *novo, info *visitado), int *nIteracao)
{
    int result;
    noF *novoNoFila = NULL, *visitado = NULL;

    if ((novoNoFila = (noF *)malloc(sizeof(noF))) != NULL)
    {
        memcpy(&(novoNoFila->dados), novo, sizeof(info));
        if (p->primeiro == NULL && p->ultimo == NULL) /*fila vazia*/
        {
            //insere o elemento ja que a fila esta vazia
            novoNoFila->atras = NULL;
            novoNoFila->frente = NULL;
            p->primeiro = novoNoFila;
            p->ultimo = novoNoFila;
            (*nIteracao)++;
        }
        else
        {
            //compara o novo elemento com a cauda
            result = (*compara)(novo, &(p->ultimo->dados));

            if (result == -1 || result == 0) //novo elemento é o de menor prioridade ou igual ao de menor prioridade
            {
                //menor prioridade
                novoNoFila->atras = NULL;
                novoNoFila->frente = p->ultimo;
                p->ultimo->atras = novoNoFila;
                p->ultimo = novoNoFila;
                (*nIteracao)++;
            }
            else
            {
                //compara o novo elemento com o primerio da fila
                result = (*compara)(novo, &(p->primeiro->dados));

                if (result == 1)
                {
                    // maior prioridade
                    novoNoFila->frente = NULL;
                    novoNoFila->atras = p->primeiro;
                    p->primeiro->frente = novoNoFila;
                    p->primeiro = novoNoFila;
                    (*nIteracao)++;
                }
```

```c
    else
    {
        //verifica se ta mais perto do primeiro ou do ultimo
        int difMaior = p->primeiro->dados.ranking - novo->ranking;
        int difMenor = novo->ranking - p->ultimo->dados.ranking;

        if (difMaior < difMenor)
        {
            //mais perto do primeiro
            visitado = p->primeiro->atras;//segunda maior prioridade

            (*nIteracao)++;

            //percorre a fila a partir do primeiro
            while (visitado->atras != NULL && (*compara)(novo, &(visitado->dados)) != 1)
            {
                visitado = visitado->atras; //comparou(A,B) e A < B ou A == B
                (*nIteracao)++;
            }

            //insere o novo no na posicao correta para ele
            novoNoFila->atras = visitado;
            novoNoFila->frente = visitado->frente;
            visitado->frente->atras = novoNoFila;
            visitado->frente = novoNoFila;
            (*nIteracao)++;
        }
        else
        {
            //sera mais perto do ultimo
            visitado = p->ultimo->frente;

            (*nIteracao)++;

            //percore a fila a partir da frente
            while (visitado->frente != NULL && (*compara)(novo, &(visitado->dados)) == 1)
            {
                visitado = visitado->frente;
                (*nIteracao)++;
            }

            //insere o novo no na posicao correta
            novoNoFila->atras = visitado->atras;
            novoNoFila->frente = visitado;
            visitado->atras->frente = novoNoFila;
            visitado->atras = novoNoFila;
            (*nIteracao)++;
```

Uma breve síntese sobre a função insereA():

- começa alocando memória para o novo no;
- se estiver vazia a fila irá inserir no começo e final da fila;
- se não estiver vazia fará uma comparação com a cauda, se tiver com prioridade igual ou menor ele será inserido como nova cauda;

- se o elemento não condiz com a comparação acima fará uma comparação com o primeiro elemento da fila e será inserido como primeiro se tiver prioridade maior;
- caso as comparações acima não sejam o suficiente para inserir o novo elemento (o elemento não for nem o primeiro nem o último da fila e ela nao estiver vazia) ele fará uma comparação entre o primeiro e último para estimar qual está mais próximo do elemento novo e irá percorrer a fila a partir da análise até encontrar sua posição ideal.
- durante o processo de inserção o número de interações necessárias para encontrar a posição correta do novo nó é contada e armazenada na variável nInteracao;

O restante das funções relacionadas a FDE sem refMovel foram implementadas pelo próprio professor e nos baseamos no código disponível do moodle, apenas comentamos a função insereA por termos feitos leves modificações;

Partindo para as funções referentes a FDE com refMovel temos de início a função criaB() com a seguinte estrutura (única diferença de criaB é que também iniciamos reMovel como NULL):

```c
/*************** CRIA FILA COM REF***************/
descFDEcRef *criaB(int tamInfo)
{
    descFDEcRef *desc = (descFDEcRef *)malloc(tamInfo * sizeof(descFDEcRef));
    if (desc != NULL)
    {
        desc->ultimo = NULL;
        desc->primeiro = NULL;
        desc->refMovel = NULL;
        desc->tamInfo = tamInfo;
    }
    return desc;
}
```

A função insereB inicialmente realiza os mesmos processos de verificação da função insereA, onde verifica se a fila está vazia e verifica os extremos (frente e cauda):

```c
int result;
noF *novoNoFila = NULL, *visitado = NULL;
if ((novoNoFila = (noF *)malloc(sizeof(noF))) != NULL)
{
    memcpy(&(novoNoFila->dados), novo, sizeof(info));
    if (p->primeiro == NULL && p->ultimo == NULL) /*fila vazia*/
    {
        // printf("FILA VAZIA\n");
        novoNoFila->atras = NULL;
        novoNoFila->frente = NULL;
        p->primeiro = novoNoFila;
        p->ultimo = novoNoFila;
        p->refMovel = novoNoFila;
        (*nIteracao)++;
    }
    else
    {
        result = (*compara)(novo, &(p->ultimo->dados));
        if (result == -1 || result == 0) /*novo elemento é o de menor prioridade ou igual ao de menor prioridade */
        {
            // printf("MENOR PRIORI\n");
            novoNoFila->atras = NULL;
            novoNoFila->frente = p->ultimo;
            p->ultimo->atras = novoNoFila;
            p->ultimo = novoNoFila;
            p->refMovel = novoNoFila;
            (*nIteracao)++;
        }
```

```c
result = (*compara)(novo, &(p->primeiro->dados));
if (result == 1) // maior priori ou igual
{
    // printf("MAIOR PRIORI\n");
    novoNoFila->frente = NULL;
    novoNoFila->atras = p->primeiro;
    p->primeiro->frente = novoNoFila;
    p->primeiro = novoNoFila;
    p->refMovel = novoNoFila;
    (*nIteracao)++;
}
```

Depois define a diferença entre os rankings, verificando o sinal da diferença de rankings entre o elemento apontado pelo referencial móvel e o novo elemento:

```c
int difMaior = (p->primeiro->dados.ranking) - (novo->ranking), difMenor = (novo->ranking) - (p->ultimo->dados.ranking);
int difRef = (novo->ranking) - (p->refMovel->dados.ranking);
if (difRef < 0)
{
    difRef *= (-1);
}
```

Então ele verifica qual é a menor diferença e a partir disso insere o novo elemento:

Inserção quando o novo ranking é mais próximo do primeiro elemento:

```c
if (difMaior < difMenor)
{
    if (difMaior < difRef)
    {
        // printf("MAIS PROX DO PRIMEIRO\n");
        visitado = p->primeiro->atras; /*segunda maior prioridade */
        (*nIteracao)++;
        int show = 2;
        // printf("VERIFICANDO O %d ELEMENTO\n", show);
        while (visitado->atras != NULL && (*compara)(novo, &(visitado->dados)) != 1)
        {
            show++;
            //    printf("VERIFICANDO O %d ELEMENTO\n", show);
            visitado = visitado->atras; /*comparou(A,B) e A < B ou A == B*/
            (*nIteracao)++;
        }
        novoNoFila->atras = visitado;
        novoNoFila->frente = visitado->frente;
        visitado->frente->atras = novoNoFila;
        visitado->frente = novoNoFila;
        p->refMovel = novoNoFila;
        (*nIteracao)++;
```

Inserção quando é mais próximo e maior que o referencial móvel:

```c
noF *visitado PROX DO REF MOVEL\n");
visitado = p->refMovel;
(*nIteracao)++;
// printf("COMPARANDO COM O REF MOVEL\n");
// printf("COMPARANDO COM OS PROX:\n");
int show = 1;
while (visitado->frente != NULL && (*compara)(novo, &(visitado->dados)) == 1)
{
    // printf("%d\n", show);
    show++;
    visitado = visitado->frente;
    (*nIteracao)++;
}
novoNoFila->atras = visitado->atras;
novoNoFila->frente = visitado;
visitado->atras->frente = novoNoFila;
visitado->atras = novoNoFila;
p->refMovel = novoNoFila;
(*nIteracao)++;
```

Inserção quando é mais próximo do último elemento:

```c
if (difMenor <= difRef)
{
    // printf("MAIS PROX DO ULTIMO\n");
    visitado = p->ultimo->frente;
    (*nIteracao)++;
    int show = 1;
    // printf("COMPARANDO COM O %d A FRENTE DO ULTIMO\n", show);
    while (visitado->frente != NULL && (*compara)(novo, &(visitado->dados)) == 1)
    {
        show++;
        //    printf("COMPARANDO COM O %d A FRENTE DO ULTIMO\n", show);
        visitado = visitado->frente;
        (*nIteracao)++;
    }
    novoNoFila->atras = visitado->atras;
    novoNoFila->frente = visitado;
    visitado->atras->frente = novoNoFila;
    visitado->atras = novoNoFila;
    p->refMovel = novoNoFila;
    (*nIteracao)++;
```

Inserção quando é mais próximo e menor que o referencial móvel:

```c
// printf("MAIS PROX DO REF MOVEL\n");
visitado = p->refMovel;
(*nIteracao)++;
// printf("COMPARANDO COM O REF MOVEL\n");
// printf("COMPARANDO COM OS PROX:\n");
int show = 1;
while (visitado->atras != NULL && (*compara)(novo, &(visitado->dados)) != 1)
{
    // printf("%d\n", show);
    show++;
    visitado = visitado->atras;
    (*nIteracao)++;
}
novoNoFila->atras = visitado;
novoNoFila->frente = visitado->frente;
visitado->frente->atras = novoNoFila;
visitado->frente = novoNoFila;
p->refMovel = novoNoFila;
(*nIteracao)++;
```

Função remove _b já possui comentários para melhor compreensão:

```c
/************** REMOVE DA FILA COM REF**************/
int remove_B(info *reg, descFDEcRef *p, int chave)
{
    int ret = 0;
    noF *aux = p->ultimo;//auxiliar apontando para o ultimo no da fila

    if (p->ultimo != NULL && p->primeiro != NULL)//verifica se a fila esta vazi
    {
        memcpy(reg, &(p->primeiro->dados), p->tamInfo);

        if (aux == p->primeiro)
        { // caso tenha 1 elemento apenas
            free(p->primeiro);
            p->primeiro = p->ultimo = NULL;
        }
        else
        {
            if (chave == 1)
            {
                //analisa se a remocao sera to primeiro elemento da fila
                p->primeiro = p->primeiro->atras;
                free(p->primeiro->frente);
                p->primeiro->frente = NULL;
            }
            else
            {
                //atualizara para apontar para o anterior
                p->ultimo = p->ultimo->frente;
                free(p->ultimo->atras);
                p->ultimo->atras = NULL;
            }
        }
        ret = 1;
    }

    return ret;
}
```

As funcoes buscaNaFrenteB, buscaNaCaudaB, testaVaziaB, tamanhoDaFilaB são iguais as da FDE sem refMovel disponíveis no moodle da disciplina, apenas de diferenciam da fila A pois o parâmetro do descritor passado possui o refMovel.

A função reiniciaB() possui de diferença o refMovel sendo setado como NULL:

```c
/*************** PURGA FILA COM REF***************/
int reiniciaB(descFDEcRef *p)
{
    int ret = 0;
    noF *aux = NULL;

    if (p->primeiro != NULL && p->ultimo != NULL)
    {
        aux = p->ultimo->frente;
        while (aux != NULL)
        {
            free(p->ultimo);
            p->ultimo = aux;
            aux = aux->frente;
        }

        free(p->ultimo);
        p->primeiro = NULL;
        p->ultimo = NULL;
        p->refMovel = NULL;
        ret = 1;
    }
    return ret;
}
```

E por ultimo temos a função inveteB, que no codigo testesFuncFDEcomRefMovel.c apresentou ótimo desempenho com números baixos de inserção na fila e garantimos seu funcionamento (código já possui comentários):

```c
/*************INVERTE FILA COM REF***************/
int inverteB(descFDEcRef *p)
{
    if (testaVaziaB(p) == 1)//verifica se a fila esta vazia
    {
        printf("FILA VAZIA!\n\n");
        return 0;
    }
    if (p->primeiro == p->ultimo)//verifica se a fila possui apenas um
    {
        printf("FILA COM UM ELEMENTO\n");
        return 0;
    }

    noF *aux = p->primeiro->atras;//auxiliar para andar pela fila

        while (aux->atras != NULL)
        {
            //inverte os ponteiros atras e frente
            noF *helper = aux->atras;
            aux->atras = aux->frente;
            aux->frente = helper;
            aux = helper;
        }

    //atualizacao dos ponteiros do primeiro e ultimo elemento da fila
    noF *aux2 = p->primeiro;
    p->primeiro->frente = p->primeiro->atras;
    p->primeiro->atras = NULL;
    p->ultimo->atras = p->ultimo->frente;
    p->ultimo->frente = NULL;
    p->primeiro = p->ultimo;
    p->ultimo = aux2;

    return 1;//sucesso
}
```

Finalizando a análise do arq.h e do arq.c onde respeitamos o mesmo menu para ambas as filas vamos agora partir para a análise do código testesFuncFDEcomRefMovel.c, o qual

possui testes para verificar a funcionalidade das implementações que desenvolvemos e exibirá o funcionamento da fila.

O arquivo pode ser compilado utilizando o comando:

gcc -Wall arq.c testesFuncFDEcomRefMovel.c -Larq.h && ./a.out

De início temos nosso main() compacto devido a função menu que será apresentada mais abaixo :

```c
int main(void)
{
    descFDEcRef *ptr = NULL;
    int tamFila = 0;
    printf("Insira o tamanho da fila:\n");
    scanf("%d", &tamFila);
    ptr = criaB(tamFila);

    if (ptr == NULL)//verificacao durante a criacao
        printf("Erro fatal durante a criacao da fila!!\n\n");
    else
        menu(ptr);
    return 1;
}
```

Teremos como teste de mesa:

```
Insira o tamanho da fila:
4
------------------------MENU----------------------------
-------------INSERIR ELEMENTO NA FILA(1)----------------
------------REMOVER PRIMEIRO ELEMENTO(2)----------------
---------REMOVER ULTIMO ELEMENTO DA FILA(3)-------------
------------------BUSCA NA CAUDA(4)---------------------
-----------------BUSCA NA FRENTE(5)---------------------
-----------------TAMANHO DA FILA(6)---------------------
-----------------INVERTE A FILA(7)----------------------
---------------------REINICIA(8)-------------------------
---------------------DESTROI(9)--------------------------
-------------------TESTE VAZIA(10)----------------------
```

Observações: os casos como inserir mais elementos do que selecionado, remover, buscar, e inverter de uma lista vazia foram testados e funcionam mas não vamos testá-los aqui devido a extensão do relatório.

Passo 1 (teste de inserção) digitar 1 e inserir os dados para inserção:

```
1
Insira o NOME:
Gilmario
Insira a matricula:
1234
Insira o ranking:
10
Insira o curso:
BCC
Sucesso na insecao!
```

Teremos assim a fila (por ranking): _ _ _ 10

Passo 2: completar a fila com mais 3 elementos

```
1
Insira o NOME:
Gustavo
Insira a matricula:
12345
Insira o ranking:
9
Insira o curso:
BCC
Sucesso na insecao!
```

Teremos assim a fila (por ranking): _ _ 9 10

```
1
Insira o NOME:
Felipe
Insira a matricula:
12345
Insira o ranking:
11
Insira o curso:
BCC
Sucesso na insecao!
```

Teremos assim a fila (por ranking): _ 9 10 11

```
1
Insira o NOME:
Mario
Insira a matricula:
43321
Insira o ranking:
6
Insira o curso:
BCC
Sucesso na insecao!
```

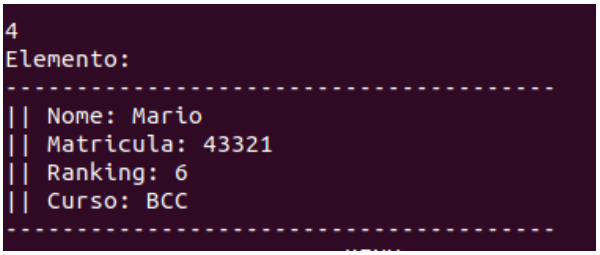
Teremos assim a fila (por ranking): 6 9 10 11

Passo 3 (quantidade de elementos):

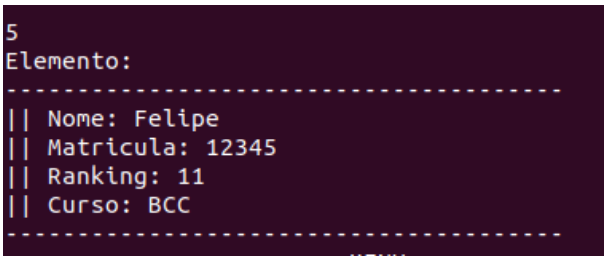
```
------------------------MENU--------------------------
-------------INSERIR ELEMENTO NA FILA(1)---------------
------------REMOVER PRIMEIRO ELEMENTO(2)---------------
---------REMOVER ULTIMO ELEMENTO DA FILA(3)------------
------------------BUSCA NA CAUDA(4)---------------------
-----------------BUSCA NA FRENTE(5)---------------------
-----------------TAMANHO DA FILA(6)---------------------
-----------------INVERTE A FILA(7)----------------------
---------------------REINICIA(8)-----------------------
--------------------DESTROI(9)-------------------------
------------------TESTE VAZIA(10)----------------------

6
Tamanho: 4
```

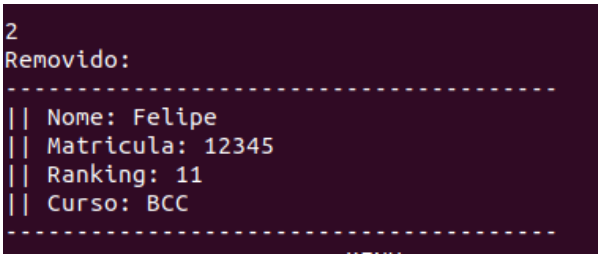
Passo 4 (consultar cauda):



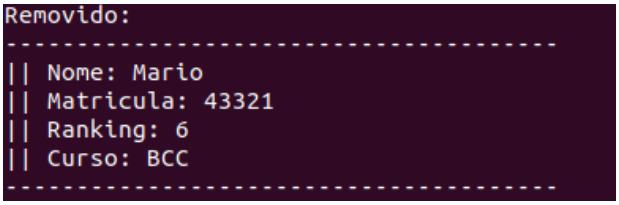
Passo 5 (consultar frente):



Passo 6 (remoção do primeiro):



Passo 7 (remoção do último, aqui pode haver alguns erros de exibições porém a remoção acontece normalmente) :



Passo 8 (inversão da fila, agora só temos [_ _ 9 10]):

```
7
Fila normal:

Elemento: 1
|| Nome: Gilmario
|| Matricula: 1234
|| Ranking: 10
|| Curso: bcc
Elemento: 2
|| Nome: Gustavo
|| Matricula: 12345
|| Ranking: 9
|| Curso: BCC
Sucesso na inversao!!

Fila invertida:

Elemento: 1
|| Nome: Gustavo
|| Matricula: 12345
|| Ranking: 9
|| Curso: BCC
Elemento: 2
|| Nome: Gilmario
|| Matricula: 1234
|| Ranking: 10
|| Curso: bcc
```

Passo 9 (reinicialização da fila seguido da verificação do estado vazio):

```
8
Sucesso na reinicializacao!!

------------------------MENU--------------------------
-------------INSERIR ELEMENTO NA FILA(1)---------------
------------REMOVER PRIMEIRO ELEMENTO(2)---------------
---------REMOVER ULTIMO ELEMENTO DA FILA(3)------------
------------------BUSCA NA CAUDA(4)--------------------
-----------------BUSCA NA FRENTE(5)--------------------
-----------------TAMANHO DA FILA(6)--------------------
-----------------INVERTE A FILA(7)---------------------
---------------------REINICIA(8)-----------------------
---------------------DESTROI(9)------------------------
-------------------TESTE VAZIA(10)---------------------

10
FILA VAZIA!
```

Passo 10 (por fim destruição da fila):

```
9
Fila destruida!

SAINDO...
```

Finalizando o teste de mesa para as funções relacionadas com a fila com referencial móvel vamos partir para a análise do código fila.c, o qual foi desenvolvido com o intuito de medir as diferentes formas de aplicações de uma fila e conseguir comparar as estratégias de implementação.

Como foi citado no começo do relatório, a linha de comando para o teste que compara as implementações é:

gcc -Wall arq.c fila.c -Larq.h && ./a.out dataset_v1.csv

No início da análise do main() de fila.c temos a abertura do arquivo da mesma forma que foi feita na TAREFA1A :

```c
int main(int argc, char const *argv[])
{
    srand((unsigned)time(NULL));
    FILE *file;
    char const *filename;

    // arquivo ta ok
    if (argc != 2)
    {
        printf("Uso: %s arquivo.csv\n", argv[0]);
        return 1;
    }

    // nome do csv
    filename = argv[1];

    // abertura do arquivo para leitura
    file = fopen(filename, "r");

    // verificacao se o arquivo foi aberto corretamente
    if (file == NULL)
    {
        printf("Erro ao abrir o arquivo.\n");
        return 1;
    }
```

Logo abaixo temos a declaração dos descritores da fila com e sem refMovel e a variável tamDesejado, essa variável está setada agora de início com o valor 100, os valores a ela atribuído representarão o número de casos que serão efetuados. Em seguida temos o vetor sOun que conterá valores gerados aleatoriamente que representam os índices dos rankings para inserção e o vetor valoresDeInserção que conterá os valores a serem inseridos. :

```c
//declarando os descritores das duas filas
descFDEsRef *filaSemRefMovel;
descFDEcRef *filaComRefMovel;

int tamDesejado = 100; //variavel que controla os testes

//sOuN eh um vetor gerado randomicamente que contem os indices
int sOuN[tamDesejado];
int valoresDeInsercao[tamDesejado];
```

Abaixo temos a criação das filas através das funções criaA e criaB e a declaração das variáveis que usaremos para contabilizar as interações:

```c
//criacao das filas
filaSemRefMovel = criaA(tamDesejado);
filaComRefMovel = criaB(tamDesejado);

int qnt = 0, indice = 0;
int qntIteracoesA = 0, qntIteracoesB = 0;
```

Aqui temos a criação aleatória e ordenação do vetor que contem os índices que serão inseridos nas filas:

```
//criando os valores aleatorios
for (int i = 0; i < tamDesejado; i++)
{
    sOuN[i] = 1 + (rand() % 9999);
    int correcao = sOuN[i];
    int flag = 0;
    while (flag == 0)
    {
        flag = 1;
       for (int j = 0; j <= i; j++)
    {

        if (correcao == sOuN[j] && i != j)
        {
            sOuN[i] = 1 + (rand() % 9999);
            correcao = sOuN[i];
            flag = 0;
        }

    }
    }
}

//ordenacao dos numeros gerados
for (int i = 0; i < tamDesejado; i++)
{
    for (int j = 0; j < tamDesejado; j++)
    {
        if (sOuN[i] < sOuN[j])
        {

            int c = sOuN[j];
            sOuN[j] = sOuN[i];
            sOuN[i] = c;
        }
    }
}
```

Aqui temos um teste de mesa dos valores aleatórios já ordenados do vetor sOuN[100] com a variável tamDesejado = 100. Vale ressaltar que vamos deixar esse print comentado pois quando estamos fazendo casos grandes como 1k ou mais o terminal do Linux não irá printar toda a saída do programa então será indecente para exibição.

```
Vetor com os indices que serão inseridos nas filas:
74 81 257 322 342 406 565 699 833 974 1051 1201 1319 1325 1337 1394 1445 1527 1679 1756 1796 1848 1947 1998 2182 2392 260
1 2713 3192 3376 3379 3408 3465 3545 3600 3716 3729 3771 3884 3923 3938 4399 4475 4503 4577 4647 4776 4801 4810 5053 5138
 5449 5596 5615 5774 5783 5880 5927 6008 6075 6137 6165 6375 6383 6482 6738 6910 6975 7061 7090 7149 7211 7216 7479 7528
7548 7552 7830 7863 7872 7989 8054 8062 8092 8107 8169 8297 8971 9037 9043 9054 9056 9275 9305 9336 9657 9762 9830 9863 9
983
```

Em seguida teremos no main um loop que só será fechado caso acabe os caracteres do arquivo ou o tamanho desejado de casos seja atingido, enquanto o laço se manter teremos a cópia dos dados do arquivo csv para as filas da seguinte forma:

```c
//loop para ler o arquivo e inserir os dados na fila
while (aux != NULL && qnt < tamDesejado && !(feof(file)))
{

    info inserir;

    //inserindo os dados do csv de cada usario para a a struct info
    fscanf(file, "\n%[^,]s", aux);
    strcpy(inserir.nome, aux);
    fscanf(file, ",%[^,]s", aux);
    int matricula = atoi(aux);//convertendo a string para inteiro
    inserir.matricula = matricula;
    fscanf(file, ",%[^,]s", aux);
    int rank = atoi(aux);
    inserir.ranking = rank;
    fscanf(file, ",%s", aux);
    strcpy(inserir.curso, aux);

    //verifica se o indice atual condiz com o indice gerado aleatoriamente
    if (sOuN[qnt] == indice)
    {
        insereA(&inserir, filaSemRefMovel, compara, &qntIteracoesA);//insere na fila sem referencial movel
        insereB(&inserir, filaComRefMovel, compara, &qntIteracoesB);//insere na fila com referencial movel

        printf("\nInsercao: %d", inserir.ranking);
        printf("\nIteracoes A: %d\n", qntIteracoesA);
        printf("Iteracoes B: %d\n\n", qntIteracoesB);
        valoresDeInsercao[qnt] = inserir.ranking;
        qnt++;
    }
    indice++;
}
```

Um teste de mesa para facilitar a compreensão de tal aplicação é dado com tamDesejado = 100:

```
Insercao: 10
Iteracoes A: 1
Iteracoes B: 1


Insercao: 10
Iteracoes A: 2
Iteracoes B: 2


Insercao: 12
Iteracoes A: 3
Iteracoes B: 3


Insercao: 13
Iteracoes A: 4
Iteracoes B: 4


Insercao: 13
Iteracoes A: 6
Iteracoes B: 6


Insercao: 13
Iteracoes A: 9
Iteracoes B: 8
```

E por fim temos a exibição dos valores que serão inseridos nas filas e dos valores totais de interações de cada fila:

```c
printf("Valores de ranking para inserir na fila:\n");
for (int i = 1; i <= tamDesejado; i++)
{
    printf("%d, ", valoresDeInsercao[i-1]);
    if (i % 30 == 0)
    {
        printf("\n");
    }

}

printf("\nTotal de iteracoes A: %d\n\n", qntIteracoesA);
printf("Total de iteracoes B: %d\n", qntIteracoesB);

fclose(file);
return 0;
```

Teste de mesa com tamDesejado = 100:

```
Insercao: 99
Iteracoes A: 157
Iteracoes B: 139


Insercao: 100
Iteracoes A: 158
Iteracoes B: 140

Valores de ranking para inserir na fila:
10, 10, 12, 13, 13, 13, 15, 16, 17, 18, 19, 21, 22, 22, 22, 22, 23, 23, 25, 26, 26, 26, 27, 28, 30, 32, 33, 35, 39, 40,
40, 41, 41, 42, 42, 43, 44, 44, 45, 45, 45, 50, 50, 51, 51, 52, 53, 53, 54, 56, 57, 59, 61, 61, 62, 63, 64, 64, 65, 65,
66, 66, 68, 68, 69, 71, 73, 73, 74, 74, 75, 75, 75, 78, 78, 78, 78, 81, 81, 81, 82, 83, 83, 83, 83, 84, 85, 91, 92, 92,
92, 92, 94, 94, 94, 97, 98, 99, 99, 100,
Total de iteracoes A: 158

Total de iteracoes B: 140
```

TESTES DE IMPLEMENTAÇÕES

Observação: para cada quantidade de casos foi constituída uma base de interações que foram encontradas a partir da média com arredondamento de 6 compilações do código com as mesmas especificações.

TESTE (tamDesejado = 500):

```
Total de iteracoes A: 1900
Total de iteracoes B: 908
```

```
Total de iteracoes A: 1871
Total de iteracoes B: 905
```

```
Total de iteracoes A: 1863
Total de iteracoes B: 905
```

```
Total de iteracoes A: 1808
Total de iteracoes B: 904
```

```
Total de iteracoes A: 1852
Total de iteracoes B: 906
```

```
Total de iteracoes A: 1823
Total de iteracoes B: 901
```

Total de A: (1900+1871+1808+1852+1863+1823)/6 = 1853
Total de B: (908+905+904+906+905+901)/6 = 905

TESTE (tamDesejado = 1000):

```
Total de iteracoes A: 6586
Total de iteracoes B: 1902
```

```
Total de iteracoes A: 6373
Total de iteracoes B: 1895
```

```
Total de iteracoes A: 6433
Total de iteracoes B: 1897
```

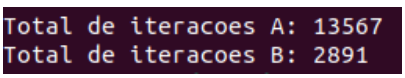
```
Total de iteracoes A: 6462
Total de iteracoes B: 1901
```

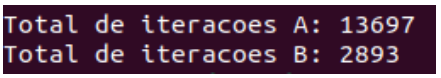
```
Total de iteracoes A: 6340
Total de iteracoes B: 1899
```

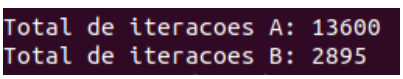
```
Total de iteracoes A: 6374
Total de iteracoes B: 1898
```

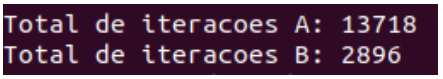
Total de A: (6586+6373+6433+6462+6340+6374)/6= 6428
Total de B: (1902+1895+1897+1901+1899+1898)/6= 1899

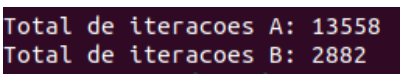
TESTE (tamDesejado = 1500):

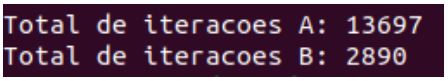






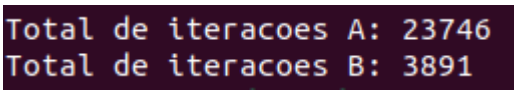


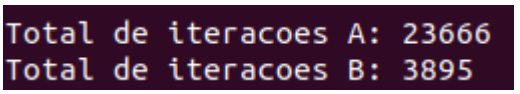


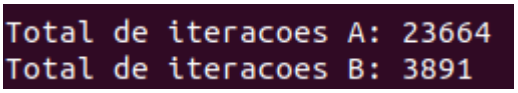


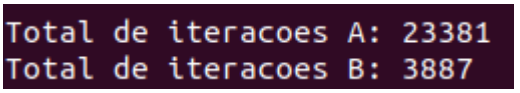
Total de A: (1357+13697+13600+13718+13558+13697)/6 = 11605
Total de B: (2891+2893+2895+2896+2882+2890)/6 = 2891

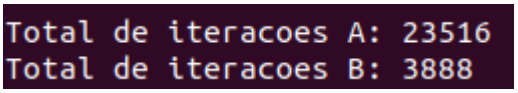
TESTE (tamDesejado = 2000):

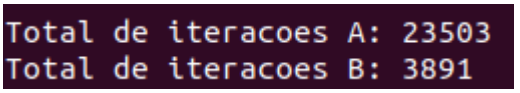






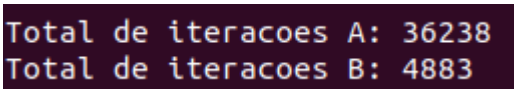


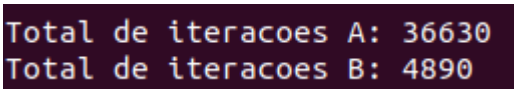


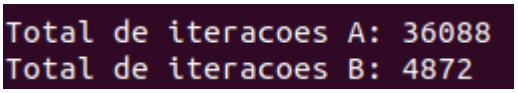


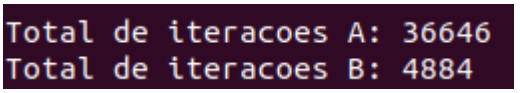
Total de A: (23746+ 23666+23664+23381+23516+23503)/6 = 23579
Total de B: (3891+ 3895+3891+3887+3888+3891)/6 = 3891

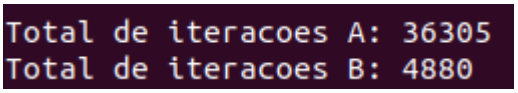
TESTE (tamDesejado = 2500):

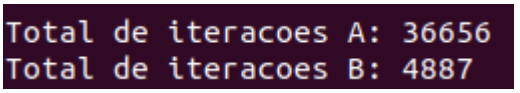






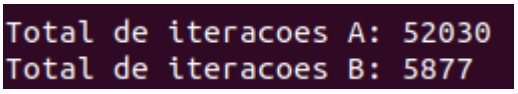


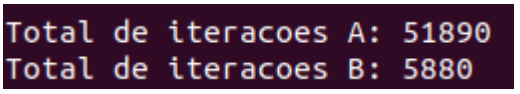


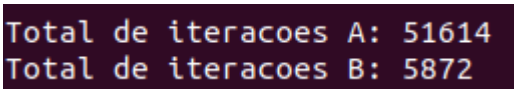


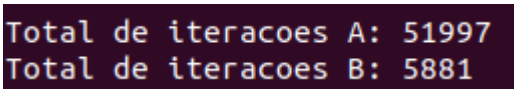
Total de A: (36238+36630+36088+36646+36305+36656)/6 = 36427
Total de B: (4883+4890+4872+4884+4880+4887)/6 = 4883

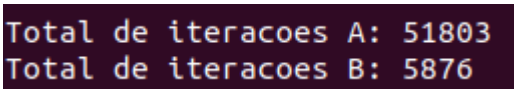
TESTE (tamDesejado = 3000):

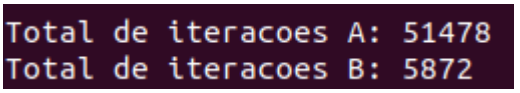












Total de A: (52030+21890+51614+51997+51478+51803+51478)/6 = 55381
Total de B: (5877+5880+5881+5872+5876+5872)/6 = 5876

TESTE (tamDesejado = 3500):

```
Total de iteracoes A: 70060
Total de iteracoes B: 6870
```
```
Total de iteracoes A: 69089
Total de iteracoes B: 6864
```
```
Total de iteracoes A: 69833
Total de iteracoes B: 6872
```
```
Total de iteracoes A: 69956
Total de iteracoes B: 6874
```
```
Total de iteracoes A: 70019
Total de iteracoes B: 6879
```
```
Total de iteracoes A: 69870
Total de iteracoes B: 6872
```

Total de A: (70060+69089+69833+69956+70019+69870)/6 = 69805
Total de B: (6870+6864+6874+6872+6879+6872)/6 = 6871

TESTE (tamDesejado = 4000):

```
Total de iteracoes A: 90325
Total de iteracoes B: 7862
```
```
Total de iteracoes A: 90990
Total de iteracoes B: 7874
```
```
Total de iteracoes A: 90849
Total de iteracoes B: 7863
```
```
Total de iteracoes A: 90468
Total de iteracoes B: 7868
```
```
Total de iteracoes A: 91231
Total de iteracoes B: 7870
```
```
Total de iteracoes A: 90584
Total de iteracoes B: 7871
```

Total de A: (90325+90990+90849+90468+91231+90584)/6 = 90741
Total de B: (7862+7874+7863+7868+7870+7871)/6 = 7868

TESTE (tamDesejado = 4500):

```
Total de iteracoes A: 114166
Total de iteracoes B: 8870
```
```
Total de iteracoes A: 114503
Total de iteracoes B: 8866
```
```
Total de iteracoes A: 113988
Total de iteracoes B: 8862
```
```
Total de iteracoes A: 113951
Total de iteracoes B: 8861
```
```
Total de iteracoes A: 114738
Total de iteracoes B: 8867
```
```
Total de iteracoes A: 114553
Total de iteracoes B: 8856
```

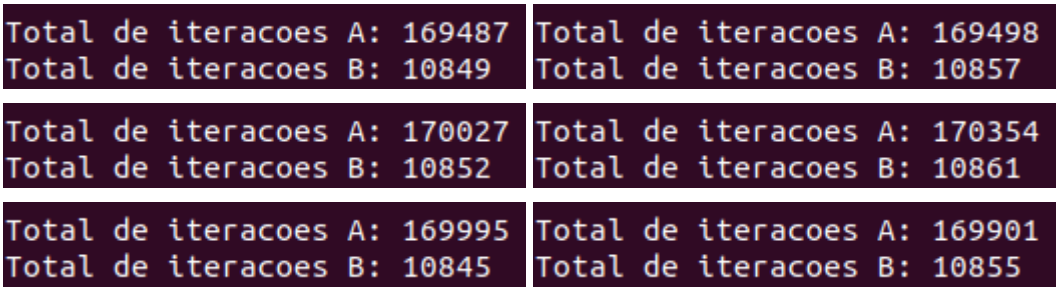
Total de A: (114166+114503+113988+113951+114738+114553)/6 = 114317
Total de B: (8870+8866+8862+8861+8856+8867)/6 = 8864

TESTE (tamDesejado = 5000):

```
Total de iteracoes A: 141124
Total de iteracoes B: 9861
```
```
Total de iteracoes A: 140449
Total de iteracoes B: 9856
```
```
Total de iteracoes A: 140444
Total de iteracoes B: 9856
```
```
Total de iteracoes A: 140742
Total de iteracoes B: 9859
```
```
Total de iteracoes A: 141123
Total de iteracoes B: 9857
```
```
Total de iteracoes A: 140701
Total de iteracoes B: 9862
```

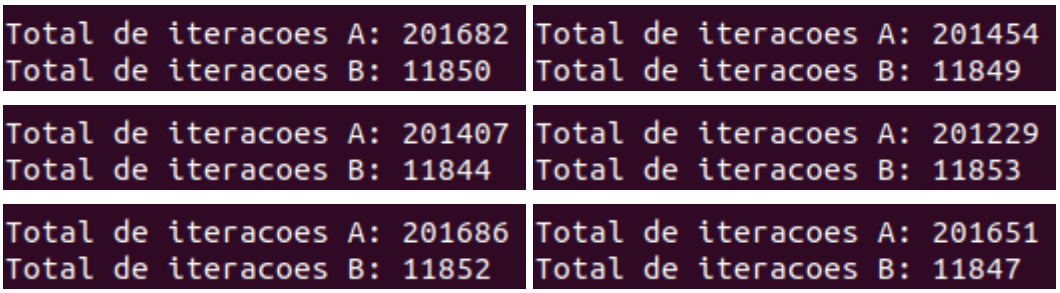
Total de A: (141124+140449+140444+140742+141123+140701)/6 = 140764
Total de B: (9861+9856+9856+9859+9857+9862)/6 = 9859

TESTE (tamDesejado = 5500):



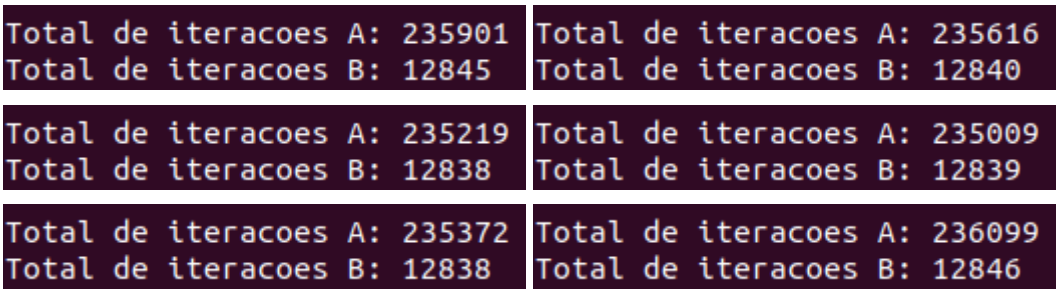
Total de A: (169487+169498+170027+170354+169995+169901)/6 = 169877
Total de B: (10849+10857+10852+10861+10845+10855)/6 = 10853

TESTE (tamDesejado = 6000):



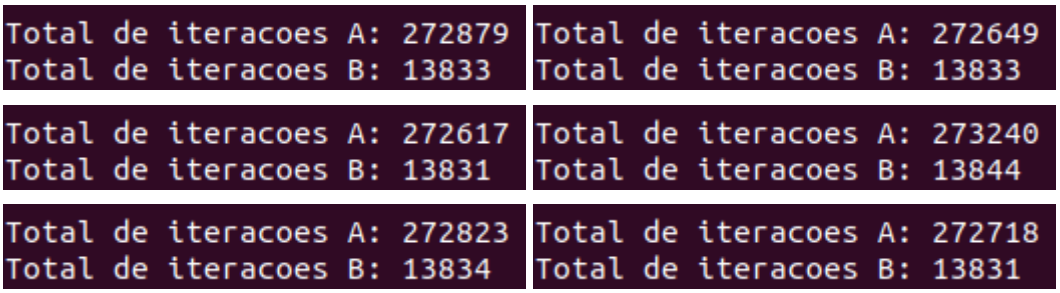
Total de A: (201682+201454+201407+201229+201686+201651)/6 = 201518
Total de B: (11850+11849+11844+11853+11852+11847)/6 = 11849

TESTE (tamDesejado = 6500):



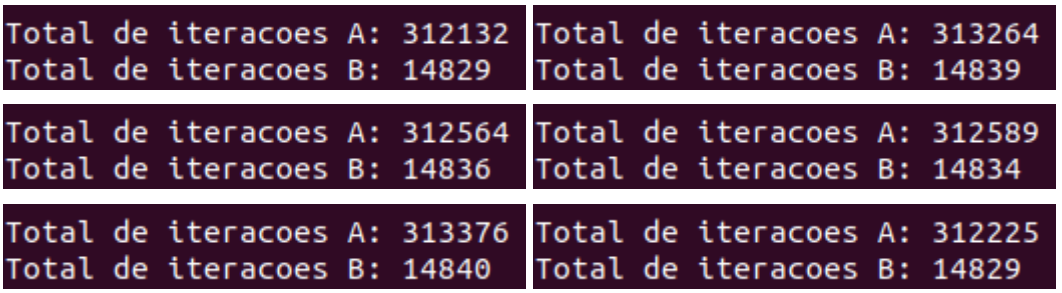
Total de A: (235901+235616+235219+235009+235372+236099)/6 = 235536
Total de B: (12845+12840+12838+12839+12838+12846)/6 = 12841

TESTE (tamDesejado = 7000):



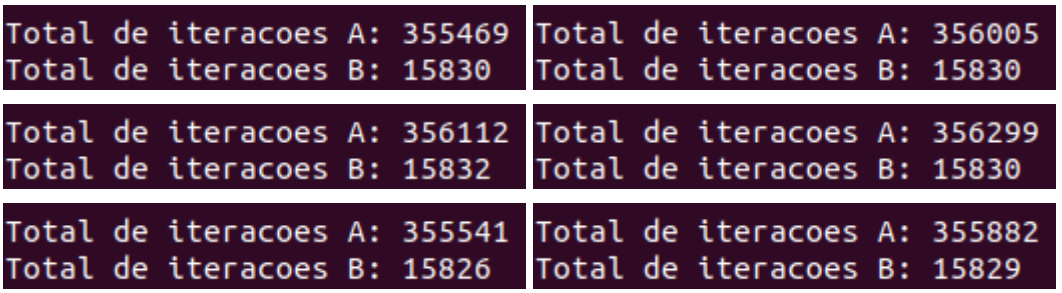
Total de A: (272879+272649+273240+272617+272823+272718)/6 = 272821
Total de B: (13833+13833+13831+13844+13834+13831)/6 = 13834

TESTE (tamDesejado = 7500):



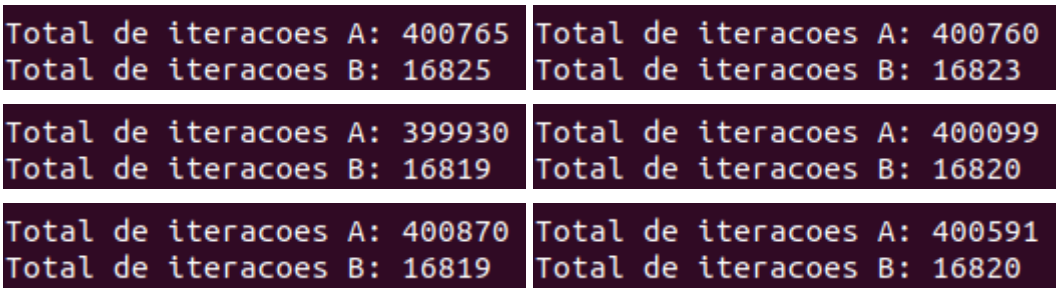
Total de A: (312132+313264+312564+312589+313376+312225)/6 = 312692
Total de B: (14829+14839+14836+14834+14840+14829)/6 = 14835

TESTE (tamDesejado = 8000):



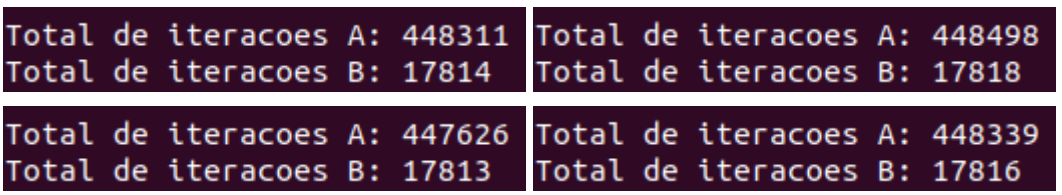
Total de A: (355469+356005+356112+356299+355541+355882)/6 = 355885
Total de B: (15830+15830+15832+15830+15826+15829)/6 = 15830

TESTE (tamDesejado = 8500):



Total de A: (400765+400760+399930+400099+400870+400591)/6 = 400503
Total de B: (16825+16823+16819+16820+16819+16820)/6 = 16821

TESTE (tamDesejado = 9000):



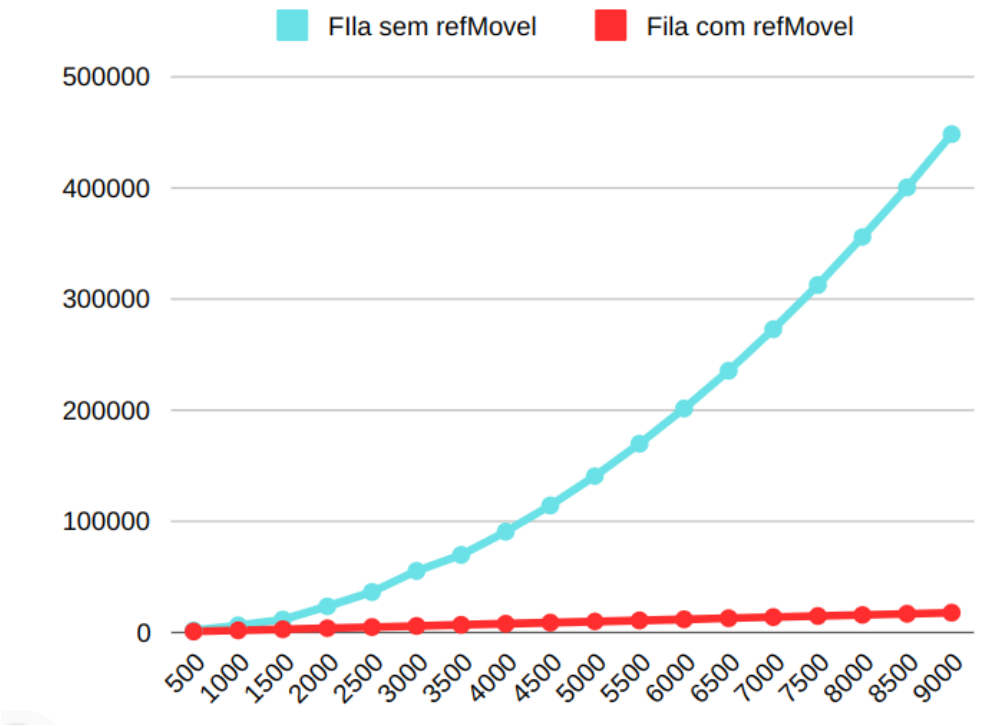
```
Total de iteracoes A: 448924
Total de iteracoes B: 17818
```

```
Total de iteracoes A: 448673
Total de iteracoes B: 17820
```

Total de A: (448924+448673+448311+448498+447626+448339)/6 = 448395
Total de B: (17814+17818+17813+17816+17818+17820)/6 = 17817

Gráfico com a evolução das interações

O gráfico foi desenvolvido com os valores médios calculados acima e possui a seguinte estrutura: (número de interações X número de casos testados)



Conclusão: ao analisarmos o gráfico podemos concluir que o comportamento do número de interações possui um caráter exponencial na fila sem refMovel, já na fila com refMovel tem um carácter linear de crescimento e podemos concluir que o desempenho é melhor com a inserção na fila com referencial móvel pois o mesmo facilita o acesso e manipulação dos elementos da estrutura. Pelo o que conseguimos encontrar na internet e baseado nos nossos dados obtidos podemos concluir que a inserção em uma fila com referencial móvel possui uma operação O(1) pois o ponteiro da cauda é movido em tempo constante independente do tamanho da fila. Em contrapartida, uma fila sem referencial móvel para ter uma adição de um novo elemento é necessário deslocar todos os elementos para abrir espaço, esse deslocamento é uma operação de complexidade O(n).