Trabalho Prático - AEDS II
Relatório de Desenvolvimento dos Algoritmos Propostos

Bruno dos Reis Gomes[1], **Estevão Henrique Moura de Oliveira** [2], **Flaviane Vitoria Cruz Ferrares**[3], **Gustavo Silva da Fonseca** [4], **Ismael Prado da Cruz Costa**[5], **Mateus Ferreira Martins** [6], **Sweney Meneses** [7], **Victor Elias Torquato Barboza** [8]

[1]Instituto de Ciências Exatas e Aplicadas – Universidade Federal de Ouro Preto (UFOP)
Caixa Postal 155 - 35931-008- João Monlevade - MG - Brasil

{estevao.oliveira; flaviane.ferrares; gustavo.fonseca;}@aluno.ufop.edu.br

{ismael.cruz; sweney.meneses; victor.barboza;}@aluno.ufop.edu.br

{bruno.reis1; mateus.martins}@aluno.ufop.edu.br

Resumo. *Este relatório tem como objetivo descrever os detalhes de desenvolvimento do algoritmos propostos pela docente Janniele Aparecida Soares de Araújo em uma atividade passada como método de avaliação da disciplina Algoritmos e Estruturas de Dados II.*

Abstract. *This report aims to describe the details of the development of the algorithms proposed by professor Janniele Aparecida Soares de Araújo in an activity passed as a method of evaluation of the discipline Algorithms and Data Structures II.*

1. Introdução

As informações apresentadas neste relatório e também na elaboração dos códigos foram efetuadas pelos discentes: Bruno, Estevão, Flaviane, Gustavo, Ismael, Mateus, Sweney e Victor, ambos alunos do curso de Sistemas de Informação, todos de períodos diferenciados.

Inicialmente, foi decidido que cada membro do grupo pegaria de forma individual uma parte do código na qual tivesse habilidades/conhecimentos para a implementação e assumisse a responsabilidade e, posteriormente, os mesmos iriam se reunir (via meet) novamente para a montagem do trabalho proposto. Contudo, devido às dificuldades encontradas, não foi possível que cada um fizesse algo de forma individual como foi proposto inicialmente, sendo assim, reuniões virtuais aconteceram e todos participaram de forma igualitária, ajudando no código e no relatório como um todo. Portanto, em todo o trabalho proposto, há um toque de cada discente em todas as partes do projeto apresentado. Vale ressaltar que obtivemos um grande avanço vindo dos esclarecimentos de dúvidas efetuados pelo discente Victor Elias, no qual possibilitou a conclusão de ideias para o trabalho apresentado. O presente relatório foi escrito pelas discentes Flaviane Vitória e Sweney Meneses e revisado por todos os demais integrantes do grupo.

2. Funções Criadas

Devido ao esqueleto do projeto ter sido apresentado, houve a necessidade de criar as seguintes funcionalidades:

2.1. particoes.h

Na biblioteca *particoes.h* fizemos a adoção de assinaturas de funções criadas, e realizamos algumas alterações na função *selecao_natural* que julgamos serem necessárias durante o processo de desenvolvimento do algoritmo.

Listing 1. Biblioteca particoes.h

```
#ifndef PARTICOES_H
#define PARTICOES_H

#include <stdio.h>
#include <stdbool.h>

#include "lista.h"

void classificacao_interna(FILE *arq, Lista *nome_arquivos_saida, int M, int nFunc);

void selecao_natural(FILE *arq, Lista *nome_arquivos_saida, int M, int nFunc, int n, int* nParticoes);

TFunc* getFunc(FILE* arq, int* contLidos);

int menorParaInicio(TFunc* v[], int M, int menor);

int elementosNoVetor(bool controle[], int tam);

void atualizaNomesParticao(Lista* nomes, int* nParticoes);

int getMenor(TFunc* v[], int tam);

int possuiElementos(TFunc* v[], int tam);

#endif
```

2.2. particoes.c

No programa *particoes.c* desenvolvemos as funções das quais fizemos a assinatura na biblioteca *particoes.h*, descreveremos agora a funcionalidades de cada função.

2.3. Função getFunc

A função lê um funcionário do arquivo e o retorna, também incrementa o total de funcionários lidos.

Listing 2. Função getFunc

```
TFunc* getFunc(FILE* arq, int* contLidos)
{
    fseek(arq, *contLidos * tamanho_registro(), SEEK_SET);
    TFunc* func = le_funcionario(arq);
    (*contLidos)++;
    return func;
}
```

2.4. Função atualizaNomesParticao

O procedimento cria novas partições caso necessário.

Listing 3. Função atualizaNomesParticao

```
void atualizaNomesParticao(Lista* nomes, int* nParticoes)
{
    if(nomes->prox == NULL)
    {
        char* newNome = malloc(5 * sizeof(char));
        (*nParticoes)++;
        sprintf(newNome, "p%d.dat", *nParticoes);
        nomes->prox = cria(newNome, NULL);
    }
}
```

2.5. Função possuiElementos

A função verifica se um vetor de tamanho tam possui elementos.

Listing 4. Função possuiElementos

```
int possuiElementos(TFunc* v[], int tam)
{
    for (int i = 0; i < tam; i++)
    {
        if(v[i] != NULL){
            return(1);
        }
    }
    return(0);
}
```

2.6. Função getMenor

A função retorna o indice do menor registro do vetor.

Listing 5. Função getMenor

```
int getMenor(TFunc* v[], int tam)
{
    int menor = 0;
    for (int i = 1; i < tam; i++)
    {
        if(v[menor]->cod > v[i]->cod)
        {
            menor = i;
        }
    }
    return menor;
}
```

2.7. Função menorParaInicio

A função recebe o indice do menor elemento do vetor, joga ele para o inicio e retorna sua posição.

Listing 6. Função menorParaInicio

```
int menorParaInicio(TFunc* v[], int M, int menor)
{
    TFunc* aux;
    if(menor == 0)
    {
        return(0);
    }
    else
    {
        for(int i = M-1; i > 0; i--)
        {
            if (i == menor)
            {
                aux = v[menor];
                v[i] = v[i-1];
                v[i-1] = aux;
                menor--;
            }
        }
    }
    return(menor);
}
```

2.8. Função elementosNoVetor

A função retorna a quantidade de elementos dentro do vetor

Listing 7. Função elementosNoVetor

```
int elementosNoVetor(bool controle[], int tam)
{
    int aux = 0;
    for (int i = 0; i < tam; i++)
    {
        if (controle[i] == false)
        {
            aux++;
        }
    }
    return(aux);
}
```

2.9. Função selecao_natural

A função gera as partições do arquivo.

Listing 8. Função selecao_natural

```
void selecao_natural(FILE *arq, Lista *nome_arquivos_saida, int M, int nFunc, int n, int* nParticoes)
{
    Lista* nomes = nome_arquivos_saida;
    Lista* nome_part = nomes;
    FILE* particao = NULL;
    int cursorPart = 0;
    int totalLidos = 0;

    TFunc** v = malloc(M * sizeof(TFunc*));

    FILE* res = fopen("reservatorio.dat", "w+");
    int cursorRes = 0;
    bool reservatorio_cheio;

    bool controleVet[M];
    int i = 0;

    // M registros do arquivo para a memoria
    while (i < M && (!feof(arq)))
    {
        v[i] = getFunc(arq, &totalLidos);
        controleVet[i] = false;
        i++;
    }

    // Abre arquivo de particao
    if(particao == NULL)
    {
        char* nome = nomes->nome;
        particao = fopen(nome, "wb+");
        nome_part = nomes;

        atualizaNomesParticao(nomes, nParticoes);
        nomes = nomes->prox;
    }

    // Enquanto ha elementos no vetor e
    // todos os registros no foram lidos
    while (possuiElementos(v, M) && totalLidos <= nFunc)
    {
        reservatorio_cheio = false;
        // Enquanto nao chega ao fim do arquivo e
        // o reservatorio nao esta cheio
        while (!feof(arq) && !reservatorio_cheio)
        {
            // Caso nao haja particao aberta
            // abre a proxima particao
            if(particao == NULL)
```

```
49              {
50                  char* nome = nomes->nome;
51                  particao = fopen(nome, "wb+");
52                  nome_part = nomes;
53
54                  atualizaNomesParticao(nomes, nParticoes);
55                  nomes = nomes->prox;
56              }
57              // Caso a particao atual esteja cheia
58              if(cursorPart >= M)
59              {
60                  fclose(particao);
61                  particao = NULL;
62                  cursorPart = 0;
63                  continue;
64              }
65              // Pega o menor valor do vetor
66              int menor = getMenor(v, M);
67              menor = menorParaInicio(v, M, menor);
68
69              // Salva o menor registro na particao
70              fseek(particao, cursorPart * tamanho_registro(), SEEK_SET);
71              salva_funcionario(v[menor], particao);
72
73              // Salva o cod do registro particionado acima
74              int lastKey = v[menor]->cod;
75              cursorPart++;
76              nome_part->tamanho++;
77              controleVet[menor] = true;
78
79              // Enquanto o indice do registro particionado no for trocado e
80              // nenhuma das condicoes de parada anteriores for verdadeira
81              while (controleVet[menor] && !reservatorio_cheio && !feof(arq) && totalLidos < nFunc)
82              {
83                  if (!feof(arq))
84                  {
85                      // Troca o registro ja particionado com prximo registro do arquivo
86                      v[menor] = getFunc(arq, &totalLidos);
87
88                      // Se for maior do que o registro da particao
89                      // grava na particao normalmente
90                      if (v[menor]->cod >= lastKey)
91                      {
92                          // Caso a particao atual esteja cheia
93                          // abre a proxima particao
94                          if(cursorPart >= M)
95                          {
96                              fclose(particao);
97                              particao = NULL;
```

```
98                              cursorPart = 0;
99
100                             char* nome = nomes->nome;
101                             particao = fopen(nome, "wb+");
102                             nome_part = nomes;
103
104                             atualizaNomesParticao(nomes, nParticoes);
105                             nomes = nomes->prox;
106                         }
107                         fseek(particao, cursorPart * tamanho_registro(), SEEK_SET);
108                         salva_funcionario(v[menor], particao);
109                         controleVet[menor] = false;
110                     }
111                     // Se for menor do que o registro da particao
112                     // grava no reservatorio e substitui pelo proximo
113                     else
114                     {
115                         fseek(res, cursorRes * tamanho_registro(), SEEK_SET);
116                         salva_funcionario(v[menor], res);
117                         cursorRes++;
118                         if (cursorRes == n)
119                         {
120                             reservatorio_cheio = true;
121                         }
122                     }
123                 }
124             }
125         }
126         // Se ja leu todos os registros est no fim da execucao
127         // Abre nova particao e armazena os registros restantes
128         if(totalLidos == nFunc)
129         {
130             fclose(particao);
131             particao = NULL;
132             cursorPart = 0;
133             char* nome = nomes->nome;
134             particao = fopen(nome, "wb+");
135             nome_part = nomes;
136
137             for(i = 0; i < M; i++)
138             {
139                 fseek(particao, i * tamanho_registro(), SEEK_SET);
140                 salva_funcionario(v[i], particao);
141             }
142             cursorPart = 0;
143             cursorRes = 0;
144         }
145         // Se a particao esta cheia abre a proxima
146         if(cursorPart >= M)
```

```
147             {
148                 fclose(particao);
149                 particao = NULL;
150                 cursorPart = 0;
151 
152                 char* nome = nomes->nome;
153                 particao = fopen(nome, "wb+");
154                 nome_part = nomes;
155 
156                 atualizaNomesParticao(nomes, nParticoes);
157                 nomes = nomes->prox;
158             }
159             // Se ha elementos no reservatorio, os passa para o vetor
160             if (cursorRes > 0)
161             {
162                 for (i = 0; i < cursorRes; i++)
163                 {
164                     fseek(res, i * tamanho_registro(), SEEK_SET);
165                     v[i] = le_funcionario(res);
166                     controleVet[i] = false;
167                     cursorRes--;
168                 }
169             }
170             int cont = elementosNoVetor(controleVet, M);
171 
172             // Se ha espao no vetor, o preenche com registros do arquivo
173             if (cont != M && (!feof(arq)))
174             {
175                 while (cont < M && (!feof(arq)))
176                 {
177                     v[cont] = getFunc(arq, &totalLidos);
178                     controleVet[cont] = false;
179                     cont++;
180                 }
181             }
182         }
183         // Libera a memoria alocada
184         for (int i = 0; i < M; i++)
185         {
186             free(v[i]);
187         }
188         fclose(particao);
189         free(v);
190 }
```

3. Resultados Obtidos

Abaixo, temos figuras representando os dados antes e depois da ordenação externa:

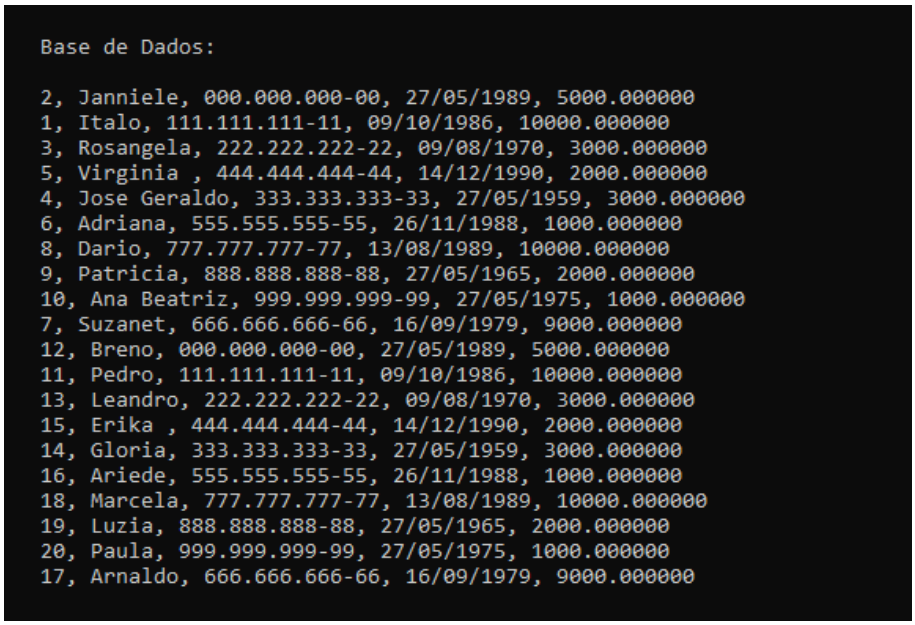

Figura 1. Base de dados antes da ordenação

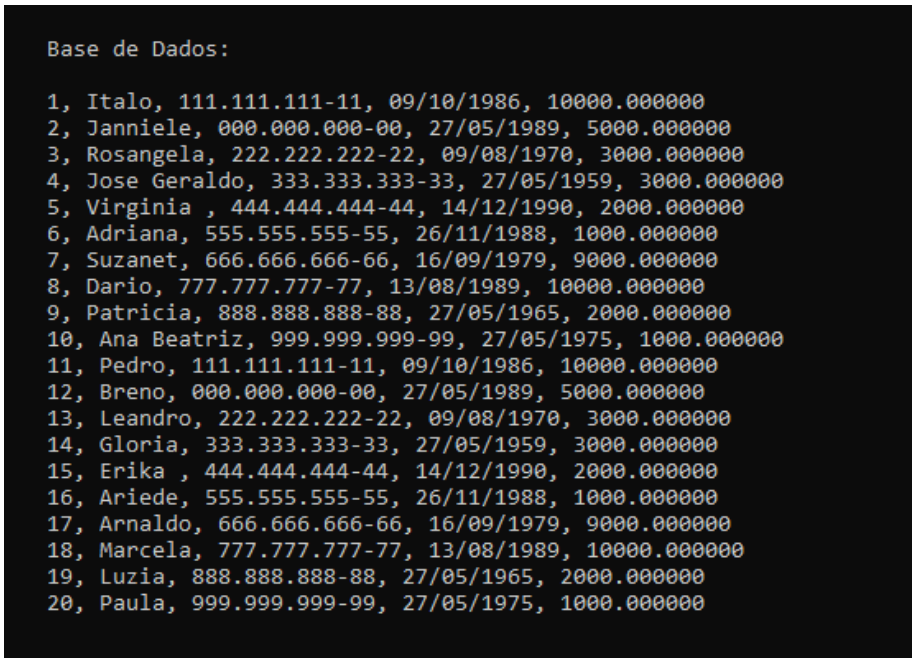

Figura 2. Base de dados depois da ordenação

Podemos perceber que pelo fato de não serem tantos dados a serem ordenados, ao executar o código, o tempo gasto foi consideravelmente bom (rápido - milésimos de segundos) para a ordenação.

4. Código Desenvolvido

O código desenvolvido está disponível em:

`https://github.com/VictorTorquato/Trabalho-Pratico-AEDS2`

Códigos que foram desenvolvidos:

Listing 9. particoes.h

```
#ifndef PARTICOES_H
#define PARTICOES_H

#include <stdio.h>
#include <stdbool.h>

#include "lista.h"

/* Algoritmo de geracao de particoes por Classificacao Interna, recebe como parametro
o arquivo de dados de entrada, a lista contendo os nomes dos arquivos de saida das particoes,
o numero de elementos M a ser armazenado em cada particao e o numero de registro total do arquivo.*/
void classificacao_interna(FILE *arq, Lista *nome_arquivos_saida, int M, int nFunc);

/*Algoritmo a ser implementado no trabalho de geracao de particoes por Selecao Natural recebe como pa
o arquivo de dados de entrada, a lista contendo os nomes dos arquivos de saida das particoes,
o numero de elementos M a ser armazenado em cada particao e o numero de registro total do arquivo e o
Fiquem a vontade para modificar a estrutura, foi uma sugestao.*/
void selecao_natural(FILE *arq, Lista *nome_arquivos_saida, int M, int nFunc, int n, int* nParticoes);

/*Funcao que le um funcionario do arquivo e o retorna
   tambem incrementa o total de funcionarios lidos*/
TFunc* getFunc(FILE* arq, int* contLidos);

/*Funcao que recebe o indice do menor elemento do vetor,
   joga ele para o inicio e retorna sua posicao*/
int menorParaInicio(TFunc* v[], int M, int menor);

/*Funcao que retorna a quantidade de elementos dentro do vetor*/
int elementosNoVetor(bool controle[], int tam);

/*Procedimento que cria novas particoes caso necessario*/
void atualizaNomesParticao(Lista* nomes, int* nParticoes);

/*Funcao que retorna o indice do menor registro do vetor*/
int getMenor(TFunc* v[], int tam);

/*Funcao para verificar se um vetor de tamanho tam possui elementos*/
int possuiElementos(TFunc* v[], int tam);

#endif
```

Listing 10. particoes.c

```c
#include "funcionarios.h"
#include <stdlib.h>
#include <limits.h>
#include <string.h>
#include <stdbool.h>

#include "particoes.h"

// Funcao que le um funcionario do arquivo e o retorna
// tambem incrementa o total de funcionarios lidos
TFunc* getFunc(FILE* arq, int* contLidos)
{
    fseek(arq, *contLidos * tamanho_registro(), SEEK_SET);
    TFunc* func = le_funcionario(arq);
    (*contLidos)++;
    return func;
}

// Funcao que recebe o indice do menor elemento do vetor,
// joga ele para o inicio e retorna sua posicao
int menorParaInicio(TFunc* v[], int M, int menor)
{
    TFunc* aux;
    if(menor == 0)
    {
        return(0);
    }
    else
    {
        for(int i = M-1; i > 0; i--)
        {
            if (i == menor)
            {
                aux = v[menor];
                v[i] = v[i-1];
                v[i-1] = aux;
                menor--;
            }
        }
    }
    return(menor);
}

// Funcao que retorna a quantidade de elementos dentro do vetor
int elementosNoVetor(bool controle[], int tam)
{
    int aux = 0;
    for (int i = 0; i < tam; i++)
    {
```

```
50              if (controle[i] == false)
51              {
52                  aux++;
53              }
54          }
55          return(aux);
56      }
57
58      // Procedimento que cria novas particoes caso necessario
59      void atualizaNomesParticao(Lista* nomes, int* nParticoes)
60      {
61          if(nomes->prox == NULL)
62          {
63              char* newNome = malloc(5 * sizeof(char));
64              (*nParticoes)++;
65              sprintf(newNome, "p%d.dat", *nParticoes);
66              nomes->prox = cria(newNome, NULL);
67          }
68      }
69
70      // Funcao que retorna o indice do menor registro do vetor
71      int getMenor(TFunc* v[], int tam)
72      {
73          int menor = 0;
74          for (int i = 1; i < tam; i++)
75          {
76              if(v[menor]->cod > v[i]->cod)
77              {
78                  menor = i;
79              }
80          }
81          return menor;
82      }
83
84      // Funcao para verificar se um vetor de tamanho tam possui elementos
85      int possuiElementos(TFunc* v[], int tam)
86      {
87          for (int i = 0; i < tam; i++)
88          {
89              if(v[i] != NULL){
90                  return(1);
91              }
92          }
93          return(0);
94      }
95
96      void classificacao_interna(FILE *arq, Lista *nome_arquivos_saida, int M, int nFunc)
97      {
98          rewind(arq); //posiciona cursor no inicio do arquivo
```

```
99      int reg = 0;
100
101     while (reg != nFunc)
102     {
103         //le o arquivo e coloca no vetor
104         TFunc *v[M];
105         int i = 0;
106         while (!feof(arq))
107         {
108             fseek(arq, (reg) * tamanho_registro(), SEEK_SET);
109             v[i] = le_funcionario(arq);
110             // imprime_funcionario(v[i]);
111             i++;
112             reg++;
113             if(i>=M)
114                 break;
115         }
116
117         //ajusta tamanho M caso arquivo de entrada tenha terminado antes do vetor
118         if (i != M)
119         {
120             M = i;
121         }
122
123         //faz ordenacao
124         for (int j = 1; j < M; j++)
125         {
126             TFunc *f = v[j];
127             i = j - 1;
128             while ((i >= 0) && (v[i]->cod > f->cod))
129             {
130                 v[i + 1] = v[i];
131                 i = i - 1;
132             }
133             v[i + 1] = f;
134         }
135
136         //cria arquivo de particao e faz gravacao
137         char *nome_particao = nome_arquivos_saida->nome;
138         nome_arquivos_saida = nome_arquivos_saida->prox;
139         printf("\n%s\n", nome_particao);
140         FILE *p;
141         if ((p = fopen(nome_particao, "wb+")) == NULL)
142         {
143             printf("Erro criar arquivo de saida\n");
144         }
145         else
146         {
147             for (int i = 0; i < M; i++)
```

```
148                 {
149                     fseek(p, (i) * tamanho_registro(), SEEK_SET);
150                     salva_funcionario(v[i], p);
151                     imprime_funcionario(v[i]);
152                 }
153                 fclose(p);
154             }
155             for(int jj = 0; jj<M; jj++)
156                 free(v[jj]);
157         }
158     }
159
160     void selecao_natural(FILE *arq, Lista *nome_arquivos_saida, int M, int nFunc, int n, int* nParticoes)
161     {
162         Lista* nomes = nome_arquivos_saida;
163         Lista* nome_part = nomes;
164         FILE* particao = NULL;
165         int cursorPart = 0;
166         int totalLidos = 0;
167
168         TFunc** v = malloc(M * sizeof(TFunc*));
169
170         FILE* res = fopen("reservatorio.dat", "w+");
171         int cursorRes = 0;
172         bool reservatorio_cheio;
173
174         bool controleVet[M];
175         int i = 0;
176
177         // M registros do arquivo para a memoria
178         while (i < M && (!feof(arq)))
179         {
180             v[i] = getFunc(arq, &totalLidos);
181             controleVet[i] = false;
182             i++;
183         }
184
185         // Abre arquivo de particao
186         if(particao == NULL)
187         {
188             char* nome = nomes->nome;
189             particao = fopen(nome, "wb+");
190             nome_part = nomes;
191
192             atualizaNomesParticao(nomes, nParticoes);
193             nomes = nomes->prox;
194         }
195
196         // Enquanto ha elementos no vetor e
```

```
197     // todos os registros nao foram lidos
198     while (possuiElementos(v, M) && totalLidos <= nFunc)
199     {
200         reservatorio_cheio = false;
201         // Enquanto nao chega ao fim do arquivo e
202         // o reservatorio nao esta cheio
203         while (!feof(arq) && !reservatorio_cheio)
204         {
205             // Caso nao haja particao aberta
206             // abre a proxima particao
207             if(particao == NULL)
208             {
209                 char* nome = nomes->nome;
210                 particao = fopen(nome, "wb+");
211                 nome_part = nomes;
212 
213                 atualizaNomesParticao(nomes, nParticoes);
214                 nomes = nomes->prox;
215             }
216             // Caso a particao atual esteja cheia
217             if(cursorPart >= M)
218             {
219                 fclose(particao);
220                 particao = NULL;
221                 cursorPart = 0;
222                 continue;
223             }
224             // Pega o menor valor do vetor
225             int menor = getMenor(v, M);
226             menor = menorParaInicio(v, M, menor);
227 
228             // Salva o menor registro na particao
229             fseek(particao, cursorPart * tamanho_registro(), SEEK_SET);
230             salva_funcionario(v[menor], particao);
231 
232             // Salva o cod do registro particionado acima
233             int lastKey = v[menor]->cod;
234             cursorPart++;
235             nome_part->tamanho++;
236             controleVet[menor] = true;
237 
238             // Enquanto o indice do registro particionado nao for trocado e
239             // nenhuma das condicoes de parada anteriores for verdadeira
240             while (controleVet[menor] && !reservatorio_cheio && !feof(arq) && totalLidos < nFunc)
241             {
242                 if (!feof(arq))
243                 {
244                     // Troca o registro ja particionado com proximo registro do arquivo
245                     v[menor] = getFunc(arq, &totalLidos);
```

```
246
247                     // Se for maior do que o registro da particao
248                     // grava na particao normalmente
249                     if (v[menor]->cod >= lastKey)
250                     {
251                         // Caso a particao atual esteja cheia
252                         // abre a proxima particao
253                         if(cursorPart >= M)
254                         {
255                             fclose(particao);
256                             particao = NULL;
257                             cursorPart = 0;
258
259                             char* nome = nomes->nome;
260                             particao = fopen(nome, "wb+");
261                             nome_part = nomes;
262
263                             atualizaNomesParticao(nomes, nParticoes);
264                             nomes = nomes->prox;
265                         }
266                         fseek(particao, cursorPart * tamanho_registro(), SEEK_SET);
267                         salva_funcionario(v[menor], particao);
268                         controleVet[menor] = false;
269                     }
270                     // Se for menor do que o registro da particao
271                     // grava no reservatorio e substitui pelo proximo
272                     else
273                     {
274                         fseek(res, cursorRes * tamanho_registro(), SEEK_SET);
275                         salva_funcionario(v[menor], res);
276                         cursorRes++;
277                         if (cursorRes == n)
278                         {
279                             reservatorio_cheio = true;
280                         }
281                     }
282                 }
283             }
284         }
285         // Se ja leu todos os registros esta no fim da execucao
286         // Abre nova particao e armazena os registros restantes
287         if(totalLidos == nFunc)
288         {
289             fclose(particao);
290             particao = NULL;
291             cursorPart = 0;
292             char* nome = nomes->nome;
293             particao = fopen(nome, "wb+");
294             nome_part = nomes;
```

```
295
296             for(i = 0; i < M; i++)
297             {
298                 fseek(particao, i * tamanho_registro(), SEEK_SET);
299                 salva_funcionario(v[i], particao);
300             }
301             cursorPart = 0;
302             cursorRes = 0;
303         }
304         // Se a particao esta cheia abre a proxima
305         if(cursorPart >= M)
306         {
307             fclose(particao);
308             particao = NULL;
309             cursorPart = 0;
310
311             char* nome = nomes->nome;
312             particao = fopen(nome, "wb+");
313             nome_part = nomes;
314
315             atualizaNomesParticao(nomes, nParticoes);
316             nomes = nomes->prox;
317         }
318         // Se ha elementos no reservatorio, os passa para o vetor
319         if (cursorRes > 0)
320         {
321             for (i = 0; i < cursorRes; i++)
322             {
323                 fseek(res, i * tamanho_registro(), SEEK_SET);
324                 v[i] = le_funcionario(res);
325                 controleVet[i] = false;
326                 cursorRes--;
327             }
328         }
329         int cont = elementosNoVetor(controleVet, M);
330
331         // Se ha espao no vetor, o preenche com registros do arquivo
332         if (cont != M && (!feof(arq)))
333         {
334             while (cont < M && (!feof(arq)))
335             {
336                 v[cont] = getFunc(arq, &totalLidos);
337                 controleVet[cont] = false;
338                 cont++;
339             }
340         }
341     }
342     // Libera a memoria alocada
343     for (int i = 0; i < M; i++)
```

```
344         {
345             free(v[i]);
346         }
347         fclose(particao);
348         free(v);
349 }
```