1.1/main.c

```c
#include "MSE.h"

int main(){
    criaArquivoTeste("dados.txt", 1, 1000);
    system("cp dados.txt dados2.txt");
    MergeSortExterno("dados.txt");
    return 0;
}
```

1.1/MSE.h

```c
#ifndef MSE_H
#define MSE_H

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 100

typedef struct sArq{
    FILE *f;
    int pos, max, *buffer;
}Arquivo;

void criaArquivoTeste(char* nome, int ini, int fim){
    int i;
    FILE *f = fopen(nome, "w");
    srand(time(NULL));
    for(i=1; i<1000; i++)
        fprintf(f, "%d\n", ini + rand() % (fim-ini + 1));
    fprintf(f, "%d", ini + rand() % (fim-ini + 1));
    fclose(f);
}


//QuickSort
void troca(int* a, int *b){
    int aux = *a;
    *a = *b;
    *b = aux;
}

int particao(int *v, int ini, int fim){
    int i = ini, j = fim;
    int pivo = v[(ini+fim)/2];
    while (1) {
        while(v[i] < pivo) i++; //procura algum >= pivo do lado esquerdo
        while(v[j] > pivo) j--; //procura algum <= pivo do lado direito

        if(i<j){
            troca(&v[i], &v[j]); //troca os elementos encontrados
            i++;
            j--;
        }else
            return j; //retorna o local onde foi feita a particao
    }
}

void QuickSort(int *v, int ini, int fim){
    if(ini < fim){
        int q = particao(v, ini, fim);
        QuickSort(v, ini, q);
        QuickSort(v, q+1, fim);
    }
}

//-----------------------------------
```

```
58
59
60  //Cria Arquivos Ordenados
61  void salvaArquivo(char *nome, int *v, int tam, int mudaLinhaFinal){
62      int i;
63      FILE *f = fopen(nome, "a");
64      for(i=0; i<tam-1; i++)
65          fprintf(f, "%d\n", v[i]);
66      if(mudaLinhaFinal == 0)
67          fprintf(f, "%d", v[tam-1]);
68      else
69          fprintf(f, "%d\n", v[tam-1]);
70      fclose(f);
71  }
72
73  int criaArquivosOrdenados(char *nome){
74      int *v = (int*) malloc (N*sizeof(int));
75      char novo[20];
76      int K = 0, total = 0;
77      FILE *f = fopen(nome, "r");
78      while(!feof(f)){
79          fscanf(f, "%d", &v[total]);
80          total++;
81          if(total == N){
82              K++;
83              sprintf(novo, "Temp%d.txt", K);
84              QuickSort(v, 0, N-1);
85              salvaArquivo(novo, v, total, 0);
86              total = 0;
87          }
88      }
89      if(total > 0){
90          K++;
91          sprintf(novo, "Temp%d.txt", K);
92          QuickSort(v, 0, total-1);
93          salvaArquivo(novo, v, total, 0);
94      }
95      fclose(f);
96      free(v);
97      return K;
98  }
99
100
101 //----------------------
102
103 //Multiway Merging
104 void preencheBuffer(Arquivo* arq, int T){
105     int i;
106     if(arq->f == NULL) return;
107     arq->pos = 0;
108     arq->max = 0;
109     for(i=0; i<T; i++){
110         if(!feof(arq->f)){
111             fscanf(arq->f, "%d", &arq->buffer[arq->max]);
112             arq->max++;
113         }else{
114             fclose(arq->f);
115             arq->f = NULL;
116             break;
117         }
```

```
118     }
119 }
120 
121 int procuraMenor(Arquivo* arq, int K, int T, int* menor){
122     int i, idx = -1;
123     for(i=0; i<K; i++){
124         if(arq[i].pos < arq[i].max){
125             if(idx == -1)
126                 idx = i;
127             else{
128                 if(arq[i].buffer[arq[i].pos] < arq[idx].buffer[arq[idx].pos])
129                     idx = i;
130             }
131         }
132     }
133     if(idx != -1){
134         *menor = arq[idx].buffer[arq[idx].pos];
135         arq[idx].pos++;
136         if(arq[idx].pos == arq[idx].max)
137             preencheBuffer(&arq[idx], T);
138         return 1;
139     }else
140         return 0;
141 }
142 
143 
144 void multiWayMerge(char *nome, int K, int T){
145     char novo[20];
146     int i;
147     int *saida = (int*) malloc (T*sizeof(int));
148     Arquivo *arq;
149     arq = (Arquivo*) malloc (K*sizeof(Arquivo));
150     for(i=0; i<K; i++){
151         sprintf(novo, "Temp%d.txt", i+1);
152         arq[i].f = fopen(novo, "r");
153         arq[i].buffer = (int*) malloc (T*sizeof(int));
154         preencheBuffer(&arq[i], T);
155     }
156     int menor, qtdSaida = 0;
157     while(procuraMenor(arq, K, T, &menor) == 1){
158         saida[qtdSaida] = menor;
159         qtdSaida++;
160         if(qtdSaida == T){
161             salvaArquivo(nome, saida, T, 1);
162             qtdSaida = 0;
163         }
164     }
165     if(qtdSaida != 0)
166         salvaArquivo(nome, saida, qtdSaida, 1);
167 
168     for(i=0; i<K; i++) free(arq[i].buffer);
169     free(arq);
170     free(saida);
171 }
172 
173 //----------------------
174 
175 void MergeSortExterno(char* nome){
176     char novo[20];
177     int K = criaArquivosOrdenados(nome);
```

```
178     int i, T = N / (K + 1);
179     remove(nome);
180     multiWayMerge(nome, K, T);
181     for(i=0; i<K; i++){
182         sprintf(novo, "Temp%d.txt", i+1);
183         remove(novo);
184     }
185 }
186 
187 #endif
```

Roteiro 13

1.1
Função criarArquivoTeste():
- Gera um arquivo de teste contendo números aleatórios no intervalo entre os valores ini e fim.
- Utilizada para criar arquivos de entrada destinados ao algoritmo de ordenação externa.

Funções troca(), fparticao() e QuickSort():
- Implementam o algoritmo de ordenação QuickSort, o qual é empregado na criação de arquivos ordenados.

Função salvarArquivo():
- Armazena um conjunto de inteiros em um arquivo, acrescentando os números separados por quebras de linha.

Função criarArquivosOrdenados():
- Lê o arquivo que contém uma sequência de números inteiros, fragmentando-o em sequências menores que são ordenadas por meio do QuickSort.

Funções preencherBuffer() e encontrarMenor():
- Funções auxiliares para executar a operação de merge. A função preencherBuffer() lê T elementos de um arquivo para um buffer, enquanto a função encontrarMenor() identifica o menor elemento entre os buffers dos arquivos.

Função multiWayMerge():
- Realiza o merge dos arquivos ordenados por meio da técnica de merge sort externo.
- Utiliza a função encontrarMenor() para localizar o menor elemento entre os buffers dos arquivos e, posteriormente, os insere no arquivo de saída.

Função MergeSortExterno:
- Conduz o processo de ordenação externa:
    - Invoca criarArquivosOrdenados() para gerar os arquivos ordenados.
    - Exclui o arquivo original (nome).
    - Chama multiWayMerge() para efetuar o merge dos arquivos ordenados, resultando no arquivo final ordenado.
    - Remove os arquivos temporários criados durante o processo.

1.2/ArvoreB.h

```c
/*---------------- File: ArvoreB.h  --------------------+
|TAD: Arvore B                                          |
|                                                       |
| Baseado no material do Prof. Rafael Sachetto          |
| Implementado por Guilherme C. Pena em 02/12/2023      |
+-------------------------------------------------------+ */

#ifndef ARVOREB_H
#define ARVOREB_H

#include <stdio.h>
#include <stdlib.h>

#define M 2
#define MM (M*2) //2M
#define FALSE 0
#define TRUE 1

typedef struct Registro{
    int chave;
    /*outros componentes*/
}Registro;

typedef struct Pagina* Apontador;

typedef struct Pagina{
    int n; //Qtd de registros na pagina
    Registro r[MM];
    Apontador p[MM + 1];
}Pagina;

typedef struct Pagina* ArvoreB;

ArvoreB* criaArvoreB(){
    ArvoreB* raiz;
    raiz = (ArvoreB*) malloc (sizeof(ArvoreB));
    if(raiz != NULL)
        *raiz = NULL;
    return raiz;
}

Pagina* criapagina(){
    Pagina* pag;
    pag = (Pagina*) malloc (sizeof(Pagina));
    pag->n = 0;
    int i;
    for(i=0; i<(MM + 1); i++)
        pag->p[i] = NULL;
    return pag;
}

void destroiPagina(Pagina *pag){
    if(pag == NULL) return;
    int i;
    for(i=0; i<(MM + 1); i++)
        destroiPagina(pag->p[i]);
    free(pag);
```

```c
}

void destroiArvoreB(ArvoreB *raiz){
    if(raiz != NULL){
        destroiPagina(*raiz);
        free(raiz);
    }
}

int estaVazia(ArvoreB* raiz){
    if(raiz == NULL) return 0;
    return (*raiz == NULL);
}

void pre_ordem(Pagina* raiz, int nivel){
    if(raiz != NULL){
        int i;
        printf("Nivel %d: ", nivel);
        for(i=0; i<raiz->n; i++)
            printf("%d ", raiz->r[i].chave);
        printf("\n");

        //int contP = 0;
        //for(i=0; i<MM+1; i++) if(raiz->p[i] != NULL) contP++;
        //printf("%d ponteiros\n", contP);

        for(i=0; i<raiz->n+1; i++)
            pre_ordem(raiz->p[i], nivel+1);
    }
}

void imprimeArvoreB(ArvoreB* raiz){
    if(raiz == NULL) return;
    if(estaVazia(raiz)){
        printf("Arvore B Vazia!\n");
        return;
    }
    pre_ordem(*raiz, 0);
    printf("\n");
}

//Procedimento de PESQUISAR

void pesquisaRec(Registro *x, Apontador ap){
    if (ap == NULL) {
        printf("Registro nao esta presente na arvore\n");
        return;
    }
    int i = 1;
    while (i < ap->n && x->chave > ap->r[i - 1].chave) i++;
    if (x->chave == ap->r[i - 1].chave) {
        printf("Registro (chave %d) encontrado!\n", x->chave);
        *x = ap->r[i - 1]; // Atribui reg
        return;
    }
    if (x->chave < ap->r[i - 1].chave)
        pesquisaRec(x, ap->p[i - 1]);
    else
        pesquisaRec(x, ap->p[i]);
}
```

```
118
119 void pesquisaArvoreB(ArvoreB *raiz, Registro *reg){
120     if(raiz == NULL) return;
121     if(estaVazia(raiz)){
122         printf("Arvore B Vazia!\n");
123         return;
124     }
125     pesquisaRec(reg, *raiz);
126 }
127
128
129 //Procedimento de INSERIR
130
131 void insereNaPagina(Apontador ap, Registro reg, Apontador apDir) {
132     int k, NaoAchouPosicao;
133     k = ap->n;
134     NaoAchouPosicao = (k > 0);
135
136     while (NaoAchouPosicao) {
137         if (reg.chave >= ap->r[k - 1].chave) {
138             NaoAchouPosicao = FALSE;
139             break;
140         }
141
142         ap->r[k] = ap->r[k - 1]; // Atribui reg
143         ap->p[k + 1] = ap->p[k];
144         k--;
145
146         if (k < 1) NaoAchouPosicao = FALSE;
147     }
148
149     ap->r[k] = reg; // Atribui reg
150     ap->p[k + 1] = apDir;
151     ap->n++;
152 }
153
154
155
156 void insereRec(Registro reg, Apontador ap, int *Cresceu, Registro *regRetorno,
    Apontador *apRetorno) {
157     int i = 1, j;
158     Apontador apTemp;
159
160     if (ap == NULL) {
161         printf("Inserio %d..\n", reg.chave);
162         *Cresceu = TRUE;
163         *regRetorno = reg; // Atribui reg
164         *apRetorno = NULL;
165         return;
166     }
167
168     while (i < ap->n && reg.chave > ap->r[i - 1].chave) i++;
169
170     if (reg.chave == ap->r[i - 1].chave) {
171         printf("Erro: Registro ja esta presente\n");
172         *Cresceu = FALSE;
173         return;
174     }
175
176     if (reg.chave < ap->r[i - 1].chave) i--;
```

```
177
178     insereRec(reg, ap->p[i], Cresceu, regRetorno, apRetorno);
179
180     if (!*Cresceu) return;
181
182     if (ap->n < MM) { /* Página tem espaço */
183         insereNaPagina(ap, *regRetorno, *apRetorno);
184         *Cresceu = FALSE;
185         return;
186     }
187
188     /* Overflow: Página tem que ser dividida */
189     //Original Comentado
190     //apTemp = (Pagina*) malloc (sizeof(Pagina));
191     //apTemp->n = 0;
192     //apTemp->p[0] = NULL;
193
194     apTemp = criapagina();
195
196     if (i < (M + 1) ) {
197         insereNaPagina(apTemp, ap->r[MM - 1], ap->p[MM]);
198         ap->n--;
199         ap->p[MM] = NULL;
200         insereNaPagina(ap, *regRetorno, *apRetorno);
201     } else {
202         insereNaPagina(apTemp, *regRetorno, *apRetorno);
203     }
204
205     for (j = M + 2; j <= MM; j++) {
206         insereNaPagina(apTemp, ap->r[j - 1], ap->p[j]);
207         ap->p[j] = NULL;
208     }
209
210     ap->n = M;
211     apTemp->p[0] = ap->p[M + 1];
212     ap->p[M + 1] = NULL;
213     *regRetorno = ap->r[M]; // Atribui reg
214     *apRetorno = apTemp;
215 }
216
217 void insereArvoreB(ArvoreB *raiz, Registro reg) {
218     if(raiz == NULL) return;
219
220     int Cresceu;
221     Registro regRetorno;
222     Pagina *apRetorno, *apTemp;
223
224     insereRec(reg, *raiz, &Cresceu, &regRetorno, &apRetorno);
225
226     if (Cresceu) { /* Árvore cresce na altura pela raiz */
227         //apTemp = (Pagina *) malloc (sizeof(Pagina));
228         apTemp = criapagina();
229         apTemp->n = 1;
230         apTemp->r[0] = regRetorno;
231         apTemp->p[1] = apRetorno;
232         apTemp->p[0] = *raiz;
233         *raiz = apTemp;
234     }
235 }
236
```

```
237
238 //Procedimento de RETIRAR
239 void Reconstitui(Apontador apPag, Apontador apPai, int PosPai, int *Diminuiu) {
240     Pagina *Aux;
241     int DispAux, j;
242
243     if (PosPai < apPai->n) { /* Aux = Pagina a direita de apPag */
244         Aux = apPai->p[PosPai + 1];
245         DispAux = (Aux->n - M + 1) / 2;
246
247         apPag->r[apPag->n] = apPai->r[PosPai];
248         apPag->p[apPag->n + 1] = Aux->p[0];
249         apPag->n++;
250
251         if (DispAux > 0) { /* Existe folga: transfere de Aux para apPag */
252             for (j = 1; j < DispAux; j++)
253                 insereNaPagina(apPag, Aux->r[j - 1], Aux->p[j]);
254
255             apPai->r[PosPai] = Aux->r[DispAux - 1];
256             Aux->n -= DispAux;
257
258             for (j = 0; j < Aux->n; j++)
259                 Aux->r[j] = Aux->r[j + DispAux];
260
261             for (j = 0; j <= Aux->n; j++)
262                 Aux->p[j] = Aux->p[j + DispAux];
263
264             *Diminuiu = FALSE;
265         } else { /* Fusão: intercala Aux em apPag e libera Aux */
266             for (j = 1; j <= M; j++)
267                 insereNaPagina(apPag, Aux->r[j - 1], Aux->p[j]);
268
269             free(Aux);
270
271             for (j = PosPai + 1; j < apPai->n; j++) {
272                 apPai->r[j - 1] = apPai->r[j];
273                 apPai->p[j] = apPai->p[j + 1];
274             }
275
276             apPai->n--;
277
278             if (apPai->n >= M) *Diminuiu = FALSE;
279
280         }
281     } else { /* Aux = Pagina a esquerda de apPag */
282         Aux = apPai->p[PosPai - 1];
283         DispAux = (Aux->n - M + 1) / 2;
284
285         for (j = apPag->n; j >= 1; j--)
286             apPag->r[j] = apPag->r[j - 1];
287
288         apPag->r[0] = apPai->r[PosPai - 1];
289
290         for (j = apPag->n; j >= 0; j--)
291             apPag->p[j + 1] = apPag->p[j];
292
293         apPag->n++;
294
295         if (DispAux > 0) { /* Existe folga: transfere de Aux para apPag */
296             for (j = 1; j < DispAux; j++)
```

```
297                 insereNaPagina(apPag, Aux->r[Aux->n - j], Aux->p[Aux->n - j + 1]);
298
299             apPag->p[0] = Aux->p[Aux->n - DispAux + 1];
300             apPai->r[PosPai - 1] = Aux->r[Aux->n - DispAux];
301             Aux->n -= DispAux;
302             *Diminuiu = 0;
303         } else { /* Fusão: intercala apPag em Aux e libera apPag */
304             for (j = 1; j <= M; j++)
305                 insereNaPagina(Aux, apPag->r[j - 1], apPag->p[j]);
306
307             free(apPag);
308             apPai->n--;
309
310             if (apPai->n >= M) *Diminuiu = FALSE;
311         }
312     }
313 }
314
315 void Antecessor(Apontador ap, int i, Apontador apPai, int *Diminuiu) {
316     if (apPai->p[apPai->n] != NULL) {
317         Antecessor(ap, i, apPai->p[apPai->n], Diminuiu);
318         if (*Diminuiu)
319             Reconstitui(apPai->p[apPai->n], apPai, (int)apPai->n, Diminuiu);
320         return;
321     }
322
323     ap->r[i - 1] = apPai->r[apPai->n - 1];
324     apPai->n--;
325     *Diminuiu = (apPai->n < M);
326 }
327
328 void removeRec(int Ch, Apontador *ap, int *Diminuiu) {
329     int j, i = 1;
330     Apontador Pag;
331
332     if (*ap == NULL) {
333         printf("Erro: registro nao esta na arvore\n");
334         *Diminuiu = FALSE;
335         return;
336     }
337
338     Pag = *ap;
339
340     while (i < Pag->n && Ch > Pag->r[i - 1].chave) i++;
341
342     if (Ch == Pag->r[i - 1].chave) {
343         if (Pag->p[i - 1] == NULL) { /* Pagina folha */
344             Pag->n--;
345             *Diminuiu = (Pag->n < M);
346
347             for (j = i; j <= Pag->n; j++) {
348                 Pag->r[j - 1] = Pag->r[j];
349                 Pag->p[j] = Pag->p[j + 1];
350             }
351
352             return;
353         }
354
355         /* Pagina nao e folha: trocar com antecessor */
356         Antecessor(*ap, i, Pag->p[i - 1], Diminuiu);
```

```
357
358         if (*Diminuiu) Reconstitui(Pag->p[i - 1], *ap, i - 1, Diminuiu);
359
360         return;
361     }
362
363     if (Ch > Pag->r[i - 1].chave) i++;
364
365     removeRec(Ch, &Pag->p[i - 1], Diminuiu);
366
367     if (*Diminuiu) Reconstitui(Pag->p[i - 1], *ap, i - 1, Diminuiu);
368 }
369
370 void removeArvoreB(ArvoreB *raiz, Registro reg) {
371     int Diminuiu;
372     Apontador Aux;
373
374     printf("Removendo %d..\n", reg.chave);
375
376     removeRec(reg.chave, raiz, &Diminuiu);
377
378     if (Diminuiu && (*raiz)->n == 0) { /* Arvore diminui na altura */
379         Aux = *raiz;
380         *raiz = Aux->p[0];
381         free(Aux);
382     }
383 }
384
385
386
387
388 #endif
```

1.2/MainArvoreB.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "ArvoreB.h"

int main(){
    ArvoreB *B = criaArvoreB();
    int valores[21] = {20, 10, 40, 50, 30, 55, 3, 11, 4, 28, 36, 33, 52, 17, 25, 13, 45, 9, 43, 8, 48};
    int remover[17] = {45, 30, 28, 50, 8, 10, 4, 20, 40, 55, 17, 33, 11, 36, 3, 9, 52};

    int i;
    Registro r;
    char c;

    printf("\n\n----Arvore B - INSERCAO:\n");
    for(i=0; i<21; i++){
        r.chave = valores[i];
        insereArvoreB(B, r);
        //imprimeArvoreB(B);
        //printf("\n");
        //c = getchar();
    }
    printf("\n\n----Arvore B - FINAL:\n");
    imprimeArvoreB(B);

    printf("\n\n----Arvore B - PESQUISA:\n");
    for(i=0; i<21; i++){
        r.chave = valores[i];
        pesquisaArvoreB(B, &r);
    }

    printf("\n\n----Arvore B - FINAL:\n");
    imprimeArvoreB(B);

    printf("\n\n----Arvore B - REMOCAO:\n");
    for(i=0; i<17; i++){
        r.chave = remover[i];
        removeArvoreB(B, r);
        //imprimeArvoreB(B);
        //printf("\n\n");
        //c = getchar();
    }

    printf("\n\n----Arvore B - FINAL:\n");
    imprimeArvoreB(B);

    destroiArvoreB(B);
    return 0;
}
```

1.2/Patricia.h

```c
/*---------------- File: Patricia.h  -------------------+
|TAD: Arvore Patricia                                  |
|                                                      |
| Do livro do Ziviani                                  |
| Adaptado por Guilherme C. Pena em 04/12/2023         |
+------------------------------------------------------+ */

#ifndef PATRICIA_H
#define PATRICIA_H

#include <stdio.h>
#include <stdlib.h>

#define D 6 // depende de TipoChave

typedef unsigned char TipoChave; // a definir, depende da aplicacao
typedef unsigned char TipoIndexAmp;

typedef enum {
    Interno,
    Externo
} TipoNo;

typedef struct patriciaNO *ArvorePat;

typedef struct patriciaNO {
    TipoNo nt; //NO type - Uma flag para dizer se o NO eh Externo ou Interno
    union {
        struct {
            TipoIndexAmp Index;
            ArvorePat Esq, Dir;
        } NInterno;
        TipoChave Chave;
    } NO;
} patriciaNO;

//Outra Tentativa da Funcao
int valorBit2(int i, TipoChave k){
    //Todo numero sera considerado com D bits
    //Exemplo: chave 5 em bin (101) -> (00000101)
    //Exemplo: chave 10 em bin (1010) -> (00001010)
    //Exemplo: chave 11 em bin (1011) -> (00001011)

    int n_bits = D;
    int p = 1, j, r;
    //int r = p << n_bits;//R teria o bit 8 ligado (10000000)

    // //Encontra o bit mais a esquerda primeiro
    // //Esse sera o bit 1
    // while((k & r) != r){ n_bits--; r = p << n_bits; }
    // n_bits++;

    //Desloca i bits a partir da esquerda
    for(j=1; j<=i; j++)
        n_bits--;

    if(n_bits < 0) return 0;
```

```
58      r = p << n_bits;
59
60      return ((k & r) == r);//Retorna se o i-esimo bit eh 1 ou 0
61  }
62
63  int retornaNBits(TipoChave k){
64      int n_bits = D;
65      int p = 1;
66      int r = p << n_bits;
67      //Encontra o bit mais a esquerda primeiro
68      //Esse sera o bit 1
69      while((k & r) != r){ n_bits--; r = p << n_bits; }
70      n_bits++;
71      return n_bits;
72  }
73
74  int valorBit(int i, TipoChave k){
75
76      int n_bits = D;
77      int p = 1, j;
78      int r = p << n_bits;
79      //Encontra o bit mais a esquerda primeiro
80      //Esse sera o bit 1
81      while((k & r) != r){ n_bits--; r = p << n_bits; }
82      n_bits++;
83
84
85
86      //Desloca i bits a partir da esquerda
87      for(j=1; j<=i; j++)
88          n_bits--;
89
90      if(n_bits < 0) return 0;
91      r = p << n_bits;
92
93      //printf("Bit[%d] em %d: %d\n", i, k, (k & r) == r);
94
95      return ((k & r) == r);//Retorna se o i-esimo bit eh 1 ou 0
96  }
97
98  int EExterno(ArvorePat p) {
99      // Verifica se p^ eh um nodo externo
100     return (p->nt == Externo);
101 }
102
103 ArvorePat CriaNoInt(int i, ArvorePat *Esq, ArvorePat *Dir) {
104     ArvorePat p;
105     p = (ArvorePat)malloc(sizeof(patriciaNO));
106     p->nt = Interno;
107     p->NO.NInterno.Esq = *Esq;
108     p->NO.NInterno.Dir = *Dir;
109     p->NO.NInterno.Index = i;
110     return p;
111 }
112
113 ArvorePat CriaNoExt(TipoChave k) {
114     ArvorePat p;
115     p = (ArvorePat)malloc(sizeof(patriciaNO));
116     p->nt = Externo;
117     p->NO.Chave = k;
```

```
118     return p;
119 }
120 
121 int Pesquisa(TipoChave k, ArvorePat t) {
122     if (EExterno(t)) {
123         if (k == t->NO.Chave){
124             printf("Elemento %d encontrado!\n", k);
125             return 1;
126         }
127         else{
128             printf("Elemento %d NAO encontrado!\n", k);
129             return 0;
130         }
131     }
132     if (valorBit2(t->NO.NInterno.Index, k) == 0)
133         return Pesquisa(k, t->NO.NInterno.Esq);
134     else
135         return Pesquisa(k, t->NO.NInterno.Dir);
136 }
137 
138 ArvorePat InsereEntre(TipoChave k, ArvorePat *t, int i) {
139     //printf("Insere Entre %d, [%d]\n", k, i);
140     ArvorePat p;
141     if (EExterno(*t) || i < (*t)->NO.NInterno.Index) {
142 
143         int nb1 = retornaNBits((*t)->NO.Chave);
144         int nb2 = retornaNBits(k);
145         // cria um novo no externo
146         p = CriaNoExt(k);
147         if(nb1 <= nb2){
148             if (valorBit2(i, k) == 1)
149                 return CriaNoInt(i, t, &p);
150             else
151                 return CriaNoInt(i, &p, t);
152         }else{
153             return CriaNoInt(i, &p, t);
154         }
155     } else {
156         if (valorBit2((*t)->NO.NInterno.Index, k) == 1)
157             (*t)->NO.NInterno.Dir = InsereEntre(k, &((*t)->NO.NInterno.Dir), i);
158         else
159             (*t)->NO.NInterno.Esq = InsereEntre(k, &((*t)->NO.NInterno.Esq), i);
160         return (*t);
161     }
162 }
163 
164 ArvorePat Insere(TipoChave k, ArvorePat *t) {
165     printf("Inserindo %d..\n", k);
166     ArvorePat p;
167     int i;
168     if (*t == NULL)
169         return CriaNoExt(k);
170     else {
171         if(Pesquisa(k, *t)){
172             printf("Erro: chave ja esta na arvore\n");
173             return (*t);
174         }
175         p = *t;
176         while (!EExterno(p)) {
177             if (valorBit2(p->NO.NInterno.Index, k) == 1)
```

```
178                 p = p->NO.NInterno.Dir;
179             else
180                 p = p->NO.NInterno.Esq;
181         }
182         // acha o primeiro bit diferente
183         i = 1;
184         while ((i <= D) && ( valorBit2((int)i, k) == valorBit2((int)i, p->
    NO.Chave) ) )
185             i++;
186         printf("Bit diferente eh: [%d]\n", i);
187         if (i > D) {
188             //fazer outra coisa
189             //No caso de exemplo, inserir 5 (Bin: 101) antes de 10 (Bin: 1010),
    funciona corretamente
190             //No entanto, ainda nao funciona se inserer 10 antes de 5
191             return InsereEntre(k, t, 1);
192         } else
193             return InsereEntre(k, t, i);
194     }
195 }
196
197
198 void pre_ordem(ArvorePat raiz, int nivel){
199     if(raiz != NULL){
200         printf("Nivel %d: ", nivel);
201         if(raiz->nt == Interno){
202             printf("(INT) %d\n", raiz->NO.NInterno.Index);
203             pre_ordem(raiz->NO.NInterno.Esq, nivel+1);
204             pre_ordem(raiz->NO.NInterno.Dir, nivel+1);
205         }else{
206             printf("(EXT) %d\n", raiz->NO.Chave);
207         }
208     }
209 }
210
211 void imprimePatricia(ArvorePat raiz){
212     if(raiz == NULL) return;
213     pre_ordem(raiz, 0);
214     printf("\n");
215 }
216
217 #endif
```

1.2/MainPatricia.c

```c
#include <stdio.h>
#include "Patricia.h"

void binary(int n){
    if(n<2)
          printf("%d ", n%2);
    else{
          binary(n/2);
      printf("%d ", n%2);
    }
}

int main(){

    int valoresLivro[7] = {18, 19, 24, 33, 40, 54, 34};
    //char valoresLivroChar[7] = {'B', 'C', 'H', 'J', 'Q', 'W', 'K'};

    ArvorePat P = NULL;
    int x, i;

    printf("----------Insercao:\n");

    for(i=0; i<7; i++){
        printf("%d em binario: ", valoresLivro[i]); binary(valoresLivro[i]); printf("\n");
        P = Insere(valoresLivro[i], &P);
        imprimePatricia(P);
    }

    // x = 5;
    // printf("%d em binario: ", x); binary(x); printf("\n");
    // P = Insere(x, &P);
    // imprimePatricia(P);

    // x = 10;
    // printf("%d em binario: ", x); binary(x); printf("\n");
    // P = Insere(x, &P);
    // imprimePatricia(P);

    // x = 11;
    // printf("%d em binario: ", x); binary(x); printf("\n");
    // P = Insere(x, &P);
    // imprimePatricia(P);

    // x = 12;
    // printf("%d em binario: ", x); binary(x); printf("\n");
    // P = Insere(x, &P);
    // imprimePatricia(P);

    printf("\n\n----------Busca:\n");
    x = 9;
    printf("%d em binario: ", x); binary(x); printf("\n");
    Pesquisa(x, P);

    x = 11;
    printf("%d em binario: ", x); binary(x); printf("\n");
    Pesquisa(x, P);
```

```
57
58
59      printf("\n\n----------Impressao FINAL:\n");
60      imprimePatricia(P);
61
62
63      return 0;
64  }
```

1.2/Trie.h

```c
#ifndef TRIE_H
#define TRIE_H

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

// Tamanho do alfabeto (# de símbolos)
#define TAMANHO_ALFABETO (26)

// Converte o caractere atual da chave em indice
// use apenas 'a' a 'z' em minúsculas
#define CHAR_PARA_INDICE(c) ((int)c - (int)'a')

// NO da trie
typedef struct TrieNode{
    struct TrieNode* filhos[TAMANHO_ALFABETO];
    // fimPalavra é verdadeiro se o NO representa
    // o final de uma palavra
    bool fimPalavra;
}NO;

// Retorna um novo NO da trie (inicializado como NULL)
NO* criaNO(){
    NO *p = NULL;
    p = (NO *)malloc(sizeof(NO));
    if (p){
        int i;
        p->fimPalavra = false;
        for (i = 0; i < TAMANHO_ALFABETO; i++)
            p->filhos[i] = NULL;
    }
    return p;
}

// Se não estiver presente, insere a chave na trie
// Se a chave eh um prefixo de um NO da trie, apenas marca o NO folha
void inserir(NO *raiz, const char *chave){
    int nivel;
    int comprimento = strlen(chave);
    int indice;
    NO *aux = raiz;
    for (nivel = 0; nivel < comprimento; nivel++){
        indice = CHAR_PARA_INDICE(chave[nivel]);
        if (!aux->filhos[indice])
            aux->filhos[indice] = criaNO();
        aux = aux->filhos[indice];
    }
    // marca o ultimo NO como folha
    aux->fimPalavra = true;
}

// Retorna verdadeiro se a chave estiver presente na trie, caso contrario, falso
bool buscar(NO *raiz, const char *chave){
    int nivel;
    int comprimento = strlen(chave);
```

```
58      int indice;
59      NO *aux = raiz;
60      for (nivel = 0; nivel < comprimento; nivel++){
61          indice = CHAR_PARA_INDICE(chave[nivel]);
62          if (!aux->filhos[indice])
63              return false;
64          aux = aux->filhos[indice];
65      }
66      return (aux->fimPalavra);
67  }
68
69  void pre_ordem(NO* raiz, int nivel){
70      if(raiz != NULL){
71          int i;
72          for(i=0; i<26; i++){
73              if(raiz->filhos[i] != NULL){
74                  printf("Nivel %d: ", nivel);
75                  printf("%c ", (char) 'a' + i);
76                  if(raiz->filhos[i]->fimPalavra) printf("*");
77                  printf("\n");
78                  pre_ordem(raiz->filhos[i], nivel+1);
79              }
80          }
81
82      }
83  }
84
85  void imprimeTrie(NO* raiz){
86      if(raiz == NULL) return;
87      pre_ordem(raiz, 0);
88      printf("\n");
89  }
90
91  #endif
```

1.2/MainTrie.c

```c
#include <stdio.h>
#include "Trie.h"


// Função principal
int main(){
    // Chaves de entrada (use apenas 'a' a 'z' em minúsculas)
    char chaves[][8] = {"the", "a", "there", "answer", "any", "by", "bye", "their"};
    char saida[][32] = {"Nao encontrada na TRIE", "Encontrada na TRIE"};

    NO *raiz = criaNO();
    // Construir a trie
    int i;
    for (i = 0; i < 8; i++)
        inserir(raiz, chaves[i]);

    // Buscar por diferentes chaves
    printf("%s --- %s\n", "the", saida[buscar(raiz, "the")]);
    printf("%s --- %s\n", "these", saida[buscar(raiz, "these")]);
    printf("%s --- %s\n", "their", saida[buscar(raiz, "their")]);
    printf("%s --- %s\n", "thaw", saida[buscar(raiz, "thaw")]);

    imprimeTrie(raiz);

    return 0;
}
```

2.2

ARVORE B:

```
[lucascosta@fedora 1.2]$ gcc MainArvoreB.c -o main
[lucascosta@fedora 1.2]$ ./main


----Arvore B - INSERCAO:
Inserio 20..
Inserio 10..
Inserio 40..
Inserio 50..
Inserio 30..
Inserio 55..
Inserio 3..
Inserio 11..
Inserio 4..
Inserio 28..
Inserio 36..
Inserio 33..
Inserio 52..
Inserio 17..
Inserio 25..
Inserio 13..
Inserio 45..
Inserio 9..
Inserio 43..
Inserio 8..
Inserio 48..


----Arvore B - FINAL:
Nivel 0: 30
Nivel 1: 10 20
Nivel 2: 3 4 8 9
Nivel 2: 11 13 17
Nivel 2: 25 28
Nivel 1: 40 50
Nivel 2: 33 36
Nivel 2: 43 45 48
Nivel 2: 52 55



----Arvore B - PESQUISA:
Registro (chave 20) encontrado!
Registro (chave 10) encontrado!
Registro (chave 40) encontrado!
Registro (chave 50) encontrado!
Registro (chave 30) encontrado!
Registro (chave 55) encontrado!
Registro (chave 3) encontrado!
Registro (chave 11) encontrado!
Registro (chave 4) encontrado!
Registro (chave 28) encontrado!
Registro (chave 36) encontrado!
Registro (chave 33) encontrado!
Registro (chave 52) encontrado!
Registro (chave 17) encontrado!
```

```
Registro (chave 25) encontrado!
Registro (chave 13) encontrado!
Registro (chave 45) encontrado!
Registro (chave 9) encontrado!
Registro (chave 43) encontrado!
Registro (chave 8) encontrado!
Registro (chave 48) encontrado!


----Arvore B - FINAL:
Nivel 0: 30
Nivel 1: 10 20
Nivel 2: 3 4 8 9
Nivel 2: 11 13 17
Nivel 2: 25 28
Nivel 1: 40 50
Nivel 2: 33 36
Nivel 2: 43 45 48
Nivel 2: 52 55



----Arvore B - REMOCAO:
Removendo 45..
Removendo 30..
Removendo 28..
Removendo 50..
Removendo 8..
Removendo 10..
Removendo 4..
Removendo 20..
Removendo 40..
Removendo 55..
Removendo 17..
Removendo 33..
Removendo 11..
Removendo 36..
Removendo 3..
Removendo 9..
Removendo 52..


----Arvore B - FINAL:
Nivel 0: 13 25 43 48

[lucascosta@fedora 1.2]$
```

.

Árvore Patricia:

```
[lucascosta@fedora 1.2]$ ./main
----------Insercao:
18 em binario: 1 0 0 1 0
Inserindo 18..
Nivel 0: (EXT) 18

19 em binario: 1 0 0 1 1
Inserindo 19..
Elemento 19 NAO encontrado!
Bit diferente eh: [6]
Nivel 0: (INT) 6
Nivel 1: (EXT) 18
Nivel 1: (EXT) 19

24 em binario: 1 1 0 0 0
Inserindo 24..
Elemento 24 NAO encontrado!
Bit diferente eh: [3]
Nivel 0: (INT) 3
Nivel 1: (INT) 6
Nivel 2: (EXT) 18
Nivel 2: (EXT) 19
Nivel 1: (EXT) 24

33 em binario: 1 0 0 0 0 1
Inserindo 33..
Elemento 33 NAO encontrado!
Bit diferente eh: [1]
Nivel 0: (INT) 1
Nivel 1: (INT) 3
Nivel 2: (INT) 6
Nivel 3: (EXT) 18
Nivel 3: (EXT) 19
Nivel 2: (EXT) 24
Nivel 1: (EXT) 33

40 em binario: 1 0 1 0 0 0
Inserindo 40..
Elemento 40 NAO encontrado!
Bit diferente eh: [3]
Nivel 0: (INT) 1
Nivel 1: (INT) 3
Nivel 2: (INT) 6
Nivel 3: (EXT) 18
Nivel 3: (EXT) 19
Nivel 2: (EXT) 24
Nivel 1: (INT) 3
Nivel 2: (EXT) 33
Nivel 2: (EXT) 40

54 em binario: 1 1 0 1 1 0
Inserindo 54..
Elemento 54 NAO encontrado!
Bit diferente eh: [2]
Nivel 0: (INT) 1
Nivel 1: (INT) 3
```

```
Nivel 2: (INT) 6
Nivel 3: (EXT) 18
Nivel 3: (EXT) 19
Nivel 2: (EXT) 24
Nivel 1: (INT) 2
Nivel 2: (INT) 3
Nivel 3: (EXT) 33
Nivel 3: (EXT) 40
Nivel 2: (EXT) 54

34 em binario: 1 0 0 0 1 0
Inserindo 34..
Elemento 34 NAO encontrado!
Bit diferente eh: [5]
Nivel 0: (INT) 1
Nivel 1: (INT) 3
Nivel 2: (INT) 6
Nivel 3: (EXT) 18
Nivel 3: (EXT) 19
Nivel 2: (EXT) 24
Nivel 1: (INT) 2
Nivel 2: (INT) 3
Nivel 3: (INT) 5
Nivel 4: (EXT) 33
Nivel 4: (EXT) 34
Nivel 3: (EXT) 40
Nivel 2: (EXT) 54



----------Busca:
9 em binario: 1 0 0 1
Elemento 9 NAO encontrado!
11 em binario: 1 0 1 1
Elemento 11 NAO encontrado!


----------Impressao FINAL:
Nivel 0: (INT) 1
Nivel 1: (INT) 3
Nivel 2: (INT) 6
Nivel 3: (EXT) 18
Nivel 3: (EXT) 19
Nivel 2: (EXT) 24
Nivel 1: (INT) 2
Nivel 2: (INT) 3
Nivel 3: (INT) 5
Nivel 4: (EXT) 33
Nivel 4: (EXT) 34
Nivel 3: (EXT) 40
Nivel 2: (EXT) 54

[lucascosta@fedora 1.2]$
```

.

Trie

```
[lucascosta@fedora 1.2]$ ./main
the --- Encontrada na TRIE
these --- Nao encontrada na TRIE
their --- Encontrada na TRIE
thaw --- Nao encontrada na TRIE
Nivel 0: a *
Nivel 1: n
Nivel 2: s
Nivel 3: w
Nivel 4: e
Nivel 5: r *
Nivel 2: y *
Nivel 0: b
Nivel 1: y *
Nivel 2: e *
Nivel 0: t
Nivel 1: h
Nivel 2: e *
Nivel 3: i
Nivel 4: r *
Nivel 3: r
Nivel 4: e *

[lucascosta@fedora 1.2]$
```