

- Hashing
  - 2 registros por bloco
  - 77573 buckets
  - blocos de 4096 bytes
  - função de hash = resto da divisão do id / número de buckets
  
- Árvore B+Tree (índice primário)
  - 510 chaves por bloco/nó
  - 511 ponteiros por bloco/nó
  - blocos de 4096 bytes
  - chave e ponteiro são números inteiros, o ponteiro é um offset da raiz para a posição do Bloco.

## TP2

---

OBS: tentar comentar ou descomentar o `#define CSV_IO_NO_THREAD` na primeira linha do arquivo `upload.cpp` para tentar rodar dentro do container.

## Como rodar o projeto

Rodar em 1 comando:

```
make run-all
```

Rodar o container:

```
make run
```

Alocar espaço para o hash:

```
make hash-allocate
```

Compilar os arquivos cpp:

```
make compile-upload  
make compile-findrec  
make compile-seek1
```

To end:

```
make end
```

## Authors

**Student:** Aldemir Rodrigues da Silvar

**Email:** [aldemir.silva@icomp.ufam.edu.br](mailto:aldemir.silva@icomp.ufam.edu.br)

**Student:** Erlon Pereira Bié

**Email:** [erlon.bie@icomp.ufam.edu.br](mailto:erlon.bie@icomp.ufam.edu.br)

**Student:** Glenn Aguiar de Oliveira da Fonseca

**Email:** [glenn.fonseca@icomp.ufam.edu.br](mailto:glenn.fonseca@icomp.ufam.edu.br)

# Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:

|             |  |
|-------------|--|
| csv.h       |  |
| dados.h     |  |
| findrec.cpp |  |
| functions.h |  |
| seek1.cpp   |  |
| upload.cpp  |  |

# Referência do Arquivo dados.h

```
#include <iostream>
```

Vá para o código-fonte desse arquivo.

## Componentes

```
struct tipoArtigoLeitura
```

```
struct tipoArtigo
```

```
struct tipoBloco
```

```
struct dadoBusca
```

```
struct cabecalhoArvore
```

```
struct parNo
```

```
struct noArvore
```

```
struct noArvoreTemp
```

```
struct noArvoreTempPai
```

## Definições e Macros

```
#define QUANTIDADE_PONTEIROS 511
```

```
#define QUANTIDADE_BUCKETS 774573
```

```
#define TAMANHO_BLOCO 4096
```

## Definições e macros

### ◆ QUANTIDADE\_BUCKETS

```
#define QUANTIDADE_BUCKETS 774573
```

### ◆ QUANTIDADE\_PONTEIROS

```
#define QUANTIDADE_PONTEIROS 511
```

### ◆ TAMANHO\_BLOCO

```
#define TAMANHO_BLOCO 4096
```

## Referência do Arquivo findrec.cpp

---

```
#include "csv.h"  
#include "functions.h"  
#include <cmath>  
#include <iostream>
```

### Funções

---

```
int main (int argc, char *argv[])
```

---

### Funções

---

◆ main()

```

int main ( int    argc,
           char * argv[]
         )

{
    FILE *ponteiroHash = fopen("./hash.bin", "rb");
    std::string sEntrada = argv[1];
    int idPesquisa = stoi(sEntrada);

    tipoBloco bloco;
    tipoArtigo artigo;
    bool achou = false;

    fseek(ponteiroHash, funcaoHash(idPesquisa)*TAMANHO_BLOCO, SEEK_SET);
    fread(&bloco, sizeof(tipoBloco), 1, ponteiroHash);

    for(int i = 0; i < bloco.quantidadeArtigos; i++)
    {
        if(bloco.vetorArtigos[i].ID == idPesquisa)
        {
            achou = true;
            artigo = bloco.vetorArtigos[i];
            break;
        }
    }

    if(achou)
    {
        std::cout << "Id: " << artigo.ID << std::endl;
        std::cout << "Título: " << artigo.Titulo << std::endl;
        std::cout << "Ano: " << artigo.Ano << std::endl;
        std::cout << "Autores: " << artigo.Autores << std::endl;
        std::cout << "Citações: " << artigo.Citacoes << std::endl;
        std::cout << "Atualização: " << artigo.Atualizacao << std::endl;
        std::cout << "Snippet: " << artigo.Snippet << std::endl;
        std::cout << "Total de blocos lidos: " << 1 << std::endl; // sempre será
1 aqui por ser um hash
        std::cout << "Total de blocos do arquivo: " << QUANTIDADE_BUCKETS << std
    }
    else
    {
        std::cout << "Registro não encontrado!" << std::endl;
    }

    fclose(ponteiroHash);
    return 0;
}

```

## Referência do Arquivo functions.h

```
#include "dados.h"
#include <bits/stdc++.h>
#include <iostream>
```

Vá para o código-fonte desse arquivo.

### Funções

```
int funcaoHash (int chave)
int contaChaves (noArvore noDado)
int contaPonteiros (noArvore noDado)
void apagaParesNo (noArvore *no)
void copiaParesNo (noArvore *no, noArvoreTemp *tmp, int inicioTemp, int fim)
void copiaTodosParesNo (noArvore *no, noArvoreTemp *tmp)
void moveParesNo (noArvore *no, int pos, int quantidadeChaves)
void moveParesNo (noArvoreTemp *no, int pos, int quantidadeChaves)
int posicaoPai (int node_offset)
void copiaPaiNo (noArvore *P, noArvoreTempPai *TP, int chave, int offsetChave)
void copiaPorPonteiro (noArvore *no, noArvoreTempPai *TP, int inicioTemp, int fim)
void copiaPorPonteiro2 (noArvore *no, noArvoreTempPai *TP, int inicioTemp, int fim)
void printDadosBusca (dadoBusca d)
```

### Variáveis

```
std::vector< int > vetorPais
std::vector< int > vetorPaisFind
```

### Funções

#### ◆ apagaParesNo()

```
void apagaParesNo ( noArvore * no )
```

Apaga os valores de chave e ponteiro, setando para -1

```
45 {
46     for (int i = 0; i < QUANTIDADE_PONTEIROS - 1; i++)
47     {
48         (*no).pares[i].chave = -1;
49         (*no).pares[i].endereco = -1;
50     }
51     (*no).ponteiroM = -1;
52 }
```

## ◆ contaChaves()

```
int contaChaves ( noArvore noDado )
```

Retorna a quantidade de chaves existentes em um nó

```
21 {
22     int counter = 0;
23     while (noDado.pares[counter].chave != -1 && counter != QUANTIDADE_PONTEIROS
24         - 1)
25         counter++;
26     return counter;
```

## ◆ contaPonteiros()

```
int contaPonteiros ( noArvore noDado )
```

Retorna a quantidade de ponteiros existente em um nó

```
31 {
32     int counter = 0;
33     while (noDado.pares[counter].endereco != -1 && counter !=
34         QUANTIDADE_PONTEIROS - 1)
35         counter++;
36     if (noDado.ponteiroM != -1)
37     {
38         counter++;
39     }
40     return counter;
```

## ◆ copiaPaiNo()

```
void copiaPaiNo ( noArvore *      P,
                  noArvoreTempPai * TP,
                  int             chave,
                  int             offsetChave
                  )
```

Copia todos os ponteiros e chaves de um nó **noArvore** para um nó **noArvoreTempPai**, e adiciona chave e offsetChave no final

```
113 {
114     int i;
115     for (i = 0; i < QUANTIDADE_PONTEIROS - 1; i++)
116     {
117         (*TP).pares[i] = (*P).pares[i];
118     }
119     (*TP).pares[i].endereco = (*P).ponteiroM;
120     (*TP).pares[i].chave = chave;
121     (*TP).ponteiroM = offsetChave;
122 }
```



## ◆ copiaParesNo()

```
void copiaParesNo ( noArvore *      no,
                   noArvoreTemp * tmp,
                   int              inicioTemp,
                   int              fim
                   )
```

Copia os valores de um nó auxiliar (**noArvoreTemp**) para o nó **noArvore** de uma determinada posição a outra

```
57 {
58     for (int i = inicioTemp, j = 0; i <= fim; i++, j++)
59     {
60         (*no).pares[j] = (*tmp).pares[i];
61     }
62 }
```

## ◆ copiaPorPonteiro()

```
void copiaPorPonteiro ( noArvore *      no,
                       noArvoreTempPai * TP,
                       int              inicioTemp,
                       int              fim
                       )
```

Copia os valores de um nó **noArvoreTempPai** para um nó **noArvore**, de 0 à teto(N/2)-1

```
127 {
128     int i, j;
129     for (i = inicioTemp, j = 0; i < fim; i++, j++)
130     {
131         (*no).pares[j] = (*TP).pares[i];
132     }
133     (*no).pares[j].endereco = (*TP).pares[i].endereco;
134 }
```

## ◆ copiaPorPonteiro2()

```
void copiaPorPonteiro2 ( noArvore *      no,
                        noArvoreTempPai * TP,
                        int               inicioTemp,
                        int               fim
                      )
```

Copia os valores de um nó **noArvoreTempPai** para um nó **noArvore**, de teto(N/2) à até o fim(N)

```
139 {
140     int i, j;
141     for (i = inicioTemp, j = 0; i < fim; i++, j++)
142     {
143         (*no).pares[j] = (*TP).pares[i];
144     }
145     (*no).pares[j].endereco = (*TP).ponteiroM;
146 }
```

### ◆ copiaTodosParesNo()

```
void copiaTodosParesNo ( noArvore *      no,
                        noArvoreTemp * tmp
                      )
```

Copia todos os valores de **noArvore** para uma **noArvoreTemp**

```
67 {
68     for (int i = 0; i < QUANTIDADE_PONTEIROS - 1; i++)
69     {
70         (*tmp).pares[i] = (*no).pares[i];
71     }
72     (*tmp).pares[QUANTIDADE_PONTEIROS - 1].chave = -1;
73     (*tmp).pares[QUANTIDADE_PONTEIROS - 1].endereco = -1;
74 }
```

### ◆ funcaoHash()

```
int funcaoHash ( int chave )
```

Cria um hash para uma determinada chave

A implementação foi a mais simples possível com uma função mod %

```
14 {
15     return (chave - 1) % QUANTIDADE_BUCKETS;
16 }
```

### ◆ moveParesNo() [1/2]

```
void moveParesNo ( noArvore * no,
                  int      pos,
                  int      quantidadeChaves
                )
```

Desloca os valores de um determinado nó para a inserção de uma nova chave

```
79 {
80   for (int j = quantidadeChaves + 1; j > pos; j--)
81   {
82     (*no).pares[j] = (*no).pares[j - 1];
83   }
84 }
```

#### ◆ moveParesNo() [2/2]

```
void moveParesNo ( noArvoreTemp * no,
                  int      pos,
                  int      quantidadeChaves
                )
```

Desloca os valores de um determinado nó para a inserção de uma nova chave

```
89 {
90   for (int j = quantidadeChaves; j > pos; j--)
91   {
92     (*no).pares[j] = (*no).pares[j - 1];
93   }
94 }
```

#### ◆ posicaoPai()

```
int posicaoPai ( int node_offset )
```

Retorna a posição do pai de um determinado nó em relação ao caminho feito na inserção ou busca

```
99 {
100   for (int i = 0; i < vetorPais.size(); i++)
101   {
102     if (node_offset == vetorPais[i])
103     {
104       return vetorPais[i - 1];
105     }
106   }
107   return 0;
108 }
```

#### ◆ printDadosBusca()

```
void printDadosBusca ( dadoBusca d )
```

Imprime os dados de um bloco no arquivo de hash se a consulta for bem sucedida

```
152 {  
153     std::cout << "ID: " << d.artigoDado.ID << std::endl;  
154     std::cout << "Titulo: " << d.artigoDado.Titulo << std::endl;  
155     std::cout << "Ano : " << d.artigoDado.Ano << std::endl;  
156     std::cout << "Autores: " << d.artigoDado.Autores << std::endl;  
157     std::cout << "Citacoes: " << d.artigoDado.Citacoes << std::endl;  
158     std::cout << "Atualizacao: " << d.artigoDado.Atualizacao << std::endl;  
159     std::cout << "Snippet: " << d.artigoDado.Snippet << std::endl;  
160     std::cout << "Blocos lidos: " << d.node_level << std::endl;  
161     std::cout << "Quantidade de blocos: " << d.quantidadeBlocos + 1 << std::endl;  
162     std::cout << "-----" << std::endl;  
163 }
```

## Variáveis

### ◆ vetorPais

```
std::vector<int> vetorPais
```

### ◆ vetorPaisFind

```
std::vector<int> vetorPaisFind
```

Gerado por **doxygen** 1.9.3

## Referência do Arquivo seek1.cpp

---

```
#include "csv.h"  
#include "functions.h"  
#include <cmath>  
#include <iostream>
```

### Funções

---

**dadoBusca** **buscaNaArvore** (int chave)  
int **main** (int argc, char \*argv[])

---

### Variáveis

---

FILE \* **ponteiroArvore**

---

### Funções

---

◆ **buscaNaArvore()**

**dadosBusca** buscaNaArvore ( int chave )

Busca uma chave na árvore (índice).

- Se a busca for bem sucedida, retorna as informações da registro cuja chave corresponde à chave de busca
- Caso contrário, retorna uma mensagem informando que ao registro não pode ser encontrado

```

15 {
16     FILE *ponteiroHash = fopen("./hash.bin", "r");
17     cabecalhoArvore dadoCabecalho;
18     noArvore noDado;
19     tipoBloco bloco;
20     int noAtual;
21     fseek(ponteiroArvore, 0, SEEK_SET);
22     fread(&dadoCabecalho, sizeof(cabecalhoArvore), 1,
23         ponteiroArvore);
24     if (dadoCabecalho.enderecoRaiz == -1)
25     {
26         std::cout << "Error: Ávore vazia!!" << std::endl;
27     }
28     else
29     {
30         fseek(ponteiroArvore, 0, SEEK_SET);
31         fread(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
32         fseek(ponteiroArvore, (dadoCabecalho.enderecoRaiz * TAMANHO_BLOCO),
33             SEEK_CUR);
34
35         int nivelAtual = 1;
36         noAtual = dadoCabecalho.enderecoRaiz;
37         while (nivelAtual !=
38             dadoCabecalho.alturaArvore)
39         {
40             int i = 0;
41             fread(&noDado, sizeof(noArvore), 1,
42                 ponteiroArvore);
43             while (true)
44             {
45 chave)         if (i == QUANTIDADE_PONTEIROS - 1 && noDado.pares[i - 1].chave <
46                 {
47                     noAtual = noDado.ponteiroM;
48                     fseek(ponteiroArvore, sizeof(cabecalhoArvore) + noAtual * TAMANHO_BLOCO,
49                         SEEK_SET);
50                     nivelAtual++;
51                     break;
52                 }
53                 else if (noDado.pares[i].chave > chave ||
54                     noDado.pares[i].chave ==
55                         -1)
56                 {
57                     noAtual = noDado.pares[i].endereco;
58                     fseek(ponteiroArvore, sizeof(cabecalhoArvore) + noAtual * TAMANHO_BLOCO,
59                         SEEK_SET);
60                     nivelAtual++;
61                     break;
62                 }
63                 i++;
64             }
65         }
66         fread(&noDado, sizeof(noArvore), 1, ponteiroArvore);
67         int quantidadeChaves = contaChaves(noDado);
68         for (int i = 0; i < quantidadeChaves; i++)
69         {
70             if (noDado.pares[i].chave == chave)
71             {
72                 int offsetChave = noDado.pares[i].endereco;
73                 fseek(ponteiroHash, offsetChave * TAMANHO_BLOCO,
74                     SEEK_SET);
75                 fread(&bloco, TAMANHO_BLOCO, 1,
76                     ponteiroHash);
77                 for (int j = 0; j < bloco.quantidadeArtigos; j++)
78                 {
79                     if (bloco.vetorArtigos[j].ID == chave) {

```

```
80         dadoBusca result = {bloco.vetorArtigos[j], dadoCabecalho.quantidadeBlocos,
81                               dadoCabecalho.alturaArvore};
82         printDadosBusca(result);
83         fclose(ponteiroHash);
84         return result;
85     }
86 }
87 }
88 }
89 }
90 std::cout << "Não encontrou a chave" << std::endl;
91 fclose(ponteiroHash);
92 return {{}, dadoCabecalho.quantidadeBlocos, dadoCabecalho.alturaArvore};
93 }
```

## ◆main()

```
int main ( int    argc,
           char * argv[]
        )
```

```
96 {
97     std::string sEntrada = argv[1];
98     int nomeEntrada = stoi(sEntrada);
99     ponteiroArvore = fopen("./primarytree.bin", "r");
100     buscaNaArvore(nomeEntrada);
101 }
102
103 return 0;
104 }
```

## Variáveis

### ◆ponteiroArvore

FILE\* ponteiroArvore

## Referência do Arquivo upload.cpp

---

```
#include "csv.h"
#include "functions.h"
#include <cmath>
#include <iostream>
```

### Definições e Macros

---

```
#define CSV_IO_NO_THREAD
```

---

### Funções

---

```
void insereNaFolha (noArvore *no, int chave, int P, int quantidadeChaves)
void insereNaFolha (noArvoreTemp *tmp, int chave, int P)
void insereNoPai (noArvore *no, int chave, int P, int offsetNo)
void insere (int chave, int P)
int main (int argc, char *argv[])
```

---

### Variáveis

---

```
FILE * ponteiroArvore
```

---

### Definições e macros

---

◆ CSV\_IO\_NO\_THREAD

```
#define CSV_IO_NO_THREAD
```

### Funções

---

◆ insere()



```
void insere ( int chave,
             int P
            )
```

Função principal que começa o processo de inserção.

```
166 {
167     vetorPais.clear();
168     cabecalhoArvore dadoCabecalho;
169     noArvore noDado;
170     int noAtual;
171     fseek(ponteiroArvore, 0, SEEK_SET);
172     fread(&dadoCabecalho, sizeof(cabecalhoArvore), 1,
173         ponteiroArvore);
174     if (dadoCabecalho.enderecoRaiz == -1)
175     {
176         noArvore noVazio;
177         for (int i = 0; i < QUANTIDADE_PONTEIROS - 1; i++)
178         {
179             noVazio.pares[i].chave = -1;
180             noVazio.pares[i].endereco = -1;
181         }
182         noVazio.ponteiroM = -1;
183         dadoCabecalho.enderecoRaiz = 0;
184         dadoCabecalho.alturaArvore = 1;
185         fwrite(&noVazio, sizeof(noArvore), 1, ponteiroArvore);
186         fseek(ponteiroArvore, 0, SEEK_SET);
187         fwrite(&dadoCabecalho, sizeof(cabecalhoArvore), 1,
188             ponteiroArvore);
189     }
190     else
191     {
192         fseek(ponteiroArvore, 0, SEEK_SET);
193         fread(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
194         fseek(ponteiroArvore, (dadoCabecalho.enderecoRaiz * TAMANHO_BLOCO),
195             SEEK_CUR);
196
197         int nivelAtual = 1;
198         noAtual = dadoCabecalho.enderecoRaiz;
199         while (nivelAtual !=
200             dadoCabecalho.alturaArvore)
201         {
202             int i = 0;
203             fread(&noDado, sizeof(noArvore), 1,
204                 ponteiroArvore);
205             vetorPais.push_back(noAtual);
206             while (true)
207             {
208 chave)         if (i == QUANTIDADE_PONTEIROS - 1 && noDado.pares[i - 1].chave <
209                 {
210                     noAtual = noDado.ponteiroM;
211                     fseek(ponteiroArvore, sizeof(cabecalhoArvore) + noAtual * TAMANHO_BLOCO,
212                         SEEK_SET);
213                     nivelAtual++;
214                     break;
215                 }
216                 else if (noDado.pares[i].chave > chave ||
217                     noDado.pares[i].chave ==
218                         -1)
219                 {
220                     noAtual = noDado.pares[i].endereco;
221                     fseek(ponteiroArvore, sizeof(cabecalhoArvore) + noAtual * TAMANHO_BLOCO,
222                         SEEK_SET);
223                     nivelAtual++;
224                     break;
225                 }
226             }
227             i++;
228         }
229     }
230     vetorPais.push_back(noAtual);
231     fread(&noDado, sizeof(noArvore), 1,
232         ponteiroArvore);
```

```

233 int quantidadeChaves = contaChaves(noDado);
234 if (quantidadeChaves < QUANTIDADE_PONTEIROS - 1)
235 {
236     insereNaFolha(&noDado, chave, P, quantidadeChaves);
237     fseek(ponteiroArvore, -sizeof(noArvore), SEEK_CUR);
238     fwrite(&noDado, sizeof(noArvore), 1, ponteiroArvore);
239 }
240 else
241 {
242     noArvore novoNoL;
243     apagaParesNo(&novoNoL);
244     noArvoreTemp tmp;
245     copiaTodosParesNo(&noDado, &tmp);
246     insereNaFolha(&tmp, chave, P);
247
248     novoNoL.ponteiroM = noDado.ponteiroM;
249     int novoNoOffsetL = dadoCabecalho.quantidadeBlocos + 1;
250     dadoCabecalho.quantidadeBlocos++;
251     noDado.ponteiroM = novoNoOffsetL;
252     apagaParesNo(&noDado);
253     copiaParesNo(&noDado, &tmp, 0, ceil(QUANTIDADE_PONTEIROS / 2.0) - 1); //
254     talvez voltar pra inteiro
255     copiaParesNo(&novoNoL, &tmp, ceil(QUANTIDADE_PONTEIROS / 2.0),
256     QUANTIDADE_PONTEIROS - 1); // talvez voltar pra inteiro
257     int menoK = novoNoL.pares[0].chave;
258     fseek(ponteiroArvore, -sizeof(noArvore), SEEK_CUR);
259     fwrite(&noDado, sizeof(noArvore), 1, ponteiroArvore);
260     fseek(ponteiroArvore, sizeof(cabecalhoArvore) + novoNoOffsetL *
261     TAMANHO_BLOCO, SEEK_SET);
262     fwrite(&novoNoL, sizeof(noArvore), 1, ponteiroArvore);
263     fseek(ponteiroArvore, 0, SEEK_SET);
264     fwrite(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
265     insereNoPai(&noDado, menoK, novoNoOffsetL, noAtual);
266 }
267 }

```

◆insereNaFolha() [1/2]

```
void insereNaFolha ( noArvore * no,
                    int      chave,
                    int      P,
                    int      quantidadeChaves
                    )
```

Essa função insere na folha sempre quando há espaço no nó

```
12 {
13     int i;
14     if (chave < (*no).pares[0].chave)
15     {
16         moveParesNo(no, 0, quantidadeChaves);
17         (*no).pares[0].chave = chave;
18         (*no).pares[0].endereco = P;
19     }
20     else
21     {
22         if (quantidadeChaves == 0)
23         {
24             (*no).pares[i + 1].chave = chave;
25             (*no).pares[i + 1].endereco = P;
26         }
27         else
28         {
29             for (i = quantidadeChaves - 1; i >= 0; i--)
30             {
31                 if (chave >= (*no).pares[i].chave)
32                 {
33                     moveParesNo(no, i + 1, quantidadeChaves);
34                     (*no).pares[i + 1].chave = chave;
35                     (*no).pares[i + 1].endereco = P;
36                     break;
37                 }
38             }
39         }
40     }
41 }
```

◆insereNaFolha() [2/2]

```
void insereNaFolha ( noArvoreTemp * tmp,  
                    int           chave,  
                    int           P  
                    )
```

Função chamada dentro da função insere quando a quantidade de chaves não é menor que N-1

```
46 {  
47     if (chave < (*tmp).pares[0].chave)  
48     {  
49         moveParesNo(tmp, 0, QUANTIDADE_PONTEIROS - 1);  
50         (*tmp).pares[0].chave = chave;  
51         (*tmp).pares[0].endereco = P;  
52         return;  
53     }  
54     for (int i = QUANTIDADE_PONTEIROS - 2; i >= 0; i--)  
55     {  
56         if (chave >= (*tmp).pares[i].chave)  
57         {  
58             moveParesNo(tmp, i + 1, QUANTIDADE_PONTEIROS - 1);  
59             (*tmp).pares[i + 1].chave = chave;  
60             (*tmp).pares[i + 1].endereco = P;  
61             break;  
62         }  
63     }  
64 }
```

◆insereNoPai()

```

void insereNoPai ( noArvore * no,
                  int      chave,
                  int      P,
                  int      offsetNo
                )

```

Essa função é chamada quando não há espaço suficiente na folha

É feito então um split das folhas, transferindo a chave a ser inserida para um nó pai.

Se esse processo não for suficiente, a chave a ser inserida é propagada até chegar na raiz da árvore

```

73 {
74   if (vetorPais[0] == offsetNo)
75   {
76     noArvore novaRaiz;
77     apagaParesNo(&novaRaiz);
78     novaRaiz.pares[0].chave = chave;
79     novaRaiz.pares[0].endereco = offsetNo;
80     novaRaiz.pares[1].endereco = P;
81
82     cabecalhoArvore dadoCabecalho;
83     fseek(ponteiroArvore, 0, SEEK_SET);
84     fread(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
85
86     dadoCabecalho.quantidadeBlocos += 1;
87     dadoCabecalho.alturaArvore += 1;
88     dadoCabecalho.enderecoRaiz = dadoCabecalho.quantidadeBlocos;
89
90     fseek(ponteiroArvore, sizeof(cabecalhoArvore) + TAMANHO_BLOCO * dadoCabeca
91           SEEK_SET);
92     fwrite(&novaRaiz, sizeof(noArvore), 1, ponteiroArvore);
93
94     fseek(ponteiroArvore, 0, SEEK_SET);
95     fwrite(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
96     return;
97   }
98
99   int pai = posicaoPai(offsetNo);
100
101   noArvore dadoPai;
102
103   fseek(ponteiroArvore, sizeof(cabecalhoArvore) + pai * TAMANHO_BLOCO, SEEK_SE
104   fread(&dadoPai, sizeof(noArvore), 1, ponteiroArvore);
105
106   int quantidadePonteiro = contaPonteiros(dadoPai);
107
108   if (quantidadePonteiro < QUANTIDADE_PONTEIROS)
109   {
110     int i = 0;
111     while (dadoPai.pares[i].endereco != offsetNo)
112       i++;
113
114     if (i == QUANTIDADE_PONTEIROS - 2)
115     {
116       dadoPai.pares[i].chave = chave;
117       dadoPai.ponteiroM = P;
118     }
119     else
120     {
121       dadoPai.pares[i].chave = chave;
122       dadoPai.pares[i + 1].endereco = P;
123     }
124     fseek(ponteiroArvore, -sizeof(noArvore), SEEK_CUR);
125     fwrite(&dadoPai, sizeof(noArvore), 1, ponteiroArvore);
126   }
127   else
128   {
129     noArvoreTempPai TP;
130     copiaPaiNo(&dadoPai, &TP, chave, P);
131     apagaParesNo(&dadoPai);

```

```
132
133     noArvore dadoNovoPai;
134     apagaParesNo(&dadoNovoPai);
135
136     copiaPorPonteiro(&dadoPai, &TP, 0, (ceil(QUANTIDADE_PONTEIROS / 2.0) - 1))
137     int indicePaiK =
138         ceil(QUANTIDADE_PONTEIROS / 2.0) - 1;
139     int paiK = TP.pares[indicePaiK].chave;
140
141     fseek(ponteiroArvore, sizeof(cabecalhoArvore) + TAMANHO_BLOCO * pai, SEEK_
142     fwrite(&dadoPai, sizeof(noArvore), 1, ponteiroArvore);
143
144     copiaPorPonteiro2(&dadoNovoPai, &TP, (ceil(QUANTIDADE_PONTEIROS / 2.0)),
145         QUANTIDADE_PONTEIROS);
146
147     cabecalhoArvore dadoCabecalho;
148     fseek(ponteiroArvore, 0, SEEK_SET);
149     fread(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
150
151     int dadoPosNovoPai = dadoCabecalho.quantidadeBlocos + 1;
152     fseek(ponteiroArvore, dadoPosNovoPai * TAMANHO_BLOCO, SEEK_CUR);
153     fwrite(&dadoNovoPai, sizeof(noArvore), 1, ponteiroArvore);
154
155     dadoCabecalho.quantidadeBlocos += 1;
156     fseek(ponteiroArvore, 0, SEEK_SET);
157     fwrite(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
158
159     insereNoPai(&dadoPai, paiK, dadoPosNovoPai, pai);
160 }
161 }
```

◆main()

```

int main ( int    argc,
           char * argv[]
         )

{
    268 {
    269     bool leitura = true;
    270     std::string nomeEntrada = argv[1];
    271     ponteiroArvore = fopen("./primarytree.bin", "w+");
    272     cabecalhoArvore a_cabecalho;
    273     a_cabecalho.alturaArvore = 0;
    274     a_cabecalho.quantidadeBlocos = 0;
    275     a_cabecalho.enderecoRaiz = -1;
    276     fwrite(&a_cabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
    277     io::CSVReader<7, io::trim_chars<>, io::double_quote_escape<'>', '\\\">> sample
    278     sample.set_header("ID", "Titulo", "Ano", "Autores", "Citacoes", "Atualizacao
    279                       "Snippet");
    280     FILE *ponteiroHash = fopen("./hash.bin", "r+");
    281
    282     if(ponteiroHash == NULL){
    283         std::cout << "Error create hash.bin\n";
    284         return 0;
    285     }
    286
    287     for(int i = 0; i < QUANTIDADE_BUCKETS ; i++){
    288         tipoBloco bloco;
    289         fwrite(&bloco, TAMANHO_BLOCO, 1, ponteiroHash);
    290     }
    291
    292     tipoArtigoLeitura *ta_aux =
    293     (tipoArtigoLeitura *)malloc(sizeof(tipoArtigoLeitura));
    294     while (leitura)
    295     {
    296         try
    297         {
    298             if (leitura = sample.read_row(ta_aux->ID, ta_aux->Titulo, ta_aux->Ano,
    299                                           ta_aux->Autores, ta_aux->Citacoes,
    300                                           ta_aux->Atualizacao, ta_aux->Snippet))
    301             {
    302                 int id = std::stoi(ta_aux->ID);
    303                 insere(id, funcaoHash(id));
    304
    305                 tipoArtigo artigo;
    306                 tipoBloco bloco;
    307
    308                 artigo.ID = stoi(ta_aux->ID);
    309                 strcpy(artigo.Titulo, ta_aux->Titulo.c_str());
    310                 artigo.Ano = stoi(ta_aux->Ano);
    311                 strcpy(artigo.Autores, ta_aux->Autores.c_str());
    312                 artigo.Citacoes = stoi(ta_aux->Citacoes);
    313                 strcpy(artigo.Atualizacao, ta_aux->Atualizacao.c_str());
    314                 strcpy(artigo.Snippet, ta_aux->Snippet.c_str());
    315
    316
    317                 fseek(ponteiroHash, funcaoHash(artigo.ID)*TAMANHO_BLOCO, SEEK_SET);
    318                 fread(&bloco, TAMANHO_BLOCO, 1, ponteiroHash);
    319                 bloco.vetorArtigos[bloco.quantidadeArtigos] = artigo;
    320                 bloco.quantidadeArtigos++;
    321                 fseek(ponteiroHash, funcaoHash(artigo.ID)*TAMANHO_BLOCO, SEEK_SET);
    322                 fwrite(&bloco, TAMANHO_BLOCO, 1, ponteiroHash);
    323
    324                 std::cout << "Inserindo id: " << ta_aux->ID << std::endl;
    325             }
    326         }
    327         catch (io::error::too_few_columns) {}
    328         catch (io::error::escaped_string_not_closed) {}
    329     }
    330     free(ta_aux);
    331     fclose(ponteiroArvore);
    332     fclose(ponteiroHash);
    333
    334     return 0;
    335 }

```

# Variáveis

---

◆ ponteiroArvore

FILE\* ponteiroArvore

Gerado por  1.9.3