- 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

Student: Erlon Pereira Bié
Email: erlon.bie@icomp.ufam.edu.br

Student: Glenn Aguiar de Oliveira da Fonseca
Email: glenn.fonseca@icomp.ufam.edu.br

Lista de Arquivos

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

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[]
         )
```

```
6                                       {
7       FILE *ponteiroHash = fopen("./hash.bin", "rb");
8       std::string sEntrada = argv[1];
9       int idPesquisa = stoi(sEntrada);
10
11      tipoBloco bloco;
12      tipoArtigo artigo;
13      bool achou = false;
14
15      fseek(ponteiroHash, funcaoHash(idPesquisa)*TAMANHO_BLOCO, SEEK_SET);
16      fread(&bloco, sizeof(tipoBloco), 1, ponteiroHash);
17
18      for(int i = 0; i < bloco.quantidadeArtigos; i++)
19      {
20          if(bloco.vetorArtigos[i].ID == idPesquisa)
21          {
22              achou = true;
23              artigo = bloco.vetorArtigos[i];
24              break;
25          }
26      }
27
28      if(achou)
29      {
30        std::cout << "Id: " << artigo.ID << std::endl;
31        std::cout << "Título: " << artigo.Titulo << std::endl;
32        std::cout << "Ano: " << artigo.Ano << std::endl;
33        std::cout << "Autores: " << artigo.Autores << std::endl;
34        std::cout << "Citações: " << artigo.Citacoes << std::endl;
35        std::cout << "Atualização: " << artigo.Atualizacao << std::endl;
36        std::cout << "Snippet: " << artigo.Snippet << std::endl;
37        std::cout << "Total de blocos lidos: " << 1 << std::endl; // sempre será
  1 aqui por ser um hash
38        std::cout << "Total de blocos do arquivo: " << QUANTIDADE_BUCKETS << std
39      }
40      else
41      {
42        std::cout << "Registro não encontrado!" << std::endl;
43      }
44
45      fclose(ponteiroHash);
46      return 0;
47  }
```

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 quntidadeChaves)
void	**moveParesNo** (**noArvoreTemp** *no, int pos, int quntidadeChaves)
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

Autor: Aldemir

```
53 {
54   for (int i = 0; i < QUANTIDADE_PONTEIROS - 1; i++)
55   {
56     (*no).pares[i].chave = -1;
57     (*no).pares[i].endereco = -1;
58   }
59   (*no).ponteiroM = -1;
60 }
```

◆ contaChaves()

int contaChaves (**noArvore** noDado)

Retorna a quantidade de chaves existentes em um nó *

Autor: Erlon

```
25 {
26   int counter = 0;
27   while (noDado.pares[counter].chave != -1 && counter != QUANTIDADE_PONTEIROS
   - 1)
28     counter++;
29   return counter;
30 }
```

◆ contaPonteiros()

int contaPonteiros (**noArvore** noDado)

Retorna a quantidade de ponteiros existente em um nó

Autor: Glenn

```
37 {
38   int counter = 0;
39   while (noDado.pares[counter].endereco != -1 && counter !=
  QUANTIDADE_PONTEIROS - 1)
40     counter++;
41   if (noDado.ponteiroM != -1)
42   {
43     counter++;
44   }
45   return counter;
46 }
```

◆ copiaPaiNo()

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

Copia todos os ponteiros e chavesde um nó **noArvore** para um nó **noArvoreTempPai**, e aadiciona chave e offsetChave no final

Autor: Erlon

```
133 {
134   int i;
135   for (i = 0; i < QUANTIDADE_PONTEIROS - 1; i++)
136   {
137     (*TP).pares[i] = (*P).pares[i];
138   }
139   (*TP).pares[i].endereco = (*P).ponteiroM;
140   (*TP).pares[i].chave = chave;
141   (*TP).ponteiroM = offsetChave;
142 }
```

◆ copiaParesNo()

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

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

Autor: Erlon

```
67 {
68   for (int i = inicioTemp, j = 0; i <= fim; i++, j++)
69   {
70     (*no).pares[j] = (*tmp).pares[i];
71   }
72 }
```

◆ 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

Autor: Aldemir

```
149 {
150   int i, j;
151   for (i = inicioTemp, j = 0; i < fim; i++, j++)
152   {
153     (*no).pares[j] = (*TP).pares[i];
154   }
155   (*no).pares[j].endereco = (*TP).pares[i].endereco;
156 }
```

◆ 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)

Autor: Glenn

```
163 {
164   int i, j;
165   for (i = inicioTemp, j = 0; i < fim; i++, j++)
166   {
167     (*no).pares[j] = (*TP).pares[i];
168   }
169   (*no).pares[j].endereco = (*TP).ponteiroM;
170 }
```

◆ copiaTodosParesNo()

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

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

Autor: Glenn

```
79 {
80   for (int i = 0; i < QUANTIDADE_PONTEIROS - 1; i++)
81   {
82     (*tmp).pares[i] = (*no).pares[i];
83   }
84   (*tmp).pares[QUANTIDADE_PONTEIROS - 1].chave = -1;
85   (*tmp).pares[QUANTIDADE_PONTEIROS - 1].endereco = -1;
86 }
```

◆ funcaoHash()

int funcaoHash (int chave)

Cria um hash para uma determinada chave

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

Autor: Aldemir

```
16 {
17   return (chave - 1) % QUANTIDADE_BUCKETS;
18 }
```

◆ moveParesNo() [1/2]

void moveParesNo (**noArvore** * no,
int pos,
int quntidadeChaves
)

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

Autor: Aldemir

```
93 {
94   for (int j = quntidadeChaves + 1; j > pos; j--)
95   {
96     (*no).pares[j] = (*no).pares[j - 1];
97   }
98 }
```

◆ moveParesNo() [2/2]

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

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

Autor: Erlon

```
105 {
106   for (int j = quntidadeChaves; j > pos; j--)
107   {
108     (*no).pares[j] = (*no).pares[j - 1];
109   }
110 }
```

posicaoPai()

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

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

Autor: Glenn

```
117 {
118   for (int i = 0; i < vetorPais.size(); i++)
119   {
120     if (node_offset == vetorPais[i])
121     {
122       return vetorPais[i - 1];
123     }
124   }
125   return 0;
126 }
```

printDadosBusca()

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

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

Autor: Erlon

```
178 {
179   std::cout << "ID: " << d.artigoDado.ID << std::endl;
180   std::cout << "Titulo: " << d.artigoDado.Titulo << std::endl;
181   std::cout << "Ano : " << d.artigoDado.Ano << std::endl;
182   std::cout << "Autores: " << d.artigoDado.Autores << std::endl;
183   std::cout << "Citacoes: " << d.artigoDado.Citacoes << std::endl;
184   std::cout << "Atualizacao: " << d.artigoDado.Atualizacao << std::endl;
185   std::cout << "Snippet: " << d.artigoDado.Snippet << std::endl;
186   std::cout << "Blocos lidos: " << d.node_level << std::endl;
187   std::cout << "Quantidade de blocos: " << d.quantidadeBlocos + 1 << std::endl
188   std::cout << "-----------------------------------------------" << std::endl;
189 }
```

Variáveis

◆ vetorPais

std::vector<int> vetorPais

◆ vetorPaisFind

std::vector<int> vetorPaisFind

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()

dadoBusca 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

Autor: Erlon

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

```
80          {
81            if (bloco.vetorArtigos[j].ID == chave) {
82              dadoBusca result = {bloco.vetorArtigos[j], dadoCabecalho.quantidad
83                                  dadoCabecalho.alturaArvore};
84              printDadosBusca(result);
85              fclose(ponteiroHash);
86              return result;
87            }
88          }
89        }
90      }
91    }
92    std::cout << "Não encontrou a chave" << std::endl;
93    fclose(ponteiroHash);
94    return {{}, dadoCabecalho.quantidadeBlocos, dadoCabecalho.alturaArvore};
95  }
```

◆ main()

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

```
98  {
99
100   std::string sEntrada = argv[1];
101   int nomeEntrada = stoi(sEntrada);
102   ponteiroArvore = fopen("./primarytree.bin", "r");
103   buscaNaArvore(nomeEntrada);
104
105   return 0;
106 }
```

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 quntidadeChaves)
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.

Autores: Erlon, Glenn e Aldemir

```
174 {
175   vetorPais.clear();
176   cabecalhoArvore dadoCabecalho;
177   noArvore noDado;
178   int noAtual;
179   fseek(ponteiroArvore, 0, SEEK_SET);
180   fread(&dadoCabecalho, sizeof(cabecalhoArvore), 1,
181         ponteiroArvore);
182   if (dadoCabecalho.enderecoRaiz == -1)
183   {
184     noArvore noVazio;
185     for (int i = 0; i < QUANTIDADE_PONTEIROS - 1; i++)
186     {
187       noVazio.pares[i].chave = -1;
188       noVazio.pares[i].endereco = -1;
189     }
190     noVazio.ponteiroM = -1;
191     dadoCabecalho.enderecoRaiz = 0;
192     dadoCabecalho.alturaArvore = 1;
193     fwrite(&noVazio, sizeof(noArvore), 1, ponteiroArvore);
194     fseek(ponteiroArvore, 0, SEEK_SET);
195     fwrite(&dadoCabecalho, sizeof(cabecalhoArvore), 1,
196            ponteiroArvore);
197   }
198   else
199   {
200     fseek(ponteiroArvore, 0, SEEK_SET);
201     fread(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
202     fseek(ponteiroArvore, (dadoCabecalho.enderecoRaiz * TAMANHO_BLOCO),
203           SEEK_CUR);
204
205     int nivelAtual = 1;
206     noAtual = dadoCabecalho.enderecoRaiz;
207     while (nivelAtual !=
208            dadoCabecalho.alturaArvore)
209     {
210       int i = 0;
211       fread(&noDado, sizeof(noArvore), 1,
212             ponteiroArvore);
213       vetorPais.push_back(noAtual);
214       while (true)
215       {
216         if (i == QUANTIDADE_PONTEIROS - 1 && noDado.pares[i - 1].chave <
chave)
217         {
218           noAtual = noDado.ponteiroM;
219           fseek(ponteiroArvore, sizeof(cabecalhoArvore) + noAtual * TAMANHO_BL
220                 SEEK_SET);
221           nivelAtual++;
222           break;
223         }
224         else if (noDado.pares[i].chave > chave ||
225                  noDado.pares[i].chave ==
226                      -1)
227         {
228           noAtual = noDado.pares[i].endereco;
229           fseek(ponteiroArvore, sizeof(cabecalhoArvore) + noAtual * TAMANHO_BL
230                 SEEK_SET);
231           nivelAtual++;
232           break;
233         }
234         i++;
235       }
236     }
237   }
238   vetorPais.push_back(noAtual);
```

```
239     fread(&noDado, sizeof(noArvore), 1,
240           ponteiroArvore);
241     int quntidadeChaves = contaChaves(noDado);
242     if (quntidadeChaves < QUANTIDADE_PONTEIROS - 1)
243     {
244       insereNaFolha(&noDado, chave, P, quntidadeChaves);
245       fseek(ponteiroArvore, -sizeof(noArvore), SEEK_CUR);
246       fwrite(&noDado, sizeof(noArvore), 1, ponteiroArvore);
247     }
248     else
249     {
250       noArvore novoNoL;
251       apagaParesNo(&novoNoL);
252       noArvoreTemp tmp;
253       copiaTodosParesNo(&noDado, &tmp);
254       insereNaFolha(&tmp, chave, P);
255 
256       novoNoL.ponteiroM = noDado.ponteiroM;
257       int novoNoOffsetL = dadoCabecalho.quantidadeBlocos + 1;
258       dadoCabecalho.quantidadeBlocos++;
259       noDado.ponteiroM = novoNoOffsetL;
260       apagaParesNo(&noDado);
261       copiaParesNo(&noDado, &tmp, 0, ceil(QUANTIDADE_PONTEIROS / 2.0) - 1); //
    talvez voltar pra inteiro
262       copiaParesNo(&novoNoL, &tmp, ceil(QUANTIDADE_PONTEIROS / 2.0),
    QUANTIDADE_PONTEIROS - 1); // talvez voltar pra inteiro
263       int menoK = novoNoL.pares[0].chave;
264       fseek(ponteiroArvore, -sizeof(noArvore), SEEK_CUR);
265       fwrite(&noDado, sizeof(noArvore), 1, ponteiroArvore);
266       fseek(ponteiroArvore, sizeof(cabecalhoArvore) + novoNoOffsetL *
    TAMANHO_BLOCO, SEEK_SET);
267       fwrite(&novoNoL, sizeof(noArvore), 1, ponteiroArvore);
268       fseek(ponteiroArvore, 0, SEEK_SET);
269       fwrite(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
270 
271       insereNoPai(&noDado, menoK, novoNoOffsetL, noAtual);
272     }
273 }
```

◆ insereNaFolha() [1/2]

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

Essa função insere na folha sempre quando há espaço no nó

Autor: Aldemir

```
14 {
15   int i;
16   if (chave < (*no).pares[0].chave)
17   {
18     moveParesNo(no, 0, quntidadeChaves);
19     (*no).pares[0].chave = chave;
20     (*no).pares[0].endereco = P;
21   }
22   else
23   {
24     if (quntidadeChaves == 0)
25     {
26       (*no).pares[i + 1].chave = chave;
27       (*no).pares[i + 1].endereco = P;
28     }
29     else
30     {
31       for (i = quntidadeChaves - 1; i >= 0; i--)
32       {
33         if (chave >= (*no).pares[i].chave)
34         {
35           moveParesNo(no, i + 1, quntidadeChaves);
36           (*no).pares[i + 1].chave = chave;
37           (*no).pares[i + 1].endereco = P;
38           break;
39         }
40       }
41     }
42   }
43 }
```

◆ insereNaFolha() [2/2]

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

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

Autor: Glenn

```
50 {
51   if (chave < (*tmp).pares[0].chave)
52   {
53     moveParesNo(tmp, 0, QUANTIDADE_PONTEIROS - 1);
54     (*tmp).pares[0].chave = chave;
55     (*tmp).pares[0].endereco = P;
56     return;
57   }
58   for (int i = QUANTIDADE_PONTEIROS - 2; i >= 0; i--)
59   {
60     if (chave >= (*tmp).pares[i].chave)
61     {
62       moveParesNo(tmp, i + 1, QUANTIDADE_PONTEIROS - 1);
63       (*tmp).pares[i + 1].chave = chave;
64       (*tmp).pares[i + 1].endereco = P;
65       break;
66     }
67   }
68 }
```

◆ insereNoPai()

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

Essa funcão é chamada quando não há espaço sufuciente na folha

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

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

Autor: Erlon

```
79  {
80    if (vetorPais[0] == offsetNo)
81    {
82      noArvore novaRaiz;
83      apagaParesNo(&novaRaiz);
84      novaRaiz.pares[0].chave = chave;
85      novaRaiz.pares[0].endereco = offsetNo;
86      novaRaiz.pares[1].endereco = P;
87
88      cabecalhoArvore dadoCabecalho;
89      fseek(ponteiroArvore, 0, SEEK_SET);
90      fread(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
91
92      dadoCabecalho.quantidadeBlocos += 1;
93      dadoCabecalho.alturaArvore += 1;
94      dadoCabecalho.enderecoRaiz = dadoCabecalho.quantidadeBlocos;
95
96      fseek(ponteiroArvore, sizeof(cabecalhoArvore) + TAMANHO_BLOCO * dadoCabeca
97            SEEK_SET);
98      fwrite(&novaRaiz, sizeof(noArvore), 1, ponteiroArvore);
99
100     fseek(ponteiroArvore, 0, SEEK_SET);
101     fwrite(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
102     return;
103   }
104
105   int pai = posicaoPai(offsetNo);
106
107   noArvore dadoPai;
108
109   fseek(ponteiroArvore, sizeof(cabecalhoArvore) + pai * TAMANHO_BLOCO, SEEK_SE
110   fread(&dadoPai, sizeof(noArvore), 1, ponteiroArvore);
111
112   int quantidadePonteiro = contaPonteiros(dadoPai);
113
114   if (quantidadePonteiro < QUANTIDADE_PONTEIROS)
115   {
116     int i = 0;
117     while (dadoPai.pares[i].endereco != offsetNo)
118       i++;
119
120     if (i == QUANTIDADE_PONTEIROS - 2)
121     {
122       dadoPai.pares[i].chave = chave;
123       dadoPai.ponteiroM = P;
124     }
125     else
126     {
127       dadoPai.pares[i].chave = chave;
128       dadoPai.pares[i + 1].endereco = P;
129     }
130     fseek(ponteiroArvore, -sizeof(noArvore), SEEK_CUR);
131     fwrite(&dadoPai, sizeof(noArvore), 1, ponteiroArvore);
132   }
133   else
134   {
```

```
135     noArvoreTempPai TP;
136     copiaPaiNo(&dadoPai, &TP, chave, P);
137     apagaParesNo(&dadoPai);
138
139     noArvore dadoNovoPai;
140     apagaParesNo(&dadoNovoPai);
141
142     copiaPorPonteiro(&dadoPai, &TP, 0, (ceil(QUANTIDADE_PONTEIROS / 2.0) - 1))
143     int indicePaiK =
144         ceil(QUANTIDADE_PONTEIROS / 2.0) - 1;
145     int paiK = TP.pares[indicePaiK].chave;
146
147     fseek(ponteiroArvore, sizeof(cabecalhoArvore) + TAMANHO_BLOCO * pai, SEEK_
148     fwrite(&dadoPai, sizeof(noArvore), 1, ponteiroArvore);
149
150     copiaPorPonteiro2(&dadoNovoPai, &TP, (ceil(QUANTIDADE_PONTEIROS / 2.0)),
151                       QUANTIDADE_PONTEIROS);
152
153     cabecalhoArvore dadoCabecalho;
154     fseek(ponteiroArvore, 0, SEEK_SET);
155     fread(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
156
157     int dadoPosNovoPai = dadoCabecalho.quantidadeBlocos + 1;
158     fseek(ponteiroArvore, dadoPosNovoPai * TAMANHO_BLOCO, SEEK_CUR);
159     fwrite(&dadoNovoPai, sizeof(noArvore), 1, ponteiroArvore);
160
161     dadoCabecalho.quantidadeBlocos += 1;
162     fseek(ponteiroArvore, 0, SEEK_SET);
163     fwrite(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
164
165     insereNoPai(&dadoPai, paiK, dadoPosNovoPai, pai);
166   }
167 }
```

◆ main()

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

```
276 {
277   bool leitura = true;
278   std::string nomeEntrada = argv[1];
279   ponteiroArvore = fopen("./primarytree.bin", "w+");
280   cabecalhoArvore a_cabecalho;
281   a_cabecalho.alturaArvore = 0;
282   a_cabecalho.quantidadeBlocos = 0;
283   a_cabecalho.enderecoRaiz = -1;
284   fwrite(&a_cabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
285   io::CSVReader<7, io::trim_chars<>, io::double_quote_escape<';', '\"'>> sampl
286   sample.set_header("ID", "Titulo", "Ano", "Autores", "Citacoes", "Atualizacao
287                     "Snippet");
288     FILE *ponteiroHash = fopen("./hash.bin", "r+");
289 
290     if(ponteiroHash == NULL){
291         std::cout << "Error create hash.bin\n";
292         return 0;
293     }
294 
295     for(int i = 0; i < QUANTIDADE_BUCKETS ; i++){
296         tipoBloco bloco;
297         fwrite(&bloco, TAMANHO_BLOCO, 1, ponteiroHash);
298     }
299 
300   tipoArtigoLeitura *ta_aux =
301       (tipoArtigoLeitura *)malloc(sizeof(tipoArtigoLeitura));
302   while (leitura)
303   {
304     try
305     {
306       if (leitura = sample.read_row(ta_aux->ID, ta_aux->Titulo, ta_aux->Ano,
307                                     ta_aux->Autores, ta_aux->Citacoes,
308                                     ta_aux->Atualizacao, ta_aux->Snippet))
309       {
310         int id = std::stoi(ta_aux->ID);
311         insere(id, funcaoHash(id));
312 
313         tipoArtigo artigo;
314         tipoBloco bloco;
315 
316         artigo.ID = stoi(ta_aux->ID);
317         strcpy(artigo.Titulo, ta_aux->Titulo.c_str());
318         artigo.Ano = stoi(ta_aux->Ano);
319         strcpy(artigo.Autores, ta_aux->Autores.c_str());
320         artigo.Citacoes = stoi(ta_aux->Citacoes);
321         strcpy(artigo.Atualizacao, ta_aux->Atualizacao.c_str());
322         strcpy(artigo.Snippet, ta_aux->Snippet.c_str());
323 
324 
325         fseek(ponteiroHash, funcaoHash(artigo.ID)*TAMANHO_BLOCO, SEEK_SET);
326         fread(&bloco, TAMANHO_BLOCO, 1, ponteiroHash);
327         bloco.vetorArtigos[bloco.quantidadeArtigos] = artigo;
328         bloco.quantidadeArtigos++;
329         fseek(ponteiroHash, funcaoHash(artigo.ID)*TAMANHO_BLOCO, SEEK_SET);
330         fwrite(&bloco, TAMANHO_BLOCO, 1, ponteiroHash);
331 
332         std::cout << "Inserindo id: " << ta_aux->ID << std::endl;
333       }
334     }
335     catch (io::error::too_few_columns) {}
336     catch (io::error::escaped_string_not_closed) {}
337   }
338   free(ta_aux);
339   fclose(ponteiroArvore);
340   fclose(ponteiroHash);
341 
342   return 0;
343 }
```

Variáveis

◆ ponteiroArvore

FILE* ponteiroArvore

- apagaParesNo() : **functions.h**
- buscaNaArvore() : **seek1.cpp**
- contaChaves() : **functions.h**
- contaPonteiros() : **functions.h**
- copiaPaiNo() : **functions.h**
- copiaParesNo() : **functions.h**
- copiaPorPonteiro() : **functions.h**
- copiaPorPonteiro2() : **functions.h**
- copiaTodosParesNo() : **functions.h**
- funcaoHash() : **functions.h**
- insere() : **upload.cpp**
- insereNaFolha() : **upload.cpp**
- insereNoPai() : **upload.cpp**
- main() : **findrec.cpp**, **seek1.cpp**, **upload.cpp**
- moveParesNo() : **functions.h**
- posicaoPai() : **functions.h**
- printDadosBusca() : **functions.h**

- 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

Student: Erlon Pereira Bié
Email: erlon.bie@icomp.ufam.edu.br

Student: Glenn Aguiar de Oliveira da Fonseca
Email: 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[]
         )
```

```
6                                       {
7       FILE *ponteiroHash = fopen("./hash.bin", "rb");
8       std::string sEntrada = argv[1];
9       int idPesquisa = stoi(sEntrada);
10
11      tipoBloco bloco;
12      tipoArtigo artigo;
13      bool achou = false;
14
15      fseek(ponteiroHash, funcaoHash(idPesquisa)*TAMANHO_BLOCO, SEEK_SET);
16      fread(&bloco, sizeof(tipoBloco), 1, ponteiroHash);
17
18      for(int i = 0; i < bloco.quantidadeArtigos; i++)
19      {
20          if(bloco.vetorArtigos[i].ID == idPesquisa)
21          {
22              achou = true;
23              artigo = bloco.vetorArtigos[i];
24              break;
25          }
26      }
27
28      if(achou)
29      {
30        std::cout << "Id: " << artigo.ID << std::endl;
31        std::cout << "Título: " << artigo.Titulo << std::endl;
32        std::cout << "Ano: " << artigo.Ano << std::endl;
33        std::cout << "Autores: " << artigo.Autores << std::endl;
34        std::cout << "Citações: " << artigo.Citacoes << std::endl;
35        std::cout << "Atualização: " << artigo.Atualizacao << std::endl;
36        std::cout << "Snippet: " << artigo.Snippet << std::endl;
37        std::cout << "Total de blocos lidos: " << 1 << std::endl; // sempre será
  1 aqui por ser um hash
38        std::cout << "Total de blocos do arquivo: " << QUANTIDADE_BUCKETS << std
39      }
40      else
41      {
42        std::cout << "Registro não encontrado!" << std::endl;
43      }
44
45      fclose(ponteiroHash);
46      return 0;
47  }
```

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 quntidadeChaves)
void	**moveParesNo** (**noArvoreTemp** *no, int pos, int quntidadeChaves)
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
   - 1)
24     counter++;
25   return counter;
26 }
```

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 !=
  QUANTIDADE_PONTEIROS - 1)
34     counter++;
35   if (noDado.ponteiroM != -1)
36   {
37     counter++;
38   }
39   return counter;
40 }
```

copiaPaiNo()

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

Copia todos os ponteiros e chavesde um nó **noArvore** para um nó **noArvoreTempPai**, e aadiciona 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 um 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 quntidadeChaves
)

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

```
79 {
80   for (int j = quntidadeChaves + 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 quntidadeChaves
)

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

```
89 {
90   for (int j = quntidadeChaves; 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

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()

dadoBusca 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          if (i == QUANTIDADE_PONTEIROS - 1 && noDado.pares[i - 1].chave <
chave)
46          {
47            noAtual = noDado.ponteiroM;
48            fseek(ponteiroArvore, sizeof(cabecalhoArvore) + noAtual * TAMANHO_BL
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_BL
59                  SEEK_SET);
60            nivelAtual++;
61            break;
62          }
63          i++;
64        }
65      }
66      fread(&noDado, sizeof(noArvore), 1, ponteiroArvore);
67      int quntidadeChaves = contaChaves(noDado);
68      for (int i = 0; i < quntidadeChaves; 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.quantidad
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
98    std::string sEntrada = argv[1];
99    int nomeEntrada = stoi(sEntrada);
100   ponteiroArvore = fopen("./primarytree.bin", "r");
101   buscaNaArvore(nomeEntrada);
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 quntidadeChaves)
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         if (i == QUANTIDADE_PONTEIROS - 1 && noDado.pares[i - 1].chave <
chave)
209         {
210           noAtual = noDado.ponteiroM;
211           fseek(ponteiroArvore, sizeof(cabecalhoArvore) + noAtual * TAMANHO_BL
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_BL
222                 SEEK_SET);
223           nivelAtual++;
224           break;
225         }
226         i++;
227       }
228     }
229   }
230   vetorPais.push_back(noAtual);
231   fread(&noDado, sizeof(noArvore), 1,
232         ponteiroArvore);
```

```
233     int quntidadeChaves = contaChaves(noDado);
234     if (quntidadeChaves < QUANTIDADE_PONTEIROS - 1)
235     {
236       insereNaFolha(&noDado, chave, P, quntidadeChaves);
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); //
    talvez voltar pra inteiro
254       copiaParesNo(&novoNoL, &tmp, ceil(QUANTIDADE_PONTEIROS / 2.0),
    QUANTIDADE_PONTEIROS - 1); // talvez voltar pra inteiro
255       int menoK = novoNoL.pares[0].chave;
256       fseek(ponteiroArvore, -sizeof(noArvore), SEEK_CUR);
257       fwrite(&noDado, sizeof(noArvore), 1, ponteiroArvore);
258       fseek(ponteiroArvore, sizeof(cabecalhoArvore) + novoNoOffsetL *
    TAMANHO_BLOCO, SEEK_SET);
259       fwrite(&novoNoL, sizeof(noArvore), 1, ponteiroArvore);
260       fseek(ponteiroArvore, 0, SEEK_SET);
261       fwrite(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
262 
263       insereNoPai(&noDado, menoK, novoNoOffsetL, noAtual);
264     }
265 }
```

◆ insereNaFolha() [1/2]

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

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, quntidadeChaves);
17     (*no).pares[0].chave = chave;
18     (*no).pares[0].endereco = P;
19   }
20   else
21   {
22     if (quntidadeChaves == 0)
23     {
24       (*no).pares[i + 1].chave = chave;
25       (*no).pares[i + 1].endereco = P;
26     }
27     else
28     {
29       for (i = quntidadeChaves - 1; i >= 0; i--)
30       {
31         if (chave >= (*no).pares[i].chave)
32         {
33           moveParesNo(no, i + 1, quntidadeChaves);
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 funcão 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 funcão é chamada quando não há espaço sufuciente na folha

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

Se esse processo não for sufuciente, 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<';', '\"'>> sampl
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