Universidade Federal de Viçosa - Campus Florestal
Ciência da Computação
Análise de Algoritmos e Estrutura de Dados II

ÁRVORE B

Aluno: Samuel Silva Costa Nascimento Mat: 2662

Aluno: Vinícius Simim Ribeiro Mat: 2645

Aluno: Jonathan Lopes Mat: 2666

Professora: Gláucia Braga e Silva

Florestal
2018

1 Introdução

Com o avanço da tecnologia os termos hardware e software se tornaram substantivos muito utilizados em nosso cotidiano, o hardware pode ser definido como equipamentos físicos que se aplica à dispositivos de entrada e saída, memórias, unidade central de processamento e entre outras. O software é a parte digital e basicamente a parte lógica (conjunto de instruções), que utilizam dos circuitos eletrônicos contidos nos hardware para dar fim a uma utilidade dita pelo ser humano. Como esses conceitos tecnológicos são de suma importância para estudantes de computação, no decorrer de sua formação acadêmica há um estudo bem aprofundado sobre esses e diversos outros. A disciplina Analise de Algoritmos e Estrutura de Dados II é uma das matérias cursadas no curso de ciência da computação que traz bem a fundo explicações em geral sobre a importância do software. A arvore B é uma das estruturas explicitados na disciplina e com intuito de nos proporcionar um melhor entendimento dela é que nossa Professora Gláucia Braga e Silva aplicou um trabalho pratico onde o objetivo principal foi implementar a estrutura da arvore B mostrando suas operações e visando trabalhar com a resolução de problemas foi que tivemos que utilizar a ferramenta GIMP TOOL KIT (GTK) para produzir uma interface gráfica para o sistema.

2 Desenvolvimento

Para a realização desse trabalho foi utilizado da linguagem de programação C, da IDE Code Blocks 17.12 para compilar o código produzido, de informações buscadas na internet para fim de acompanhamento da implementação da arvore B, materiais para instalação e produção de interfaces no GTK e por ultimo e não menos importante de um embasamento teórico aprofundado na matéria "Arvore B".

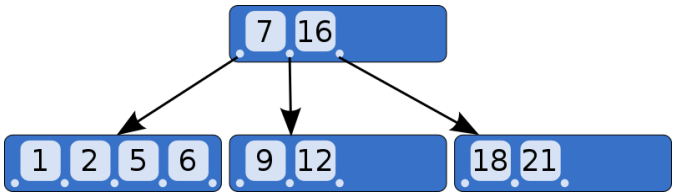

Figura 1: **Arvore B** - Ilustração Arvore B

Por decisão de projeto, a implementação da nossa arvore B foi realizada dividindo os arquivos em pontos C e H como ensinado na matéria de AEDS I. Os arquivos foram separados em um ponto H onde no arvoreB.h está contido a estrutura da árvore e o cabeçalho das funções e dois pontos C onde no arvoreB.c está incluso o desenvolvimento das funções e o arquivo main.c contendo as chamadas das funções e o GTK.

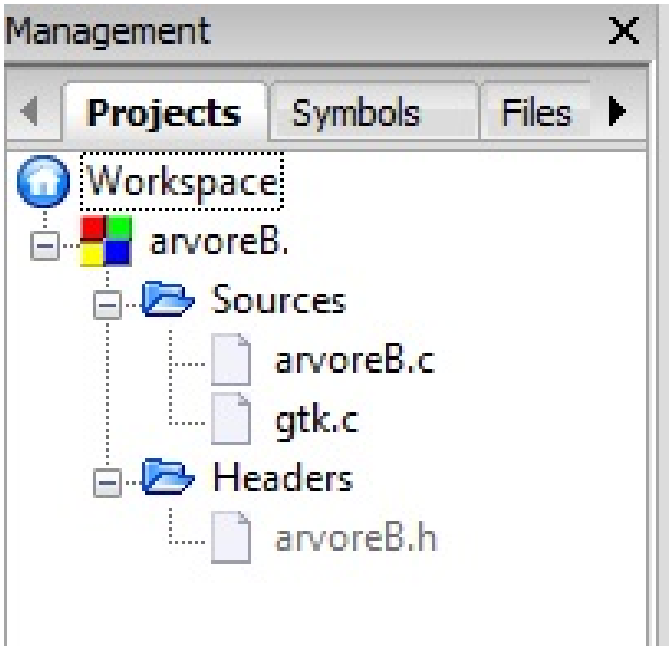

Figura 2: **Arquivos** - Ilustração da Divisão dos Arquivos

2.1 ArvoreB.h

O arquivo ArvoreB.h é responsável por encapsular as informações e funções a serem usadas no decorrer do código, nele está contido a estrutura da arvoreB implementada utilizando como referência o código do prof Nivio Ziviani e os cabeçalhos das funções. Exemplos da estrutura produzida em C logo a baixo:

2.1.1 Código em C ArvoreB.h

Listing 1: ArvoreB.h! em C

```
1  #ifndef ARVOREB_H_INCLUDED
2  #define ARVOREB_H_INCLUDED
3  #define M 8
4  #define MM  (M * 2)
5  #define FALSE 0
6  #define TRUE  1
7
8  typedef struct{
9      int chave;
10 }reg;
11
12 typedef struct Tipo_pag *apontador;
13
14 typedef struct Tipo_pag{
15     int n;
16     reg registros[MM];
17     apontador p [MM +1];
18
19 }Tipo_pag;
20
21 void Inicializa(apontador *no) ;
22 int Pesquisa(reg registro,apontador no);
23 void Insere_pagina (apontador ap, reg registro, apontador ↩
   pdir);
24 void Ins(reg registro, apontador ap,short *Cresceu, reg *↩
   retorno_reg, apontador *retorno_apontador  );
25 void Insere(reg registro , apontador *ap);
26 void Reconstitui (apontador p_pag,apontador p_pai,int ↩
   posicao, short *diminuiu);
27 void Antecessor (apontador ap,int indice,apontador p_pai,↩
   short *diminuiu);
28 void Ret (int item, apontador *ap,short *diminuiu);
29 void Retira(int item, apontador *ap);
30 void Printar (apontador ap,int nivel);
31 void Imprime (apontador p);
32
33 #endif // ARVOREB_H_INCLUDED
```

2.2 ArvoreB.c

O arquivo ArvoreB.c é responsável por implementar as funcionalidades das funções. Exemplos das funções produzida em C e alto nível logo a baixo:

2.2.1 Código em C Função Inicializa

Listing 2: Função Inicializa! em C

```
1       void Inicializa(apontador *No){
2           *No = NULL;
3       }
4
```

2.2.2 Alto Nível Função Inicializa

Figura 3: Inicializando Árvore B.

2.2.3 Código em C Função Pesquisa

Listing 3: Função Pesquisa! em C

```
1       int Pesquisa(reg registro,apontador no){
2           int i = 1;
3
4           if(no == NULL)
5           {
6               printf("Registro nao encontrado\n");
7               return 0;
8           }
9           while(i<no->n && registro.chave > no->registros[i-1].chave)
10              i++ ;
11          if(registro.chave == no->registros[i-1].chave)
12          {
13              printf("Registro Encontrado\n");
14              registro = no->registros[i-1];
15              return registro.chave ;
16          }
17          if(registro.chave < no->registros[i-1].chave)
18              Pesquisa(registro,no->p[i-1]);
19          else
20              Pesquisa(registro,no->p[i]);
21      }
22
```

2.2.4 Alto Nível Função Pesquisa

Figura 4: Pesquisa Árvore B.

2.2.5 Código em C Função Inserir

Listing 4: Função Inserir! em C

```
1       void Insere_pagina (apontador ap, reg registro, apontador pdir){
2           short nao_achou;
3           int k;
4           k = ap->n;
5           nao_achou = (k>0);
6           while(nao_achou)
7           {
8               if(registro.chave>=ap->registros[k-1].chave)
9               {
10                  nao_achou = FALSE;
11                  break;
12              }
13              ap->registros[k] = ap->registros[k-1];
14              ap->p[k+1] = ap->p[k];
15              k--;
16              if(k<1)
17                  nao_achou = FALSE;
18          }
19          ap->registros[k] = registro;
20          ap->p[k+1] = pdir;
21          ap->n++;
22      }
23
24      void Ins(reg registro, apontador ap,short *Cresceu, reg *retorno_reg, apontador *retorno_apontador  ){
25          int i = 1;
26          int j;
27          apontador ap_temp;
28
29          if(ap == NULL){
30              (*Cresceu) = TRUE;
31              (*retorno_reg) = registro;
32              (*retorno_apontador) = NULL;
33              return;
34          }
35          while ( i < ap->n && registro.chave > ap->registros[i-1].chave){
36              i++ ;
37          }
38          if(registro.chave == ap->registros[i-1].chave){
39              printf("Registro ja existe na arvore");
40              (*Cresceu )= FALSE;
41              return;
42          }
```

```
43          if(registro.chave<ap->registros[i-1].chave)
44              i--;
45          Ins(registro,ap->p[i],Cresceu,retorno_reg,↩
        retorno_apontador);
46          if(!*Cresceu)
47              return ;
48          if(ap->n < MM){
49              Insere_pagina(ap,*retorno_reg,*retorno_apontador)↩
        ;
50              (*Cresceu) = FALSE;
51              return;
52          }
53          ap_temp = (apontador) malloc(sizeof(Tipo_pag));
54          ap_temp->n = 0;
55          ap_temp->p[0] = NULL;
56          if(i<M+1){
57              Insere_pagina(ap_temp,ap->registros[MM-1],ap↩
        ->p[MM]);
58              ap->n -- ;
59              Insere_pagina(ap,*retorno_reg,*retorno_apontador)↩
        ;
60          }
61          else
62              Insere_pagina(ap_temp,*retorno_reg,*↩
        retorno_apontador);
63          for ( j = M + 2; j<= MM; j++)
64              Insere_pagina(ap_temp,ap->registros[j-1],ap->p↩
        [j]);
65          ap->n = M;
66          ap_temp->p[0] = ap->p[M+1];
67          *retorno_reg = ap->registros[M];
68          *retorno_apontador = ap_temp;
69      }
70
71      void Insere(reg registro, apontador *ap){
72          short Cresceu;
73          reg Retornoreg;
74          Tipo_pag *ap_retorno, *ap_temp;
75
76          Ins(registro, *ap,&Cresceu,&Retornoreg,&ap_retorno);
77          if(Cresceu){
78              ap_temp = (Tipo_pag*) malloc(sizeof(Tipo_pag));
79              ap_temp->n =1 ;
80              ap_temp->registros[0] = Retornoreg;
81              ap_temp->p[1] = ap_retorno;
82              ap_temp->p[0] =  *ap;
83              *ap = ap_temp;
84          }
85      }
86
```

2.2.6 Árvore antes de Inserir

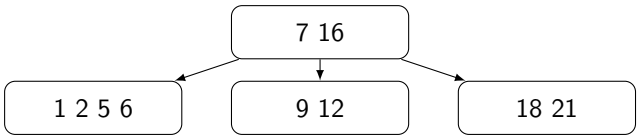

Figura 5: Ilustração da Árvore B antes de Inserir.

2.2.7 Árvore depois de chamar Função Inserir para chave 13

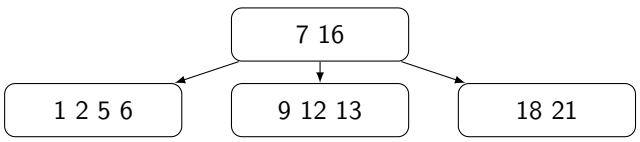

Figura 6: Ilustração da Árvore B depois de Inserir.

2.2.8 Código em C Função Retira

Listing 5: Função Retira! em C

```
1       void Retira(int item, apontador *ap){
2           short diminuiu;
3           apontador aux;
4           Ret(item,ap,&diminuiu);
5           if(diminuiu && (*ap)->n == 0)
6           {
7               aux = *ap;
8               *ap = aux->p[0];
9               free(aux);
10          }
11      }
12
13      void Ret (int item, apontador *ap,short *diminuiu){
14          int j;
15          int indice;
16          apontador pag;
17          if(*ap == NULL){
18              printf("Registro nao existente");
19              *diminuiu = FALSE;
20              return;
21          }
22          pag = *ap;
23          while (indice < pag->n && item>pag->registros[↩
        indice-1].chave)
24              indice ++ ;
25          if(item == pag->registros[indice -1].chave){
26              if(pag->p[indice -1 ] == NULL){
27                  pag->n -- ;
28                  *diminuiu = (pag->n < M);
29                  for(j = indice; j<=pag->n; j++){
30                      pag->registros[j-1] = pag->registros[j];
31                      pag->p[j]= pag->p[j+1];
32                  }
33                  return;
34              }
35              Antecessor(*ap,indice,pag->p[indice-1],diminuiu);
36              if(*diminuiu)
37                  Reconstitui(pag->p[indice-1],*ap,indice-1,↩
        diminuiu);
38              return;
39          }
40          if(item>pag->registros[indice-1].chave)
41              indice ++;
42          Ret(item,&pag->p[indice-1],diminuiu);
43          if(*diminuiu)
44              Reconstitui(pag->p[indice-1],*ap,indice-1,↩
        diminuiu  );
45      }
46
47      void Antecessor (apontador ap,int indice,apontador p_pai,↩
        short *diminuiu){
48          if(p_pai->p[p_pai->n] != NULL){
49              Antecessor(ap,indice,p_pai->p[p_pai->n],↩
        diminuiu);
50              if(*diminuiu){
51                  Reconstitui(p_pai->p[p_pai->n],p_pai,p_pai↩
        ->n,diminuiu);
52                  return;
53              }
54              ap->registros[indice - 1] = p_pai->registros[↩
        p_pai->n -1];
55              p_pai->n --;
56              *diminuiu = (p_pai->n < M);
57          }
58      }
59
60      void Reconstitui (apontador p_pag,apontador p_pai,int ↩
        posicao_pai, short *diminuiu){
61          Tipo_pag *aux;
62          int D_aux;
63          int j;
64          if( posicao_pai < p_pai->n){
65              aux = p_pai->p[posicao_pai+1];
66              D_aux = (aux->n - M +1) /2 ;
67              p_pag->registros[p_pag->n] = p_pai->registros↩
        [posicao_pai];
68              p_pag->p[p_pag->n +1 ] = aux->p[0];
69              p_pag->n ++ ;
70              if(D_aux>0){
71                  for(j=1; j<D_aux; j++)
72                      Insere_pagina(p_pag, aux->registros[j-1],↩
        aux->p[j]);
73                  p_pai->registros[posicao_pai] = aux->↩
        registros[D_aux - 1];
74                  aux->n -=D_aux;
75                  for(j=0; j<aux->n; j++)
76                      aux->registros[j] = aux->registros[j+↩
        D_aux];
```

```
77                  for(j=0; j<=aux->n; j++)
78                      aux->p[j] = aux->p[j+D_aux];
79                  *diminuiu = FALSE;
80              }
81              else{
82                  for(j=1; j<=M; j++)
83                      Insere_pagina(p_pag, aux->registros[j-1],↩
        aux->p[j]);
84                  free(aux);
85                  for(j = posicao_pai+1; j<p_pai->n; j++){
86                      p_pai->registros[j-1] = p_pai->registros[j↩
        ];
87                      p_pai->p[j] = p_pai->p[j+1];
88                  }
89                  p_pai->n -- ;
90                  if(p_pai->n >= M)
91                      *diminuiu = FALSE;
92              }
93
94          }
95          else{
96              aux = p_pai->p[posicao_pai - 1];
97              D_aux = (aux->n - M +1) / 2 ;
98              for(j = p_pag->n; j>=1; j--)
99                  p_pag->registros[j] = p_pag->registros[j-1];
100             p_pag->registros[0] = p_pai->registros[↩
        posicao_pai - 1];
101             for(j = p_pag->n; j>= 0; j--)
102                 p_pag->p[j+1] = p_pag->p[j];
103             p_pag->n ++ ;
104             if(D_aux > 0){
105                 for(j = 1; j< D_aux; j++)
106                     Insere_pagina(p_pag,aux->registros[aux->↩
        n-j],aux->p[aux->n-j +1]);
107                 p_pag->p[0] = aux->p[aux->n -D_aux ↩
        +1];
108                 p_pai->registros[posicao_pai - 1 ] = aux->↩
        registros[aux->n -D_aux ];
109                 aux->n -= D_aux;
110                 *diminuiu = FALSE;
111             }
112             else{
113                 for(j =1; j<=M; j++)
114                     Insere_pagina(aux,p_pag->registros[j-1],↩
        p_pag->p[j]);
115                 free(p_pag);
116                 p_pai->n --;
117                 if(p_pai->n >= M)
118                     *diminuiu = FALSE;
119             }
120         }
121     }
122
```

2.2.9 Árvore antes de chamar Função Remove

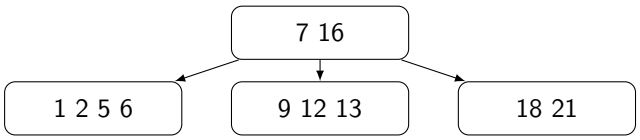

Figura 7: Ilustração da Árvore B antes de Remover.

2.2.10 Árvore depois de chamar Função Remove para Chave 18

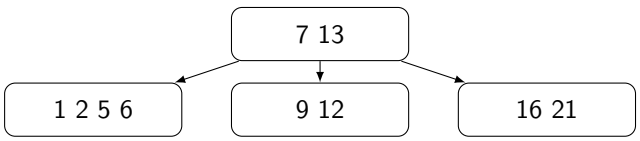

Figura 8: Ilustração da Árvore B depois de Remover Chave 18.

2.2.11 Código em C Função Imprime

Listing 6: Função Imprime! em C

```
1   void Imprime (apontador p){
2       int n=0;
3       Printar(p,n);
4   }
5
6   void Printar (apontador ap,int nivel){
7       int i;
8       if(ap == NULL)
9           return;
10      printf("Nivel  %d :\n",nivel);
11      for(i=0;i< ap->n;i++){
12          printf("Chave: %d\n",ap->registros[i].chave);
13      }
14      nivel++;
15      for(i=0;i<=ap->n;i++){
16          Printar(ap->p[i],nivel);
17      }
18  }
19
```

2.2.12 Alto Nível Função Imprime

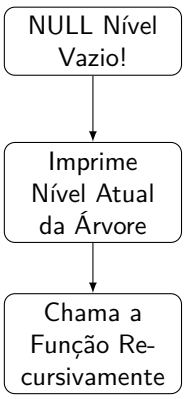

Figura 9: Alto Nível Função Imprime.

2.3 Main.c

O arquivo Main.c é responsável por criar a interface gráfica GTK e a partir dos botões implementados chamar as funções Inserir, Remover e Pesquisar da árvore b, mostrando seus respectivos resultados. Ilustrações da Interface a baixo:

2.3.1 Código em C Main GTK

Listing 7: Main GTK! em C

```
1   int main( int argc, char *argv[]){
2       Tipo_pag *op;
3       reg teste;
4
5       GtkWidget *window;
6       GtkWidget *box;
7
8       GtkWidget *insercao;
9       GtkWidget *retirada;
10      GtkWidget *pesquisa;
11      GtkWidget *fechar;
12      GtkWidget *halign;
13
14      gtk_init(&argc, &argv);
15
16      window = gtk_window_new(↩
    GTK_WINDOW_TOPLEVEL);
17      gtk_window_set_position(GTK_WINDOW(window), ↩
    GTK_WIN_POS_CENTER);
18      gtk_window_set_title(GTK_WINDOW(window), "Main↩
    ");
19      gtk_window_set_default_size(GTK_WINDOW(window)↩
    , 230, 250);
20      gtk_container_set_border_width(GTK_CONTAINER(↩
    window), 5);
21
```

```
22          box = gtk_vbox_new(TRUE, 1);
23          gtk_container_add(GTK_CONTAINER(window), box);
24
25          insercao = gtk_button_new_with_mnemonic("Insercao"↩
        );
26          retirada = gtk_button_new_with_label("Retirada");
27          pesquisa = gtk_button_new_with_label("Pesquisa");
28
29          gtk_box_pack_start(GTK_BOX(box), insercao, TRUE, ↩
        TRUE, 0);
30          gtk_box_pack_start(GTK_BOX(box), retirada, TRUE, ↩
        TRUE, 0);
31          gtk_box_pack_start(GTK_BOX(box), pesquisa, TRUE, ↩
        TRUE, 0);
32
33          g_signal_connect(insercao, "clicked",
34              G_CALLBACK(print_msg), NULL);
35
36          halign = gtk_alignment_new(0, 0, 0, 0);
37          gtk_container_add(GTK_CONTAINER(halign), insercao↩
        );
38          gtk_container_add(GTK_CONTAINER(window), halign↩
        );
39
40          g_signal_connect(retirada, "clicked",
41              G_CALLBACK(retira_elemento), NULL);
42
43          halign = gtk_alignment_new(0, 0, 0, 0);
44          gtk_container_add(GTK_CONTAINER(halign), retirada↩
        );
45          gtk_container_add(GTK_CONTAINER(window), halign↩
        );
46
47          g_signal_connect(pesquisa, "clicked",
48              G_CALLBACK(pesquisa_elemento), NULL);
49
50          halign = gtk_alignment_new(0, 0, 0, 0);
51          gtk_container_add(GTK_CONTAINER(halign), ↩
        pesquisa);
52          gtk_container_add(GTK_CONTAINER(window), halign↩
        );
53
54           fechar = gtk_button_new_from_stock (↩
        GTK_STOCK_CLOSE);
55              g_signal_connect_swapped (fechar, "clicked",
56                          G_CALLBACK (gtk_widget_destroy),
57                          window);
58          gtk_box_pack_start (GTK_BOX (box), fechar, TRUE,↩
         TRUE, 0);
59          gtk_widget_set_can_default (fechar, TRUE);
60          gtk_widget_grab_default (fechar);
61          gtk_widget_show (fechar);
62
63          gtk_widget_show_all(window);
64
65          g_signal_connect(G_OBJECT(window), "destroy",
66              G_CALLBACK(gtk_main_quit), NULL);
67
68          gtk_main();
69
70          return 0;
71      }
72
73      int aux(){}
74
75      void enter_callback( GtkWidget *widget, GtkWidget *entry)↩
          {}
76
77      void deletado( GtkWidget *widget, GtkWidget *entry){}
78
79      void pesquisando( GtkWidget *widget, GtkWidget *entry){}
80
81      void print_msg(GtkWidget *widget, gpointer window){}
82
83      void retira_elemento(GtkWidget *widget, gpointer window){}
84
85      void pesquisa_elemento(GtkWidget *widget, gpointer window↩
          ){}
86
```

2.3.2 Main GTK

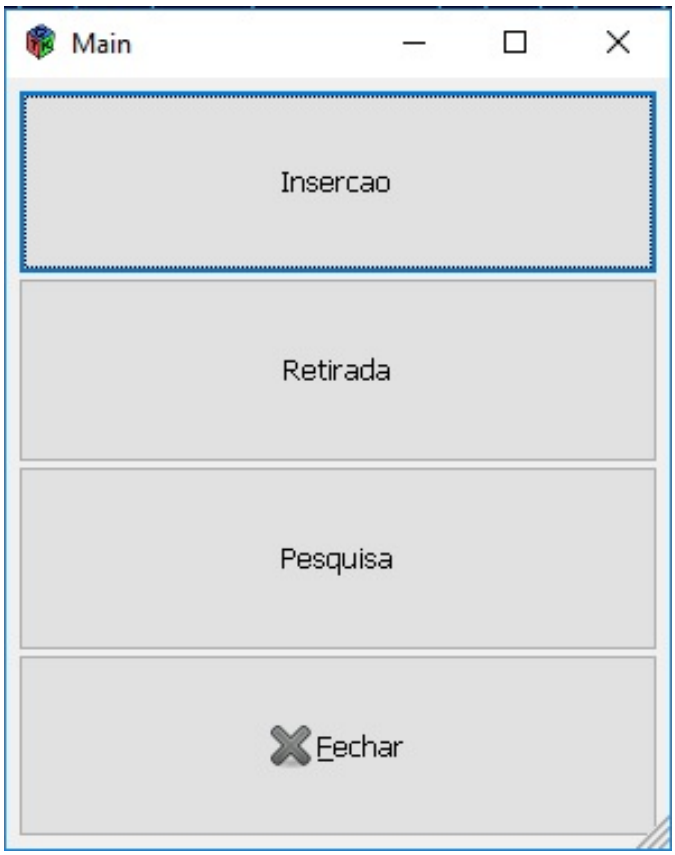

Figura 10: **Main GTK** - Ilustração do Main GTK

2.3.3 Inserção GTK

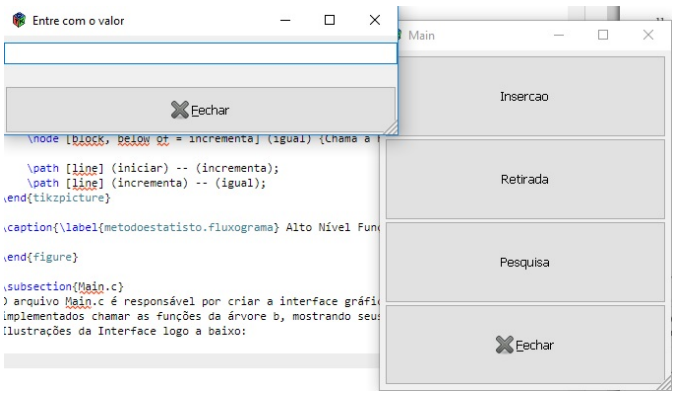

Figura 11: **Inserção GTK** - Ilustração ao Pressionar Botão de Inserção

2.3.4 Inserindo Valor GTK

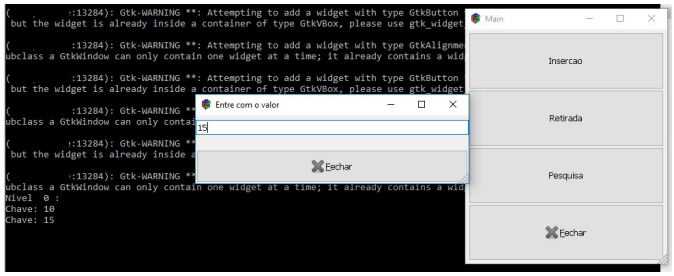

Figura 12: **Inserindo Valor GTK** - Ilustração ao Inserir Chave 10

2.3.5 Pesquisa GTK

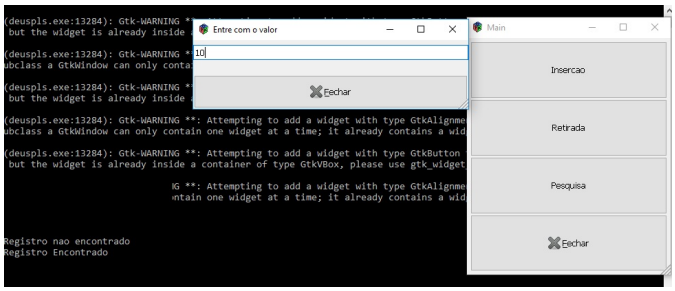

Figura 13: **Pesquisa GTK** - Ilustração ao Pesquisar Chave 10

3 Testes

Para realização dos testes foram utilizados 3 valores divergentes de ordem da árvore (m = 8, 64, 512), através dos tamanhos diferentes e dos resultados da comparação de cada um foi montado um gráfico para fins comparativos. Gráfico disponível a baixo:

4 Conclusão

Durante todo o período, estudamos mais profundamente os tipos de arvores e suas particularidades. O trabalho final da disciplina, que teve como objetivo a implementação da Árvore B em C, uniu todos os conhecimentos adquiridos durante boa parte da matéria. Podemos observar ao implementar a Árvore B, as vantagens que ela traz no quesito redução no número de acessos ao disco. Por fim, o trabalho nos mostrou a importância e as vantagens que a Árvore B nos traz.