Universidade Federal da Bahia
Estrutura de Dados e Algoritmos I – MATA40
Departamento de Ciências da Computação
Professor: Danilo Santos

PROJETO 2

Base de dados para uma Arvore Genealógica

Alunos:
Antônio Jose Azevedo
Icaro Pereira
Pedro Gabriel Carrano
Tiago Severo

Salvador
Dezembro de 2014

1. INTRODUCAO

Este projeto visa construir uma base de dados para uma arvore genealógica em linguagem C. Este base devera ser estruturada em dois módulos diferentes, de forma que o primeiro modulo aloque os membros da arvore e o segundo modulo devera conter funções para que o usuário possa consultar a arvore.

Na estrutura deste projeto utilizamos uma arvore, de modo que cada elemento desta conteria os seguintes dados:

- Nome da pessoa
- Sexo
- Ponteiro para o pai
- Ponteiro para o irmão mais proximo
- Ponteiro para o conjugue
- Ponteiro para o primeiro filho

Dessa forma a medida que são inseridos os elementos da arvore, e construída uma arvore genealógica, relacionando os indivíduos. As funções de inserção para a construção da arvore são:

- Adicionar Conjugue

- Adicionar Irmao

- Adicionar Pai

Esta arvore sera armazenada em um registro, que poderá ser utilizado pelo usuário a partir do modulo II. Esse modulo conterá funções para que se possa buscar um certo elemento na tabela, e além disso descobrir quais os seus parentes (conjugue, filhos, pais, irmãos, sobrinhos, tios, avos, entre outros) ou reconhecer os indivíduos por sexo.

2. ESTRUTURA DE DADOS UTILIZADA

A estrutura utilizada para formar a arvore foi:

```
 6
 7  typedef struct dados{
 8
 9      char nome;
10      char sexo;
11      struct dados *pai;
12      struct dados *proxIrmao;
13      struct dados *conjuge;
14      struct dados *primeiroFilho;
15
16  }informacoes;
17
18  typedef struct {
19
20      char nome;
21      informacoes *registro;
22
23  }celula;
24
25  typedef struct {
26
27      celula planilha[MAX];
28      int cont ;
29
30  } tabela;
31
```

Um vetor planilha contendo células, cada célula representaria um individuo e possuiria o nome do mesmo e suas respectivas informações. Estas informações formariam o link do elemento com os demais indivíduos da arvore, apontando para a respectiva posição dos membros no vetor.

Ou seja, dado uma célula da planilha, poderão ser extraídas informações sobre quem e seu pai, seu primeiro irmão, primeiro filho e conjugue, pela posição destes na planilha.

Para o nosso método de inserção e busca, eh importante que os nomes de todos os indivíduos seja diferente, assim, caso haja necessidade de inserir dois elementos com o mesmo nome, um deles devera ter um caractere adicional.

Dividimos o programa em um header com as assinaturas das funções a serem utilizadas e uma implementação dessas funções, de forma que ao chamar as assinaturas no main, estas chamavam as funções implementadas.

3. ALGORITMO

Fizemos nosso código buscando preencher os requisitos do sistema. Aqui temos uma lista das nossas funções principais, divididas em dois módulos e em seguida iremos detalhar algumas delas.

3.1 MODULO I

```
31
32  // MODULO I
33
34  void iniciarTabela(tabela *p);
35
36
37  void info(tabela *t, char name, char sex);
38
39
40  int adicionarConjuge(tabela *t, char nome1, char nome2);
41
42
43  int adicionarPaiFilho(tabela *t, char nome1, char nome2);
44
45
46  int adicionarIrmao(tabela *t, char nome1, char nome2);
47
48
```

Seguindo os requerimentos do projeto, fizemos as funções de inserção com base em Conjugue, Irmao e Pai. Iremos detalhar aqui somente a função adicionarConjugue, já que todas seguem aproximadamente o mesmo padrão.

```
148
149 int adicionarConjuge(tabela *t, char nome1, char nome2){
150
151     int  temp1, temp2;
152
153     if(nome1 == nome2){
154         printf("NOMES IGUAIS!!\n");
155         return;
156     }
157     temp1 = buscarTabela(t, nome1);
158
159     if(temp1 == -1){} // VERIFICA SE AS PESSOAS ESTAO INSERIDAS NA TABELA
163     temp2 = buscarTabela(t, nome2);
164
165     if(temp2 == -1){} // VERIFICA SE AS PESSOAS ESTAO INSERIDAS NA TABELA
169     if(t->planilha[temp1].registro->conjuge != NULL  || t->planilha[temp2].registro->conjuge != NULL){
170         printf("Pessoa jÃ¡ casada!\n");
171         return;
172     }
173     if((t->planilha[temp1].registro->sexo) == (t->planilha[temp2].registro->sexo)){
174         printf("Pessoas do mesmo sexo\n");
175         return ;
176     }
177
178     //irmÃ£o - VERIFICA SE EH O IRMAO
179     if(t->planilha[temp1].registro->proxIrmao == t->planilha[temp2].registro){}
183     //filho - VERIFICA SE EH O FILHO
184     if(t->planilha[temp1].registro->primeiroFilho == t->planilha[temp2].registro){}
188     else{
189         t->planilha[temp1].registro->conjuge = t->planilha[temp2].registro;
190         t->planilha[temp2].registro->conjuge = t->planilha[temp1].registro;
191     }
192     return 1;
193 }
194
```

Dados dois nomes, a funcao busca ambos e verifica se estao corretos. Após isso adiciona o elemento na arvore, por ponteiro. Ao modificar o ponteiro conjugue, os elementos estarao “linkados”.

3.2 MODULO II

```
48
49  //MODULO II
50
51  informacoes* buscarPais(tabela *t, char nome, int valor);
52
53
54  informacoes* buscarConjuge(tabela *t, char nome);
55
56
57  informacoes* buscarIrmao(tabela *t, char nome);
58
59
60  informacoes* buscarTioTia(tabela *t, char nome);
61
62
63  informacoes* buscarSobrinho(tabela *t, char nome1);
64
```

Para o modulo II, implementamos as funções de consulta (buscarPais, buscarConjugue, buscarIrmao, buscarTioTia, buscarSobrinho). Todas funcionam da seguinte forma, dado um nome, a função ira percorrer a planilha ate encontra-lo, descobrindo assim sua célula, e quais suas informações, dessas informações, poderam ser acessados esses dados.

Iremos detalhar as funções buscarPais e buscarTioTia.

A função buscarPais, pode alcançar qualquer casal de ancestrais, desde que ambos existam na planilha, isto eh, o usuário ira inserir qual o grau de ancestralidade (para pai grau 0, avo grau 1, bisavo grau 2, e assim por diante).

```
196
197 informacoes* buscarPais(tabela *t, char nome1, int valor){
198
199     int temp, i;
200     informacoes *aux;
201
202     temp = buscarTabela(t, nome1);
203
204     if(temp == -1){
205         printf("Primeira pessoa nÃ£o encontrada!\n");
206         return 0;
207     }
208     if(t->planilha[temp].registro->pai == NULL){
209         printf("NAO TEM");
210         return NULL;
211     }
212     aux = t->planilha[temp].registro;
213
214     for(i = 0; i < valor; i++){
215         aux = aux ->pai;
216         if(aux->pai == NULL)
217             break;
218     }
219     if(i!=valor)
220         printf("NÃƒO TEM ESTE ANCESTRAL\n");
221     else{
222         printf("Masculino: %c\n",aux->nome );
223         printf("Feminino: %c\n", aux->conjuge->nome);
224     }
225     return aux;
226 }
227
```

Inicialmente a função vai encontrar a pessoa, e verificar se ela não eh o topo da arvore (se ela tem um pai). A partir dai, a função vai ate o grau definido pelo usuário, e após isso imprime o ancestral requerido e seu conjugue.

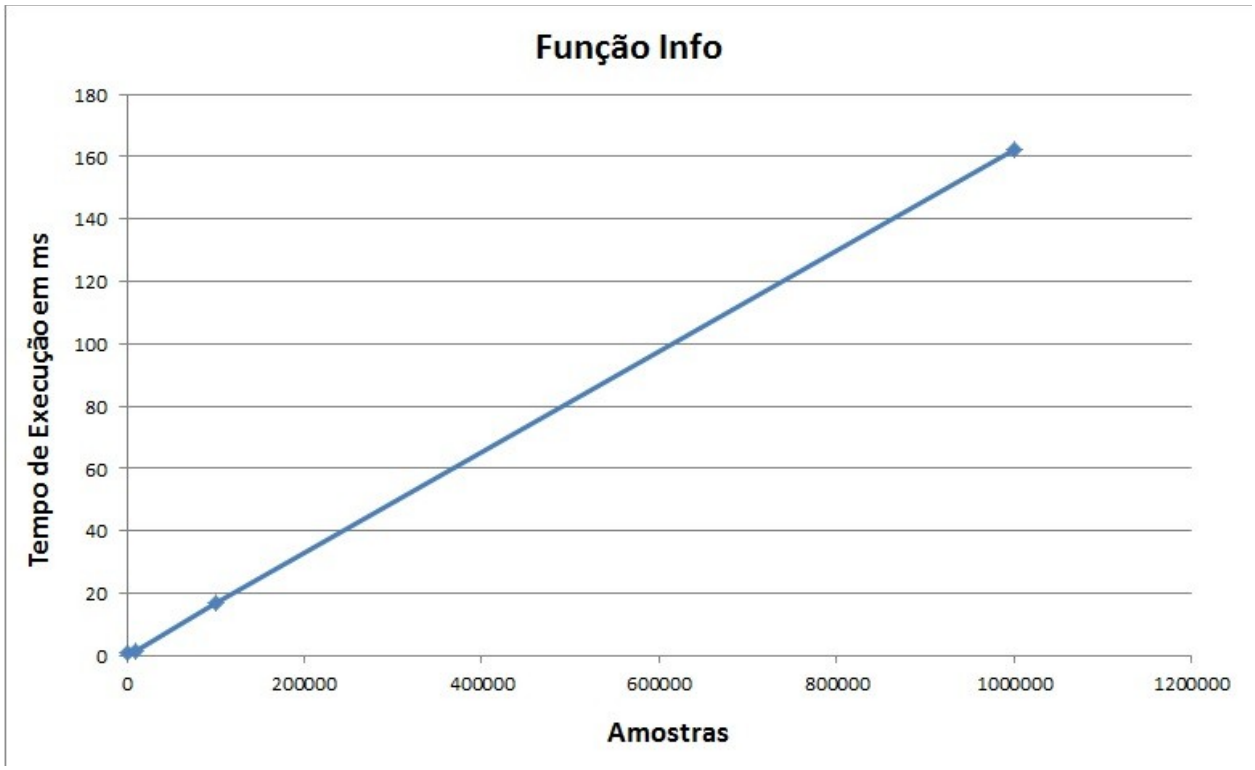
```
informacoes* buscarTioTia(tabela *t, char nome1){

    int temp;
    informacoes *aux;

    temp = buscarTabela(t, nome1);
    if(temp == -1){
        printf("Primeira pessoa nÃ£o encontrada!\n");
        return 0;
    }
    aux = t->planilha[temp].registro->pai->pai->primeiroFilho;

    if (aux->proxIrmao == NULL){
        printf("SEM TIOS");
        return 0;
    }
    if(aux->nome == t->planilha[temp].registro->pai->nome)
        aux = aux->proxIrmao;

    printf("TIO(A): ");

    while(aux!=NULL){

        printf("%c ", aux->nome);
        aux = aux->proxIrmao;

        if(aux == t->planilha[temp].registro->pai)
            aux = aux->proxIrmao;
        if(aux == NULL)
            exit(0);
    }
    return t->planilha[temp].registro->pai->proxIrmao;
}
```

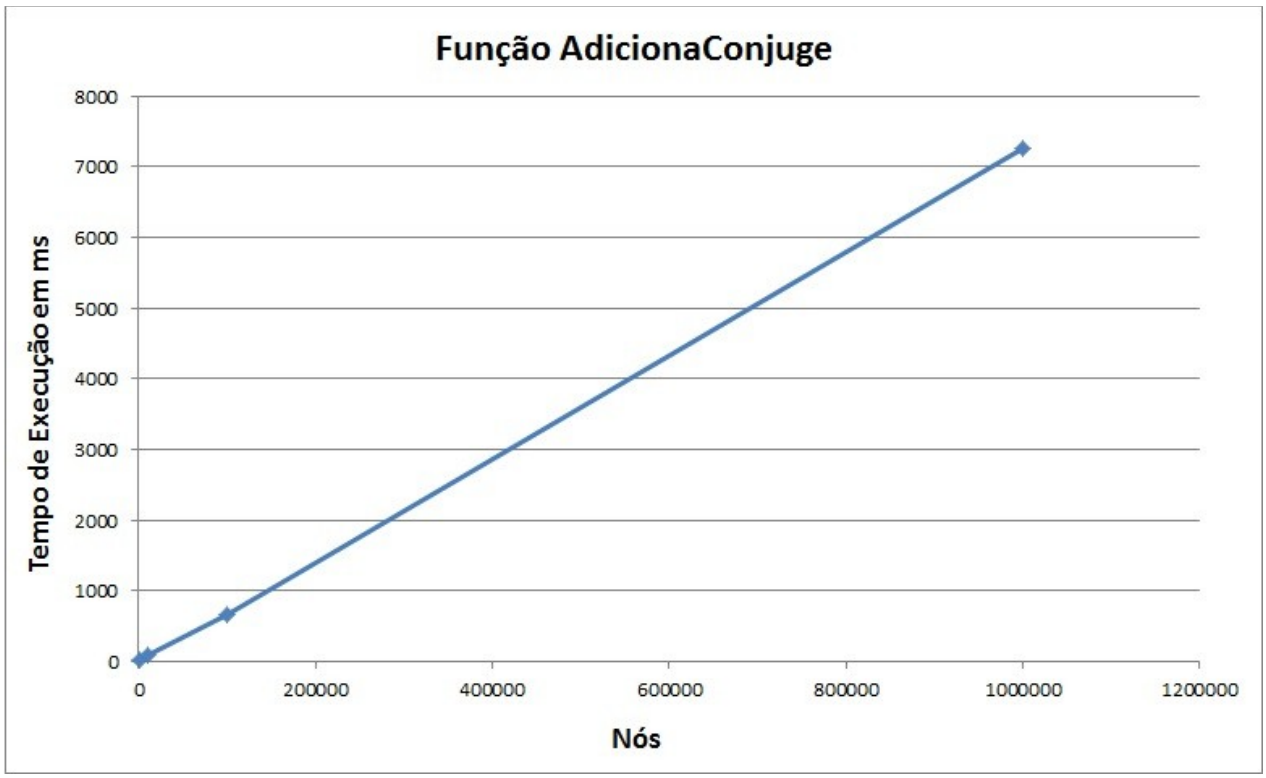
Para a função buscarTioTia, o usuário deve informar a arvore genealógica e o nome da pessoa que terá os tios encontrados. Primeiramente a função ira buscar esta pessoa na tabela e verificar se tem pais ou tios. Para verificar se existem tios, ela busca qual o avo deste individuo e pega o apontamento do seu primeiro filho. A partir dai, com o ponteiro de proximoIrmao, eh possível encontrar cada tio deste individuo.

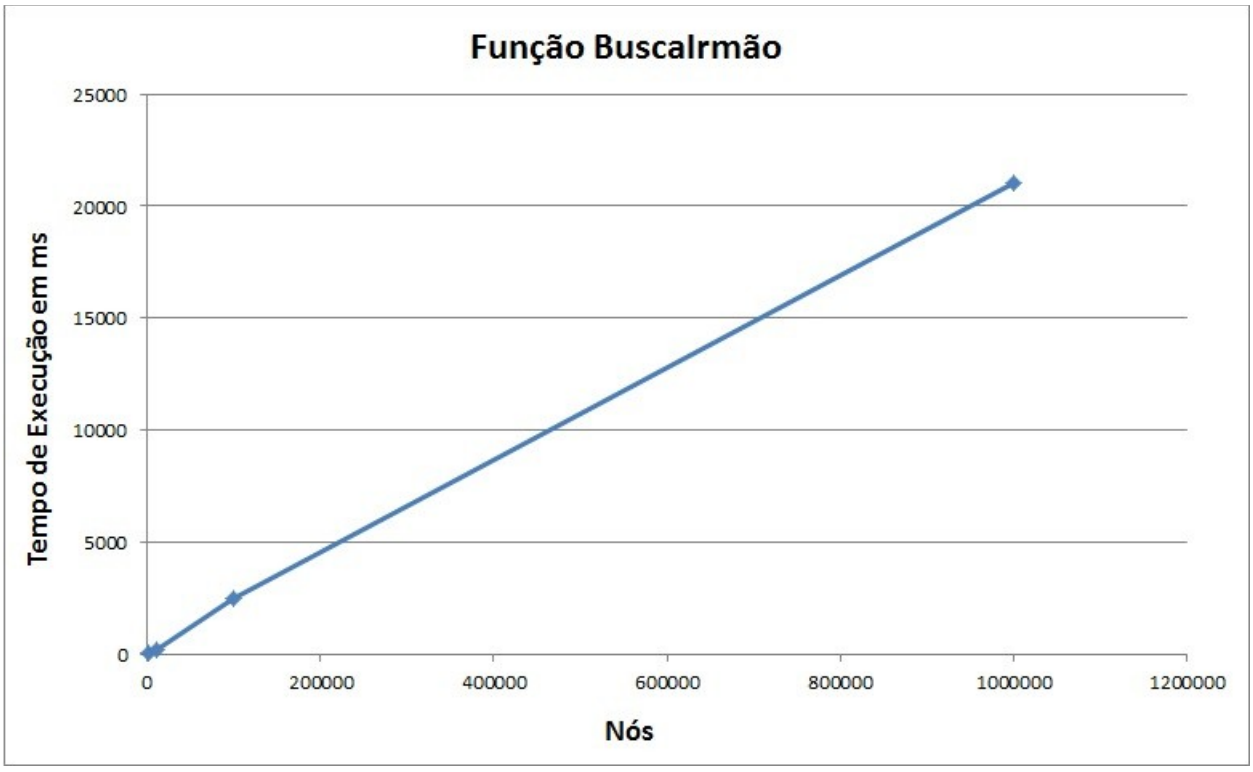
4. ANALISE EXPERIMENTAL

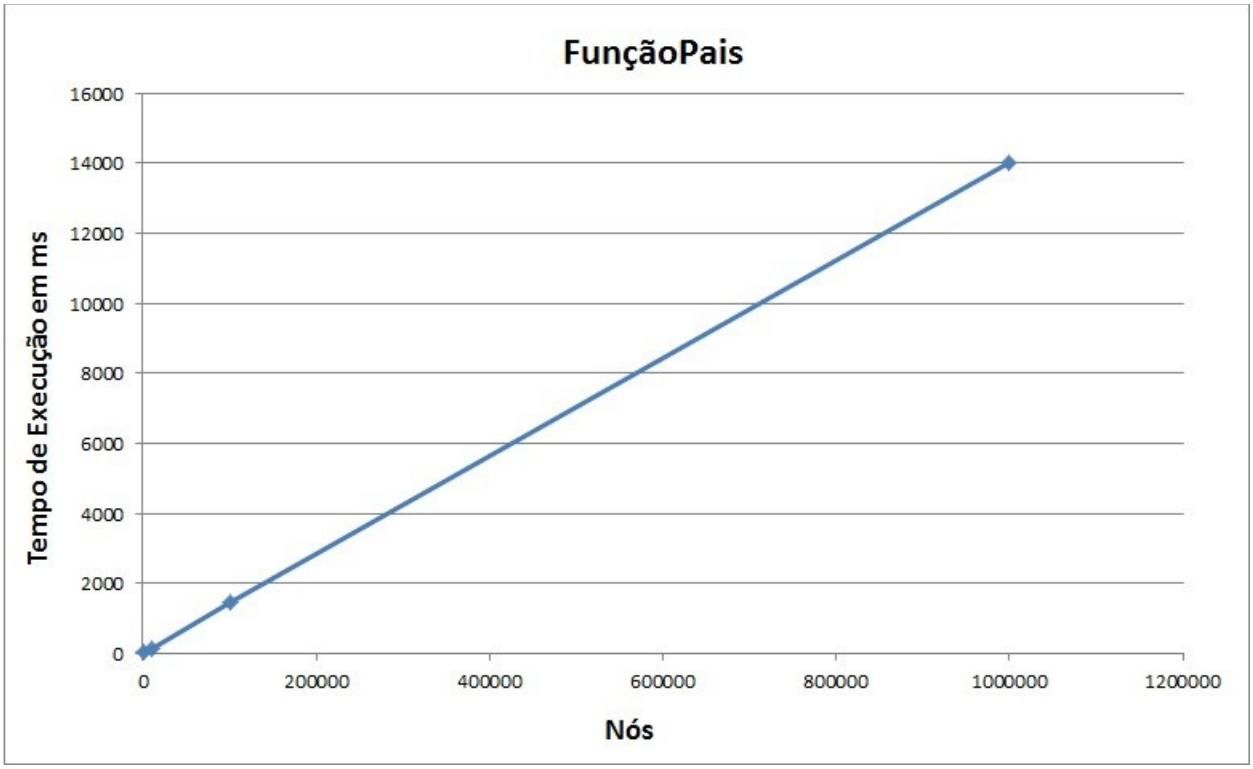
Para verificar a eficiência do código, fizemos os testes de algumas funções, tomando nota do tempo de execução para determinado numero de amostras.

As amostras utilizadas foram as celulas.


Função Info
Tempo de Execução em ms
180
160
140
120
100
80
60
40
20
0
0
200000
400000
600000
800000
1000000
1200000
Amostras


Função AdicionaConjuge
Tempo de Execução em ms
8000
7000
6000
5000
4000
3000
2000
1000
0
0
200000
400000
600000
800000
1000000
1200000
Nós





Todas as funções, seguem uma progressão linear, independente do numero de Nos, Isso nos leva a concluir que a complexidade das funções utilizadas eh O(n), o que demonstra um desempenho satisfatório do programa.