Analise de complexidade -- Árvores – Fernando Laragnoit de Oliveira Neto, Leonardo Moret de Oliveira

**Exercício 1:**

Código:  
#include <stdio.h>

#include <stdlib.h>

#define MAX\_NO 100

#define MAX\_FILHOS 3

struct no {

char letra;

int filhos[MAX\_FILHOS];

};

struct no exemplo1[MAX\_NO];

void inicializar\_arvore() {

int i, j;

for (i = 0; i < MAX\_NO; i++) {

exemplo1[i].letra = '\0';

for (j = 0; j < MAX\_FILHOS; j++) {

exemplo1[i].filhos[j] = -1;

}

}

}

void inserir\_no(int indice, char letra) {

if (indice >= 0 && indice < MAX\_NO) {

exemplo1[indice].letra = letra;

}

}

void adicionar\_filho(int pai, int filho, int posicao) {

if (pai >= 0 && pai < MAX\_NO && filho >= 0 && filho < MAX\_NO && posicao >= 0 && posicao < MAX\_FILHOS) {

exemplo1[pai].filhos[posicao] = filho;

}

}

void remover\_no(int indice) {

if (indice >= 0 && indice < MAX\_NO) {

exemplo1[indice].letra = '\0';

for (int i = 0; i < MAX\_FILHOS; i++) {

int filho = exemplo1[indice].filhos[i];

if (filho != -1) {

remover\_no(filho);

exemplo1[indice].filhos[i] = -1;

}

}

}

}

void imprimir\_arvore() {

for (int i = 0; i < MAX\_NO; i++) {

if (exemplo1[i].letra != '\0') {

printf("\nPai: %c\n", exemplo1[i].letra);

for (int j = 0; j < MAX\_FILHOS; j++) {

if (exemplo1[i].filhos[j] != -1) {

printf("Filho %d: %c ", j, exemplo1[exemplo1[i].filhos[j]].letra);

} else {

printf("Filho %d: - ", j);

}

}

printf("\n");

}

}

}

int busca\_em\_profundidade(int raiz, char letra) {

int pilha[MAX\_NO];

int topo = -1;

pilha[++topo] = raiz;

while (topo != -1) {

int atual = pilha[topo--];

if (atual != -1 && exemplo1[atual].letra != '\0') {

if (exemplo1[atual].letra == letra) {

return atual;

}

for (int i = MAX\_FILHOS - 1; i >= 0; i--) {

if (exemplo1[atual].filhos[i] != -1) {

pilha[++topo] = exemplo1[atual].filhos[i];

}

}

}

}

return -1;

}

int busca\_em\_largura(int raiz, char letra) {

int fila[MAX\_NO];

int inicio = 0, fim = 0;

fila[fim++] = raiz;

while (inicio < fim) {

int atual = fila[inicio++];

if (atual != -1 && exemplo1[atual].letra != '\0') {

if (exemplo1[atual].letra == letra) {

return atual;

}

for (int i = 0; i < MAX\_FILHOS; i++) {

if (exemplo1[atual].filhos[i] != -1) {

fila[fim++] = exemplo1[atual].filhos[i];

}

}

}

}

return -1;

}

int main() {

int opcao, pai, posicao, indice, indice\_atual = 0;

char letra;

inicializar\_arvore();

while (1) {

printf("\nMenu:\n");

printf("1. Inserir nó\n");

printf("2. Imprimir árvore\n");

printf("3. Remover nó\n");

printf("4. Buscar em profundidade (DFS)\n");

printf("5. Buscar em largura (BFS)\n");

printf("6. Sair\n");

printf("Escolha uma opção: ");

scanf("%d", &opcao);

switch (opcao) {

case 1:

printf("Digite o novo nó (letra): ");

scanf(" %c", &letra);

inserir\_no(indice\_atual, letra);

if (indice\_atual > 0) {

printf("Este nó é filho de qual nó (índice)? ");

scanf("%d", &pai);

printf("Qual a posição deste filho (0, 1 ou 2)? ");

scanf("%d", &posicao);

adicionar\_filho(pai, indice\_atual, posicao);

}

indice\_atual++;

break;

case 2:

imprimir\_arvore();

break;

case 3:

printf("Digite o índice do nó a ser removido: ");

scanf("%d", &indice);

remover\_no(indice);

break;

case 4:

printf("Digite a letra do nó a ser encontrado: ");

scanf(" %c", &letra);

indice = busca\_em\_profundidade(0, letra);

if (indice != -1) {

printf("Nó encontrado no índice: %d\n", indice);

} else {

printf("Nó não encontrado.\n");

}

break;

case 5:

printf("Digite a letra do nó a ser encontrado: ");

scanf(" %c", &letra);

indice = busca\_em\_largura(0, letra);

if (indice != -1) {

printf("Nó encontrado no índice: %d\n", indice);

} else {

printf("Nó não encontrado.\n");

}

break;

case 6:

exit(0);

default:

printf("Opção inválida. Tente novamente.\n");

}

}

return 0;

}

**Inicializar\_no**

void inicializar\_arvore() {

int i, j;

for (i = 0; i < MAX\_NO; i++) {

exemplo1[i].letra = '\0';

for (j = 0; j < MAX\_FILHOS; j++) {

exemplo1[i].filhos[j] = -1;

}

}

}

int i, j;

Esse passo só ocorre uma vez;

for (i = 0; i < MAX\_NO; i++) {

exemplo1[i].letra = '\0';

O laço *for* acontece de acordo com o número de iterações determinada, tendo uma notação O de N (O(n))

for (j = 0; j < MAX\_FILHOS; j++) {

exemplo1[i].filhos[j] = -1;

}

Há um outro laço *for*, que também tem notação O de n, sendo assim, temos uma complexidade de ordem de n^2 (O(N^2));

**Inserir\_no**

void inserir\_no(int indice, char letra) {

if (indice >= 0 && indice < MAX\_NO) {

exemplo1[indice].letra = letra;

}

}

O condicional *if* acontece de forma constante, sendo uma única vez quando chamado, portanto tem O(1)

**Adicionar filho**

void adicionar\_filho(int pai, int filho, int posicao) {

if (pai >= 0 && pai < MAX\_NO && filho >= 0 && filho < MAX\_NO && posicao >= 0 && posicao < MAX\_FILHOS) {

exemplo1[pai].filhos[posicao] = filho;

}

}

Como na ultima função, o condicional *if* ocorre somente uma vez

**Remover\_no**

void remover\_no(int indice) {

if (indice >= 0 && indice < MAX\_NO) {

exemplo1[indice].letra = '\0';

for (int i = 0; i < MAX\_FILHOS; i++) {

int filho = exemplo1[indice].filhos[i];

if (filho != -1) {

remover\_no(filho);

exemplo1[indice].filhos[i] = -1;

}

}

}

}

if (indice >= 0 && indice < MAX\_NO) {

Essa passagem ocorre somente uma vez quando a função é chamada

for (int i = 0; i < MAX\_FILHOS; i++) {

int filho = exemplo1[indice].filhos[i];

O laço *for* ocorre de acordo com as condições dadas, portanto, possui O(N)

if (filho != -1) {

remover\_no(filho);

exemplo1[indice].filhos[i] = -1;

}

O condicional *if* ocorre somente uma vez por interação do laço *for*, portanto, a função possui complexidade de O(n)

**Imprimir\_arvore**

void imprimir\_arvore() {

for (int i = 0; i < MAX\_NO; i++) {

if (exemplo1[i].letra != '\0') {

printf("\nPai: %c\n", exemplo1[i].letra);

for (int j = 0; j < MAX\_FILHOS; j++) {

if (exemplo1[i].filhos[j] != -1) {

printf("Filho %d: %c ", j, exemplo1[exemplo1[i].filhos[j]].letra);

} else {

printf("Filho %d: - ", j);

}

}

printf("\n");

}

}

}

for (int i = 0; i < MAX\_NO; i++) {

if (exemplo1[i].letra != '\0') {

printf("\nPai: %c\n", exemplo1[i].letra);

O laço *for* ocorre de acordo com o número de iterações passadas, portanto, possuí uma complexidade O(n)

if (exemplo1[i].letra != '\0') {

printf("\nPai: %c\n", exemplo1[i].letra);

O condicional *if* ocorre apenas uma vez por iteração do laço *for*, portanto tem complexidade constante

for (int j = 0; j < MAX\_FILHOS; j++) {

if (exemplo1[i].filhos[j] != -1) {

printf("Filho %d: %c ", j, exemplo1[exemplo1[i].filhos[j]].letra);

} else {

printf("Filho %d: - ", j);

}

}

printf("\n");

O segundo laço *for* também possui complexidade N, e os condicionais ocorrem somente uma vez a cada iteração, a função portanto, tem complexidade O(N^2)

**Busca\_em\_complexidade**

int busca\_em\_profundidade(int raiz, char letra) {

int pilha[MAX\_NO];

int topo = -1;

pilha[++topo] = raiz;

while (topo != -1) {

int atual = pilha[topo--];

if (atual != -1 && exemplo1[atual].letra != '\0') {

if (exemplo1[atual].letra == letra) {

return atual;

}

for (int i = MAX\_FILHOS - 1; i >= 0; i--) {

if (exemplo1[atual].filhos[i] != -1) {

pilha[++topo] = exemplo1[atual].filhos[i];

}

}

}

}

return -1;

while (topo != -1) {

int atual = pilha[topo--];

if (atual != -1 && exemplo1[atual].letra != '\0') {

if (exemplo1[atual].letra == letra) {

return atual;

}

O laço while ocorre numa orden de N, e os condicionais acontecem somente uma vez por iteração

for (int i = MAX\_FILHOS - 1; i >= 0; i--) {

if (exemplo1[atual].filhos[i] != -1) {

pilha[++topo] = exemplo1[atual].filhos[i];

}

O segundo for possui uma complexidade N, sendo assim, temos uma complexidade de O(N^2)

**Busca\_em\_Largura**

int busca\_em\_largura(int raiz, char letra) {

int fila[MAX\_NO];

int inicio = 0, fim = 0;

fila[fim++] = raiz;

while (inicio < fim) {

int atual = fila[inicio++];

if (atual != -1 && exemplo1[atual].letra != '\0') {

if (exemplo1[atual].letra == letra) {

return atual;

}

for (int i = 0; i < MAX\_FILHOS; i++) {

if (exemplo1[atual].filhos[i] != -1) {

fila[fim++] = exemplo1[atual].filhos[i];

}

}

}

}

Nessa função temos um laço *while* e um laço *for*, dentro dele, sendo assim, temos uma complexidade de ordem O(N^2)

**Exercício 2**

**Código:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_FILHOS 3

#define TAMANHO\_PILHA\_FILA 100

struct no {

char letra;

struct no\* filhos[MAX\_FILHOS];

struct no\* prox;

};

struct no\* inicio = NULL;

struct no\* nos[100];

int indice\_no = 0;

struct no\* criar\_no(char letra) {

struct no\* novo\_no = (struct no\*)malloc(sizeof(struct no));

novo\_no->letra = letra;

for (int i = 0; i < MAX\_FILHOS; i++) {

novo\_no->filhos[i] = NULL;

}

novo\_no->prox = NULL;

return novo\_no;

}

void inserir\_no(char letra) {

struct no\* novo\_no = criar\_no(letra);

nos[indice\_no++] = novo\_no;

if (inicio == NULL) {

inicio = novo\_no;

} else {

struct no\* temp = inicio;

while (temp->prox != NULL) {

temp = temp->prox;

}

temp->prox = novo\_no;

}

}

void adicionar\_filho(struct no\* pai, struct no\* filho, int posicao) {

if (pai != NULL && posicao >= 0 && posicao < MAX\_FILHOS) {

pai->filhos[posicao] = filho;

}

}

void remover\_no(struct no\* pai, struct no\* no\_a\_remover) {

if (no\_a\_remover == NULL) return;

for (int i = 0; i < MAX\_FILHOS; i++) {

if (no\_a\_remover->filhos[i] != NULL) {

remover\_no(no\_a\_remover, no\_a\_remover->filhos[i]);

no\_a\_remover->filhos[i] = NULL;

}

}

if (pai != NULL) {

for (int i = 0; i < MAX\_FILHOS; i++) {

if (pai->filhos[i] == no\_a\_remover) {

pai->filhos[i] = NULL;

}

}

}

if (inicio == no\_a\_remover) {

inicio = no\_a\_remover->prox;

} else {

struct no\* temp = inicio;

while (temp->prox != NULL && temp->prox != no\_a\_remover) {

temp = temp->prox;

}

if (temp->prox == no\_a\_remover) {

temp->prox = no\_a\_remover->prox;

}

}

free(no\_a\_remover);

}

void imprimir\_arvore(struct no\* no\_atual) {

if (no\_atual == NULL) return;

printf("Pai: %c\n", no\_atual->letra);

for (int i = 0; i < MAX\_FILHOS; i++) {

if (no\_atual->filhos[i] != NULL) {

printf(" Filho %d: %c\n", i, no\_atual->filhos[i]->letra);

} else {

printf(" Filho %d: -\n", i);

}

}

if (no\_atual->prox != NULL) {

imprimir\_arvore(no\_atual->prox);

}

}

int busca\_em\_profundidade(struct no\* raiz, char letra) {

if (raiz == NULL) return -1;

struct no\* pilha[TAMANHO\_PILHA\_FILA];

int topo = -1;

pilha[++topo] = raiz;

while (topo != -1) {

struct no\* atual = pilha[topo--];

if (atual != NULL) {

if (atual->letra == letra) {

for (int i = 0; i < indice\_no; i++) {

if (nos[i] == atual) {

return i;

}

}

}

for (int i = MAX\_FILHOS - 1; i >= 0; i--) {

if (atual->filhos[i] != NULL) {

pilha[++topo] = atual->filhos[i];

}

}

}

}

return -1;

}

int busca\_em\_largura(struct no\* raiz, char letra) {

if (raiz == NULL) return -1;

struct no\* fila[TAMANHO\_PILHA\_FILA];

int inicio = 0, fim = 0;

fila[fim++] = raiz;

while (inicio < fim) {

struct no\* atual = fila[inicio++];

if (atual != NULL) {

if (atual->letra == letra) {

for (int i = 0; i < indice\_no; i++) {

if (nos[i] == atual) {

return i;

}

}

}

for (int i = 0; i < MAX\_FILHOS; i++) {

if (atual->filhos[i] != NULL) {

fila[fim++] = atual->filhos[i];

}

}

}

}

return -1;

}

int main() {

int opcao, posicao;

char letra, pai\_letra;

while (1) {

printf("\nMenu:\n");

printf("1. Inserir nó\n");

printf("2. Imprimir árvore\n");

printf("3. Remover nó\n");

printf("4. Busca em profundidade (DFS)\n");

printf("5. Busca em largura (BFS)\n");

printf("6. Sair\n");

printf("Escolha uma opção: ");

scanf("%d", &opcao);

switch (opcao) {

case 1:

printf("Digite o novo nó (letra): ");

scanf(" %c", &letra);

inserir\_no(letra);

if (indice\_no > 1) {

printf("Este nó é filho de qual nó (letra)? ");

scanf(" %c", &pai\_letra);

printf("Qual a posição deste filho (0, 1 ou 2)? ");

scanf("%d", &posicao);

struct no\* pai = inicio;

while (pai != NULL && pai->letra != pai\_letra) {

pai = pai->prox;

}

if (pai != NULL) {

adicionar\_filho(pai, nos[indice\_no - 1], posicao);

}

}

break;

case 2:

imprimir\_arvore(inicio);

break;

case 3:

printf("Digite o nó a ser removido (letra): ");

scanf(" %c", &letra);

struct no\* pai = NULL;

struct no\* no\_a\_remover = inicio;

while (no\_a\_remover != NULL && no\_a\_remover->letra != letra) {

pai = no\_a\_remover;

no\_a\_remover = no\_a\_remover->prox;

}

if (no\_a\_remover != NULL) {

remover\_no(pai, no\_a\_remover);

}

break;

case 4:

printf("Digite a letra do nó a ser encontrado: ");

scanf(" %c", &letra);

int indice = busca\_em\_profundidade(inicio, letra);

if (indice != -1) {

printf("Nó encontrado no índice: %d\n", indice);

} else {

printf("Nó não encontrado.\n");

}

break;

case 5:

printf("Digite a letra do nó a ser encontrado: ");

scanf(" %c", &letra);

indice = busca\_em\_largura(inicio, letra);

if (indice != -1) {

printf("Nó encontrado no índice: %d\n", indice);

} else {

printf("Nó não encontrado.\n");

}

break;

case 6:

exit(0);

default:

printf("Opção inválida. Tente novamente.\n");

}

}

return 0;

}

**Criar\_no**

struct no\* criar\_no(char letra) {

struct no\* novo\_no = (struct no\*)malloc(sizeof(struct no));

novo\_no->letra = letra;

for (int i = 0; i < MAX\_FILHOS; i++) {

novo\_no->filhos[i] = NULL;

}

novo\_no->prox = NULL;

return novo\_no;

}

Alocar memória: malloc(sizeof(struct no)) é O(1).

Atribuir o valor de letra ao novo nó é O(1).

O loop inicializa cada ponteiro filhos com NULL. O loop é O(MAX\_FILHOS), onde MAX\_FILHOS é uma constante.

Atribuir prox a NULL é O(1).

A função retorna o ponteiro para o novo nó, O(1).

Complexidade total: O(1), porque MAX\_FILHOS é uma constante.

**Inserir\_no**

void inserir\_no(char letra) {

struct no\* novo\_no = criar\_no(letra);

nos[indice\_no++] = novo\_no;

if (inicio == NULL) {

inicio = novo\_no;

} else {

struct no\* temp = inicio;

while (temp->prox != NULL) {

temp = temp->prox;

}

temp->prox = novo\_no;

}

}

Criar um novo nó é O(1).

Inserir o novo nó no array nos é O(1).

Verificar se inicio é NULL e atribuir o novo nó a inicio é O(1).

Se inicio não é NULL, percorrer a lista ligada até encontrar o último nó é O(n), onde n é o número de nós na lista.

Atribuir prox do último nó ao novo nó é O(1).

Complexidade total: O(n), onde n é o número de nós na lista ligada.

**Adicionar\_filho**

void adicionar\_filho(struct no\* pai, struct no\* filho, int posicao) {

if (pai != NULL && posicao >= 0 && posicao < MAX\_FILHOS) {

pai->filhos[posicao] = filho;

}

}

Verificar se o pai é não nulo e se a posição é válida é O(1).

Atribuir o filho na posição especificada é O(1).

Complexidade total: O(1).

**Remover\_no**

void remover\_no(struct no\* pai, struct no\* no\_a\_remover) {

if (no\_a\_remover == NULL) return;

for (int i = 0; i < MAX\_FILHOS; i++) {

if (no\_a\_remover->filhos[i] != NULL) {

remover\_no(no\_a\_remover, no\_a\_remover->filhos[i]);

no\_a\_remover->filhos[i] = NULL;

}

}

if (pai != NULL) {

for (int i = 0; i < MAX\_FILHOS; i++) {

if (pai->filhos[i] == no\_a\_remover) {

pai->filhos[i] = NULL;

}

}

}

if (inicio == no\_a\_remover) {

inicio = no\_a\_remover->prox;

} else {

struct no\* temp = inicio;

while (temp->prox != NULL && temp->prox != no\_a\_remover) {

temp = temp->prox;

}

if (temp->prox == no\_a\_remover) {

temp->prox = no\_a\_remover->prox;

}

}

free(no\_a\_remover);

}

Se no\_a\_remover é NULL, retornar é O(1).

O loop para remover os filhos de no\_a\_remover é O(MAX\_FILHOS). Cada chamada recursiva para remover\_no é O(m), onde m é o número de nós na subárvore.

Verificar e remover no\_a\_remover dos filhos do pai é O(MAX\_FILHOS).

Atualizar inicio se no\_a\_remover é o início é O(1).

Percorrer a lista ligada para encontrar o nó anterior ao no\_a\_remover é O(n), onde n é o número de nós na lista.

Liberar a memória do nó removido é O(1).

Complexidade total: O(n + m), onde n é o número de nós na lista ligada e m é o número de nós na subárvore.

**Imprimir\_arvore**

void imprimir\_arvore(struct no\* no\_atual) {

if (no\_atual == NULL) return;

printf("Pai: %c\n", no\_atual->letra);

for (int i = 0; i < MAX\_FILHOS; i++) {

if (no\_atual->filhos[i] != NULL) {

printf(" Filho %d: %c\n", i, no\_atual->filhos[i]->letra);

} else {

printf(" Filho %d: -\n", i);

}

}

if (no\_atual->prox != NULL) {

imprimir\_arvore(no\_atual->prox);

}

}

Se no\_atual é NULL, retornar é O(1).

Imprimir o valor do nó atual é O(1).

O loop para imprimir os filhos do nó atual é O(MAX\_FILHOS).

Chamar recursivamente para o próximo nó na lista ligada é O(n), onde n é o número de nós na lista.

Complexidade total: O(n \* MAX\_FILHOS). Como MAX\_FILHOS é uma constante, simplificamos para O(n).

**Busca\_em\_profundidade**

int busca\_em\_profundidade(struct no\* raiz, char letra) {

if (raiz == NULL) return -1;

struct no\* pilha[TAMANHO\_PILHA\_FILA];

int topo = -1;

pilha[++topo] = raiz;

while (topo != -1) {

struct no\* atual = pilha[topo--];

if (atual != NULL) {

if (atual->letra == letra) {

for (int i = 0; i < indice\_no; i++) {

if (nos[i] == atual) {

return i;

}

}

}

for (int i = MAX\_FILHOS - 1; i >= 0; i--) {

if (atual->filhos[i] != NULL) {

pilha[++topo] = atual->filhos[i];

}

}

}

}

return -1;

}

Se a raiz é NULL, retornar -1 é O(1).

Inicializar a pilha é O(1).

Empilhar a raiz é O(1).

O loop principal roda enquanto a pilha não está vazia. No pior caso, percorre todos os nós, O(n).

Desempilhar o nó atual é O(1).

Comparar o valor da letra é O(1).

Procurar o índice no array nos é O(n).

O loop para empilhar os filhos do nó atual é O(MAX\_FILHOS).

Complexidade total: O(n \* MAX\_FILHOS). Simplificada para O(n) porque MAX\_FILHOS é constante.

**Busca\_em\_largura**

int busca\_em\_largura(struct no\* raiz, char letra) {

if (raiz == NULL) return -1;

struct no\* fila[TAMANHO\_PILHA\_FILA];

int inicio = 0, fim = 0;

fila[fim++] = raiz;

while (inicio < fim) {

struct no\* atual = fila[inicio++];

if (atual != NULL) {

if (atual->letra == letra) {

for (int i = 0; i < indice\_no; i++) {

if (nos[i] == atual) {

return i;

}

}

}

for (int i = 0; i < MAX\_FILHOS; i++) {

if (atual->filhos[i] != NULL) {

fila[fim++] = atual->filhos[i];

}

}

}

}

return -1;

Se a raiz é NULL, retornar -1 é O(1).

Inicializar a fila é O(1).

Enfileirar a raiz é O(1).

O loop principal roda enquanto a fila não está vazia. No pior caso, percorre todos os nós, O(n).

Desenfileirar o nó atual é O(1).

Comparar o valor da letra é O(1).

Procurar o índice no array nos é O(n).

O loop para enfileirar os filhos do nó atual é O(MAX\_FILHOS).

Complexidade total: O(n \* MAX\_FILHOS). Simplificada para O(n) porque MAX\_FILHOS é constante.