SCC0220 - Laboratório Introdução à Ciência da Computação II

Relatório de execução do trabalho prático 9

Alunos	**NUSP**
Alec Campos Aoki	15436800
Juan Henrique Passos	15464826

Trabalho 9 – O melhor caminho

O melhor caminho

Comentário

O trabalho prático 9, o melhor caminho, consistia em dado uma árvore de busca binária de tamanho n, sendo $0 < n < 10^5$, procurar um determinado elemento k, sendo $0 < k < 10^6$, partindo da raiz da árvore. Observa-se que todos os elementos da árvore possuem uma chave que compreende esse intervalo de k, e conforme se busca o k, deve-se imprimir o caminho a ser feito, até chegar em k. Caso não seja encontrado no final do caminho, imprimi-se -1.

A árvore binária de busca é uma estrutura de dados que permite uma ótima eficiência de memória, haja vista sua implementação por meio de nós, que são alocados dinamicamente conforme a necessidade de guardar um valor. Sua construção é feita da seguinte forma: começa-se a inserir o elemento pela raiz, se o elemento for menor que o nó atual, busca-se ir para o filho da esquerda, se for maior, vai para o filho da direita(observa-se que não trabalhamos com chaves iguais). Essa lógica segue até o nó for nulo, pois assim, como em cada chamada retorna-se o nó, basta criar um nó, atribuir a chave a ele, e retorná-lo, pois em cada chamada, o ponteiro para a direção que foi escolhida, recebe esse nó. Vale ressaltar, que se a árvore estiver vazia, também funciona, tendo em vista que cairá no caso de ser nulo na primeira chamada, e a função de inserir não recursiva, receberá esse nó como sendo a raiz.

Além disso, cabe ressaltar que é possível inserir, buscar e remover um elemento em O(logn) no melhor caso(árvore balanceada), tendo em vista a sua implementação que na esquerda possui um elemento menor que o nó atual e a direita um elemento maior que o nó atual. Dessa forma, seguimos apenas uma caminho da árvore(aquele em que a corresponde a chave seguindo o algoritmo supracitado) e se ela estiver balanceada, elimina-se metade dos elementos, semelhante a busca binária em um array.

Código

```c
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<time.h>

typedef struct no_ NO;
```

```c
struct no_
{
    int chave;
    NO *esq, *dir;
};

typedef struct arvore_
{
    NO *raiz;
    int tamanho;
}ARVORE;

// Modularização
void imprimi(NO *no);
NO* pegarSuccessor(NO* no);
NO* inserir(NO *raiz, int chave);
void apagar(NO **raiz);
bool pertence(NO *raiz, int chave);
NO *remover(NO* raiz, int chave);
void caminho(NO *raiz, int chave);

ARVORE *arvore_criar()
{
    ARVORE *arvore = (ARVORE*) malloc(sizeof(ARVORE));
    if(arvore != NULL)
    {
        arvore->raiz = NULL;
        arvore->tamanho = 0;
    }
    return arvore;
}

void arvore_apagar(ARVORE **arvore)
{
    if(*arvore != NULL)
    {
        apagar(&((*arvore)->raiz));
        free(*arvore);
```

```
        *arvore = NULL;
    }
}

void apagar(NO **raiz)
{
    if(*raiz == NULL)
        return;

    apagar(&((*raiz)->esq));
    apagar(&((*raiz)->dir));

    free(*raiz);
    *raiz = NULL;
}

void arvore_inserir(ARVORE *arvore, int chave)
{
    if(arvore != NULL)
    {
        arvore->raiz = inserir(arvore->raiz, chave);
        arvore->tamanho++;
    }
}

NO* inserir(NO *raiz, int chave)
{
    // Adiciona o no quando o no NULO é encontrado.
    if (raiz == NULL) {
        NO* novo_no = (NO*) malloc(sizeof(NO));
        novo_no->chave = chave;
        novo_no->esq =  NULL;
        novo_no->dir = NULL;
        return novo_no;
    }

    // Percorra a subárvore esquerda se os dados
    // forem menores que o nó atual
```

```
    if (chave < raiz->chave) {
        raiz->esq = inserir(raiz->esq, chave);
        return raiz;
    }

    // Percorra a subárvore direita se os dados
    // forem maiores que o nó atual
    else if (chave > raiz->chave) {
        raiz->dir = inserir(raiz->dir, chave);
        return raiz;
    }
    else
        return raiz;
}

// Retorna se esse elemento pertence ao arvore.
bool arvore_pertence(ARVORE *arvore, int chave)
{
    if(arvore != NULL)
    {
        return pertence(arvore->raiz, chave);
    }
    return false;
}

// Verifica se o elemento está no NO, caso não, verifica-se para os filhos.
bool pertence(NO *raiz, int chave)
{
    if (raiz == NULL)
        return 0;

    if(raiz->chave == chave)
        return true;

    return pertence(raiz->esq, chave) || pertence(raiz->dir, chave);
}

void menor_caminho(ARVORE *arvore, int chave)
```

```
{
    if(arvore != NULL)
    {
        // Buscar e imprimir caminho
        caminho(arvore->raiz, chave);
    }

    return;
}

void caminho(NO *raiz, int chave)
{
    if(raiz != NULL)
    {
        printf("%d ", raiz->chave);

        if(raiz->chave == chave)
            return;
        else if(raiz->chave < chave)
            caminho(raiz->dir, chave);
        else
            caminho(raiz->esq, chave);
    }
    else printf("%d ", -1);
    return;
}

int main()
{
    int tamanho_arvore;

    scanf("%d", &tamanho_arvore);

    clock_t start, end;
    double cpu_tempo_ms;

    ARVORE *arvore = arvore_criar();
```

```
    for(int i = 0; i < tamanho_arvore; i++)
    {
        int aux;
        scanf("%d", &aux);
        arvore_inserir(arvore, aux);
    }

    int numero_buscado;
    scanf("%d", &numero_buscado);

    start = clock();

    menor_caminho(arvore, numero_buscado);

    end = clock();

    cpu_tempo_ms = (double) (end - start)/ CLOCKS_PER_SEC *1000;

    printf("\nTempo de execucao: %lfms", cpu_tempo_ms);

    arvore_apagar(&arvore);

    return 0;

}
/*
Caso teste:
9
8 3 10 1 6 14 4 7 13
7
*/
```

Saída

```
→  Aula09 git:(main) ✗ ./BestPath < casosteste/10.in
499747 620559 522549 599175 535843 583163 575340 547049 539411 544968 54329
3 539935 542123 539979 541608 540240 540535 541387 540867 541130 541041 541
019 540982 540952 540891 540944 540926 540897
Tempo de execucao: 0.011000ms
```

```
→  Aula09 git:(main) ✗ ./BestPath < casosteste/10.in
499747 620559 522549 599175 535843 583163 575340 547049 539411 544968 54329
3 539935 542123 539979 541608 540240 540535 541387 540867 541130 541041 541
019 540982 540952 540891 540944 540926 540897
Tempo de execucao: 0.009000ms
```

```
→  Aula09 git:(main) ✗ ./BestPath < casosteste/10.in
499747 620559 522549 599175 535843 583163 575340 547049 539411 544968 54329
3 539935 542123 539979 541608 540240 540535 541387 540867 541130 541041 541
019 540982 540952 540891 540944 540926 540897
Tempo de execucao: 0.013000ms
```

```
→  Aula09 git:(main) ✗ ./BestPath < casosteste/10.in
499747 620559 522549 599175 535843 583163 575340 547049 539411 544968 54329
3 539935 542123 539979 541608 540240 540535 541387 540867 541130 541041 541
019 540982 540952 540891 540944 540926 540897
Tempo de execucao: 0.010000ms
```

Gráfico


Tempo de execução pelo tamanho da árvore
60
40
20
0
ms
20000
40000
60000
80000
Tamanho