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, insere-se no filho da esquerda, se for maior, insere-se no filho da direita (observa-se que não trabalhamos com chaves iguais). A lógica de busca é análoga e segue até o nó for nulo. Vale ressaltar que se a árvore estiver vazia, a busca também funciona, tendo em vista que cairá no caso de ser nulo na primeira chamada.

As operaçẽos de inserir, buscar e remover um elemento são $O(log(n))$ 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. No pior caso, a complexidade dessas operações é $O(n)$.

Para comparação, utilizamos também uma busca em profundidade (recursiva), análoga à busca sequencial em um array. Nessa busca, percorremos todos os nós esquerdos ao nó raiz, conferindo se a chave de um eles é a buscada. Ao chamarmos a função em um raiz nula, retornamos (*backtracking)* até a raíz e executamos a busca com os nós à direita dela. A complexidade dessa busca é $O(n)$.

Código

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

```c
typedef struct no_ NO;

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


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


// Modularização
NO* inserir(NO *raiz, int chave);
void apagar(NO **raiz);
bool pertence(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;
   }
```

```
    return;
}

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

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


    free(*raiz);
    *raiz = NULL;

    return;
}

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

    return;
}

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;

    printf("%d ", raiz->chave);

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


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

```c
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(void){
    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);
    //pertence(arvore->raiz, numero_buscado);

    end = clock();

    printf("\nBusca binária\n");
    //printf("\nBusca sequencial\n");

    cpu_tempo_ms = (double) (end - start)/ CLOCKS_PER_SEC *1000;
    printf("\nTempo de execucao: %lfms\n", cpu_tempo_ms);

    arvore_apagar(&arvore);

    return 0;
}
```

Saída

```
92 540190 540186 540202 540217 540224 540238 540230 540535 540514 540316
540283 540263 540285 540309 540287 540292 540291 540486 540482 540420 540
376 540342 540350 540380 540399 540467 540430 540424 540457 540452 540465
 540498 540492 540495 541387 540867 540821 540724 540605 540590 540567 54
0547 540543 540579 540574 540603 540722 540718 540618 540660 540646 54062
2 540626 540652 540685 540673 540667 540753 540744 540733 540801 540763 5
40840 540830 540826 540822 540838 540849 541130 541041 541019 540982 5409
52 540891 540944 540926 540897
Busca sequencial

Tempo de execucao: 24.170000ms
→  Aula09 git:(main) ✗
```

```
→  Aula09 git:(main) ✗ gcc BestPath.c -o BestPath -std=c99 -Wall
→  Aula09 git:(main) ✗ ./BestPath < casosteste/10.in
499747 620559 522549 599175 535843 583163 575340 547049 539411 544968 543
293 539935 542123 539979 541608 540240 540535 541387 540867 541130 541041
 541019 540982 540952 540891 540944 540926 540897
Busca binária

Tempo de execucao: 0.028000ms
→  Aula09 git:(main) ✗
```

Gráfico