Algoritmos e Estruturas de Dados 3 Trabalho 1 - AVL

Jorge Lucas Vicilli Jabczenski & Vinicius Tikara Venturi Date

21 de Dezembro 2020

Estrutura Nodo

A estrutura de dados do nodo foi construída sem nodo pai, apenas nodo da direita e esquerda, pois a utilização do nodo pai tornava as operações mais complexas e não apresentava benefícios para nossa implementação das outras funções.

Também eram guardados os valores da chave e altura, sendo este último um valor comumente usado para operações, tais como as que envolvem o fator de balanceamento da AVL.

```
struct tNodo {
    struct tNodo *esquerda, *direita;
    int chave, altura;
}
```

Altura

É calculada de tal forma que, em toda inclusão e exclusão, a altura dos nodos que participaram dessas operações é atualizada recursivamente. Além disso, a altura é arrumada nas rotações, visto que os dois nodos participantes tem seus lugares trocados e, consequentemente, suas alturas.

Busca e Ordenação

Nas funções de inclusão e exclusão foi utilizado o conceito de árvore binária de busca para achar o nodo correto para as operações, e assim, simultaneamente, os ordenando.

```
tNodo *foo(tNodo *nodo,int chave){
...
    if(chave < nodo->chave)
        nodo->esquerda = foo(nodo->esquerda, chave);
    else
        nodo->direita  = foo(nodo->direita, chave);
...
```

Exclusão

Tratamos quatro casos na exclusão, sendo três deles triviais: quando é nodo folha, tem apenas o filho da direita ou tem apenas filho da esquerda.

O quarto e mais complicado caso, aquele no qual o nodo a ser excluído tem ambos os filhos, tem como base pegar o maior nodo de sua subárvore à esquerda e trocar os valores destes nodos, a fim de manter os ponteiros coerentes.

```
...
    /* Como sabemos que o nodo tem 2 filhos, podemos acessar o nodo->esquerda sem
    perigo de acessar uma area de memoria nao alocada */
    tNodo *nodoAntecessor = maximo(nodo->esquerda);
    /* Substituir as chaves do nodo atual com o do Antecessor */
    nodo->chave = nodoAntecessor->chave;
    /* Excluir o Antecessor */
    nodo->esquerda = excluir(nodo->esquerda, nodoAntecessor->chave);
...
```

Balanceamento

O balanceamento da AVL foi feito de tal forma que, quando ocorre uma operação nos nodos - como a inserção e a exclusão - é preciso checar a propriedade da AVL em todos os nodos que foram visitados. A função de balanceamento é chamada em todos os passos das funções anteriormente citadas, até chegar na raiz.

Para checar se a propriedade da AVL é válida, calculamos o fator de balanceamento:

$$fator = altura(nodo \rightarrow esquerda) - altura(nodo \rightarrow direita)$$

Caso esteja desrespeitando a propriedade da AVL, ou seja, fora do conjunto $\{-1, 0, 1\}$, é necessário rotacionar de acordo, com um exemplo a seguir de um dos casos:

```
...
    /* Desbalanceado para a Direita */
    if (fatorBalanceamento(nodo) < -1)
    {
         /* Direita-Esquerda (ZigZag) */
        if (fatorBalanceamento(nodo->direita) > 0)
            nodo->direita = rotacaoDireita(nodo->direita);
        /* Direita-Direita OU segundo passo da Direita-Esquerda */
        return rotacaoEsquerda(nodo);
    }
...
```

Sendo o caso acima aquele que a subárvore da direita é a causadora do desbalanceamento, e, então, checando se é necessário uma rotação simples, ou se precisamos arrumar antes com outra rotação (caso ZigZag).

O caso da subárvore da esquerda é simétrico (quando o fator de balanceamento é maior que 1).

Profundidade

O algoritmo de profundidade foi baseado no já conhecido *EmOrdem*, utilizado para andar por uma árvore e imprimir seus valores sequencialmente. A alteração feita foi a adição de uma variável que guarda o valor do quão "fundo" o nodo está, baseado nas chamadas recursivas.

A primeira chamada é feita no nodo raiz com profundidade zero e as demais profundidades são calculadas recursivamente.

```
...
    if (nodo != NULL){
        profundidade(nodo->esquerda, n+1);
        printf("%d,%d\n", nodo->chave, n);
        profundidade(nodo->direita , n+1);
    }
...
/* Chamada na main(), apos a leitura dos dados */
profundidade(raiz,0);
```

Entrada de Dados

Para ler os dados de teste que vão ser inseridos ou removidos da árvore, foi utilizado o *fgets*, desta forma lendo até o fim do arquivo de entrada. Ainda enquanto os dados são recebidos, os elementos da linha lida - contendo a operação e a chave - são separados pelo *sscanf* e consequentemente invocando a função apropriada.

```
...
  while(fgets(line, sizeof(line), stdin) != NULL){
    sscanf(line, "%c %d\n", &op,&n); //Separa da linha lida o operador e o numero
    if (op == 'i') nodo = inserir(nodo, n);
    if (op == 'r') nodo = excluir(nodo, n);
  }
  return nodo;
}
```