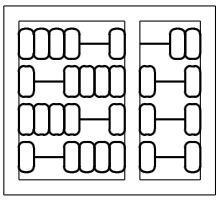

MC202 – Laboratório 06
Árvores binárias de busca com altura limitada
Instituto de Computação — UNICAMP
Prof.: Anderson Rocha
anderson.rocha@ic.unicamp.br

Objetivos

Implementar uma árvore de busca binária não balanceada mas que tenha sua altura máxima limitada.

Descrição do problema

Sem balanceamento, árvores binárias podem se degenerar para uma forma pior de lista ligada, em que cada busca dura tempo $O(n)$, ou seja, para n buscas o tempo de execução seria quadrático. Em algumas situações, não é necessário garantir que todos os elementos inseridos continuem na árvore: se uma busca falhar por um elemento ter sido retirado, o resultado será pior, mas não errado. Assim, é possível garantir tempos logarítmicos ou constantes para as operações na árvore mesmo sem incorrer no custo de efetuar balanceamento a cada inserção e remoção.

Um uso possível é na compressão de dados, em que a cada caractere do texto é necessário fazer uma inserção (com a busca ocorrendo durante essa inserção) e, após um certo número de caracteres, uma remoção pois a janela de busca tem que ser limitada devido à memória.

Neste laboratório, implementaremos uma forma bem simples deste tipo de árvore, sem propriamente uma função de remoção.

Especificação

1. A primeira entrada para seu programa será um número inteiro positivo que indicará a profundidade máxima para a árvore, seguido de um `\n`. A profundidade é contada a partir da raiz da árvore, que tem profundidade zero. Cada nó tem uma profundidade 1 a mais do que seu pai.

2. Após o caractere `I`, virá um espaço e um número inteiro positivo para ser inserido na árvore. Você não deve inseri-lo caso, ao fazer isso, a árvore ultrapasse sua altura máxima. Caso o número já exista na árvore, você deve apenas atualizar a *timestamp* dele para que pareça que foi inserido agora. Nada deve ser impresso durante uma inserção.

3. A cada comando de inserir recebido, o tempo corrido (usado para gerar a timestamp) deve ser incrementado.

4. Após 4 inserções (ou seja, a partir da quinta), deve-se passar a fazer o seguinte procedimento **adicional** durante a inserção: se, dentre os nós visitados no caminho da inserção, o último nó (ponto de inserção do novo número) for o mais antigo, você deve inserir seu novo nó substituindo-o. Ou seja, sobrescrever os campos do número e do *timestamp* do nó antigo. Você deve fazer essa substituição mesmo que a árvore tenha altura baixa o suficiente para inserir seu nó sem remover esse nó. Caso nada disso se aplique, faça a inserção simples como anteriormente.

5. Ao ler o caractere `A`, seu programa deve imprimir qual o número mais antigo ainda na árvore, seguido de um `\n`. Caso a árvore esteja vazia, imprima -1.

6. Após caractere `B` virá um espaço e um número inteiro para buscar na árvore. Caso o número esteja na árvore, você deverá imprimi-lo. Caso contrário, imprima -1. Sempre imprima um `\n` ao final.

7. Ao ler o caractere `F`, seu programa deve imprimir os números presentes na árvore de busca em pré-ordem, em-ordem e, finalmente, em pos-ordem (esse último pode ser feito enquanto você desaloca os nós da árvore da memória). Após cada número (inteiro) dê um espaço. Antes de imprimir a árvore em cada ordem, deixe uma linha em branco (dois `\n`). A árvore nunca estará vazia ao chegar neste ponto do programa. Imprima um `\n` ao final.

Exemplo 1

Entrada:

```
2
I 2
I 1
I 3
I 4
I 5
B 5
A
B 2
B 4
F
```

Saída:

```
-1
2
2
4
```

```
2 1 3 4

1 2 3 4

1 4 3 2
```

Exemplo 2

Entrada:

```
2
B 0
I 12
I 14
I 6
I 2
I 9
I 12
I 10
B 10
I 6
I 10
B 10
A
B 9
F
```

Saída:

```
-1
-1
10
14
-1

12 6 2 10 14

2 6 10 12 14

2 10 6 14 12
```

Observações

1. Os laboratórios devem ser compiláveis usando o GCC 4, com a seguinte linha de execução: `gcc -lm -std=c99 -Wall -pedantic -Werror -o main arquivo.c`

2. É obrigatório liberar toda a memória alocada. Os programas que não liberarem a memória alocada serão considerados errados, independentemente da saída correta nos testes.

3. Códigos ilegíveis serão considerados errados. A legibilidade é obtida com identação correta e coerente, bons nomes de variáveis e funções, bem como boa subdivisão do código em funções auxiliares.