

# **BEE 1191**

## Recuperação de Árvore

---

Prof. Edson Alves – UnB/FGA

# Problema

A pequena Valentina gostava muito de brincar com árvores binárias. Seu jogo favorito era construir árvores binárias aleatórias com letras em maiúsculo nos nodos.

Este é um exemplo de uma de suas criações:



Para salvar suas árvores para uso futuro, ela escreveu duas strings para cada árvore: o percurso prefixo (raíz, sub-árvore esquerda, sub-árvore direita) e o percurso infixo (sub-árvore esquerda, raíz, sub-árvore direita).

Para o desenho acima o percurso prefixo é DBACEGF e o infixo é ABCDEFG.

Agora, anos depois, olhando para as strings, ela notou que reconstruir as árvores era realmente possível, mas só porque ela não havia usado a mesma letra duas vezes na mesma árvore.

Reconstruir a árvore a mão tornou-se chato.

Então agora ela pede que você escreva um programa que faça o trabalho por ela!

## Entrada

A entrada irá conter um ou mais casos de teste. Cada caso de teste consiste em uma linha contendo duas strings representando o percurso prefixo e infixo de uma árvore binária. Ambas as strings consistem de letras maiúsculas, sem repetir. (Então elas não são maiores de 26 caracteres.)

Entrada termina com EOF (fim de arquivo).

## Saída

Para cada caso de teste, imprima uma linha com o percurso posfixo (sub-árvore esquerda, sub-árvore direita, raíz).

# Exemplo de entradas e saídas

## Exemplo de Entrada

DBACEGF ABCDEFG

BCAD CBAD

## Exemplo de Saída

ACBFGED

CDAB

## Solução com complexidade $O(N^2)$

- É necessário implementar o método construtor e a rotina de inserção
- A reconstrução da árvore parte de dois fatos importantes
- O primeiro deles é que a travessia em-ordem estabelece uma ordenação que permite a comparação entre os valores das informações a serem inseridas na árvore
- Esta ordenação deve ser utilizada na inserção, substituindo a ordenação lexicográfica padrão imposta pelo operador  $<$
- A ordem de inserção dos elementos na árvore é determinada pela travessia pré-ordem
- Finalizada a inserção, basta imprimir na saída a árvore usando a travessia pós-ordem

## Solução com complexidade $O(N^2)$

```
1 #include <iostream>
2
3 struct BST {
4     struct Node {
5         char info;
6         Node *left, *right;
7     };
8
9     Node *root;
10
11     BST() : root(nullptr) {}
12
13     void postorder(const Node* node) const
14     {
15         if (node) {
16             postorder(node->left);
17             postorder(node->right);
18             std::cout << node->info;
19         }
20     }
```

## Solução com complexidade $O(N^2)$

```
22 void insert(char info, const int rank[])
23 {
24     Node **node = &root;
25
26     while (*node)
27     {
28         if ((*node)->info == info)
29             return;
30         else if (rank[info - 'A'] < rank[(*node)->info - 'A'])
31             node = &(*node)->left;
32         else
33             node = &(*node)->right;
34     }
35
36     *node = new Node { info, nullptr, nullptr };
37 }
38 };
```



## Solução com complexidade $O(N^2)$

```
40 int main() {
41     std::string preorder, inorder;
42
43     while (std::cin >> preorder >> inorder) {
44         int rank[30], nxt = 1;
45
46         for (const auto& c : inorder)
47             rank[c - 'A'] = nxt++;
48
49         BST tree;
50
51         for (const auto& c : preorder)
52             tree.insert(c, rank);
53
54         tree.postorder(tree.root);
55         std::cout << "\n";
56     }
57
58     return 0;
59 }
```