

# **BEE 1466**

## Percurso em Árvore por Nível

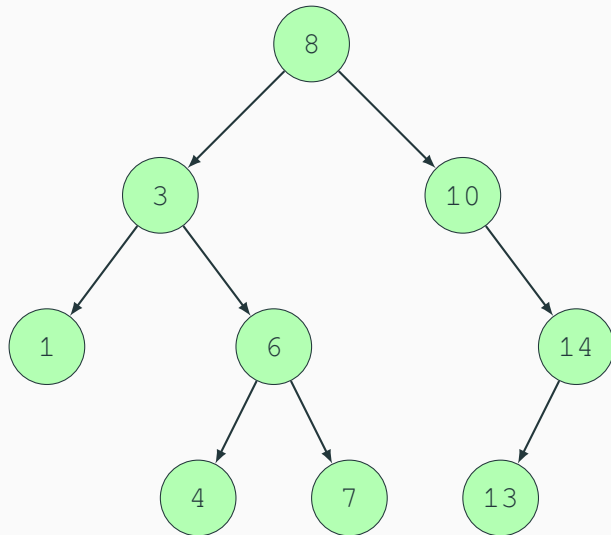
---

Prof. Edson Alves – UnB/FCTE

Em uma árvore binária, o percurso por nível é um percurso denominado **breadth first search (BFS)** ou em português, busca em largura, a qual seria não-recursiva por natureza. Este percurso utiliza uma fila ao invés de pilha para armazenar os próximos 2 nodos que devem ser pesquisados (filho à esquerda e à direita). Esta é a razão pela qual você deve percorrer os nodos na ordem FIFO ao invés da ordem LIFO, obtendo desta forma a recursão.

Portanto nossa tarefa aqui, após algumas operações de inserção sobre uma árvore binária de busca (pesquisa), é imprimir o percurso por nível sobre estes nodos. Por exemplo, uma entrada com a sequência de valores inteiros: 8 3 10 14 6 4 13 7 1 resultará na seguinte árvore:

# Problema



## Entrada

A entrada contém vários casos de teste. A primeira linha da entrada contém um inteiro  $C$  ( $C \leq 1000$ ), indicando o número de casos de teste que virão a seguir. Cada caso de teste é composto por 2 linhas. A primeira linha contém um inteiro  $N$  ( $1 \leq N \leq 500$ ) que indica a quantidade de números que deve compor cada árvore e a segunda linha contém  $N$  inteiros distintos e não negativos, separados por um espaço em branco.

## Saída

Para cada caso de teste de entrada você deverá imprimir a mensagem “Case  $n$ :”, onde  $n$  indica o número do caso de teste seguido por uma linha contendo a listagem por nível dos nodos da árvore, conforme o exemplo abaixo.

Obs: Não deve haver espaço em branco após o último item de cada linha e há uma linha em branco após cada caso de teste, inclusive após o último. A árvore resultante não terá nodos repetidos e também não terá mais do que 500 níveis.

# Exemplo de entradas e saídas

## Exemplo de Entrada

```
2
3
5 2 7
9
8 3 10 14 6 4 13 7 1
```

## Exemplo de Saída

```
Case 1:
5 2 7

Case 2:
8 3 10 1 6 14 4 7 13
```

## Solução com complexidade $O(N^2)$

- A solução deste problema é semelhante à do problema BEE 1195
- Novamente é necessário implementar o método construtor e a rotina de inserção (a qual tem complexidade  $O(S)$  no pior caso, onde  $S$  é o tamanho da árvore)
- Além disso, é preciso implementar a travessia por largura (BFS) (cuja complexidade de cada travessia também é  $O(S)$ )
- Esta implementação é iterativa e requer o uso de uma fila para a organização do processamento dos nós
- Por fim, para cada caso de teste, basta instanciar uma árvore, inserir os elementos indicados e produzir a saída usando a travessia por largura

# Solução com complexidade $O(N^2)$

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 struct BST {
6     struct Node
7     {
8         int info;
9         Node *left, *right;
10    };
11
12    Node *root;
13
14    BST() : root(nullptr) {}
15
16    void BFS() const
17    {
18        vector<int> xs;
19        queue<Node *> q;
20        q.push(root);
```



## Solução com complexidade $O(N^2)$

```
22     while (not q.empty()) {
23         auto node = q.front();
24         q.pop();
25
26         if (node) {
27             xs.push_back(node->info);
28             q.push(node->left);
29             q.push(node->right);
30         }
31     }
32
33     for (size_t i = 0; i < xs.size(); ++i)
34         cout << xs[i] << (i + 1 == xs.size() ? '\n' : ' ');
35     cout << '\n';
36 }
37
38 void insert(int info)
39 {
40     Node **node = &root;
```

## Solução com complexidade $O(N^2)$

```
42     while (*node)
43     {
44         if ((*node)->info == info)
45             return;
46         else if (info < (*node)->info)
47             node = &(*node)->left;
48         else
49             node = &(*node)->right;
50     }
51
52     *node = new Node { info, nullptr, nullptr };
53 }
54 };
55
56 int main()
57 {
58     ios::sync_with_stdio(false);
59
60     int C;
61     cin >> C;
```

## Solução com complexidade $O(N^2)$

```
63     for (int test = 1; test <= C; ++test)
64     {
65         int N;
66         cin >> N;
67
68         BST tree;
69
70         while (N--)
71         {
72             int info;
73             cin >> info;
74             tree.insert(info);
75         }
76
77         cout << "Case " << test << ":\n";
78         tree.BFS();
79     }
80
81     return 0;
82 }
```