

Lab: Trees Representation and Traversal (BFS, DFS)

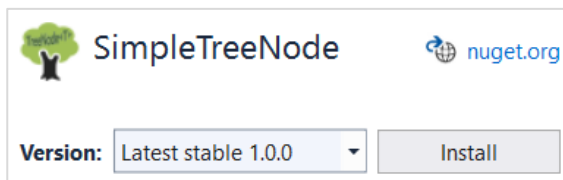
Problems for exercises and homework for the "Data Structures and Algorithms Basics" course from the official "Applied Programmer" curriculum.

You can check your solutions here: <https://judge.softuni.bg/Contests/2932/Trees-BFS-and-DFS-Lab>

1. Build a Tree

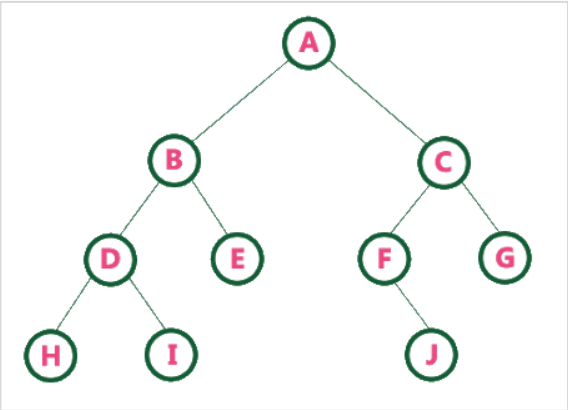
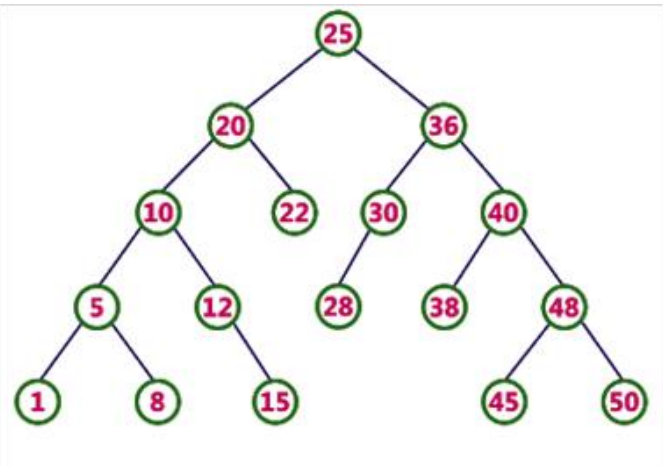
In this task, you need to **build** trees, using the **TreeNode<int>** class. When ready, **print** the tree.

Note: In order to use the class, install the **SimpleTreeNode** NuGet package.



Look at the **TreeNode<int>** class implementation and examples at <https://github.com/nakov/SimpleTreeNode> for more understanding of the data structure.

Examples

Input	Output
	<pre> A B D H I C E F J G </pre>
	<pre> 25 20 36 5 1 8 12 15 22 30 40 28 38 48 45 50 </pre>

Build the trees and **print** them **separately** on the console. **Submit** each of them to the corresponding task in Judge.

Hints

For the first example, it is a good idea to create the **root node** (A), then all its **child nodes** (B,C), then their child nodes, etc. This way you can have more clearance of the **tree structure** and the place of each node in it. You can start like this:

```
TreeNode<char> tree = new TreeNode<char>('A',  
    new TreeNode<char>('B',  
        new TreeNode<char>('D')),  
    new TreeNode<char>('C'));
```

2. BFS Traverse Folders

As you already know, **Breadth-first search (BFS)** is an algorithm for traversing or searching **tree** data structures. It starts at the **tree root** and explores all of the **neighbor nodes** at the present depth prior to moving on to the nodes at the next depth level.

We want to implement the algorithm to **traverse** our **file system**, as it has a **tree structure**.

First, write the **TraverseDirBFS()** method, which should accept **directory path**. Add a **Queue<DirectoryInfo>**, which will hold **visited** folders:

```
public static void TraverseDirBFS(string directoryPath)  
{  
    Queue<DirectoryInfo> visitedDirsQueue = new Queue<DirectoryInfo>();
```

Note that the **DirectoryInfo** class allows us to **access directories** and **subdirectories** of our **file system**. Then, get the first **DirectoryInfo** element from the **queue** like this:

```
visitedDirsQueue.Enqueue(new DirectoryInfo(directoryPath));
```

Next, create a loop to **traverse** through folders until the queue is **empty**.

```
while (visitedDirsQueue.Count > 0)  
{  
}
```

In the loop, get the **current folder**, remove it from the queue and print its name.

```
while (visitedDirsQueue.Count > 0)  
{  
    DirectoryInfo currentDir = visitedDirsQueue.Dequeue();  
    Console.WriteLine(currentDir.FullName);  
}
```

Then, use the **GetDirectories()** method to get **sub-folders** of the current folder. Add each of them to the queue:

```
while (visitedDirsQueue.Count > 0)
{
    DirectoryInfo currentDir = visitedDirsQueue.Dequeue();
    Console.WriteLine(currentDir.FullName);

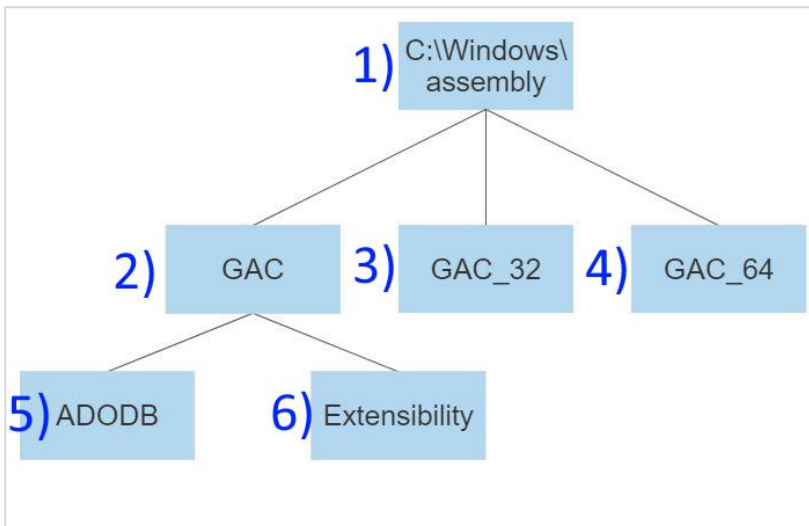
    DirectoryInfo[] children = currentDir.GetDirectories();

    foreach (DirectoryInfo child in children)
    {
        visitedDirsQueue.Enqueue(child);
    }
}
```

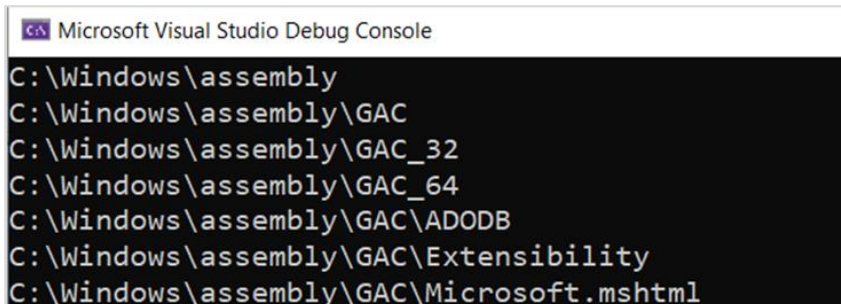
Finally, as our **TraverseDirBFS()** method is ready, invoke it from **Main()**. Manually, point out the directory **path** to the method. In our example, we will traverse "**C:\Windows\assembly**":

```
static void Main()
{
    TraverseDirBFS(@"C:\Windows\assembly");
}
```

This is an example of part of the file system. This is the **order**, in which folders are traversed, using the **BFS algorithm** we wrote.



In this case, with the file structure from the picture, the result should be like this:



```
Microsoft Visual Studio Debug Console
C:\Windows\assembly
C:\Windows\assembly\GAC
C:\Windows\assembly\GAC_32
C:\Windows\assembly\GAC_64
C:\Windows\assembly\GAC\ADODB
C:\Windows\assembly\GAC\Extensibility
C:\Windows\assembly\GAC\Microsoft.mshtml
```

Test the program with your **file system**, as the result might differ. It is a good idea to **debug** the code to see how the **BFS algorithms** traverses the folders.

Submit your solution to the Judge system. Note that you should first **comment** the **TraverseDirBFS()** invocation in the **Main()**:

```
static void Main()
{
    //TraverseDirBFS(@"C:\Windows\assembly");
}
```

Add the **.cs** and **.csproj** files to a **.zip** archive and submit it to Judge.

3. DFS Traverse Folders

Depth-first search (DFS) is an algorithm for traversing or searching tree data structures. The algorithm starts at the **root node** and explores **as far as possible** along each branch before **backtracking**.

Let's use it to traverse our folders, as we did using the BFS algorithm.

First, create **TraverseDirDFS()** method, which accepts **directory path**. Then, create an overload of the **TraverseDirDFS()** method to accept **DirectoryInfo** and **string** (we will use it for keeping spaces, which will help us create and print the output result with a tree-like structure):

```
private static void TraverseDirDFS(DirectoryInfo dir, string spaces)
{
}

0 references
public static void TraverseDirDFS(string directoryPath)
{
}
}
```

In the **TraverseDirDFS(string directoryPath)** method, invoke the other **TraverseDirDFS** method like this:

```
public static void TraverseDirDFS(string directoryPath)
{
    TraverseDirDFS(new DirectoryInfo(directoryPath), string.Empty);
}
```

Now, let's write the other method, as it is the one, which actually holds the **DFS algorithm implementation**. First, get the **sub-directories** of the current directory and use a **loop** to print each of the sub-folders. Note that you need to use **recursion** to go back to previous folders when you reach a leaf of the file system tree.

```
private static void TraverseDirDFS(DirectoryInfo dir, string spaces)
{
    DirectoryInfo[] children = dir.GetDirectories();

    foreach (DirectoryInfo child in children)
    {
        TraverseDirDFS(child, spaces + " ");
    }
}
```

Also, it is important that we **print** the directory name at each invoke of the method before it is recursively invoked again:

```
private static void TraverseDirDFS(DirectoryInfo dir, string spaces)
{
    Console.WriteLine(spaces + dir.FullName);

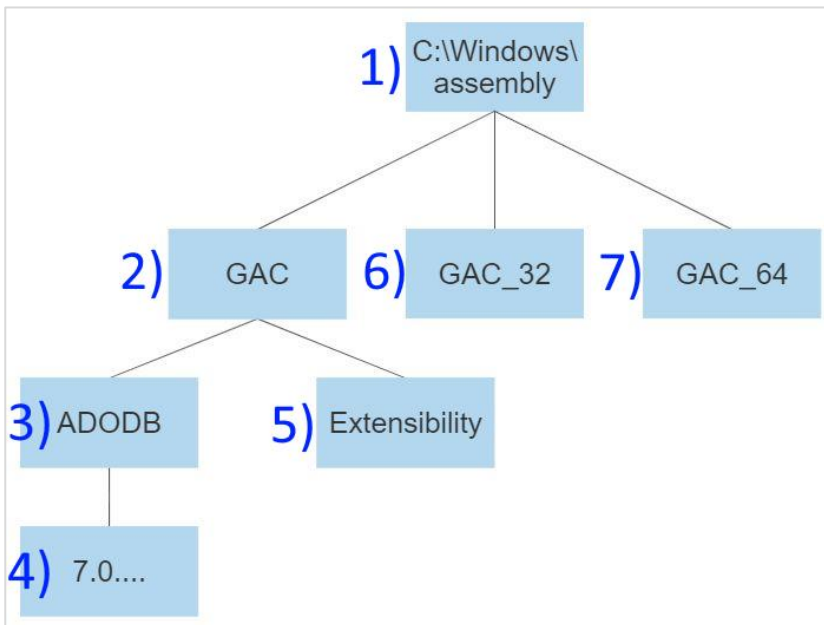
    DirectoryInfo[] children = dir.GetDirectories();

    foreach (DirectoryInfo child in children)
    {
        TraverseDirDFS(child, spaces + "  ");
    }
}
```

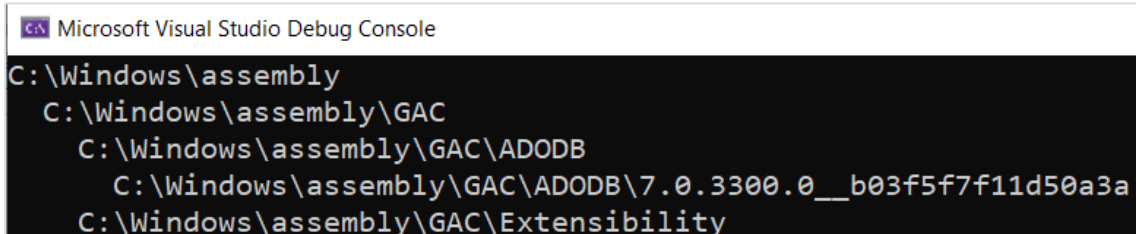
Finally, we should invoke the **TraverseDirDFS(string directoryPath)** method from the **Main()** with the directory path we want to traverse in. Let's use **"C:\Windows\assembly"** again:

```
static void Main()
{
    TraverseDirDFS(@"C:\Windows\assembly");
}
```

In this case, our **file system tree** should be traversed like this:



With the above structure, the result looks like this:



```

Microsoft Visual Studio Debug Console
C:\Windows\assembly
C:\Windows\assembly\GAC
C:\Windows\assembly\GAC\ADODB
C:\Windows\assembly\GAC\ADODB\7.0.3300.0__b03f5f7f11d50a3a
C:\Windows\assembly\GAC\Extensibility
  
```

Test the program with your **file system**, as the result might differ. It is a good idea to **debug** the code to see how the **DFS algorithms** traverses the folders. Note that **spaces** helped us create a tree-like structure of output.

Submit your solution to Judge. Do not forget to comment the **TraverseDirDFS()** method in the **Main()**.

4. DFS Traverse Folders and Files

Now we will use the code from the **previous task** and expand it to **print files** in our folders, as well. The only thing you should do is to use the **FileInfo** class, as well as the **GetFiles()** method to get **files**. Do it like this:

```
private static void TraverseDirDFS(DirectoryInfo dir, string spaces)
{
    Console.WriteLine(spaces + dir.FullName);

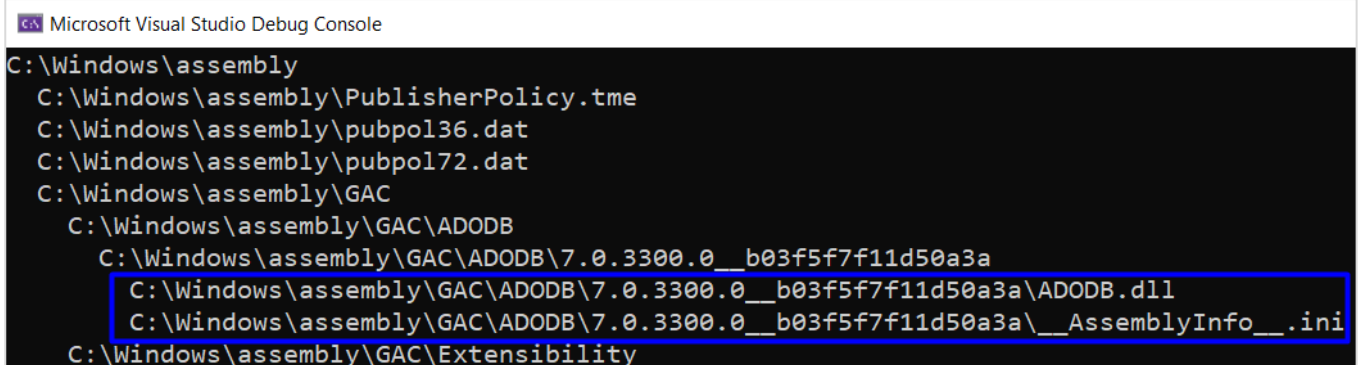
    FileInfo[] files = dir.GetFiles();

    foreach (var file in files)
    {
        Console.WriteLine(spaces + "  " + file.FullName);
    }

    DirectoryInfo[] children = dir.GetDirectories();

    foreach (DirectoryInfo child in children)
    {
        TraverseDirDFS(child, spaces + "  ");
    }
}
```

The result should be something like this:



```
Microsoft Visual Studio Debug Console
C:\Windows\assembly
C:\Windows\assembly\PublisherPolicy.tme
C:\Windows\assembly\pubpol36.dat
C:\Windows\assembly\pubpol72.dat
C:\Windows\assembly\GAC
C:\Windows\assembly\GAC\ADODB
C:\Windows\assembly\GAC\ADODB\7.0.3300.0__b03f5f7f11d50a3a
C:\Windows\assembly\GAC\ADODB\7.0.3300.0__b03f5f7f11d50a3a\ADODB.dll
C:\Windows\assembly\GAC\ADODB\7.0.3300.0__b03f5f7f11d50a3a\__AssemblyInfo__.ini
C:\Windows\assembly\GAC\Extensibility
```

Test your program. You should see your **files** printed on the console.

Submit your solution to Judge. Do not forget to comment the **TraverseDirDFS()** method in the **Main()**.