



Diploma in Infocomm Security Management

EM303

DATA STRUCTURES AND ALGORITHMS

Authors:

Swaathi Lakshmanan (2227171), DISM/FT/1B/05

Class Lecturer:

Mr Tan Hu-Shein

Due Date: 6 August 2023, 11.59 pm

Content Page

Content Page	2
Introduction	2
Checklist	2
Data Flow and Classes	7
Node.cs	8
Tree.cs	8
BuildTree.cs	9
getNode.cs	10
FileIO.cs	11
Depth First Traversal through nodes	11
Enhancements	12
Change Role/ Reporting Officer: Employee with 2 Project Leader Roles.	12
Change Role/ Reporting Officer: Which node represents which role.	12
Add Employee: Adding Employee to a team with a project.	13
Ensuring added nodes do not have the same name as another node.	13
Reflection	13
References	14

Introduction

I prepared this report to guide the readers through the different aspects of my CA2 Data Structures and Algorithms assignment, which is a Project Human Resource Management Application. I have called this application, the Project Manager program. This report includes the checklist for requirements, a walkthrough of how the data flows in the Project Manager program, enhancements I have made to the existing specifications, my reflection and references.

Checklist

Manage Role Form	
Functionalities/ Requirements/ Features	Done/ Half-done/ Not Done
Save and load the binary file	Done
Insert role	Done
Update role	Done
Delete role	Done
Validation	
Minimum validations are required during the insert/update/delete role. The validation logic includes:	
A leaf node is enforced below a project leader-type role node.	Done
A role should not be deleted if the node is used by any employee data.	Done

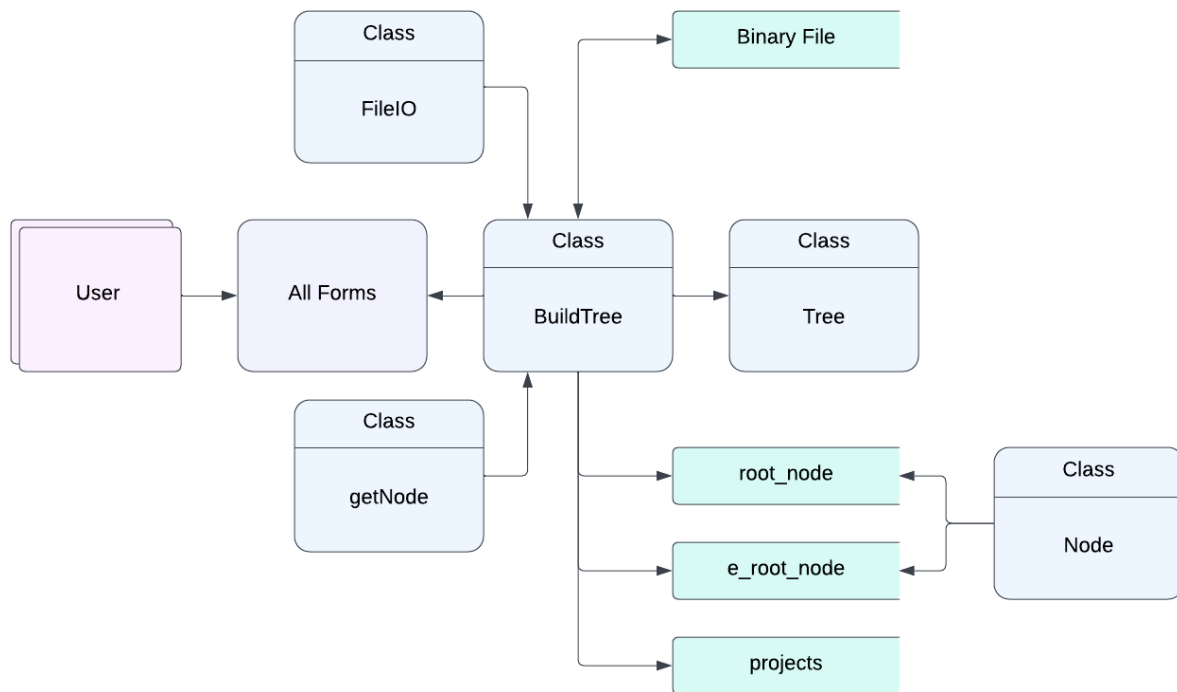
A project leader type role should not be set to a non-project type leader role when the role is already being used by employee data to create a team structure.	Done
Manage employee form	
Save and load the binary file	Done
Insert employee	Done
Update employee	Done
Delete employee	Done
Basic same level swapping: Swap employee between one employee, and another employee.	Done
Insert dummy employee. Swap dummy employees with another employee.	Done
Create/Remove a secondary role for an employee and ensure that the secondary role does not break the overall employee tree structure's data integrity.	Done
<p>Validation</p> <p>Minimum validations are required during insert/update/delete employees. The validation logic includes:</p>	

Cannot delete an employee based on the rules discussed in the video.	Done
The child node employee's salary must not be above the parent node employee's salary.	Done
A project leader type role should not be set to a non-project type leader role when the role is already being used by employee data to create a team structure.	Done
Check the total cost when the user attempts to make changes to the salary of an employee who is in charge of a project. This validation is more difficult when you attempt to make changes to the "Cluster Head", "Finance Department Head" role employee who might oversee more than one project.	Done
<p>Advance swap support: This allows an employee to be swapped with another employee regardless of the level of the employee inside the tree structure. This leads to deeper thoughts on validation processes.</p> <p>Additional validation (?): If the employee is already associated with a project, and the salary is not equal to the salary of the employee being swapped, the project revenue may or may not be able to accommodate the new salary. Thus, this has to be prevented. Also, when swapping employees, their roles must remain the same to ensure the integrity of the tree structure, and from a logical perspective, ensure that the employee is working within their field of expertise.</p>	<p>Done</p> <p>Note: After input from Mr Tan, I have added extra validation to the swap. The swap can only occur:</p> <ul style="list-style-type: none"> - If the roles of the employees are the same - If the salary of the employees are the same.

Manage project form	
Save and load the binary file	Done
Proper management of form's view mode, create mode and update mode. Note that, only the update mode enables the delete project data features.	Done
Insert a new project and re-assign the new project to a valid team.	Done
Update an existing project and re-assign the project to another team.	Done
Delete an existing project and un-assign the project from a team.	Done
Validation Minimum validations are required during the insert/update/delete project. The validation logic includes:	
Ensure that the project revenue value is above or equal to the total cost of the team tree structure's total team cost.	Done
Ensure that the project cannot be assigned to a team which is already in charge of a project.	Done
Additional Validation Features in the application	

Roles/ Employees cannot be added to leaf nodes.	Done
Ensure only complete teams can be selected to be incharge of a project.	Done
Ensuring a single employee node can only hold a maximum of 2 roles.	Done
When editing, ensuring project revenue cannot be changed.	Done
Ensuring that a new employee can be created under a project leader only if the revenue of a project can accommodate the new employee's salary.	Done
Role or employee nodes with the same names cannot be created unless they are from the change role/ reporting officer form.	Done

Data Flow and Classes



Classes

1. Node.cs
2. Tree.cs
3. BuildTree.cs
4. getNode.cs
5. FileIO.cs

Forms

1. Form1 (Manage Role Form)
2. Form2 (Manage Employee Form)
3. Form3 (Manage Project Form)
4. Role Form (Add Role and Edit Role)
5. Employee Form (Add Employee, Edit Employee, Change Role/ Reporting Officer)
6. Swap Form

Node.cs

This class contains the information for a node. It is the basis of both an employee and a role node. Since the role requires lesser information, I have simply left the information the role won't require null. It contains three methods:

1. `genUuid`: The purpose of this function is to generate the unique identifiers for every node. It just contains basic code to generate a hash. It has a parameter to input the string that it will hash.

2. `create_role`: This function takes the parameters and assigns it to the object's properties to create a role node. The name of the role and whether it is a project lead will be concatenated and fed into `genUuid` to generate a uuid.
3. `create_employee`: This function also takes parameters and assigns it to the object's properties to create an employee node. The role, name of the node, its salary and the name of its reporting officer is concatenated and fed into `genUuid` to generate a uuid.

`genUuid` helps me ensure that every node added is a unique one.

Tree.cs

This class contains the code to create a tree structure under a root node. This code exclusively interacts with the root nodes in `BuildTree.cs`, which I will explain later.

This class contains various functions for Depth First Searching with various parameters. These functions include:

1. `recursive & DFS`: Search by node and return parent
2. `namerecursive & namedDFS`: Search by node, return node with same name but different uuid if it exists.
3. `get_all_projects`: Get all the projects under a node.

This class also contains few more recursive functions:

1. `delete_all_employees`: Deletes every employee node associated with every role node.
2. `check_for_full_team`: Checks whether the team under the node is a complete team.

Lastly, there are the basic classes which I use as the foundation for the more complex functions in `BuildTree.cs`:

1. `addNode`: Checks if the node does not already exist, after which it adds the node to its parent node. If the node is unique but has a similar node, the code figures it is the result of a 'Change role/ Reporting officer' action and changes the nodes accordingly to reflect that the nodes are related. Also if the node is under a project leader, it is again modified to indicate it a leaf node.
2. `deleteNode`: Uses the recursive function to find the parent node and from there, deletes the node from the tree structure.
3. `editNode`: Takes the node and a list with the new values as parameters. Uses these parameters to modify the node.

BuildTree.cs

This class was initially created to ensure that the data across all the forms will be synced. It contains three important objects: `root_node`- the root role node, `e_root_node`- the root employee node, and `projects`, a list of employee nodes that are project leaders and have projects assigned to them. Thus, this class contains the functions to connect the user's actions to the three objects. There are many nested functions, but I will only be touching on the enclosing functions.

1. `fakeTree`: Creates a fake tree structure for a root role node.

2. `role_createTree` & `role_addRemainingTree`: This function takes a `TreeView` as a parameter. It starts by clearing the `TreeView` control and then recursively running through the root nodes to create a `TreeNode`s and populate the `TreeView`.
3. `for_tree_view`: This function could be called the 'Role Form Manager'. It has all the functionalities that are required for the user to make changes to the root role node from `form1`.
4. `start_employee_form`: This function was created to start the employee form when there are no existing employees.
5. `for_employee_tree_view`: Similar to `for_tree_view`, this function has all the functionalities that are required for the user to make changes to the employee node from `form2`.
6. `savefileio`: This function puts the existing `root_node`, `e_root_node` and projects and puts it into a list. After which it saves it in binary format in the designated binary file.
7. `loadfileio`: This function basically takes the data from the designated binary file and loads into the form. It will retrieve a list containing the `root-node`, `e_root_node` and projects and then reassign it to the three objects in `BuildTree.cs`.
8. `for_projects`: This, like `for_tree_view` and `for_employee_tree_view`, contains the functions for the user to create and assign projects to the employee nodes.
9. `update_listview`: This function takes a `ListView` as the parameter. It will clear the `ListView` and populate the `ListView` with the information of the nodes in the projects list.
10. `reset_pressed`: This function 'resets' the form. It reassigns the three objects with empty root nodes (the role node contains the fake tree structure) and an empty list.

I would also like to take a brief tangent to explain how I ensure the changed tree structures are reflected in the forms.

I only invoke a single instance of the `BuildTree` class and this ensures that the data shared between all three forms always remain the same.

```
public partial class Form1 : Form
{
    public static BuildTree btree = new BuildTree();
}

public partial class Form3 : Form
{
    BuildTree btree = Form1.btree;
}
```

I simply refer to the instance I created in `Form1` in my other forms.

I also created an event in my `BuildTree.cs`:

```
public RoleForm roleForm;
public event EventHandler updateTree;
```

Every time the root nodes or the projects are modified, I simply invoke the event:

```
updateTree.Invoke(this, new EventArgs());
return true;
```

I have also added event handlers in each form to ensure that the modifications are reflected in the forms. The event handlers usually contain functions to clear and repopulate form controls like `role_createTree`.

```
1 reference
private void Btree_updateTree(object? sender, EventArgs e)
{
    btree.role_createTree(treeView1, btree.e_root_node);
    btree.update_listview(listView1);
    treeView1.ExpandAll();
}
```

getNode.cs

This class was simply created because I wanted to link the node I am referring to in a ComboBox item to the item itself. That way, by retrieving the selected item, I can also retrieve the node associated with it.

This Class contains two public variables: `text` and `node`. `Text` refers to the string that will be displayed in the ComboBox item. The node is the node the ComboBox item will be associated with. The class also contains a single method to override the `ToString` method. Instead of returning the object itself, the overridden `ToString` will return `text` instead.

```
37 references
internal class getNode
{
    10 references
    public string text { get; set; }
    14 references
    public Node node { get; set; }

    0 references
    public override string ToString()
    {
        return text;
    }
}
```

FileIO.cs

This class contains the functions to serialise and deserialize the lists that I refer to in `loadfileio` and `savefileio`. It uses `FileStream` to open the designated file and then uses `BinaryFormatter` to serialise/ deserialize the object.

Depth First Traversal through nodes

This is a method that I used throughout most of my code and it is not unreasonable to suggest that it is the method my entire program relies on. While I have used this method to return

objects that fulfil various requirements or return many objects as a list instead of one, the essence of the method is the same.

The following is one of the depth first traversal methods:

```
7 references
public Node recursive(Node x)
{
    return DFS(root_node, x, root_node);
}

2 references
private Node DFS(Node node, Node x, Node parent)
{
    if (node.uuid == x.uuid)
    {
        return parent;
    }

    foreach (var childNode in node.childnodes)
    {
        Node result = DFS(childNode, x, node);
        if (result != null)
            return result;
    }

    return null;
}
```

As you can see, the recursive function takes node x as a parameter and will eventually return the parent of node x. The recursive function calls on the DFS function which takes three parameters, a node (the node being visited), x and parent (the parent of the node being visited). The function starts off by checking if the node's uuid is equivalent to x's. And if it is, the traversal ends there and the parent node is returned. If not, DFS is called upon each of the node's childnodes, effectively traversing the whole tree. If none of the node's uuid is equivalent to x's, a null value is returned.

The recursive method serves as a wrapper for the DFS method. It is public so that BuildTree.cs will not have to access the DFS method itself. It also helps to begin the traversal by calling the root node and not reassigning the value of the root node.

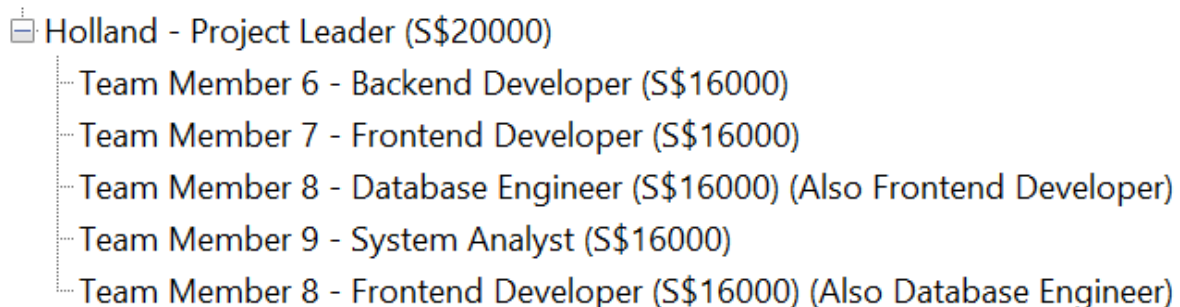
Enhancements

Change Role/ Reporting Officer: Employee with 2 Project Leader Roles.

Something that was not mentioned in the specifications was how to handle the scenario when there is an Employee holding 2 project leader roles. There was no mention of whether the employee can manage 2 projects or will just manage a single project. To accommodate this, I assumed that an Employee with 2 project leader roles can manage 2 projects. This led me to creating two different employee nodes with similar information. What will distinguish both the nodes will be their uuid. However you might also notice that the uuid textbox in the employee form will display the same uuid when you click on both employee's nodes. This is a result of the generation of a second uuid which I refer to as `display_id` in my code. The purpose of `display_id` is to ensure track whether the different nodes refer to the same employee. Since the project information is stored in the node that is incharge of the project, this will allow that single employee to manage 2 different projects with 2 different teams at the same time.

Change Role/ Reporting Officer: Which node represents which role.

While this was mentioned in the specifications video, the demo code was operating such that there was a single node with two roles and a single display name. However, with the changes I made to my code in the previous section, I was able to distinguish which node had which role and whether they referred to the same employee. Thus, I felt like I was presented with the opportunity to lay out which node referred to which role in the form itself. I decided to indicate the roles the employee has other than the one that particular node represents in brackets at the end of their display name. This will prevent any confusion if the user forgets which node he created first or which node refers to which role.



```
graph TD; A[Holland - Project Leader (S$20000)] --- B[Team Member 6 - Backend Developer (S$16000)]; A --- C[Team Member 7 - Frontend Developer (S$16000)]; A --- D[Team Member 8 - Database Engineer (S$16000) (Also Frontend Developer)]; A --- E[Team Member 9 - System Analyst (S$16000)]; A --- F[Team Member 8 - Frontend Developer (S$16000) (Also Database Engineer)];
```

Holland - Project Leader (S\$20000)

- Team Member 6 - Backend Developer (S\$16000)
- Team Member 7 - Frontend Developer (S\$16000)
- Team Member 8 - Database Engineer (S\$16000) (Also Frontend Developer)
- Team Member 9 - System Analyst (S\$16000)
- Team Member 8 - Frontend Developer (S\$16000) (Also Database Engineer)

Add Employee: Adding Employee to a team with a project.

I do not think the scenario where a team member is added to a team with a project was mentioned in the specifications. However, I used the same validation for this that I used for

checking whether the salary of an employee who is already associated with a project is valid. In addition to the other specified validation, I check whether the parent of the node being added is a project lead, and if they are, I traverse through the nodes and add the salary of the nodes that are associated with the project of the parent node. From there, I check whether the new node's salary in addition to the other nodes' salaries are less than or equal to the current revenue. If it is, I will allow the node to be created.

I have also considered the scenario where the nodes added are nodes that are not leaf nodes. I have deduced that only leaf nodes will be affected by this issue, other nodes which are newly created will naturally not have any projects associated with them.

Ensuring added nodes do not have the same name as another node.

I do not believe this was mentioned in the video, though the validation for this is pretty intricate due to the structure of my code. I had to fulfil a few conditions to ensure this function worked as intended:

1. If the node is a role node, it has to have a unique role name.
2. If the node is an employee node and is not the result of the chrole, it has to have a unique name.
3. If the node is a result of chrole, the new node must have the same name of the existing employee node.

I believe the complexity of the code increased because this validation was more of an afterthought. I was initially using the two same functions, `namerecursive` and `addNode`, to validate the node and add the node to the tree structure. In the beginning, I only made sure each node had a unique uuid. But after giving it a lot of thought, I realised that there can be scenarios where this single validation could cause confusion. For instance if the user add 2 nodes with the same names, in the same project leader roles (with different reporting officers) and decided to change the reporting officer of a third node, the user may see two of the same ComboBox items, though behind the scenes these items may refer to two completely different nodes. Therefore, in an attempt to ensure confusions such as this do not occur, I added these conditions.

Reflection

My experience with Data Structures and Algorithms has been incredibly rewarding. Initially, I was intimidated by the research-based coding assignment, as it was my first time working with C# and the Visual Studio Community user interface. However, as I began to tackle the assignment, I discovered that with determination and a willingness to learn from research, even the most challenging tasks can be accomplished. The skills I gained have proven invaluable, and I find myself applying them in my daily coding. I no longer limit myself to the resources taught in class; instead, I seek out more efficient methods when coding in other languages such

as JavaScript or Python. Additionally, I have developed a more critical approach to coding, striving to write more concise and reusable code.

This assignment went more smoothly than the assignment for CA1, as I was able to anticipate and address potential issues before they arose. For example, in the previous assignment, I did not consider how to serialise my data until near the end of the project, at which point I realised that my form controls were intertwined with my classes. To resolve this issue, I had to manually serialise and deserialize each variable or element to prevent the form controls from interfering with the binary functions in my code. In this assignment, however, I created a dedicated class for serialisation (Node.cs), which greatly simplified the binary functions and made my code more efficient.

Overall, despite my initial concerns about my lack of experience and the complexity of the assignments, I have found this module to be incredibly rewarding. The support and guidance provided and the positive learning environment in class have been instrumental in helping me get through this module. (Thank you Mr Tan for a wonderful learning experience.)

References

Stack Overflow. (n.d.). *How to create a context menu in C#*. [online]

Available at:

<https://stackoverflow.com/questions/10821835/how-to-create-a-context-menu-in-c-sharp>

[Accessed 6 Aug. 2023].

Stack Overflow. (n.d.). *Adding a right click menu to an item*. [online]

Available at: <https://stackoverflow.com/questions/9823883/adding-a-right-click-menu-to-an-item>

[Accessed 6 Aug. 2023].

Techie Delight. (2016). *Depth First Search (DFS) – Iterative and Recursive Implementation*. [online]

Available at: <https://www.techiedelight.com/depth-first-search/>.

debug.to. (2022). *Tree in Data Structure using C#*. [online]

Available at:

<https://debug.to/3253/tree-in-data-structure-using-c#:~:text=A%20tree%20data%20structure%20is> [Accessed 6 Aug. 2023].

Stack Overflow. (n.d.). *WinForms TreeView - how to manually 'highlight' node (like it was clicked)*. [online]

Available at:

<https://stackoverflow.com/questions/1838807/winforms-treeview-how-to-manually-highlight-node-like-it-was-clicked> [Accessed 6 Aug. 2023].

Stack Overflow. (n.d.). *How to manually invoke an event?* [online]

Available at: <https://stackoverflow.com/questions/8734700/how-to-manually-invoke-an-event> [Accessed 6 Aug. 2023].