Family Tree Manager

Proposal

**Marvin Newlin**

**CSC316: Data Structures for Computer Scientists**

**mwnewlin@ncsu.edu**

**North Carolina State University**

**Department of Computer Science**

**22 February 2016**

Executive Summary

This project is meant to build a family tree from user provided files and then given two names from the files, the program will state their relationship. In this document we provide the goals & benefits of this program, the designs for the main algorithms required for the project, an analysis of those algorithms, a software design and rationale, a black box test plan, and a task plan for the project.

Table of Contents

Add a table of contents here. Word processing software usually has a feature to make generating tables of contents simple and easy. Check the documentation for your specific piece of software.

For example, in Word, you should be able to right-click the table of contents below and select “Update field” -- the contents are pulled from the headers used through the rest of the document.

[Problem Statement 1](#_Toc443905815)

[Project goals & Benefits 1](#_Toc443905816)

[Algorithm Design 2](#_Toc443905817)

[Software Design 6](#_Toc443905818)

[Black Box Test Plan 7](#_Toc443905819)

[Task Plan 9](#_Toc443905820)

# Problem Statement

The new genealogy startup “AncestryTree” needs software that manages family trees. Because family trees can be extensive and take large amounts of paper to print, the company needs the trees stored in electronic format. The format for this project must allow the user to enter two names and see the relationship between the two individuals.

# Project goals & Benefits

The main goal of this program is make possibly extensive family trees easily accessible by storing them electronically. Additionally, the goal of this program is to provide the ability for a person to see how two people in a family tree are related in a quick and time-efficient manner. The benefit of this format is the time saved by the person using this software. In using this, they will save much more time than if they were attempting to do this on paper.

# Algorithm Design

Note: Dictionary will be maintained as a contiguous memory representation to preserve the order and allow constant access time to an element and to preserve the order of the pre and post traversals. The algorithms for lookup, and insert will follow standard form for looking up and inserting into an array type format. For the tree, we will use a set of linked nodes. Each node will have a parent, a list of children, its data, and a mark field.

//Assume height of tree is h = log(n), assume n nodes in tree for all algorithms, k children for all nodes except leaf nodes

**Algorithm Name:** buildTree (int size, int prestart, int poststart)

Input size is the number of nodes in the subtree to be built

Input prestart is the place in pretrav where the preorder traversal of this subtree begins

Input poststart is the place in posttrav where the postorder traversal of this subtree begins

Output a reference to the root node of subtree

If size = 1 then (1)

Return new node(pretrav(prestart)) (1)

else

originalSize 🡨 size (1)

prestart++ (1)

poststart = posttrav.indexOf(prestart) (1)

newSize 🡨 poststart – prestart (1)

while poststart < originalSize - 1 do (n)

children.add(buildTree(newSize, prestart , poststart)) (k(h-1))

poststart ++ (nk)

prestart += newSize (nk)

newSize 🡨 poststart – prestart (nk)

return children (1)

Runtime: 1 + 1 + 1 + 1 + 1 + h + n + kh – k + 3nk = (3k + 1)n + (k+1)h – k + 5 = (3k + 1)n + (k+1)log(n) – k + 5

TbT(n) = (3k + 1)n + (k+1)log(n) – k + 5

**Algorithm Name:** findRelationship(Node a, Node b)

Input: person A, person b the two people being compared

Output the relationship between the two people.

mark(a) (h)

start 🡨 b (1)

pathLenB 🡨 0 (1)

while !isMarked(start) do (h + 1)

start 🡨 start.parent (h)

commAncestor 🡨 start (1)

pathLenA 🡨 getPathLength(commAncestor, Person a) (h)

pathLenB 🡨 getPathLength(commAncestor, Person b) (h)

comparePathLengths(pathLenA, pathLenB) //Will output relationship based on numbers in project writeup (1)

clearMarks() (h)

Runtime: h + 1 + 1 + h + 1 + h + 1 + h + h + 1 + h = 7h + 5 = 7log(n) + 5

TfR(n) = 7log(n) + 5

**Algorithm Name:** mark(Node a)

Input: A person in the tree

Output: sets the mark field in each ancestor of person a to true

q 🡨 node containing a (1)

while q.hasParent do (h – 1)

q.mark 🡨 true (h)

q 🡨 q.parent (h)

q.mark 🡨 true (1)

Runtime: 1 + (h – 1) + 2h + 1 = 3h + 1 = 3log(n) + 1

Tmark(n) = 3log(n) + 1

**Algorithm Name:** getPathLength(Node commAncestor, node p)

Input: A common ancestor node, and a node containing the person’s name

Ouput: An int with the path length

q 🡨 p (1)

k 🡨 0 (1)

while !q.equals(commonAncestor) do 2(h+1)

q 🡨 q.parent (h)

k++ (h)

return k (1)

Runtime: 1 + 1 + 2(h+1) + 2h + 1 = 4h + 5 = 4log(n) + 5

TPL(n) = 4log(n) + 5

**Algorithm Name:** clearMarks(Node r)

Input: A root node r

Sets the marked field to false in all nodes of the tree rooted at r, assume k children

If r.isMarked then (1)

mark 🡨 false (1)

for each child c of r do (k)

clearMarks(c) (k)(h-1)

Runtime: 1 + 1 + k + (kh – k) = kh + 2 = k\*log(n) + 2

Tcm(n) = k\*log(n) + 2

**Algorithm Name:** levelOrderTraversal(node r) //Adapted from TS PowerPoint slides

Input: A root node r

Output: The level order traversal

Q 🡨 queue of nodes //Empty to start

If r is null then

return

Q.enqueue(r)

While Not Q.isempty() do

p 🡨 Q.dequeue()

visit(p)

while p has more children do

lmc 🡨 leftmost child that has not been considered

Q.enqueue(lmc)

Runtime: O(n) since printing every node in the tree

Total runtime: = TbT(n) = (3k + 1)n + (k+1)log(n) – k + 5

TfR(n) = 7log(n) + 5

Tmark(n) = 3log(n) + 1

TPL(n) = 4log(n) + 5

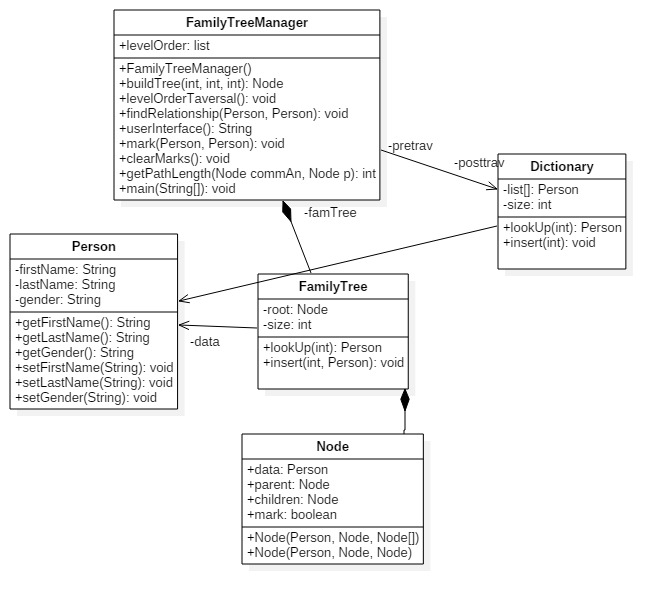
Tcm(n) = k\*log(n) + 2

TLO(n) = n

T(n) = (3k + 2)n + 2(k + 6) log(n) – k + 18

T(n) is O(n)

# Software Design



The design pattern for this program is Model-view-controller (MVC). The model consists of the whole program minus the main and userInterface methods. The view/controller is the userInterface method. It is the only thing that the user sees and interacts with, and the userInterface method will then dictate what methods get called based on user input.

# Black Box Test Plan

Contents of preorder1.txt

John Smith M

Mark Smith M

Jane Smith F

Bessie Doe F

Elizabeth Doe F

Sam Doe F

James Smith M

Contents of postorder1.txt

Jane Smith F

Mark Smith F

Elizabeth Doe F

Sam Doe M

Bessie Doe F

James Smith M

John Smith M

|  |  |  |  |
| --- | --- | --- | --- |
| Test ID | Description | Expected Results | Actual Results |
| testLevelOrder | User starts the familyTree program with the preorder1.txt and postorder1.txt files. | Level Order Traversal prints as: [John Smith, Mark Smith, Bessie Doe, James Smith, Jane Smith, Elizabeth Doe, Sam Doe] |  |
| testParentRelationShip | Precondition: User has started the program with the preorder and postorder files listed above.  User enters: John Smith James Smith | Program Prints:  Jomes Smith is John Smith’s son. |  |
| testGrandparent | Precondition: User has started the program with the preorder and postorder files listed above.  User enters: Jane Smith John Smith | Program Prints:  John Smith is Jane Smith’s grandfather. |  |
| Test cousin | Precondition: User has started the program with the preorder and postorder files listed above.  User enters: Jane Smith Sam Doe | Program Prints:  Same Doe is Jane Smith’s first cousin. |  |
| Test No Input | Precondition: User has started the program with the preorder and postorder files listed above.  User enters: “” | Program prints error message saying no input and redisplays console prompt. |  |

# Task Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task Description | Owner | Planned  Start Date | Planned  End Date | Status |
| Write algorithms | Marvin | 2/15 | 2/20 | Completed |
| Write Executive summary | Marvin | 2/15 | 2/15 | Completed |
| Write Project Goals & Benefits | Marvin | 2/16 | 2/16 | Completed |
| UML Design & Rationale | Marvin | 2/18 | 2/18 | Completed |
| Algorithm Analysis & Rationale | Marvin | 2/19 | 2/20 | Completed |
| Write BBTP | Marvin | 2/22 | 2/22 | Completed |
| Write code for part 2 | Marvin | 2/26 | 3/1 |  |
| Write tests for part 2 | Marvin | 3/1 | 3/3 |  |
| Last minute checks | Marvin | 3/4 | 3/4 |  |

**Team Contact Information**

Marvin Newlin, [mwnewlin@ncsu.edu](mailto:mwnewlin@ncsu.edu), (336) 684-6160