I implemented a family tree using a graph, basically a dumbed down graph. Graphs usually have inNodes, outNodes, inEdges, outEdges. I only needed inEdges and outEdges, which were inRelationships and outRelationships.

In a nutshell, I have a HashMap that holds the people's names which are keys, to the destinations which are an object I made called "Information." "Information" holds the person's relationships and to whom and vice versa.

I did this program last cause it was the scariest and most vague, there were so many ways to do it and implement it, and your question was quite open ended.

To answer, "how does this algorithm scale," I'm guessing you mean the searching. I chose to search "for members that match name." I wasn't really sure what that meant, but I programmed it so that after you finish entering data for the family tree, you can search and print a person's name in the tree and also people that have the same last name. Doing so will print their full names and also their "relationships."

---

searchForPerson() in FamilyTreeTester gets the full name from the user, checks if the name exists in the FamilyTree map, then prints it and its relationship information. The checking if existing is done by just attempting to get the the destination with the full name key, which is a O(1) operation. If it returns null, then I know it doesn't exist.

Code from FamilyTree:

```
public boolean checkPersonIsInMap(String fullName)
	{
		if(personMap.get(fullName.toUpperCase()) == null)
		{
			System.out.println(fullName+" is not in the family tree. You must add them before you can add relationship/search.");
			return false;
		}
		return true;
	}
```

---

printPeopleWithLastName() in FamilyTreeTester

This method gets the last name from the user then finds out if its in my map <String lastnames, Set of fullNames> with lastNameMap.containsKey(lastName). It is an O(1) operation because I use a HashMap, though worst case is O(N). I then use the set of full names and print out each of their relationship stats, which is O(N). In the end, thanks to HashMap, the method's complexity is O(N). Though I do have some extra storage because of caching the last names to set of first names, it's worth it because it's faster than going through all the full names in my personMap, which would be a linear search.