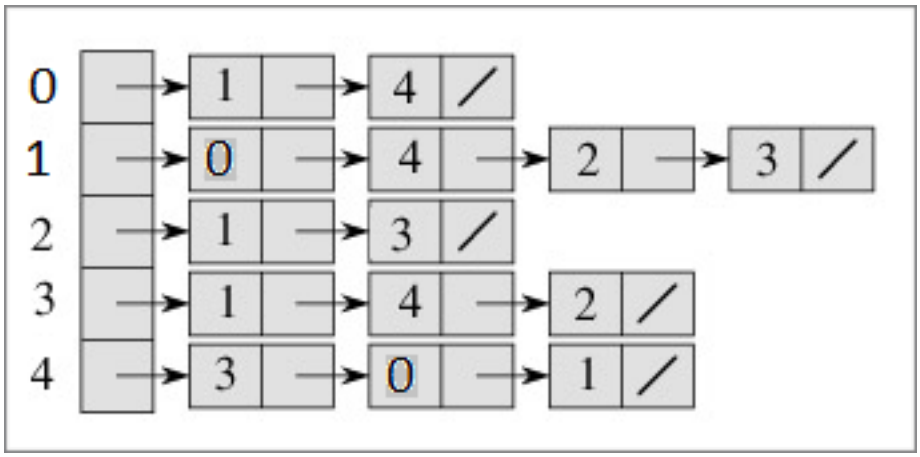
Yahoo Mini Project by Xuezi Zhang

1. Overview

Firstly, the problem can be treated simply as finding the longest route in a directed graph. Each node in the graph is a persons' full names. Then, if person A's last name is person B's first name, there will be an edge pointing from A to B.

2. Data Representation

In this project, the name graph will be represented by an adjacency list. Each node a the head cell on the left and the edges are linked list with the index of the node that the current node points to. In my project, nodes are represented by the class of "NameNode" and edges are represented by the class of "EdgeNode". The whole graph containing all the nodes and edges are represented by the class of "GameGraph".



3. Algorithm

Depth First Search is applied to solve this problem. The function called "void dfs()" is a recursive function that reaches to the next node in the graph. In each level of recursion, the function will store the current longest name chain and the level. The termination of this function will be triggered by reaching the node that with no successors or the node that have been visited before in this round of search. After the termination, if the current name chain and the level is larger than the maximum level and name chain stored in the game, the function will update both and go to the next round of search.

4. Class Design and Implementation

NameNode: Vertex in the graph
Member Variables: firstName(string), lastName(string), firstNode(EdgeNode), lastNode(EdgeNode)

EdgeNode: Edge in the graph
Member Variables: next(EdgeNode), index(int)

GameGraph: Graph of the game
Member Variables: nameNodes(NameNode[]), size(int)
Class Methods: int initVertex(), void initEdges(), int initGraph(), NameNode getNode(int), boolean isValidName(String)

NameChainGame: Game Controller
Member Variables: VISITED(static int), NOTVISITED(static int), res(List<String>), maxLevel(int), game(static NameGameChain), graph(GameGraph)
Class Methods: static NameChainGame getInstance(), void newGame(String[] names), void run(), void dfx(), List<String> getResult()

GameApp: Main Application of running the NameChainGame
Class Methods: static List<String> play(String path)

MainAppTest: Function Test testing all the whole game process
Class Methods: static void main(String[] args)

Due to the limit of time, the UML diagrams, including the class diagram and data flow diagram are omitted in this document. Also, all the detailed descriptions of each member variables and methods are written in the source code.

5. Analysis and Tests

n: the number of nodes, e: the number of edges
Building the graph: O(n ^ 2)
DFS Search: O(n + E)
Print Result: O(K)

Due to the limited time, I only conducted one function test and two unit tests without any advanced framework JUnit. I've been trained with JUnit testing and test-driven development during my internship at IBM.

Function Tests:
I downloaded the names files containing about 500000 human entries. Then I chunked this raw file, including 0, 1, 10, 100, 1000, 10000, 50000 name entries separately in different test case files. Each case is going to be tested to check the functionality of "void play()" in the GameApp class, considered as the main function of the game.

For the input with no names, the function successfully thrower out an exception that I set in my program. For the input from 1 to 10000 names entries, the test successfully output the result into the file in the output directory.

However, when testing with 50000 entries, the program running for hours and by the end of my writing, it still didn't have any output yet. It is probably due to the time complexity of building the graph being O(n ^ 2). Also after that, I found the potential limit of the program is the type of my

name node number is integer, which cannot run under a long name list such as the list containing 100000000 entries.

Unit Tests:

1. ValidNameTest: test if the function of GameGraph.isValidName(String) works

Cases: {}, {“a”}, {“a”, “d”, “d”}, {“a”, “d“}, {“”}, {" “}
Expected Result: False, False, False, True, False, False
Actual Result: False, False, False, True, False, False

2. BuildGraphTest: test if the function of initGraph() works

Cases: “Elton John”, “John Lennon”, “James Elton”, “Lebron James”, “James Faulkner”
Excepted Result:
“Elton John” -> 1 -> \
“John Lennon” -> \
“James Elton” -> 0 - > \
“Lebron James” - > 2 - > 4 -> \
“James Faulkner” - > \

Actual Result:
“Elton John” -> 1 -> \
“John Lennon” -> \
“James Elton” -> 0 - > \
“Lebron James” - > 2 - > 4 -> \
“James Faulkner” - > \