

# Six Degrees of Kevin Bacon

Daniel Kaiser and Eric Cherrstrom

University of Notre Dame, Notre Dame IN 46556, USA

**Abstract.** *Six Degrees of Kevin Bacon* is a well known game to most movie buffs and has inspired a board game version and even a book. This project sets out to create a small database of movies and actors, a set of functions to manipulate the data to find a path between actors and their films, and to create a GUI in which to play the game. We also wanted to take advantage of Design by Contract and various STL containers that we have learned about in our *Data Structures* class.

## Keywords:

breadth-first search, graph, adjacency matrix, STL container

## Introduction:

This popular game has been implemented on a website, [www.cs.virginia.edu/oracle/](http://www.cs.virginia.edu/oracle/), but the user is prompted to enter an actor and then a path to Kevin Bacon is provided. We wanted to prompt the user with an actor and have them try to find the path. We then found that it would be simple to expand the game to connect an actor to any other actor, which we call *The Movie Game*, which dramatically increases gameplay options. The program outputs one or two actors at random from the database. The user must then try to connect the two actors based on their cinematic knowledge. We do not prompt the user for any input, but rather the user must think of the answer then has the option of

getting the answer, getting the next actor in the chain, or getting a list of an actor's films. They can then compare their answer with the answer provided by the program.

### **Main:**

The main decision of this project was the representation of the actors and films the traversal of the chosen representation. We decided to use a graph to represent the actors and films and an adjacency matrix to represent this graph. We created a class Actor to store an actor's name and values for the actor to be used by the search function. The database consists of an array of actors, an adjacency matrix containing films the actors were in, a queue to store the visited nodes of the searched graph, and a queue to store the final path between actors. We used several Standard Template Library containers to store and manipulate the database and path between actors. We used a `<vector<vector<string>>>` to implement the adjacency matrix. We used a `deque<int>` to store the visited nodes of the graph and the final path. We implemented the array of actors as a `<vector<actor>>`. We used a `set<string>` to store an actor's films which was very advantageous because it took care of repeats and put the films in alphabetical order for output. The definition of the Actor and Moviebase classes are provided in appendix a. To traverse the graph and store the path we decided to use a breath-first search. This was perfectly suited to this project because it finds all nodes that are one unit of distance away from the source, then all nodes that are two units away from the source, then all nodes that are three units away from the source, and so on. This was useful because it stops as soon as the destination is reached and it stores the shortest path. We then had to reverse the path because the breadth-first search stores the path from destination to source. We then created functions to output the entire path and to output the path step by step. We also created a function to output all movies that an actor is in according to the database.

### **Results**

We succeeded in creating a game that is both fun and challenging, but with some drawbacks. The main drawbacks are the small size of the database and the inability for

the user to input an answer. The database was hard-coded and even the limited database took several hours to input. Due to the size of the database it is more likely that a movie or actor is not in database, so user input is very impractical. It is for this reason we chose to output the movies in our database so the user will know which movies he or she can use when formulating their answer. We succeeded in using our knowledge of the Standard Template Library for effectively storing data and accessing it efficiently. Our greatest difficulty came when creating the GUI in Visual Studio. This was our first time making a GUI in Visual Studio and we had several problems compiling and running the program. We persevered though, and with some help for the teaching assistants we created a simple, but well designed GUI in Visual Studio, that greatly enhances game play as compared to the console version.

### **Conclusion:**

Through this project we came up with a simple representation of a graph and created functions to manipulate the data in our graph. We took advantage of *Design by Contract* in creating our database to improve correctness and reusability of our code. We also took advantage of several Standard Template Library containers and their respective functions to store our data and to traverse the graph. We implemented a GUI in Visual Studio in which to play the game, and the GUI has several gameplay advantages over a console version of the game. We have a long way to go before this game is finished, but we accomplished all the goals that we set for ourselves, and successfully created a fun and unique game.

### **Future Work:**

We have several ideas for the future of this game and how we can enhance the gameplay. The first obvious improvement would be to increase the size of the database to include more actors and films. We would also like to make the game more interactive so the user has more control. One easy way to do this would be to have a difficulty setting. Each actor would have a difficulty rating, and the difficulty would determine which actors the game provides the user to try to connect. The difficulty rating could be based on the number of films an actor is in, an actor's popularity among fans, the number of possible paths to other actors, and other variables. Basically the higher the difficulty rating, the more obscure the actor and the more difficult to connect them to another actor. The game would also be improved by allowing the user to enter in the path that they found. The game would then verify the path and output if it were a valid path. The program would also output a shorter path if one were available. We would also like the game to provide the user with some feedback, most easily with the output of a score. The game would calculate the score based on the difficulty and the time taken by the user to find the path.

### **References:**

Cormen, Thomas H. Introduction to Algorithms:Second Edition. McGraw-Hill: Boston, 2001.

<http://www.iro.umontreal.ca/~lecuyer/papers.html>

<http://www.imdb.com>

<http://www.chixinflix.com>

<http://www.meninmovies.com>

