Degrees of @KevinBacon

Dominick Taylor

February 27, 2018

1 Introduction

This was likely the first engaging, entertaining project I've worked on that felt like it had real-world utility. Therefore, I put a lot of time into it, and although I started out strong, there were several setbacks that were a welcome challenge to overcome. Initially, I wrote a lot of code that ended up getting deleted as I tried to decide how I wanted to approach this problem. What is likely the most fundamental problem is that it is very difficult to look at two successor nodes and say "one is better than the other". If I had realized this early, I would have likely altered my approach to suit this fact.

2 Methods

2.1 Search Algorithm Decisions

My first approach was to use Ole' Reliable: a Breadth-First Search. I very quickly realized that the enormous state space of the problem made this very unrealistic, and so I altered my approach.

My next attempt saw me use a Best-First Search. However, I made it a bit more difficult for myself than I needed to. In order to ensure that the "best" nodes were expanded first, I created a Max-Heap structure that automatically heapified itself after insertions and removals. Note that in the code, the heap is erroneously named after a priority queue. Although a pqueue is the desired effect I wanted, and the heap serves the same purpose, the structures are not the same. I could have simply used a sorted listed, but implementing an efficient sort algorithm would have been relatively costly compared to the heap that I already had. Thus, this method is the one that is used in the final version of the project.

I very nearly implemented a version that used two threads. The main thread would view tweets as before. However, it would scan the mentions and add them to a seperate queue, called the "query queue". The second thread would pop this query queue and add the resulting tweets to the tweet queue. The limitation here is that, even with level 2 authorization, I could still only

make 1 query every 3 seconds. The tweet queue did not have this limitation, and so it would expand every tweet node before the query queue even had a chance to add the diversity of other handles. Although it worked, it had the same behavior as the breadth-first search. I chose to revert back to my previous implementation.

2.2 Tweet Evaluation

Once I implemented the Max-Heap, I knew I needed a way to evaluate tweets. This proved to be a second major difficulty. Initially, I valued the number of unique handles above anything else, thinking that the more material the program had to search, the better. I quickly got stuck in loops of tweets where one tweet had a dozen metnions and no real text. I decided to tweak the value so that each unique handle was only worth 1 point.

I decided a good idea would be to, right from the beginning, place a query on the target account to see what they are tweeting about. Here, I kept a set of valuable keywords (as well as a set of words to ignore, like "a") and hashtags. I particularly valued the hashtags, where the keywords ended up having very little value. Here's the problem: the Twython queries only return recent tweets. If a user is fairly incative, then there is little data to scan. This was the case of Kevin Bacon, who rarely tweets. This approach is more valuable when the target user tweets frequently.

I would have liked to add more heuristics to improve the diversity of a tweets value. A simple thing would be to evaluate verified users more than others, due to their possible celebrity links. Secondly, it might be worth it to peruse an account followers to find potential links. This would help make up the wasted 3 seconds of nothing that the Twitter API forces us to wait.

3 Results

After a lot of testing, as well as some failed attempts to improve the code, I know that the program is capable in simple cases. The file *test1.txt* shows a simple test case where the program has succeeded. However, in larger cases - for example, finding Kevin Bacon from an arbitrary user - the program lacks any real ability. The size of the state space, and the lack of ability to make an informed decision rather than relatively random ones, really inhibit the program from completing its task in realistic time. For example, I allowed my program to run while I was sleeping one night, trying to find a link from @CNN to @KevinBacon. Sure enough, it was running when I had woken up. This does prove however that a single execution can last quite a while.

4 Conclusion

In conclusion, the program definitely works (if given enormous time). I would like to have had a go at an iterative-deepening search. This would had cut down on the number of nodes to considert, and would also have provided a decent answer. In the end, easily the most difficult part was finding a way to determine when a tweet is more valuable than another. Or could it be that Kevin Bacon is just an elusive adversary? Perhaps so, but I would have liekd to find him regardless.