
Additional Required Extension

4. Choose any additional extension in which you report some interesting graph properties or solve some interesting problem on a data set different from the kevin bacon movie data set that we provided. There are a number of extensions you can make to this assignment, and rather than choosing a data set and a problem for you, we've left it open for you to decide what you want to do.

Do you want to pull in data from movie reviews to augment your weighting so you prefer more highly rated movies? Do you want to analyze friend circles from the facebook ego network and make interesting conclusions? Do you want to provide k friend recommendations to a user based on the number of mutual/common friends? Do you want to study on how tweets evolve over time?

Pretty much any cool non-trivial graph extension is fair game here. If you are concerned about whether or not your idea is "non-trivial" enough, ask on Piazza. It is important that you solve a problem that is considerably different from path-finder / actorConnections. You may not solve the same problem on a different graph.

Implementation Checklist:

- Explore the linked videos to see potential questions that can be answered with graphs (see exploring graphs below)
- Choose a dataset to work with (see dataset details below)
- Implement your idea in a program called **`extension`** (in a file called **`extension.cpp`**). It is worth 4 points and potentially one or more star points.
- Provide a short write-up (in plain text format) in the file **`extension.txt`** which includes:
  - the problem you solved
  - how you solved it
  - how you tested it
  - how the grader should run your program (If this is not provided, you will receive a 0).
  - provide a link to where we can download the entire data set in extension.txt. If you did some pre-processing to the data set, you must include that code as well in your submission and tell. You must clearly give all steps to get the data set and reproduce the results that you have provided. **DO NOT PUSH THE ENTIRE DATA SET TO GIT.**
- Include a tsv/csv/data file **for a smaller version of the graph** for us to test the program. The data file(s) that you submit **must be small** and must contain no more than 100 vertices.
  - **ADD THE LINE `!fileName` TO THE `.gitignore` FILE. (where the fileName is the fileName of the smaller version of the graph on which we can test your code).**
- Read Starpoint details below

Exploring Graphs

In this extension you will be identifying questions that you want to answer about data that you care about. So you must pick the data set and then think about what you'd like to discover about that

data. Now, that's really open ended and it might seem a little bit daunting. How do I know what to look for?

Here are some videos from a Coursera course that Prof Alvarado co-authored that will give you some motivation and direction to get started. Link 1 Link2a Link 2b Link 3 Link 4a and Link 4b. These videos give you some leads and classes of questions that you might answer using graphs. This extension is really about your interests and your creativity. Feel free to come up with a completely new idea/ solve similar problems on other datasets/ tweak the ideas described in the videos. We really encourage you to keep looking and research. Google is your friend here.

Dataset Details

You must work with a real-world data set. It does not have to be extremely large, but it must have enough data to make it interesting (usually at least a hundred nodes or more).You may use any datasource you legally obtain online. This github "awesome list" is a great place to look: https://github.com/caesar0301/awesome-public-datasets.

We have also provided a twitter and facebook dataset that you may choose to use at

```
/home/linux/ieng6/cs100w/public/pa4/SocialNetworks
```

Below are links to videos of a Coursera course that Prof Alvarado co-authored, where this data set is described.
https://www.coursera.org/learn/intermediate-programming-capstone/lecture/EqnCh/introduction-to-some-social-network-data
https://www.coursera.org/learn/intermediate-programming-capstone/lecture/IwRBd/representing-social-network-data-as-a-graph

Starpoint Details

Since this is a challenging extension that requires a substantial amount of work, the extension may be deemed Star Point worthy. Again, we won't say yes or no for any particular problem. The graders will deem it Star Point worthy based on the difficulty of the problem attempted, correctness and quality of write-up/testing/design. Super challenging, complex extensions may even receive more than 1 star point.

Running Time

With the **`-O3`** optimization flag, it should not be taking too long (5 seconds max) to run **`pathfinder`** (with <20 query paths) or **`actorconnections`** on the full **`movie_casts.tsv`** file. We will be compiling your code with the **`-O3`** optimization flag. Also, we will predominantly test your program with smaller versions of this file (as should you) containing <100 nodes. This means you will not lose more than a point or two on each of the **`pathfinder`**/**`actorconnections`** segments if your program processes the small test files successfully but takes too long on the full file. As a rule of thumb, we expect that your program runs in **less than 2x the**

runtime of the reference, so ensure that it does. Practically, this is because your code might time out if it takes far too long, but realistically, the purpose of this course is to implement fast algorithms, so within twice the time of the reference solution should be reasonable, if not expected.

In general, aim for the reference run times or better.

For Pathfinder:
Anything more than 2x the reference solution may receive a penalty.

For Actorconnections:
We expect your solutions to run under 1 minute for any of the provided input files, the largest of which is the **`movie_casts.tsv`** database and the **`pair.tsv`** file with 100 actor pairs. **Note that you are allowed to (and should) simultaneously check all of the actor pairs as you add edges to the graph/union the sets.** The reference solution runs in a few seconds. No deductions for this part unless your solution takes a really long to run (say 2 to 3 minutes)

If your implementation is slow compared to the reference program, check the following things:

Running Time Checklist:

- Make sure you have compiled with the **`-O3`** flag
- Make sure you don't have any accidental inefficient operations (e.g. linear search) in *any* part of your code
- Consider the space efficiency of your program. Often, in C++, memory issues are the cause of running time inefficiencies:
  - Get rid of all data you don't need to store; perhaps you are storing redundant data.
  - Use references instead of values.

Hints

1. Remember that a hash table is given to you as a std::unordered_set and a hash map is given to you as a std::unordered_map. You may NOT use any pre-built data structures, like the Boost Graph Library (BGL), besides what is provided in the C++ STL data structures.
2. Compilation problems? Try having a target **`.o`** file for each **`h`** that corresponds to a class. For example:

```
ActorGraph.o: ActorGraph.h Movie.o ActorNode.o OtherClass.o
Movie.o: Movie.h
ActorNode.o: ActorNode.h
OtherClass.o:  OtherClass.h
```

Grading

The grading breakdown is as follows: This assignment is meant to be open-ended, so you may define classes/methods/data-structures how ever you wish (i.e., you can rename **`ActorGraph.h/cpp`** to what ever you want). We will only grade your assignment based on the correctness of your **`pathfinder`** and **`actorconnections`** output as well as on the correctness of your extension and its write-up. We will develop our own input graph files to test all edge cases - at a minimum, these graphs will contain at least 2 nodes and 1 movie. **All testing will be performed on connected graphs**, which are graphs that contain at least one path between all pairs of nodes. **`pathfinder`** will always be tested with pairs of unique nodes as input (i.e., no self-paths). The assignment is out of 35 points and the breakdown is as follows:

1. **5 points** for unweighted **`pathfinder`** correctness at CHECKPOINT. To receive **any** points, your program must compile and output some path (in the correct format) for each input pair on a connected graph.
2. **8 points** for the complete **`pathfinder`** correctness at FINAL SUBMISSION. Note that this might include a test of unweighted as well as weighted. To receive **any** points, your program must compile and output some path (in the correct format) for each input pair on a connected graph. 1 point will go towards managing memory correctly (i.e., no memory leaks).
3. **7 points** for **`actorconnections`** correctness at FINAL SUBMISSION. We will run your program on a number of graph files, the largest of which will be the provided **`movie_casts.tsv`** file. For each input file, we will check that your output set of movie years is correct for your BFS implementation and union-find data structure. 2 points are for your analysis of how the run time of the two implementations compare.
4. **2 points** for the Actor connections running time analysis in **`Report.pdf`**
5. **4 points** for an extension of your choosing
6. **7 points** for style and class design (including the information in the report). That is, bad style can lose you up to 20% of your final grade. Stick to the Minimum Style Guide
7. **2 points** for Survey (and CAPEs). See Part 0

Format

We are giving very specific instructions on how to format your programs' output because our auto-grader will parse your output in these formats, and **any deviation from these exact formats will cause the autograder to take off points**. There will be no special attention given to submissions that do not output results in the correct format. Although we will still give partial credit to the correctness of your results, if you do not follow the exact formatting described here, you are at risk of losing all the points for that portion of the assignment. **NO EXCEPTIONS**.