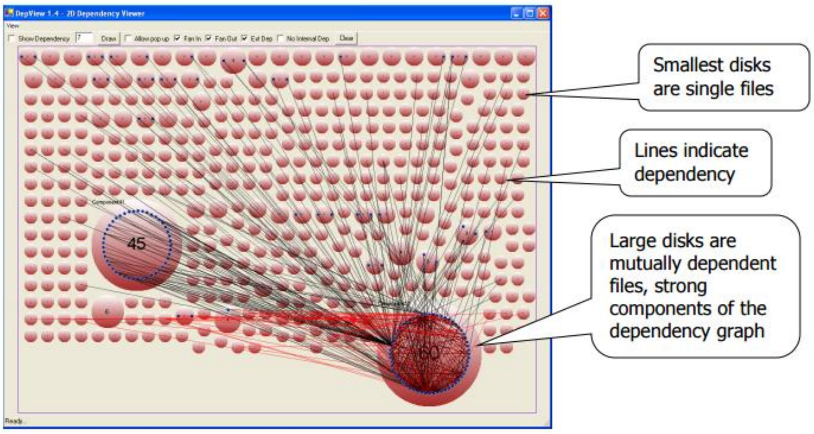
CSE 674: Advanced Data Structures and Algorithms
Due Tuesday, April 11

Project 4: File Dependency Graph Analyzer

For this project you are going to create a File Dependency Analysis tool for software packages. This tool is going to provide information about the files that can be very useful in compiling and testing. In this project we are going to focus on discovering cycles and strong components in a given set of files.

A cycle or circular reference is a very poor design decision, when not handled properly and potentially removed, will always cause compilation errors. Most compilers do not check for circular references. So this will be valuable information to show the user to understand why the code does not compile. http://stackoverflow.com/questions/625799/resolve-header-include-circular-dependencies

Strong components on the other hand give information about the testing coverage. Dependencies between software files are essential. However, dependencies complicate process of making changes and excessive dependency degrades flexibility. A strong component indicates that a change may cause new changes in all other files in the same strong component.



Smallest disks are single files

Lines indicate dependency

Large disks are mutually dependent files, strong components of the dependency graph

http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/presentations/SoftwareStructureResearchExcerpts.pdf

Your tool shall do the following:

1. Find the packages in a given directory. A package is a pair of .cpp and .h files with the same name.
2. Each package constitutes a vertex in our dependency graph. We are interested in only analyzing the #include statements. You should assume that #include statements are always on top of each file. This means that you do not have to read the entire file or develop a complex parser.
3. Construct an adjacency matrix to represent this graph. You should keep this part separate from the actual graph analysis to foster reusability. We may test your graph analyzers with some other adjacency matrix that are not necessarily file dependency graphs.
4. List the cycles and strong components.
5. List the names of vertices with DFS.
6. Show the topological sort of the vertices. (You need to first make sure that graph does not include a cycle)