Project 1 CSc 402 Kutztown

Purpose: Using the STL; Recursion; Graphs & Paths

Deadline: TBA, using the turnin script. Late submission at your own risk in force. Setup and test *turnin*. Inability to use *turnin* is not an excuse for late submission. Don't email me your program.

Description: Finding paths in a graph is a rich topic in our discipline. During class we will look at clever algorithms for finding shortest paths. But, in this project, we are going to get accustomed to advanced data structures by eschewing the efficient in favor of the naïve by designing a brute force recursive solution to the problem of finding all paths from one vertex to another in a directed graph represented by an adjacency list (not a matrix) of costs.

Start with an STL sequential container to hold a path and place the start vertex (input from the user, with error checking) in it. Then, passing the container by value, recursively add vertices reachable from the last vertex added until you find a path to the destination vertex (success) or you try to add a vertex already in the path (failure due to cycle). Continue until all possible paths have been discovered.

Each path is to be stored in an ordered (by key) STL associative container where the key is the cost of the path and the value is a container that holds all paths that have that cost. Output occurs after all paths have been found. It is to be ordered by cost, where a cost is printed followed by the vertices of all paths having that cost in a well labeled, cogent manner.

Notes:

- You may use a language other than C++, but you must get approval from the course instructor for any language other than C++, Java, Python, or C#, and you must use containers available in libraries, c.f. STL for C++, API for Java, etc.
- Number your vertices starting with 0.
- Paths may not contain cycles.
- Paths may only occur once in the output.
- Your adjacency list should use STL container(s) to handle the graph data. State your design of this list in your readme.
    - The file holding an adjacency list has, on each line, one integer representing a vertex, followed, on the same line, by pairs of integers, each representing an adjacent vertex and the cost to go to that vertex.
- Be sure the sequential container that you use to hold a path maintains the path's ordering. Use a suitable container.
- Print the paths in ascending order classified by cost. Find an associative container to accomplish this.
- Note: If you are using C++, an iterator can't be applied to a const STL object. A const iterator can.
- Provide a file named **readme.txt** that includes the following:
    - Link to your documentation site
    - Notes on the design of your adjacency list.
    - Sequential container you used to hold paths, and justification.
    - Document your major decisions in creating the algorithm.
    - Information on how to run the program.
- **GRAD STUDENTS ONLY**: You must also detect when there is no path from the start vertex to the destination and provide an analysis of the efficiency of your algorithm. Higher standards will be applied overall, as well.
- This is a senior/graduate level course. Proper style is a must. Substantial penalties, up to and including your program not being graded, will be levied for lazy, incomplete, or chintzy style.
- You must use Doxygen (for C++) or another appropriate documentation tool for your project. Information/tutorials are available on the instructor's links page.

Deliverables:

- **Turnin:**
    - The project, to be named **paths.cpp** (that exact name).
    - At least two files representing directed graphs with costs. Grads: Three files, described in readme.txt, and how they were effective in testing the algorithm, i.e. why they were useful in testing the program.
- **D2L:**
    - **readme.txt** (again, no caps) as previously specified.