CS 251, Fall 2007
Voicu Popescu
Assignment 10
Due Monday December 3rd, 11:59PM

Graphs & more

1. Implement an undirected graph data structure using adjacency lists, as a C++ class **UGraph** in UGraph.cpp. A skeleton of UGraph.cpp is provided. The class should have the following functionality:
    a. Constructor **UGraph ()**:
        i. makes an undirected graph from input file "graphinput" (always use this file name, you can hardcode the file name).
        ii. the file format is the number of vertices *n* on the first line and then one line for each vertex; a line starts with the degree of the vertex followed by the adjacency list, which gives the indices of the adjacent vertices; the constructor should maintain the adjacency list order given in the input files; for example a complete graph with 3 nodes (i.e. "triangle") is described as

```
3
2 1 2
2 0 2
2 0 1
```

        iii. a vertex stores a character label 'u' or 'v' corresponding to "unexplored" or "visited", an integer degree, an array of adjacent vertex indices modeling the adjacency list of the vertex, and an array of character edge labels with possible values 'u', 'd', 'b', or 'c', corresponding to unexplored, discovery, back, or cross edges, respectively; it is known that the maximum degree of any vertex is 20, so you can statically allocate the adjacency list array and the edge label array; initially all vertices and all edges are unexplored, indicated with labels 'u'.
    b. Destructor **~ UGraph ()**: releases all and only dynamically allocated memory.
    c. Print graph **print()**: outputs the edges of vertex with index *k* on line *k*; an edge is printed out as a vertex index and a label, separated by a space; edges are separated by a space; for example the graph above in the initial state would be printed out as

```
1 u 2 u
0 u 2 u
0 u 1 u
```

    d. Clear all labels **clearall()**: sets all vertex and edge labels to 'u'.
    e. Depth first search traversal **dfs(int vi)**: labels edges according to a traversal starting from vertex with index vi; the traversal should respect the adjacency list order; the function does not output anything, it just sets the labels.
    f. Breadth first search traversal **bfs(int vi)**l: labels edges according to a traversal starting from vertex with index vi; the traversal should respect the adjacency list order; the function does not output anything, it just sets the labels.
    g. Reachability of vertex with index *wi* from vertex with index *vi* **isreachable(int vi, int wi)**; the function prints "yes" or "no".
    h. Print shortest distance between vertices with indices *vi* and *wi* **shortestdistance(int vi, int wi)** as an integer.

i. Additional instructions
    i. Sample and code skeleton files are provided in a10_files.zip
    ii. The main.cpp file is provided, but your UGraph class should work with any valid main file. DO NOT modify the main file, the grader will use both the provided main file and another main file to test your class implementation's correctness, therefore your own code segment in the main file instead of the class files will most likely lead to a 0 for the overall program grade.
    iii. Example input and output is provided in files "sampleinput" and "sampleoutput". The provided main file will take keyboard input, and you could use "main < sampleinput" on borg machines to redirect the sampleinput file as the input. The output should be printed on the screen and you could also use "main < sampleinput > sampleoutput" to save the output into a file. Please make sure that your program can EXACTLY reproduce that output file given the input file. You may want to use the `diff' program to test this.
    iv. DO NOT change any of the function signatures. Doing so will most likely lead to a compile error when using and a 0 for your overall program grade.
    v. Use the provided Makefile.
    vi. Do not partition your code into files we haven't asked for. If you want to use auxiliary classes, please put the code inside the UGraph.cpp and UGraph.h files.

2. Extra credit 1 (2%): Show that the algorithm for testing strong connectivity given on page 615 of our text in the "Testing for Strong Connectivity" section is correct.

3. Extra credit 2 (2%): Show that a DAG has a vertex with no outgoing edges.

4. Extra credit 3 (3%):
    a. Build a text file named "paper.txt" from an ACM Transactions on ABC paper; ABC should be your favorite computer science subfield.
    b. Implement a **void kmp(char *string)** function that searches the text file for the string using the Knuth-Morris-Pratt algorithm; the function prints out the failure function for the string and prints out "found" or "not found".
    c. A main function is provided in e3.cpp, as well as a Makefile in a10_ex3.zip.

5. Turn in instructions:
    a. Assignment specific:
        i. Turn in files:
            1. Required Part: UGraph.cpp, UGraph.h, main.cpp, Makefile
            2. Extra credit:
                a. extracredit.pdf for extra credit 1 and 2 (in one pdf file) in lab10 folder.
                b. If extracredit 3 is attempted, create a folder named ex3 under your lab10 folder; inside this folder, include e3.cpp (which will include your kmp function), Makefile, and the text file paper.txt. **DO NOT** include ex3 folder if extracredit 3 is not attempted.

c. Submit the corresponding extra credit blank file.

ii. **Your code is REQUIRED to compile with /opt/csw/gcc3/bin/g++ (gcc version 3.4.5) on the borg machines. If your code doesn't compile with this compiler, then even if it compiles with another, your code will still be considered non-compiling.**

iii. Remember that assignment-specific turn-in instructions override general instructions.

iv. Directory structure: all files in the directory lab10.

v. turnin command executed in the directory containing your lab10 directory:

- turnin -c cs251 -p lab10 lab10.

b. General:
See instruction at http://www.cs.purdue.edu/cgvlab/courses/251/As/turnin.html
Make sure to check it EVERY TIME before submitting a new assignment. It is updated periodically.