Coding Assignment 5

CSE3318

Name your program `Code5_xxxxxxxxxx.c` where `xxxxxxxxxx` is your student id (not netid). My file would be named `Code5_1000074079.c`.

Please place the following files in a zip file with a name of `Code5_xxxxxxxxxx.zip` and submit the zip file.

`Code5_xxxxxxxxxx.c`

`Graph.txt`

Drawing of graph – `PDF` or `PNG` or `JPG`

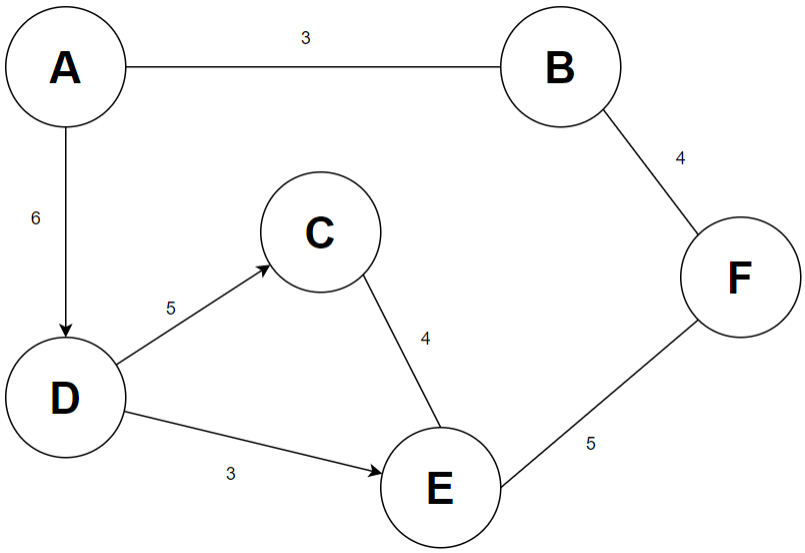
A zip file is used to avoid Canvas's file renaming convention.

Reminder – ANY compiler warning(s) or error(s) when compiled will result in an automatic 0. This apply no matter what the warning/error is.

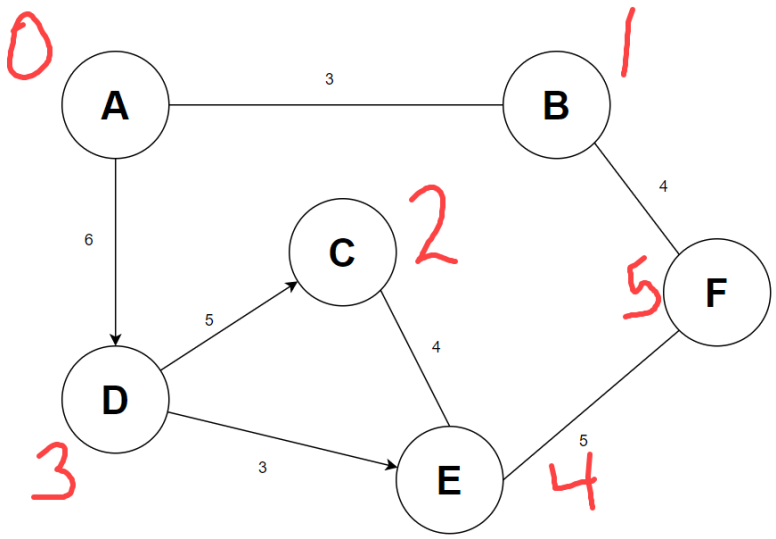
The Program

You are creating a program that can read a file of graph information and run Dijkstra's Algorithm on the graph and produce the path and the length/weight of the path between the starting Vertex and any other Vertex in the graph.

Let's say we start with this graph.



To translate graph to file, we'll first start by numbering the vertices. The number we assign to a vertex will be that vertex's index number in the Vertex Array. Just for convenience, I am going to start with A and number it 0 and go alphabetically from there. This is not a necessary order. You could start with Vertex E and 0 and mark Vertex C as 1, etc. Just pick one to be index 0 and number from there.



Now we can translate this graph to our file format. Our file format assumes that each line in the file represents a vertex and all of its edges and weights. If the file has 5 lines, then the graph has 5 vertices.

The format of a line in the file is

VertexLabel, Adjacent Vertex Index, Weight of Edge

where "Adjacent Vertex Index, Weight of Edge" can be repeated. The first line in our file would be

```
A,1,3,3,6
```

This shows that Vertex A (which is at index 0) has an edge between itself and Vertex B (which is at index 1) and that edge has a weight of 3. Vertex A also has an edge between itself and Vertex D (index 3) and that edge has a weight of 6.

The file for this graph will be

```
A,1,3,3,6
B,0,3,5,4
C,4,4
D,2,5,4,3
E,2,4,5,5
F,1,4,4,5
```

The ordering of the vertex, edge pairs does not matter – `0,3,5,4` is the same as `5,4,0,3`. You need to create a file that represents a graph you have created – it should be different than the graph shown in the assignment. Name the file `Graph.txt` and submit it with your assignment. Your program will be tested with various files/graphs.

You will also need to submit a drawing of your graph. Draw the graph by hand or use a program. However you create the graph, you must submit an electronic picture version of it. Make a PDF version, take a picture with your phone or take a screenshot – doesn't matter how you do it. Your submission must be something that can be viewed without a specialized app – use PDF, JPEG or PNG.

IMPORTANT : make NO assumptions about the ordering or the labeling of the vertices in the graph. The VertexLabel could be up to 5 characters and the vertices can be listed in any order in the file. The labels will not contain spaces.

Getting Started

Create a define that will be set to the maximum size of your graph – the maximum number of vertices that your program will load and process. Create your Vertex Array and your Adjacency Matrix using that define. Initialize the Adjacency Matrix to -1.

See **File Handling** for details on how to populate the Vertex array and the Adjacency Matrix. Prompt for the starting vertex. Your program should be able to use any vertex in the graph as the starting vertex.

File Handling

As in the previous assignments, open the file from the command line using `argv[1]`. While using `fgets()` to read through the file, use `strtok()` to parse each line into its parts. Use the parts to add vertices to the Vertex Array and edges to the Adjacency Matrix.

Conditional Compile

Add a conditional compile statement `PRINTIT` to print out the Adjacency Matrix after you have populated it (using values from the file). So, if your code is compiled as

```
gcc Code5_xxxxxxxxxx.c -D PRINTIT
```

then your adjacency matrix will be printed to the screen.

```
#ifdef PRINTIT
printf("\n");
for(i = 0; i < VertexCount; i++)
{
    for(j = 0; j < VertexCount; j++)
        printf("%5d\t", AdjMatrix[i][j]);
    printf("\n");
}
#endif
```

```
   -1     3    -1     6    -1
    3    -1    -1    -1    -1
   -1    -1    -1    -1     4
   -1    -1     5    -1     3
   -1    -1     4    -1    -1
```

The `PRINTIT` conditional compile should also be used to print out your Vertex Array after running Dijkstra.

```
#ifdef PRINTIT
printf("\n\nI\tL\tD\tP\tV\n");
for (i = 0; i < VertexCount; i++)
{
    printf("%d\t%s\t%d\t%d\t%d\n", i,
        VertexArray[i].label, VertexArray[i].distance,
        VertexArray[i].previous, VertexArray[i].visited);
}
printf("\n");
#endif
```

```
I     L     D     P     V
0     A     0     -1    1
1     B     3     0     1
2     C     11    3     1
3     D     6     0     1
4     E     9     3     1
```

I = **I**ndex, L = **L**abel, D = **D**istance, **P** = Previous, **V** = Visited

This feature will be used to grade your program. The printing should be controlled with the compiler directive (not by commenting or uncommenting code). The output of your adjacency matrix and vertex array **must** have these formats.

Printing and Prompts

The adjacency matrix must be printed AFTER being populated from the file and the vertex array must be printed AFTER running Dijkstra's Algorithm. The print outs must be after prompting for the starting vertex. The prompting for the end vertex and printing of the path may be before or after the print but must be after running Dijkstra's Algorithm on the data. The path MUST be printed in order starting with the start vertex and ending with the end/destination vertex. The path should not print from ending to starting – it must be in order.

Dijkstra's Algorithm

I suggest you use the Dijkstra code shown in class. It closely mirrors the manual method we learned. If you so choose to use the Internet as your source, be sure you are getting the C version of Dijkstra (and not some other graph traversal/variation of Dijkstra). Your Dijkstra code, regardless of where you source it, **MUST** use an adjacency matrix and a vertex array so that you can create the required output for grading.

Program Input/Output

```
student@maverick:/media/sf_VM/CSE3318$ gcc Code5_1000074079.c -g -D PRINTIT
student@maverick:/media/sf_VM/CSE3318$ ./a.out Graph.txt

   -1     3    -1     6    -1    -1
    3    -1    -1    -1    -1     4
   -1    -1    -1    -1     4    -1
   -1    -1     5    -1     3    -1
   -1    -1     4    -1    -1     5
   -1     4    -1    -1     5    -1

What is the starting vertex? A


I     L     D     P     V
0     A     0     -1    1
1     B     3     0     1
2     C     11    3     1
3     D     6     0     1
4     E     9     3     1
5     F     7     1     1

What is the destination vertex? E
The path from A to E is A->D->E and has a length of 9
```

```
student@maverick:/media/sf_VM/CSE3318$ gcc Code5_1000074079.c -g
student@maverick:/media/sf_VM/CSE3318$ ./a.out Graph.txt

What is the starting vertex? E
What is the destination vertex? A
The path from E to A is E->F->B->A and has a length of 12
```

Extra Notes

Please remember that the label variable must be a `char[6]` array - not a pointer.

If you are using `char *label`, then you must change it to `char label[6]`

`char [6]` allows for labels that are 5 characters plus room for the null terminator. Your program will NOT be tested with labels that contain spaces.

Your Dijkstra function should have an outer for loop and 2 separate inner loops. If you have a for i loop INSIDE a for i loop, your code is wrong.

```
for (x = 0; x < VertexCount-1; x++)
{
    for(i = 0; i < VertexCount; i++)
    {
        relaxing the edges
    }

    SmallestVertexIndex = -1;
    int SmallestVertex = INT_MAX;

    for(i = 0; i < VertexCount; i++)
    {
        finding the smallest
    }
}
```

One suggestion to print the path is to create a simple integer array of size `MAX`. `MAX` should be a `#define` at the top of your program (remember #defines are not variables; therefore, your `#define MAX 50` at the top of your program is not a global variable). Then, put the index of the destination vertex in `path[0]`. Then, put the destination's previous value in `path[1]`. Then, use that previous value to index into your `VertexArray` and get that previous value and put it in `path[2]`. Continue getting the previous value from `VertexArray` until it's -1 (which means you are back at the starting index). Use a loop to do this since you don't know how many vertices will be in the path.

Now use your path array to print the path. It contains the path in reverse, so loop through your path array in reverse to print the path in the correct order.

To print the vertex labels, use the value in your path array to get back to the label. For example, if `path[i]` is 1, you need to print the vertex label at element `i` in the `VertexArray`.

`VertexArray[path[i]].label`

Something to note - the assignment and the rubric both refer to `VertexArray` as a regular array of size `MAX` (where `MAX` is defined to be 50). The `addVertex()` code shown in the slides show `VertexArray` being used as an array of pointers.

THIS ASSIGNMENT USES `VertexArray` AS A REGULAR ARRAY OF TYPE `Vertex`.

You will need to translate that code from using pointers back to code using a regular array containing a structure. No mallocs and no use of ->

For dealing with a vertex with no edge

If you detect a new line (a vertex without an edge), then just remove it before using `strtok`

```
while (fgets(FileLine, sizeof(FileLine)-1, FH))
{
   if (FileLine[strlen(FileLine)-1] == '\n')
      FileLine[strlen(FileLine)-1] = '\0';
   token = strtok(FileLine, ",");
```