Algorithms in C++: Assignment 7

1. Objective

Your goal is to solve the all pairs shortest paths problem with Floyd's algorithm. For every vertex in the graph, you must find the shortest path (if a path exists) to every other vertex.

2. Problem

Your program begins by reading a text file with the initial distance matrix for a graph. The first line of the file should be an integer between 1 and 26, which represents the number of vertices in the graph. The rest of the file should contain a variable number of lines with three components: the starting vertex, the ending vertex, and the weight of the edge connecting the two vertices. Any number of problems can occur on these lines, and your solution must be equipped to handle them. The output of the program for various errors is listed below:

- No command line argument or too many arguments:

```
$ ./shortestpaths
Usage: ./shortestpaths <filename>
```

- Input file not found in the same folder as the executable:

```
$ ./shortestpaths notfound.txt
Error: Cannot open file 'notfound.txt'.
```

The rest of the input errors are in the content of the input file itself. Throughout the rest of this document, the content of each sample input file is displayed in a gray box.

- The number of vertices is not an integer between 1 and 26.

```
0
A B 1
B C 4
```

```
Error: Invalid number of vertices '0' on line 1.
```

- Each line should have three components. Check the number of components before verifying the content of each. Line 2 has only 2 components, so it's considered to be invalid edge data.

```
3
A B 1
B C
```

```
Error: Invalid edge data 'B C' on line 3.
```

- The starting vertex must be in range. For instance, if there are 3 vertices, as in the file below, the only valid vertices and A, B, and C.

```
3
X Y 1
Y Z 4
```

```
Error: Starting vertex 'X' on line 2 is not among valid values A-C.
```

- The same holds for the ending vertex. It must be a valid capital letter in the range A..(letter corresponding to the number of vertices). In the example below, there are 4 vertices permitted, namely A through D.

```
4
A C 2
A D 10
B A 7
B C 5
B D 1
C B 3
D A 2
D B 6
D NYC 6
```

```
Error: Ending vertex 'NYC' on line 10 is not among valid values A-D.
```

- Finally, make sure the edge weight is a positive integer.

```
3
A B 1
B C -5
```

```
Error: Invalid edge weight '-5' on line 3.
```

If the same edge appears more than once in the file, the subsequent line replaces the data from the previous line.

After verifying all the data, you should create a distance matrix with the weights of all the edges specified in the file. Because you must handle any valid positive integer, including INT_MAX, the matrix should be one of longs. Values on the major diagonal must be initialized to zero, while all the remaining cells should be however you choose to represent infinity.

Consider the following input file:

```
4
A C 2
A D 10
B A 7
B C 5
B D 1
C B 3
D A 2
D B 6
D C 6
```

After your program parses the file, it should have a distance matrix in memory that looks as follows, where a '-' character represents infinity:

```
Distance matrix:
   A  B  C  D
A  0  -  2 10
B  7  0  5  1
C  -  3  0  -
D  2  6  6  0
```

You will need two additional matrices of the same dimensions as the initial distance matrix – one for keeping track of the lengths of the shortest paths and the other for determining which vertex was used as an intermediate. For the same input as seem above, your program will produce the following output:

```
Path lengths:
  A B C D
A 0 5 2 6
B 3 0 5 1
C 6 3 0 4
D 2 6 4 0

Intermediate vertices:
  A B C D
A - C - C
B D - - -
C D - - B
D - - A -
```

Upon computing this information, you can easily determine the paths between any two vertices in the graphs. For each pair, the backtracking algorithm will recursive over the matrix of intermediate vertices and produce a string containing the shortest path, as seen below:

```
A -> A, distance: 0, path: A
A -> B, distance: 5, path: A -> C -> B
A -> C, distance: 2, path: A -> C
A -> D, distance: 6, path: A -> C -> B -> D
B -> A, distance: 3, path: B -> D -> A
B -> B, distance: 0, path: B
B -> C, distance: 5, path: B -> C
B -> D, distance: 1, path: B -> D
C -> A, distance: 6, path: C -> B -> D -> A
C -> B, distance: 3, path: C -> B
C -> C, distance: 0, path: C
C -> D, distance: 4, path: C -> B -> D
D -> A, distance: 2, path: D -> A
D -> B, distance: 6, path: D -> B
D -> C, distance: 4, path: D -> A -> C
D -> D, distance: 0, path: D
```

The output for this assignment includes all the matrices, as well as the listing of paths. For completeness, you can view the entire output below.

```
$ ./shortestpaths graph.txt
Distance matrix:
   A  B  C  D
A  0  -  2 10
B  7  0  5  1
C  -  3  0  -
D  2  6  6  0

Path lengths:
  A B C D
A 0 5 2 6
```

```
B 3 0 5 1
C 6 3 0 4
D 2 6 4 0

Intermediate vertices:
  A B C D
A - C - C
B D - - -
C D - - B
D - - A -

A -> A, distance: 0, path: A
A -> B, distance: 5, path: A -> C -> B
A -> C, distance: 2, path: A -> C
A -> D, distance: 6, path: A -> C -> B -> D
B -> A, distance: 3, path: B -> D -> A
B -> B, distance: 0, path: B
B -> C, distance: 5, path: B -> C
B -> D, distance: 1, path: B -> D
C -> A, distance: 6, path: C -> B -> D -> A
C -> B, distance: 3, path: C -> B
C -> C, distance: 0, path: C
C -> D, distance: 4, path: C -> B -> D
D -> A, distance: 2, path: D -> A
D -> B, distance: 6, path: D -> B
D -> C, distance: 4, path: D -> A -> C
D -> D, distance: 0, path: D
```

---

3. Advice

You may work in **pairs** on this assignment. Start early, as there are many small functions that you'll need to integrate into your final solution.

Printing the matrix can be a tedious tasks. You may use (or modify) the following function to suit your needs. Of course, you will have to decide how to represent INF so that the code makes sense and compiles:

```
/**
 * Displays the matrix on the screen formatted as a table.
 */
void display_table(long** const matrix, const string &label,
                   const bool use_letters = false) {
  cout << label << endl;
  long max_val = 0;
  for (int i = 0; i < num_vertices; i++) {
    for (int j = 0; j < num_vertices; j++) {
      long cell = matrix[i][j];
      if (cell < INF && cell > max_val) {
        max_val = matrix[i][j];
      }
    }
  }
```

```
  int max_cell_width = use_letters ? len(max_val) :
    len(max(static_cast<long>(num_vertices), max_val));
  cout << ' ';
  for (int j = 0; j < num_vertices; j++) {
    cout << setw(max_cell_width + 1) << static_cast<char>(j + 'A');
  }
  cout << endl;
  for (int i = 0; i < num_vertices; i++) {
    cout << static_cast<char>(i + 'A');
    for (int j = 0; j < num_vertices; j++) {
      cout << " " << setw(max_cell_width);
      if (matrix[i][j] == INF) {
        cout << "-";
      } else if (use_letters) {
        cout << static_cast<char>(matrix[i][j] + 'A');
      } else {
        cout << matrix[i][j];
      }
    }
    cout << endl;
  }
  cout << endl;
}
```