CS2040C Data Structures & Algorithms
Assignment #5: Single Source Shortest Path (SSSP)

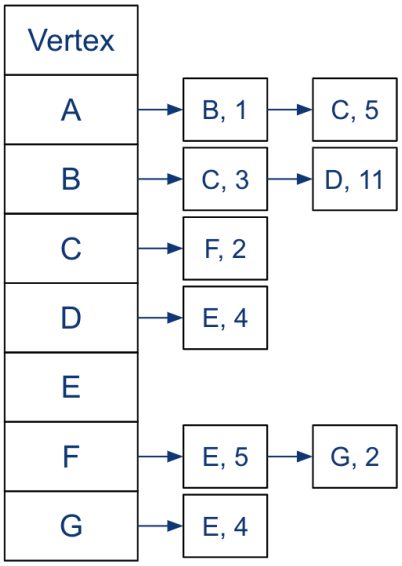
Objective

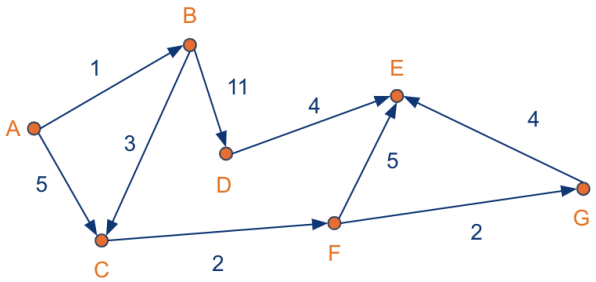
In this assignment you will be given a graph data structure and asked to compute the shortest path between two nodes. The graph will be directed and weighted (i.e. each edge has a positive integer weight) and you can assume that there are no self-loops or multi-edges (i.e. each edge has a distinct source-destination pair). You will be provided a zip file that contains:

- graph.h and graph.cpp that defines and implements the Graph data structure (detail below).
- shortest_path.h defines a Path data structure and the interface to shortestPath.
- shortest_path.cpp where you are to implement the shortestPath function.
- shortest_path_test.cpp with a number of tests.
- CMakeLists.txt for building and running the project.

Graph Data Structure

The Graph data structure represents a directed, weighted graph as an adjacency list. Consider the graph shown below.





Below is a code snippet for the class interface to Graph. Vertices are numbered starting at 0 (the diagram above uses letters to make it easier to see the connection between the data structure and the drawing) and correspond to a forward_list (singly-linked list) of edges. Each edge has type GraphEdge, which has a destination and a (positive) integer weight. The starting vertex is implicit in the construction. Use the method edges_from(i) to get the list of edges from source vertex i.

```
class Graph {
 public:
  // Create an empty graph with n vertices
  Graph(int n);

  int num_vertices() const;

  const forward_list<GraphEdge>& edges_from(int i) const;

  void addEdge(int source, int dest, int weight);

  void print() const;
};

class GraphEdge {
 public:
  int dest() const;
  int weight() const;
};
```

Task 1: Shortest Path

Your task is to implement the shortestPath function defined below to find the shortest path and distance from a given source vertex and the given destination vertex.

```
Path shortestPath(const Graph& g, int source, int dest);
```

The Path data structure holds the path as a vector (dynamic array) of vertex indices, and a single integer as the path length.

```
class Path {
 public:
  Path(int total_distance, vector<int> path);
  int total_distance() const;
  const vector<int>& path();
};
```

You are to submit only the contents of shortest_path.cpp without any include lines to Coursemology. Coursemology includes the max heap used in Assignment 4 automatically. Your code will be compiled and run against the same public unit tests as in the package, as well as additional private tests.

Libraries

As you may have noticed, Graph uses the STL libraries for vector (dynamic array) and forward_list (singly-linked list). Since the goal of this assignment is for you to learn how to apply the SSSP algorithm using existing data structures, we felt it would be more instructive to use STL for these particular data structures. See if you can use native C++ constructs (such as iterators and for loops) to simplify your code.

That said, note that you must not use any other STL libraries. To be concrete, you are allowed to use the following libraries only:

1. vector (and associated iterators)
2. forward_list (and associated iterators)
3. string
4. cout, cerr, endl
5. Standard exceptions (e.g. runtime_error)
6. Heap from Assignment 4 (no need to submit, it will be provided in Coursemology)

To be clear: you must not use a priority queue from the STL, nor any other library not listed above.

As always, you may experiment with any code you would like to on your own system, but the code you submit to Coursemology must be your own and must conform to the rules above.

Task 2: Trains

In this exercise you will learn to use the data structures and algorithm in Task 1 to solve a problem. You will be given the time table for trains between cities, such as the example below, along with a starting city and destination city. You are to find the sequence of cities that minimizes the total time traveling between cities.

```
Student ID: A0123456X
        A     B     C     D     E     F     G     H     I     J
   A              186         192
   B                                 178         147
   C          59                 190                           120
   D               180                       158         195
   E    80
   F                              118          33          98
   G   186                                                   169
   H    94
   I   170   101          52    199    62         172          68
   J                                       21
Source B
Destination A
```

In the example above, trains depart from the city listed on the left hand side and arrive at the city listed on the top of the table in the number of minutes shown. Blank entries indicate no train service. For example, the train from city A to city C takes 186 minutes, and there is no train from city C to city A. The student in this example is to find the shortest path from city B to city A.

You will be given a file trains.zip that contains a text file trains.txt. Find your student ID in the file, and use the associated table, source, and destination.

While you may use any tools you would like in order to process the data, **you must use your own code to find the solution.** Acceptable methods include:

- Processing the data in C++ using your own code directly
- Writing a Python script to convert the data into a form that can be used by your C++ code
- Using a spreadsheet to manipulate the data yourself

Unacceptable methods include:

- Using a colleague's code to manipulate the data before using your code to solve it.
- Using code from the internet for either data manipulation or solving.
- Using other tools, including ChatGPT, to either manipulate the data or solve the problem automatically.

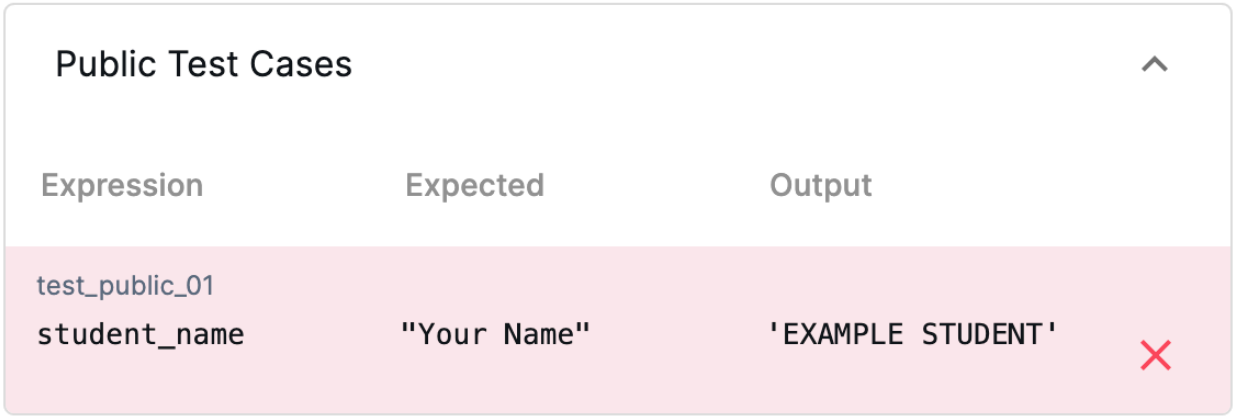
We may ask you to provide an explanation and any intermediate steps to prove your answer.

You will submit to Coursemology three lines of text as follows:

```
student_id = "A0123456X"
path = ['B', 'G', 'D', 'A']
length = 123
```

Where you replace the values with your solution. In particular, you must replace the student_id with your student ID. The path line may contain as few or as many vertices as needed, and must be letters as in the time table. The path length is the total time cost of the trip.

To make sure you have properly filled out the student ID field, Coursemology will look up your name based on your student ID and display it in a public test as below. Check that the correct name is displayed before finalizing your submission. Note that the test will always fail.



Extension Tasks

Here are some additional tasks that will further help you develop your DS&A skills:

1. What real world problems could you solve using your algorithm? Are there day-to-day problems you could measure and then solve?
2. Can you solve any of the problems from Week 11's lab using your SSSP algorithm? Specifically, the rickety bridge or Manhattan driving problem?
3. What happens if edge weights can be negative? Is there an algorithm that could solve that case?