Problem

• Find the best (briefest) flight route between two cities in an airline’s flight network.

Solution

• Model the flight network as a graph, with the cities as vertices and the available flights as edges, and use a graph optimization algorithm to find the best flight route.
Section I: Background Information

- Graphs
- Dijkstra’s Algorithm
The Graph Data Structure

Definition:

- A Graph is a collection composed of a non-empty set of vertices, and a set of pairs of vertices called edges.
- There are different types of graphs; our Graph data structure implements non-directed, weighted, cyclic graphs (can be used trivially for non-weighted graphs by setting all edge weights to 1).
Adjacency List:
- Represents a graph as a vector of $n$ source vertices, each of which contains a list of pairs of destination vertices and the weight of the edge connecting the two.
- Example:
Graph Representations II

Adjacency Matrix:
• Represents a graph as an n x n matrix (implemented as n vectors of size n). Entry i,j in the matrix contains the weight of the edge connecting vertices i and j if it exists; otherwise, the entry is zero.
• Example:

```
   A   B   C   D
A  0   5   2   0
B  5   0   3   0
C  2   3   0   8
D  0   0   8   0
```
Dijkstra's Algorithm

- An $O(n^2)$ algorithm for finding the shortest path between two vertices in a weighted graph.
- Proceeds by building up a set $S$ of vertices such that the shortest path from the source vertex to each vertex in $S$ is known. Initially, $S$ is empty, and in each iteration of the algorithm, the vertex closest to $S$ (that is the vertex not in $S$ with the shortest path from the source to the vertex that contains only vertices in $S$) is added to the set. The algorithm terminates when the destination vertex is added to $S$. 
Dijkstra’s Algorithm

Example: Find shortest path from A to D in the following graph

A: 0 (A)
B: 5 (A, B)
C: 7 (A, C)
D: 15 (A, C, D)
Section II: Implementation

- Graph classes
- Iterators
The classes

The group implemented two versions of the graph data structure, one which stores the graph internally as an adjacency matrix and one which stores the graph internally as an adjacency list.

- **Graph** - the graph base class, which defines the interface through which programs access graphs
- **AM_Graph** - adjacency matrix implementation of a graph (inherits from Graph)
- **AL_Graph** - adjacency list implementation of a graph (inherits from Graph)
- **Vertex** - holds information about a vertex in the graph; used especially by graph algorithms
- **Edge** - holds information about an edge in the graph
In order to facilitate moving through a graph, the group implemented iterators for both the AM_Graph and AL_Graph classes. It was necessary to create two types of iterators:

- **vertex_iterator** - moves through the vertices of a graph (dereferencing a vertex_iterator gives a Vertex)
- **edge_iterator** - moves through the edges from a given vertex (dereferencing an edge_iterator gives an Edge)
void AM_Graph::print()
{
  bool first;
  vertex_iterator v_it; // used to iterate through the vertices
  edge_iterator e_it;  // used to iterate through the edges

  // Print the edge information
  for (v_it = vertex_begin(); v_it != vertex_end(); v_it++)
  {
    first = true;
    cout << endl << (*v_it).getName() << ':';

    for (e_it = edge_begin(v_it); e_it != edge_end(v_it); e_it++)
    {
      if (first)
        first = false;
      else
        cout << ',';
      cout << (*e_it).getVertex2().getName()
        << '(' << (*e_it).getWeight() << ');
    }
  }
  cout << endl;
}
Section III: Demonstration

Please select your departure and destination airports.
Departing from: Atlanta Int'l Airport [ATL]
Going to: South Bend Regional Airport [SBN]
Travel Time: 3 hours, 10 minutes
Shortest Route: Atlanta Int'l Airport [ATL] to Cincinnati Airport [CVG] to Columbus [OH] Int'l Airport [CMH] to South Bend Regional Airport [SBN]