**Graphs 2**

**Weighted Adjacency Matrix: Floyd's Algorithm**

A *weighted* graph contains weight (or distance or cost) information about the edges, as shown below. Then the typical question to ask concerns the shortest (or cheapest) path.

2

3

8

3

3

10

5

5

4

2

**5** Pittsburgh

**6** Princeton

**1** Pensacola

**2** Peoria

**7** Pueblo

**3** Phoenix

**4** Pierre

**0** Pendleton

5

To simplify things, we will use graphs that do not have negative cycles. We will ignore loops, which allows us to put 0 on the major diagonal as the cost to "travel" from a vertex to the same vertex. Notice that the Princeton-to-Princeton loop 5 becomes 0 in the adjacency matrix. We use some impossibly large value (9999) to represent that no path exists. The diagram above becomes the map and the matrix below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0.** | **1.** | **2.** | **3.** | **4.** | **5.** | **6.** | **7.** |
| **0.** | 0 | 9999 | 9999 | 9999 | 9999 | 9999 | 9999 | 8 |
| **1.** | 9999 | 0 | 9999 | 5 | 9999 | 9999 | 9999 | 9999 |
| **2.** | 9999 | 9999 | 0 | 9999 | 9999 | 5 | 9999 | 3 |
| **3.** | 9999 | 9999 | 9999 | 0 | 9999 | 10 | 9999 | 3 |
| **4.** | 2 | 9999 | 9999 | 9999 | 0 | 9999 | 9999 | 9999 |
| **5.** | 9999 | 4 | 9999 | 10 | 9999 | 0 | 9999 | 9999 |
| **6.** | 9999 | 9999 | 9999 | 9999 | 9999 | 2 | 0 | 9999 |
| **7.** | 8 | 9999 | 9999 | 9999 | 3 | 9999 | 9999 | 0 |

|  |
| --- |
| 0-Pendleton 1-Pensacola 2-Peoria 3-Phoenix 4-Pierre 5-Pittsburgh 6-Princeton 7-Pueblo |

What is the lowest cost from Phoenix to Pittsburgh? \_\_\_\_

What is the lowest cost from Pittsburgh to Phoenix? \_\_\_\_

What is the lowest cost from Pueblo to Pendleton?\_\_\_\_\_  
What is the lowest cost from Pendleton to Phoenix?\_\_\_\_\_

Floyd's Algorithm is very similar to Warshall's Algorithm (and some books call it the Floyd-Warshall Algorithm). While Warshall's Algorithm solved the *all pairs reachability problem*, Floyd's algorithm solves the *all pairs lowest-cost problem*. The disadvantage is that both algorithms run in O(V3), which may make them too large to run in practice.

**Assignment**

In your AdjMat implement the Floyd interface.

interface Floyd  
 {  
 public int getCost(int from, int to);  
 public int getCost(String from, String to);  
 public void allPairsWeighted(); //Floyd's Algorithm  
 }

Floyd's is an easy modification of Warshall. Test it with FloydDriver which has been written for you. The two files are called "citymatrixweighted" and "cities". You will turn in the improved AdjMat.

**Sample Run**

Floyd's Algorithm!

Enter file with the weighted matrix: citymatrixweighted

Enter file of names: cities  
  
Adjacency Matrix  
 0 9999 9999 9999 9999 9999 9999 8   
 9999 0 9999 5 9999 9999 9999 9999   
 9999 9999 0 9999 9999 5 9999 3   
 9999 9999 9999 0 9999 10 9999 3   
 2 9999 9999 9999 0 9999 9999 9999   
 9999 4 9999 10 9999 0 9999 9999   
 9999 9999 9999 9999 9999 2 0 9999   
 8 9999 9999 9999 3 9999 9999 0

0-Pendleton  
1-Pensacola  
2-Peoria  
3-Phoenix  
4-Pierre  
5-Pittsburgh  
6-Princeton  
7-Pueblo

Cost Matrix  
 0 9999 9999 9999 11 9999 9999 8   
 13 0 9999 5 11 15 9999 8   
 8 9 0 14 6 5 9999 3   
 8 14 9999 0 6 10 9999 3   
 2 9999 9999 9999 0 9999 9999 10   
 17 4 9999 9 15 0 9999 12   
 19 6 9999 11 17 2 0 14   
 5 9999 9999 9999 3 9999 9999 0

What is the cost? Enter start city (-1 to exit): Pittsburgh  
 Enter end city: Phoenix  
9

What is the cost? Enter start city (-1 to exit): Pendleton  
 Enter end city: Phoenix  
9999

What is the cost? Enter start city (-1 to exit): -1