Data Structures and Algorithms  
**Weighted Graphs & Algorithms**  
Goodrich & Tamassia Sections 13.5 & 13.6  
- Weighted Graphs  
- Shortest Path Problems  
- A Greedy Algorithm

**Weighted Graphs**  
Sometimes want to associate some value with the edges in graph.  

```
20
1 -------> 2
/ \ / \
50/ \50 /20
/ \ /
10 v v 20
5 -------> 3 -------> 4
```

So.. label all the edges with a number. That number (called the weight) could represent:
- Distances between two locations (cities; computers on network)
- Time taken to get from one node to another (stations; states in schedule or plan).
- Cost of traversing the edge (train fares; cost of wires)

**Weighted Graph ADT**  
- Easy to modify the graph ADT(s) representations to accommodate weights  
- Also need to add operations to modify/inspect weights.

For example we can modify adjacency matrix representation so entries in array are now numbers (int or float) rather than true/false.

You can travel from a node to itself at zero cost, and if there is no connection between two nodes then the “weight” is ‘null’ (sometimes called ‘infinity’): typically a large number in simple implementations

```
1 2 3 4 5
1 0 20 50 null 50
2 null 0 20 null null
3 null null 0 20 null
4 null null null 0 null
5 null null 10 null 0
```
Weighted Edge Class

Introduce a `WeightedEdge` subclass, derived from the `Edge` class.

For genericity the weight is an `Object` it can take different classes of weights, e.g. `Integer`, `MyInteger`, `MyFloat`

```java
public class WeightedEdge extends Edge {
    // data member
    Object weight;

    // constructor
    public WeightedEdge(int theVertex1, int theVertex2, Object theWeight) {
        super(theVertex1, theVertex2);
        weight = theWeight;
    }
}
```

Weighted Graph Class

Introduce a `WeightedGraph` subclass, derived from Sahni’s `Graph` class.

```java
public class AdjacencyWDigraph extends Graph {
    int n; // number of vertices
    int e; // number of edges
    Object [][] a; // adjacency array

    // constructors
    public AdjacencyWDigraph(int theVertices) {
        // validate theVertices
        if (theVertices < 0)
            throw new IllegalArgumentException
                ("number of vertices must be >= 0");
        n = theVertices;
        a = new Object [n + 1] [n + 1];
        // default values are e = 0 and a[i][j] = null
    }

    /*put edge e into the digraph;
       if the edge is already there, update its weight to e.weight */
    public void putEdge(Object theEdge) {
        WeightedEdge edge = (WeightedEdge) theEdge;
        int v1 = edge.vertex1;
        int v2 = edge.vertex2;
        if (v1 < 1 || v2 < 1 || v1 > n || v2 > n || v1 ==
            throw new IllegalArgumentException
                ("(" + v1 + "," + v2 +
                ") is not a permissible edge");

        if (a[v1][v2] == null) // new edge
            e++;
        a[v1][v2] = edge.weight;
    }
}
```

Shortest Path Problems

Many problems can be solved using weighted graphs. For example finding the ‘shortest path’ between two nodes, e.g.,:

- shortest distance between two cities by road links.
- fastest train journey
- cheapest plane journey
- lowest cost plan

‘length’ of path is just sum of weights on relevant edges. e.g.:

N.B. the shortest path may visit more nodes!
A Shortest Path Algorithm

There are several possible shortest path problems, we consider the single source, all destinations version.

If all the weights are the same, then breadth first search finds shortest path first:

Explores paths of length N before paths of length N+1

But for arbitrary weights we need a slightly more complex algorithm developed by E.Dijkstra. My intuition is “how far can you go for your money”.

More formally, the key is From the vertices to which a shortest path has not been generated, select the one that results in the least path length

Recording Paths and Path Lengths

Observe that

• the 2nd path is a 1-edge extension of the 1st;
• the 3rd path is a 1-edge extension of the 2nd;
• the 4th path is a 1-edge extension of the 1st;
• the 5th path is a 1-edge extension of the 3rd;

So we can represent a path by recording the immediate predecessor for each vertex as a data member path.

Shortest Path Algorithm

<table>
<thead>
<tr>
<th>Path Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
</tr>
<tr>
<td>1 --------&gt; 2 0</td>
</tr>
<tr>
<td>/ \ /</td>
</tr>
<tr>
<td>50/ \50 /20 20</td>
</tr>
<tr>
<td>/ \ /</td>
</tr>
<tr>
<td>v 10 v v 20 40</td>
</tr>
<tr>
<td>5 --------&gt; 3 -------&gt; 4 50</td>
</tr>
<tr>
<td>60</td>
</tr>
</tbody>
</table>

See Weiss Section 9.3.2 for another example.

Similarly the length of the shortest path to each vertex found so far can be recorded as a data member dist.

We also need to record whether we’ve seen this visitor before known

class Vertex
{
    public boolean known;
    // Disttype is probably int or Double
    public DistType dist;
    // preceding vertex on path
    public Vertex path;
    ... // Other fields and methods
}

The last thing we require is a function Weight getWeight(Vertex v, Vertex w) that returns the weight on the edge between v and w.
Shortest Path Pseudocode
Based on Weiss Chapter 9

```plaintext
dijkstraShortestPath(Vertex s)
{
    for each vertex v {
        v.dist = INFINITY
        v.known = false
    }
    s.dist = 0
    newReachables = {s}
    while newReachables is not empty {
        delete from newReachables the v with smallest dist
        v.known = true
        for each vertex w adjacent to v
            if (!w.known) {
                add w to newReachables
                if (v.dist + getWeight(v,w) < w.dist) {
                    w.dist = v.dist + getWeight(v,w)
                    w.path = v
                }
            }
    }
}
```

Walkthrough: Initialisation

```
1  2
0  20
K  20  U
null -----> 1
/ \ / \
50/ \50 /20
/ \ /
v 10 v v 20
5 -----> 3 -----> 4
50  50  INF
U  U  U
1 1  1 null

newReachables = 1
```

Walkthrough: First Iteration

Chose vertex 1

```
1  2
0  20
K  20  U
null -----> 1
/ \ / \
50/ \50 /20
/ \ /
v 10 v v 20
5 -----> 3 -----> 4
50  50  INF
U  U  U
1 1  1 null

newReachables = 2, 3, 5
```

Walkthrough: Second Iteration

Chose vertex 2

```
1  2
0  20
K  20  K
null -----> 1
/ \ / \
50/ \50 /20
/ \ /
v 10 v v 20
5 -----> 3 -----> 4
50  40  INF
U  U  U
1  2  null

newReachables = 3, 5
```
Walkthrough: Final Graph

1  2
0  20
K  20  K
null ----> 1
/ \ / \
50/ \50 /20
/ \ / \
v 10 v v 20
5 ----> 3 ----> 4
50 40 INF
K  K  60
1  2  3

newReachables = {}

Tip: Performing walkthroughs of complex algorithms operating on a simple set of data aids understanding.

Exercise: Complete the walkthrough for the graph above, and check your results with the final graph above.

Exercise: Weiss Exercise 9.5

jgrapht Implementation

public final class DijkstraShortestPath<V, E> {
    /* Constructors ---------------
    
    /* Create & execute a new DijkstraShortestPath algorithm instance. An instance is only good for a single search; after construction, it can be accessed to retrieve information about the path found. */
    public DijkstraShortestPath(Graph<V, E> graph, V startVertex, V endVertex)
    {
        // ... }
Priority Queue Refresher

Used to retrieve items in a priority order. Uses include:

- Sorting
- Task scheduling

Can be implemented as a list or tree.

Example where small numbers have priority:

- Insert 10, 30, 20, 5
- Dequeue:
- Dequeue:
- Insert 15, 40
- Dequeue:
- Dequeue:

Exercise: Rework this exercise assuming large numbers have high priority.

Graph Traversal Reflection

The graph traversal is determined by how the next vertex to visit is selected

- **shortest path**: chose next vertex from a priority queue (priority is shortest length).
- **depth-first search**: chose next vertex from a stack
- **breadth-first search**: chose next vertex from a queue
- **random walk**: chose the next vertex randomly from a set

Summary

- Weighted graphs useful for many problems - each edge has an associated number representing weight/cost/length.
- Easy to implement as NxN array of weights, or by adding a weight to edge objects.
- Example problem: single-source, all-destinations shortest path
- Example algorithm: Dijkstra’s greedy solution.