Prim's algorithm

Table of contents

1. How to program Prim's algorithm for the Minimum Spanning Tree

One of mascopt expected use is the quick implementation of algorithm and more partcularly algorithms on graphs. This note is intended to help you in the realization of this goal by showing you how to manipulate the differents data structures involved. The followed approach is to show how one could program the Prim's algorithm that computes the Minimum Spanning Tree (MST for short) of a given graph. A formulation and a proof of this algorithm can be found in textbooks or even on the internet.

2. Where we go

We are now programming a very simple application that takes a graph, computes its MST taking into account the "length" of the edges and then outputs the resulting graph. All the GUI stuff is left as an exercise to the reader or perhaps as a subject for a forthcoming article in this documentation.

It is to be remarked that with a little bit of work mascopt enables us to implement an application accepting as input either a graph or a digraph and let the user choose the type of value used to compute the MST. This is what the PrimMST class does.

3. At least, the code

It's a java program, so first we need to declare the class (we can't do the imports now, because we don't know exactly what we'll need), let's call it MSTByPrim:

```
public class MSTByPrim
{
```

As the application is very basic, we won't need any global variable and no constructor. The default one is sufficient. So let's code the method that will do most the work (the aim of this article is not to instruct on how to do some beautiful oop but how to use mascopt - even if the latter does not exclude the former, but in this particular case the algorithm is very simple and straightforward).

```
 public Graph computeMST(Graph inputGraph)
    {
```

We are going to follow the Prim's algorithm and construct step by step the graph made of the MST, that is we explore the graph given as an input until we find the right edge and then we add that edge to the set of edges of the MST and its missing extremity to the set of vertices. In order to do that we will need a graph to store the MST and a set of already touched vertices. We introduce those variables and initialize them.

```
   VertexSet touchedVertices = new VertexSet();
   EdgeSet keptEdges = new EdgeSet(touchedVertices);
   Iterator itGN = inputGraph.getVertexSet().iterator();
   if (itGN.hasNext())
       {
      touchedVertices.add((Vertex)itGN.next());
       }
```

It has to be noted that in mascopt, you traverse linearly a set using an Iterator. Two methods of those objects are mainly used, hasnext() which test if there remains an untouched element in the set and next() which extracts the next element of the set. The drawback of this method is the fact that the returned object is of type Object and we are then obliged to explicitely cast it. Advantages include the standardization of the traverse of sets and the fact that if the set is changed, the object of type Iterator is automatically set to null.

```
   while (touchedVertices.size() < inputGraph.getVertexSet().size())
       {
      Iterator itTN = touchedVertices.iterator();
      int minLength = Integer.MAX_VALUE;
      Edge bestCandidate = null;
      Vertex neighborToAdd = null;
```

This is the main loop, we try until the MST we are constructing contains as many vertices as the initial graph. And we initialize working variables. itTN is the iterator used to parse the set of vertices of the MST. minLength is the smallest length of an outgoing edge find so far, bestCandidate is the corresponding edge and neighborToAdd is the extremity of that edge not yet belonging the MST.

```
      while (itTN.hasNext())
          {
         Vertex current = (Vertex)itTN.next();
         VertexSet neighbors = (VertexSet)current.getNeighbors(inputGraph);
         Iterator itN = neighbors.iterator();
```

Here, for every edge of the MST, we test the outgoing edges. In fact we consider the list of neighbors of a vertex which is exactly the same. It is only a little bit shorter in the following as we need to test wether the neighbor is already in the MST.

```
while (itN.hasNext())
             {
            Vertex currentNeighbor = (Vertex)itN.next();
            // we are only interested in outgoing edges
            if (!touchedVertices.contains(currentNeighbor))
                {
               EdgeSet potentialMSTEdges = (EdgeSet)
current.getEdgesTo(inputGraph,currentNeighbor);
               Iterator itpMSTe = potentialMSTEdges.iterator();
               Edge potentialEdge = null;
               // try every edge in the edgeset
```

```
               while (itpMSTe.hasNext())
                   {
                  potentialEdge = (Edge)itpMSTe.next();
                  // if the current edge is the best up to now
                  if (Integer.parseInt(potentialEdge.getValue("length"))<
minLength)
                      {
                     minLength =
Integer.parseInt(potentialEdge.getValue("length"));
                     bestCandidate = potentialEdge;
                     neighborToAdd = currentNeighbor;
                      }
                   }
                }
             }
```

If the neighbor doesn't yet belong to the MST, we consider the edges between the two vertices and in particular its length compared to the length of the already seen edges. This is Prim's algorithm.

```
          }
      touchedVertices.add(neighborToAdd);
      keptEdges.add(bestCandidate);
       }
```

After having explored all the vertices of the soon-to-be MST and all the outgoing edges, we add the one that fits.

```
   return new Graph(touchedVertices,keptEdges);
    }

}
```

Finally we return the graph of the MST.

It is possible to transcribe algorithms in mascopt in a quite simple way. In spite of the intensive use of a great number of variables the program is still quite efficient because we are not creating any variable (a great loss of time in java) but only pointing to already existing variables.

The use of iterators to describe the different sets is quite interesting because it standadize the writing of loops and we don't need to pay any more attention to the size of the different structures.

4. Sample

The above program can be found in Mascopt Dev with a test program samples/algos/MSTByPrimGUI.java using this function and a file containing a graph whose MST can be computed with this program (file samples/algos/MSTGrapheExample.mgl). You

can of course use your own graph, but in order to keep the example rather simple, we only considered connected graphs with edges that have an associated value called "length".

To test it, just do:

```
java MSTByPrimGUI samples/algos/MSTGrapheExample.mgl
```