Minimum-Cost Spanning Tree

```
import GRAPH  -- the ADT 'Graph' from Assignment #3, plus the functions

   -- ugraph sns es : the undirected graph formed from
   --                    all nodes corresponding to
   --                       the list of strings 'sns'
   --                    all edges corresponding to
   --                       the list of ( String, String, Float )-tuples 'es'

   -- putGraph g    : output the graph 'g', in adjacency-list format

   -- start e       : the start node of edge 'e'

   -- finish e      : the finish node of edge 'e'

   -- weight e      : the weight of edge 'e'

--------------------------------------------------------------------------------

-- mcst g : a minimum-cost spanning tree of the
--          connected weighted undirected graph 'g'
--          ( Prim's Algorithm )

mcst :: Graph -> Graph

mcst g = if null ng then
            emptyGraph
         else
            mcst' g ( insertNode n emptyGraph ) ( outEdges n g )
         where
            ng = nodes g
            n  = head ng

--------------------------------------------------------------------------------

-- mcst' g t out : a minimum-cost spanning tree of the
--                 connected weighted undirected graph 'g'
--                 containing the tree 't', where 'out' is a list
--                 of the edges in 'g' leading out of 't'

mcst' :: Graph -> Graph -> [ Edge ] -> Graph

mcst' _ t [ ] = t

mcst' g t out = mcst' g
                      ( insertEdge eMin ( insertNode nMin t ) )
                      ( delEdgesTo nMin out ++ outEdgesNotTo nMin g t )
                      where eMin = minWeightEdge out
                            nMin = finish eMin

--------------------------------------------------------------------------------

-- delEdgesTo n es : the list formed by deleting all edges finishing at node 'n'
--                   from the list 'es'

delEdgesTo :: Node -> [ Edge ] -> [ Edge ]

delEdgesTo n es = [ e | e <- es, finish e /= n ]

--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------

-- outEdgesNotTo n g1 g2 : a list of the outgoing edges from node 'n'
--                         in graph 'g1', not leading to nodes in graph 'g2'

outEdgesNotTo :: Node -> Graph -> Graph -> [ Edge ]

outEdgesNotTo n g1 g2 = [ e | e <- outEdges n g1, notElem ( finish e ) ng2 ]
                        where ng2 = nodes g2

--------------------------------------------------------------------------------

-- minWeightEdge es : an edge with minimum weight in the non-empty list 'es'

minWeightEdge :: [ Edge ] -> Edge

minWeightEdge ( e : es ) = foldr ( \e1 -> \e2 ->
                                     if weight e1 < weight e2 then e1 else e2 )
                                  e
                                  es

--------------------------------------------------------------------------------
```

Minimum-Cost Spanning Tree

```
g1 = emptyGraph
```

```
g2 = ugraph [ "A" ]
            [ ]
```

```
g3 = ugraph [ "A", "B", "C" ]
            [ ( "A", "B", 3.0 ), ( "A", "C", 2.0 ), ( "B", "C", 1.0 ) ]
```

```
g4 = ugraph [ "A", "B", "C", "D", "E", "F", "G", "H", "I" ]
            [ ( "A", "B", 2.0 ), ( "A", "F", 7.0 ), ( "A", "G", 3.0 ),
              ( "B", "C", 4.0 ), ( "B", "G", 6.0 ),
              ( "C", "D", 2.0 ), ( "C", "H", 2.0 ),
              ( "D", "E", 1.0 ), ( "D", "H", 8.0 ),
              ( "E", "F", 6.0 ), ( "E", "I", 2.0 ),
              ( "F", "I", 5.0 ),
              ( "G", "H", 3.0 ), ( "G", "I", 1.0 ),
              ( "H", "I", 4.0 ) ]
```

```
> putGraph g1

> putGraph ( mcst g1 )
```

```
> putGraph g2
A  -->

> putGraph ( mcst g2 )
A  -->
```

```
> putGraph g3
A  -->  B:3.0  C:2.0
B  -->  A:3.0  C:1.0
C  -->  A:2.0  B:1.0

> putGraph ( mcst g3 )
B  -->
C  -->  B:1.0
A  -->  C:2.0
```

```
> putGraph g4
A  -->  B:2.0  F:7.0  G:3.0
B  -->  A:2.0  C:4.0  G:6.0
C  -->  B:4.0  D:2.0  H:2.0
D  -->  C:2.0  E:1.0  H:8.0
E  -->  D:1.0  F:6.0  I:2.0
F  -->  A:7.0  E:6.0  I:5.0
G  -->  A:3.0  B:6.0  H:3.0  I:1.0
H  -->  C:2.0  D:8.0  G:3.0  I:4.0
I  -->  E:2.0  F:5.0  G:1.0  H:4.0

> putGraph ( mcst g4 )
F  -->
H  -->
C  -->  H:2.0
D  -->  C:2.0
E  -->  D:1.0
I  -->  F:5.0  E:2.0
G  -->  I:1.0
B  -->
A  -->  G:3.0  B:2.0
```