```
data Courses = Maths | Theory | Languages | Programming
             | Concurrency | Architecture | Parallelism
     deriving (Eq,Ord,Enum,Ix,Show)

cg = mkGraph True (Maths,Parallelism)
       [(Maths,Theory,1),(Languages,Theory,1),
        (Programming,Languages,1),(Programming,Concurrency,1),
        (Concurrency,Parallelism,1),(Architecture,Parallelism,1)]
```

A topological sort of this graph then returns the following result:

```
topologicalSort cg
  ⇒  [Architecture, Programming, Concurrency, Parallelism,
      Languages, Maths, Theory]
```

which is also a topological sort different from the one we gave before, but finishing with Maths and Theory is surely a much less pedagogical approach!

7.5 Minimum spanning tree

Given a connected, weighted undirected graph $G = \langle V, E\rangle$, a spanning tree is a subgraph of G, $G' = \langle V, E'\rangle$ such that G' is acyclic and connected. In other words, G' is a tree that 'covers' all the nodes in the graph. For example, the graph in Figure 7.4 has several spanning trees, three of them depicted below the original graph. Note that the number of edges in a spanning tree is always equal to $|V - 1|$.

The cost of a spanning tree is the sum of the weights of the edges that constitute it. In the previous example, the costs of the three spanning trees is 194, 185 and 133 respectively. The minimum spanning tree problem is to find the spanning tree with the minimum cost. This graph problem has several practical applications such as building a computer network with minimal cost or designing a telephone exchange for n locations.

7.5.1 Kruskal's algorithm

The first algorithm described is called Kruskal's algorithm. Starting with an empty tree, it works by successively adding the lowest cost edge to this tree providing that no cycles are created. An example of how the algorithm works is illustrated in Figure 7.5.

However, this algorithm could be extremely expensive to implement if we have to test a graph continuously for cycles. For this reason, it is implemented as follows:

1. Initially, there is a table containing one entry per vertex, each vertex being allocated an arbitrary distinct number (we could give every vertex its own value to start with).

2. All edges are placed in a priority queue, ordered according to the weight of the edge.

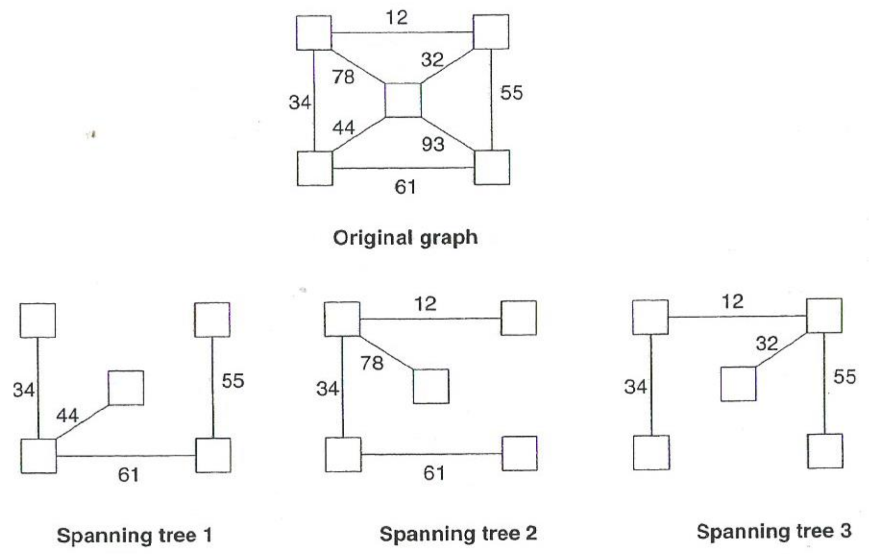

Figure 7.4 Examples of spanning trees.

3. Add the edge at the front of the priority queue (that is, with the lowest cost) until $|V| - 1$ edges have been added using the following algorithm:
   (a) if both vertices that constitute this edge have different numbers in the table, this edge is added to the solution and both numbers in the table are replaced by the same number (for example, the minimum of both of them);
   (b) otherwise the edge is rejected and the table is left untouched.

Step 3 of this algorithm uses the function `unionFind` which, given a pair of vertices and a table, returns a pair consisting of a boolean and the updated table, the boolean indicates whether the table has been updated according to the rules stated above. This function, which uses the table ADT (see Section 5.6), is defined as:

```
unionFind :: (Eq n, Ord w) => (n,n)->Table w n->(Bool,Table w n)
unionFind (x,y) t
  = let xv = findTable t x
        yv = findTable t y
    in  if (xv == yv)
        then (False,t)
        else (True,updTable (if yv<xv then (x,yv) else (y,xv)) t)
```

Now here is an implementation of Kruskal's algorithm. The function `fillPQ` places edges in the priority queue in the order `(weight,v,w)` so that the tuples are sorted according to weight.