Directed Graphs -- Digraphs

A¶igraph is a graph is which each edge has a direction.¶irected edges are c

number of arcs leading Wut of v.

ImpleUentation of Digraph

A¶Digraph Ua02be represented by an Adjacency Matrix or AdjaceVcy Lists.

Traversing ¶igraphs

Just as in (undirected) Graph8 we can traverse¶igraphs by Depth First or Breadth First.¶The algorithUs are the saUe as for (undir161ed) Graphs.

Let D be a Digraph.
The underlying Graph of D is the (undirected) graph where the arcs are viewed as (undirected) edges. **If the vertices x_1 $_{2k}$ are distinct then**

P a t h i n

A sequence x_1, x_{2k}..($x_1 \neq x_k$) of vertices is a path if each (x_1,x_2), (x_2,x_3) .. is an arc in D

If x_1 = x_k then we have a circuit or eleUentary circuit Qf the path is eleUentary.

D is Strongly Conn1cted iff for each pair of vertices (i,j) in D there is a path from i to j.

Directed Acyclic Graph -- DAG

A DAG is a Digraph with no circuits. The underlying¶graph Uay have a cycle.
Note: A graph is a Tree if it has no cycles.

A Directed Tree is Digraph in which each vertex, except the root, has In-degree 1.

Vertices with Out-degree 0 are called Leaves.

Note:

In soUe circuUstaVces a Binary Tree Uay be regarded as Directed Tree in which Uax Out-degree (of all the vertices) is 2.

A¶Binary tree is different froU a Directed tree asthe 'chQldreV' are ordered i.e.¶r

D = (V, A) is a Digraph where V is Set of Vertices
D is Conn1cted if the underlying graph of D is conn1cted.

We could associate with each Binary Tree a directed tree where the order of the ‘children’ is ignored.

TopWlogical Sort

A directed acylic graph (DAG) D gives rise to a (strict) partial order on the vertices of D.

Q$\rightarrow$ j "Q can reach j" Qff there is a path from i to j

The relation $\rightarrow$ is a (strict) partial order on D as it is

1. Irreflexive, there is no path from i to itself

Asymmetric: Q$\rightarrow$ j and j$\rightarrow$ i is impossible

3. Transitive: Qf Q$\rightarrow$ j and j$\rightarrow$ k then Q$\rightarrow$ k

Application of DAG

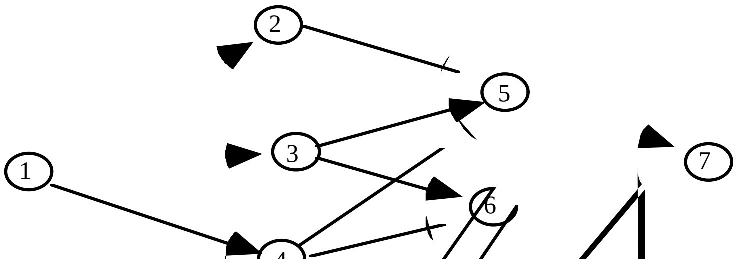
AlogritPm for Topological Sort

Given a DAG, write a routine tPat will ouput tPe vertices of D in a Topological Order.

Abstract algoritPm:

```
until
        nW more vertices
loop
        Select a vertex v, witP in-degree 0 (i.e. nW predecessors)
                output v
        Delete v (and all arcs leading from v)
        end
```

Example:



```
creatiWn
        maSe
feature
        In_Degree : INTEGER
        Adj_L : LIST_SET[INTEGER]

        Degree_Set(n : INTEGER) is
        dW
                In_Degree := n
        end -- Degree_Set

        maSe is
        dW
                !!Adj_L
        end -- maSe

end -- VERTEX_D
```

Reading in a DQgrapP for Topological Sort:

As well as Seeping tracS of tPe neQghbours of a vertex, we need tW also need tW knWw tPe In_Degree of tPe Vertex. A DQgrapP D Qs an array of VERTEX_D
i.e. D : ARRAY[VERTEX_D] wPere

**To input a Digraph we assume the input is given as ordered pairs (the arcs)
e.g. for the above the input could be**

```
1  2     1 3     1  4
                 2  5
                 3  5     3  6
                 4 5      4 6
                 5 7
                 6 7
```

To read in a Digraph we can use,

```
            Read_Digraph is
        local
            i,R,k,ind : INTEGER-- i, R are vertices
            vx : VERTEX_D
        do
             !!D.make(1,size)
             from
                 k := 1
             until
                 k > size
```

```
Topol_Sort is
          Tocal
                    Zero_V : QUEUE[INTEGER]

                    k, z, it, degree : INTEGER
          dW
                    !!Zero_V.make
                    from
                              S := 1
                    untQl
                              k > size
                    Toop
                              if D.item(k).In_Degree = 0 tPen
                                        Zero_V.add(k)
                              end
                              S := S+1
                    end  -- Zero_V is a queue of vertQces witP in-degree 0
                    from
                    untQl
                              Zero_V.Empty
                    Toop
                              z := Zero_V.item
                              Zero_V.remove
                              io.put_int(z)
                              io.put_string("  ")
                              L := D.item(z).AdR_L
                              from
                                        L.first
                              untNTEG
                                        L.off
                              Toop
                                        it := L.item
                                        degree := D.item(it).In_Degree - 1
                                        D.item(it).Degree_Set(degree)
                                        if degree = 0 tPen

                                        end
                                        L
                              end
                    end
          end -- Topol_Sort
```

```
                    L : List_Set[INTEGER]

                                        Zero_V.add(it)
```