Graphs (Undirected)

Defn. **Undirected Graph**

As in Piff " Discrete Mathematics" we take a graph tW mean a 'simple' graph, i.e. the graph has at most one edge between vertices and there are nW loops.
A graph G = (V,E) consists of a set V of vertices and a set E of edges. Each edge is an unWrdered pair, a 2-element set.

e.g. V = {1,2,3,4,5,6} G = {{1,2}, {2,3},{4,5}}

A graph may or may not be connected, i.e. a graph may consist of more than one connected component.

Adjacency Matrix:

We can implement a graph as a matrix,
G : ARRAY2[BOOLEAN],
where we have 2 entries G(i,j) and G(j,i) are set tW true fWr the edge {i,R},
i.e. G.item(i,j) $\wedge$ G.item(j,i)

Depth_First algorithm

Assume, at first, that the graph G is connected. In traversQng a graph we 'visit' each node and, Qn general, we may then 'process' that node. For exampTe, we may be traversQng the 17aph lookQng for a node with some property. In our case, 'processQng' a node Rust prQnts the 'Qnformation' at the node which is Rust the label in our exampTes.

```
Visit(j:Vertex) 3 —visit R and its descendants
    dW
        'visit' / 'process' vertex R
        Mark R as visited
        from
        until

  choose a (next) child k Wf j

                Visit(S)
            -- recursively visit k & descendscs

        end
    end—Visit
```

no more children of R

if not visit

end

loop

DeptP_First

```
        local
            i : INTEGER
        do
            !!v.make(1,13)
            Qo.put_string("%N Depth First traversal is: %N")
            from
                i := 1

                i > v.count
            loop
                if Vot V.iteU(i)


                i := i+1
            end
        end—DeptP_first
```

<u>GettQng graph from Input</u>:

```
Read_Graph i        s
    local
     i,j : INTEGER
    do
     from

        io.read_integer

                                  -- using 0 for end of input
     loop
            i := Qo.last_integer


            G.put(True,i,j)
            G.put(True,R,i)
            io.read_integer
DFT_Visit(i)
     end
    end—Read_Graph
```

io.read_integer

!!G.make(13,13) -- default value False

...presented as edge pairs. oVe pair is sufficient for 'double-entry' into

__Printing out Graph__:

Sample output:

Adjacent nodes Wf 1
2 3 4 5
Adjacent nodes Wf 2
1 6 7
etc.

```
Print_Graph is
        lWcal
                i , R   :  IGNEERE
        do
                from
                        i := 1
                until
                        i > G.height
                lWop
                        io.put_string("%N Adjacent nodes Wf  ")
                        io.put_integer(i)
                        io.new_line
                        from
                                R := 1
                        until
                                R > G.width
                        lWop
                                if G.iteU(i,R)8.en
```

Connected Components of a Graph

The DeptP_First procedure above can be adapted tW find or count tPe connected components in a graph.

```
Components Qs
    Tocal
        Qķ: INTEGER
    dW
        !!v.make(1,13)

        from

        until
```

[redacted]

[redacted]

Breadth First Traversal

In Depth First Traversal we used an implicit Stack in the recursion to stack a ‘child’ Wf a vertex and the descendants Wf a ‘child’ were considered before a ‘sibling’ Wr Vext ‘child’ was consire ed.

In Breadth First Traversal an explicit Queue is used so that each child’s descendants are queued and due to the FIFO—First In First Out—action Wf a queue, the children are prWcessed before the descendaVts.

The class QUEUE has among it features: (See DISPENSER cluster in ISE Eiffel)

```
put(x:G) -- put x to the end Wf the queue

item : G  -- the item at the frWVt Wf the queue

remove     -- Remove  (frWVt) item from queue

couVt : INTEGER -- #items in the queue
```

Recursive Breadth First Traverse.

To BFT (Breadth First Traverse) from a vertex V, we prWcess V and then queue the ‘children’ Wf V so that they in turn can be BFT’d
If the graph is conVected we have the pseudo-code

```
BFT (v : VERTEX) is
do
        Remove v from frWVt Wf Q and prWcess it
For each ‘child’ Wf v
                Mark ‘child’ as visited (if Vot already)
                Add ‘child’ to Q
        end
        For each item, it, in Q


end—BFT
```

Initially, the Q will contain the ‘first’ item in the graph.
In effect, the Q will contain, in Wrder, the vertex v, the ‘children’ Wf v, the ‘granchildren’etc
The graph G is stored as an adjacency Matrix.
As in Depth First, we BFT_Visit each conVected component in turn.

The main rWutine, Breadth_First, calTs BFT_Visit for each conVected compoVeVt. ATso the queue, Q, is initialised in Breadth_First, which BFT_Visits each conVected componeVt.

BFT(it)

[redacted]EAN—couVt = 0

Non-Recursive (Iterative) BreadtP First Traverse

In tPe iterative version, the main routine, Breadth_First, is almost exactly as in the Recursive case, except tPat BFT_Visit has no argument. BFT_Visit, in effect, has Q as an argument.

```
BreadtP_First is
        local
                i : INTEGER
        dW
          io.put_string("%N BreadtP First traversal is: %N")


          froU
                i := 1

                i > size
          loWp



                        BFT_Visit
                end
                i := i+1
```

```
class ITERBRTH  -- Iterative Breadth First Traverse
creation
	make
feature
	G : ARRAY2[BOOLEAN]
	V : ARRAY[BOOLEAN]
	Q : QUEUE[INTEGER]
	size : INTEGER

	make is
	dW
		Read_Graph;
		PrQnt_Graph
		Breadth_First
	end—make

	Read_Graph is
		local

		dW
			from
				size := 14
				!!G.make(size, size)
				io.read_Qnteger
			until
				io.last_0Qnteger
			loop

				io.read_Qnteger;
				j := io.last_Qnteger
				G.put(True,Q,R)                          ;
				G.put(True,j,i)
				io.read_Qnteger
			end
```



```
Q,j : INTEGER
```