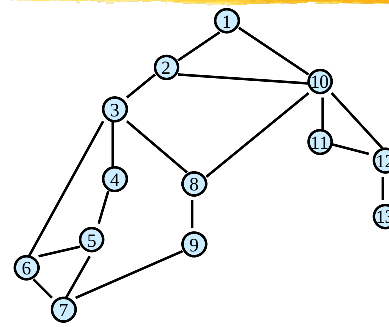
CSE 417: Algorithms and Computational Complexity

Winter 2002

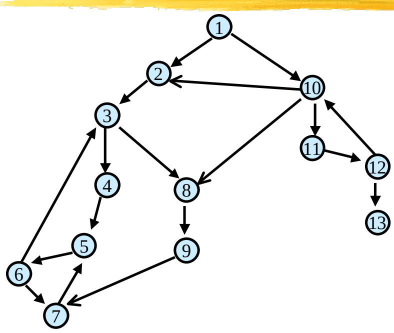
Graphs and Graph Algorithms

Larry Ruzzo

Undirected Graph G = (V,E)



Directed Graph G = (V,E)

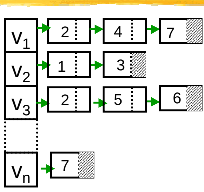


Representing Graph G=(V,E) n vertices, m edges

- Vertex set $V=\{v_1,\ldots v_n\}$
- Adjacency Matrix A
  - $A[i,j]=1$ iff $(v_i,v_j) \in E$
  - Space is n^2 bits
- Advantages:
  - O(1) test for presence or absence of edges.
  - compact in packed binary form for large m
- Disadvantages: inefficient for sparse graphs

Representing Graph G=(V,E) n vertices, m edges

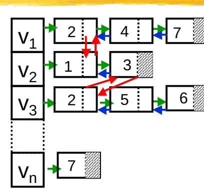
- Adjacency List:
  - O(n+m) words
  - O(log n) bits each



- Advantages:
  - Compact for sparse graphs

Representing Graph G=(V,E) n vertices, medges

- Adjacency List:
  - O(n+m) words
  - O(log n) bits each



- Back- and cross pointers more work to build, but allow easier traversal and deletion of edges
  - usually assume this format

Graph Traversal

- Learn the basic structure of a graph
- Walk from a fixed starting vertex v to find all vertices reachable from v
- Three states of vertices
  - **undiscovered**
  - **discovered**
  - **fully-explored**

Breadth-First Search

- Completely explore the vertices in order of their distance from v
- Naturally implemented using a queue

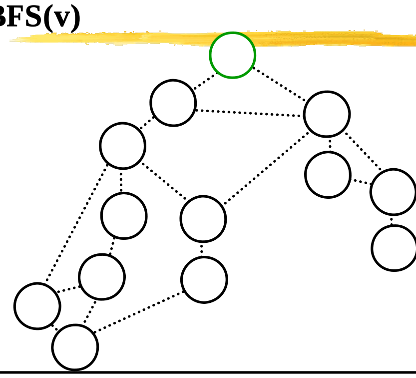
BFS(v)

Global initialization: mark all vertices "undiscovered"

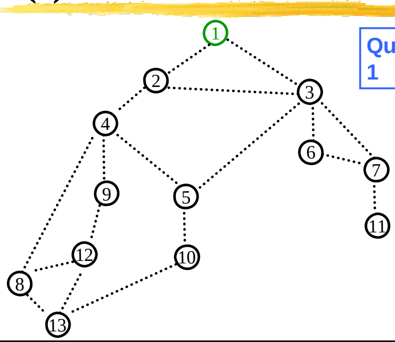
```
BFS(v)
    mark  v "discovered"
    queue = v
    while queue not empty
        u = remove_first(queue)
        for each edge {u,x}
            if (x is undiscovered)
                mark x discovered
                append x on queue
        mark u completed
```

Exercise: modify code to number vertices & compute level numbers

BFS(v)

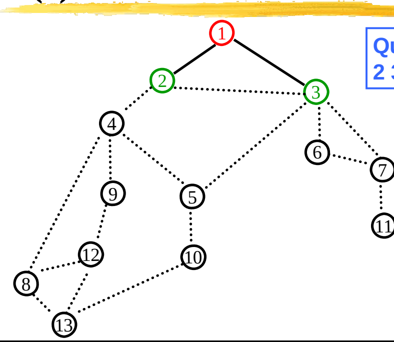


BFS(v)



Queue:
1

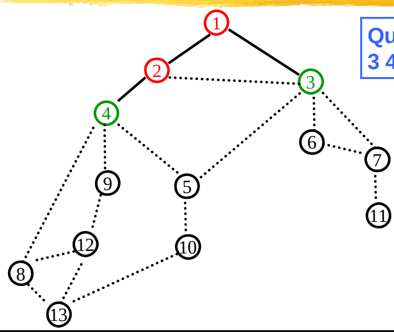
BFS(v)



Queue:
2 3

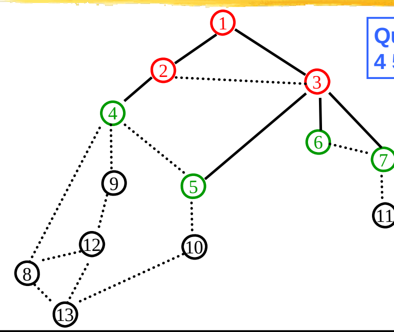
BFS(v)

Queue:
3 4



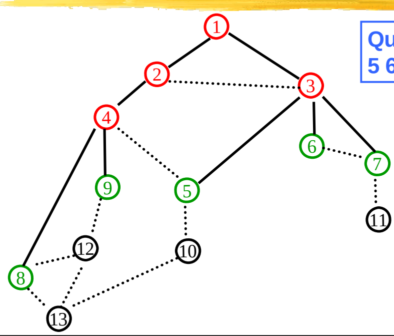
BFS(v)

Queue:
4 5 6 7



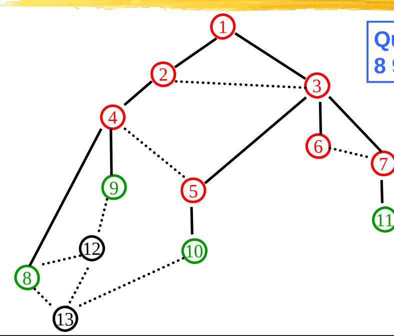
BFS(v)

Queue:
5 6 7 8 9



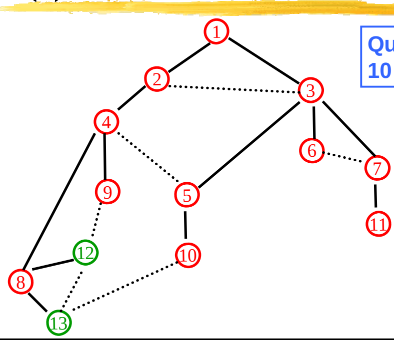
BFS(v)

Queue:
8 9 10 11



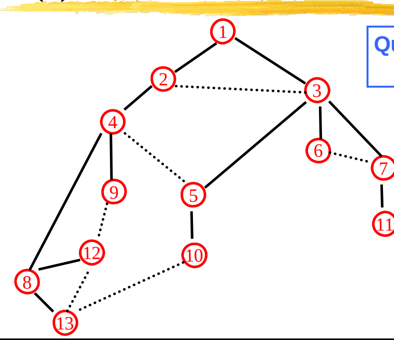
BFS(v)

Queue:
10 11 12 13



BFS(v)

Queue:



BFS analysis

- Each edge is explored once from each end-point (at most)
- Each vertex is discovered by following a different edge
- Total cost O(m) where m=# of edges

Properties of (Undirected) BFS(v)

- BFS(v) visits x if and only if there is a path in G from v to x.
- Edges into then-undiscovered vertices define a ***tree*** – the "breadth first spanning tree" of G
- Level i in this tree are exactly those vertices u such that the shortest path (in G, not just the tree) from the root v is of length i.
- ***All*** non-tree edges join vertices on the same or adjacent levels

Graph Search Application: Connected Components

- Want to answer questions of the form:
  - given vertices u and v, is there a path from u to v?
- Idea: create array A such that
  A[u] = smallest numbered vertex that is connected to u
- question reduces to whether A[u]=A[v]?

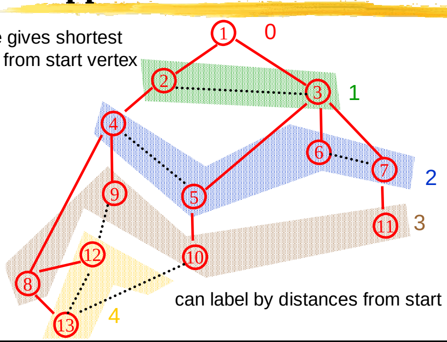
Q: Why not create 2-d array Path[u,v]?

Graph Search Application: Connected Components

- initial state: all v undiscovered
  for v=1 **to** n **do**
  if state(v)!=fully-explored **then**
  BFS(v): **setting** A[u] ←v **for each** u **found**
  (and marking u discovered/fully-explored)
  endif
  endfor
- Total cost: O(n+m)
  - each vertex an each edge is touched a constant number of times
  - works also with DFS

BFS Application: Shortest Paths

Tree gives shortest paths from start vertex



can label by distances from start

Depth-First Search

- Follow the first path you find as far as you can go
- Back up to last unexplored edge when you reach a dead end, then go as far you can
- Naturally implemented using recursive calls or a stack

DFS(v)

Global Initialization: mark all vertices "undiscovered"

DFS(v)
    mark v "discovered"
    stack = v
    while stack not empty
        u = pop(stack)
        for each edge {u,x}
            if (x is undiscovered)
                mark x discovered
                push x
        mark u completed

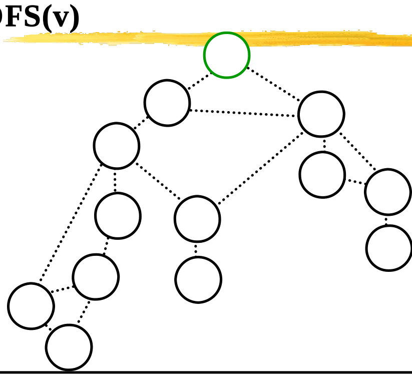
> Exercise1: recode recursively
>
> Exercise 2: modify to compute vertex numbering

DFS(v) – Recursive version

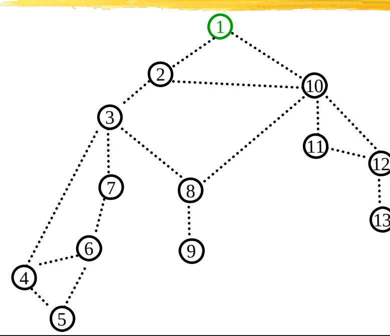
Global Initialization:
    mark all vertices v "undiscovered" via v.dfs# = -1
    dfscounter = 0

DFS(v)
    v.dfs# = dfscounter++   // mark v "discovered"
    for each edge (v,x)
        if (x.dfs# = -1)   // tree edge (x previously undiscovered)
            DFS(x)
        else …   // code for back-, fwd-, parent,
                                        // edges, if needed
    // mark v "completed," if needed

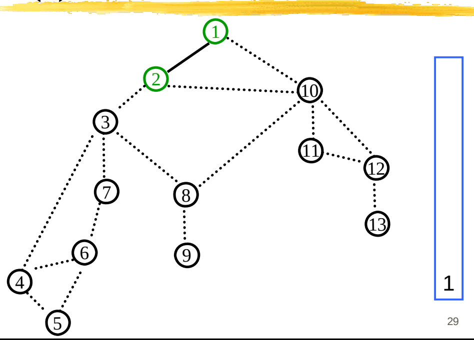
DFS(v)



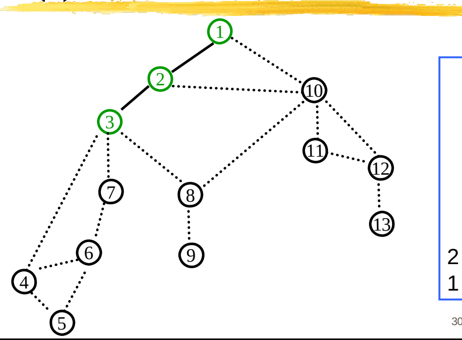
DFS(v)



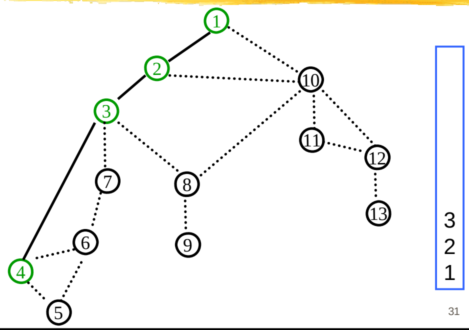
DFS(v)



DFS(v)

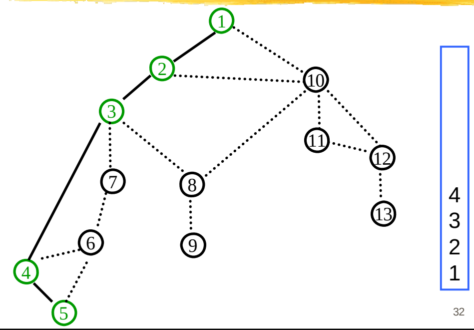


DFS(v)



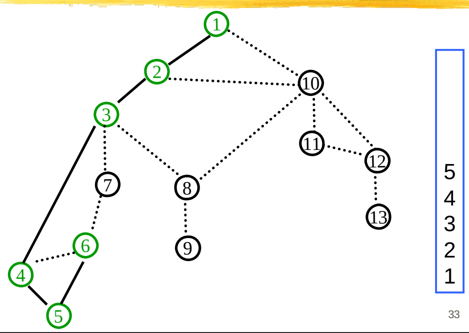
31

DFS(v)



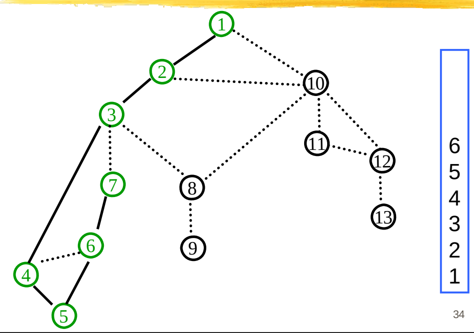
32

DFS(v)



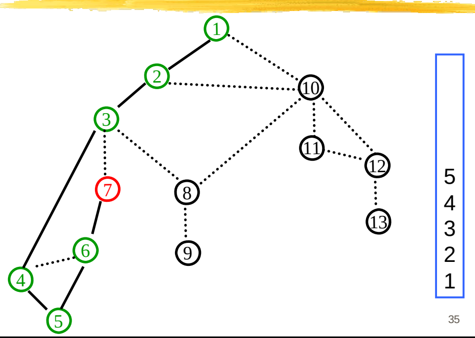
33

DFS(v)



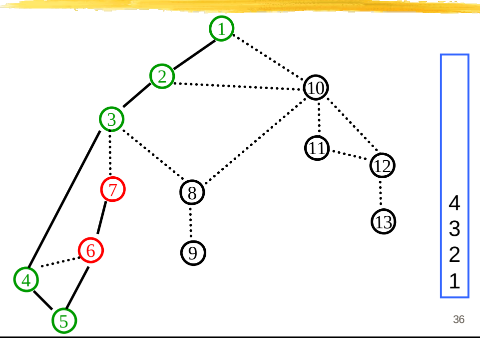
34

DFS(v)



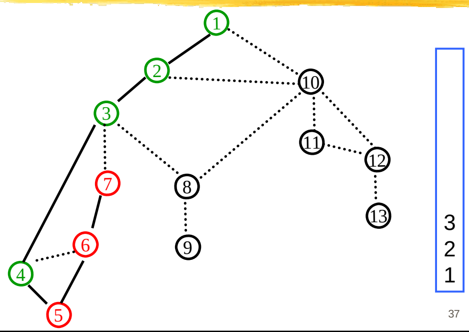
35

DFS(v)



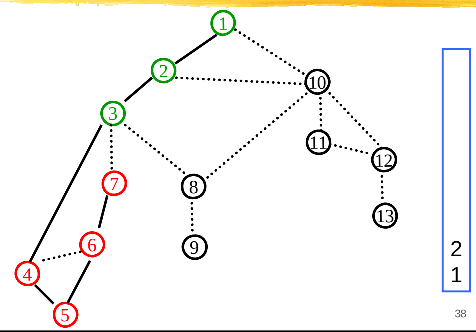
36

DFS(v)



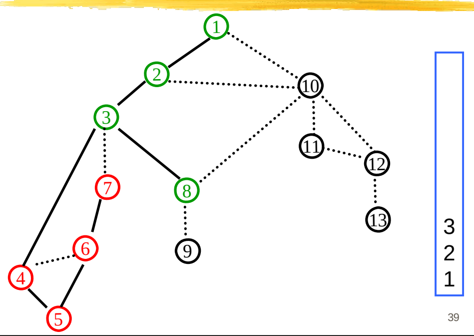
37

DFS(v)



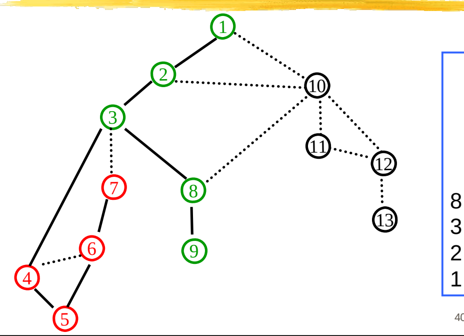
38

DFS(v)



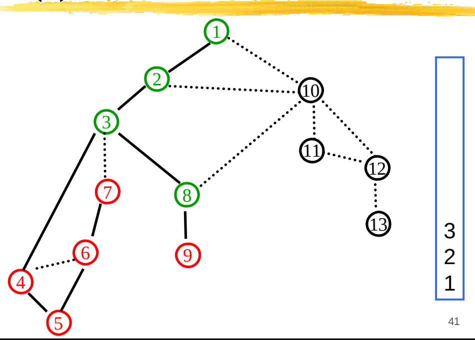
39

DFS(v)



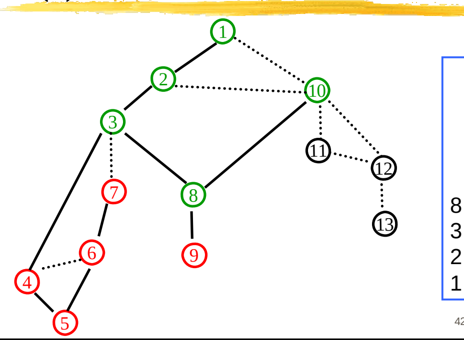
40

DFS(v)



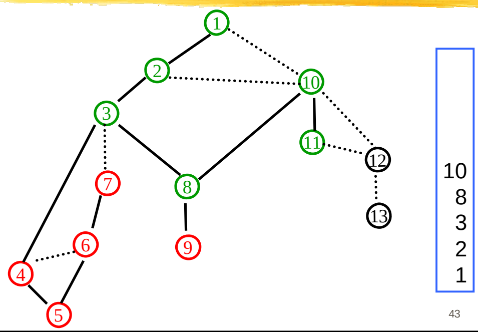
41

DFS(v)



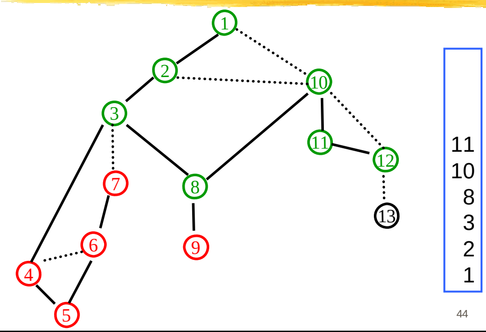
42

DFS(v)



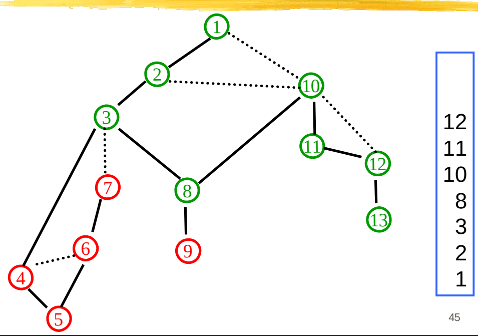
43

DFS(v)



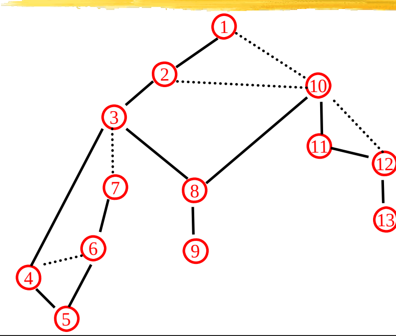
44

DFS(v)



45

DFS(v)



46

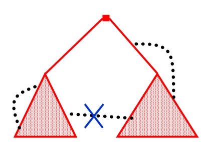
Properties of (Undirected) DFS(v)

- Like BFS(v):
  - DFS(v) visits x if and only if there is a path in G from v to x (through previously unvisited vertices)
  - Edges into then-undiscovered vertices define a ***tree*** – the "depth first spanning tree" of G
- Unlike the BFS tree:
  - the DF spanning tree isn't minimum depth
  - its levels don't reflect min distance from the root
  - non-tree edges never join vertices on the same or adjacent levels
- BUT…

47

Non-tree edges

- All non-tree edges join a vertex and one of its descendents/ancestors in the DFS tree
- No cross edges!



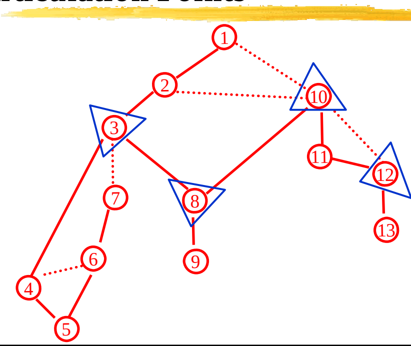
48

Application: Articulation Points

- A node in an undirected graph is an **articulation point** iff removing it disconnects the graph
- articulation points represent vulnerabilities in a network – single points whose failure would split the network into 2 or more disconnected components

49

Articulation Points



50

Brainstorming

- draw a graph, ~ 10 nodes, A-J
- redraw as via DFS
- add dsf#s & tree/back edges (solid/dashed)
- find cycles
- give alg to find cycles via dfs; does G have any?

- find articulation points
- what do cycles have to do with articulation points?
- alg to find articulation points via DFS???

51

Articulation Points from DFS

- Every interior vertex of a tree is an articulation point
  - Non-tree edges eliminate articulation points
- Root node is an articulation point iff it has more than one child

non-leaf, non-root node u is an articulation point ⟺ no non-tree edge goes above u from a sub-tree below some child of u

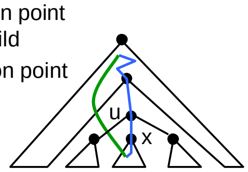
52

Articulation Points from DFS

- Root node is an articulation point iff it has more than one child
- Leaf is never an articulation point
- non-leaf, non-root node u is an articulation point

⇕

no non-tree edge goes above u from a sub-tree below some child of u



If removal of u does NOT separate x, there must be an exit from x's subtree. How? Via back edge.

53

Articulation Points: the "LOW" function

- Definition: LOW(v) is the lowest dfs# of any vertex that is either in the dfs subtree rooted at v (including v itself) or connected to a vertex in that subtree by a back edge.
- Key idea 1: if some child x of v has LOW(x) ≥ dfs#(v) then v is an articulation point.
- Key idea 2: LOW(v) =
  min ({LOW(w) | w a child of v } ∪
  { dfs#(x) | {v,x} is a back edge from v })

54

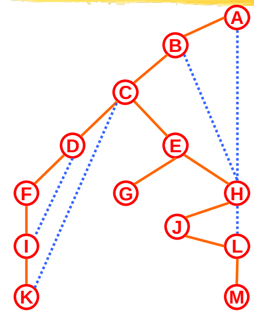
DFS(v) for Finding Articulation Points

```
Global initialization: v.dfs# = -1 for all v.
DFS(v)
 v.dfs# = dfscounter++
 v.low = v.dfs#              // initialization
 for each edge {v,x}
    if (x.dfs# == -1)        // x is undiscovered
       DFS(x)
       v.low = min(v.low, x.low)
       if (x.low >= v.dfs#)
          print "v is art. pt., separating x"
    else if (x is not v's parent)
       v.low = min(v.low, x.dfs#)
```

Except for root. Why?

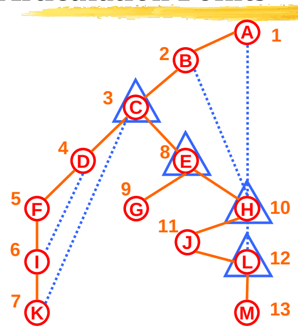
Equiv: "if({v,x} is a back edge)" Why?

Articulation Points



Vertex	DFS #	Low
A		
B		
C		
D		
E		
F		
G		
H		
I		
J		
K		
L		
M		

Articulation Points



Vertex	DFS #	Low
A	1	1
B	2	1
C	3	1
D	4	3
E	8	1
F	5	3
G	9	9
H	10	1
I	6	3
J	11	10
K	7	3
L	12	10
M	13	13