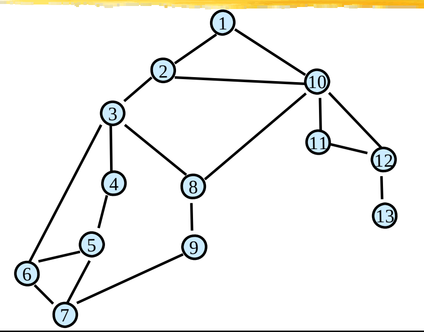
CSE 417: Algorithms and Computational Complexity

Winter 2001

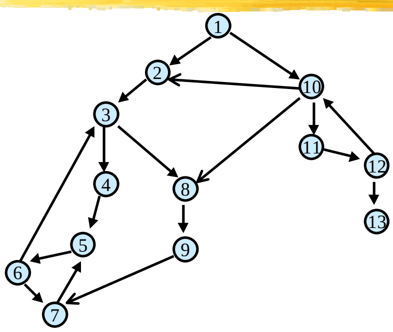
Lecture 9

Instructor: Paul Beame

Undirected Graphs



Directed Graphs

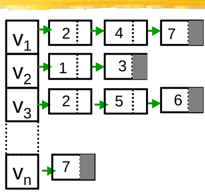


Representing Graph G=(V,E) n vertices, m edges

- 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

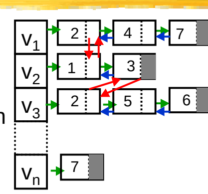
- Adjacency List:
  - $O(n+m)$ words
  - $O(\log n)$ bits each



- Advantages:
  - Compact for sparse graphs

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

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



- Back pointers and cross pointers 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**
  - **completely-explored**

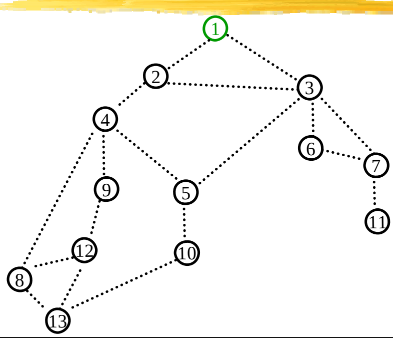
7

Breadth-First Search

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

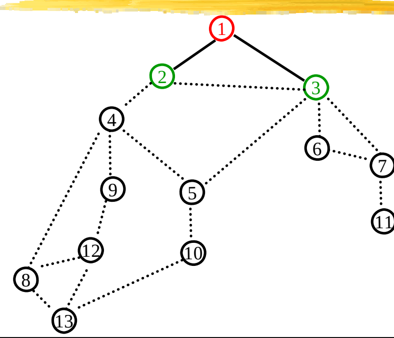
8

BFS(v)



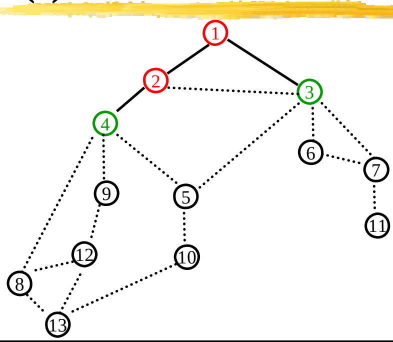
9

BFS(v)



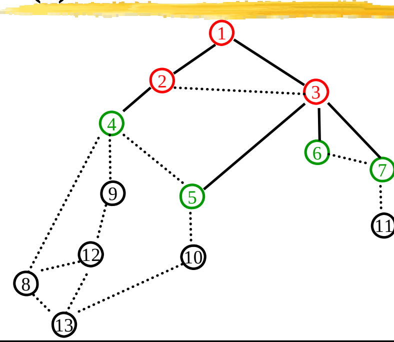
10

BFS(v)



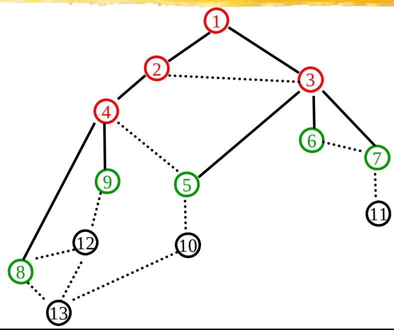
11

BFS(v)

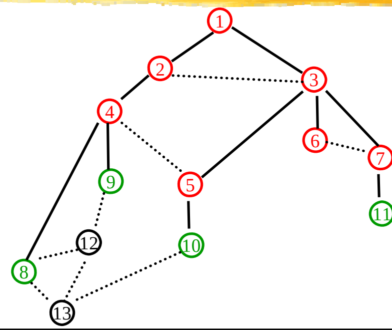


12

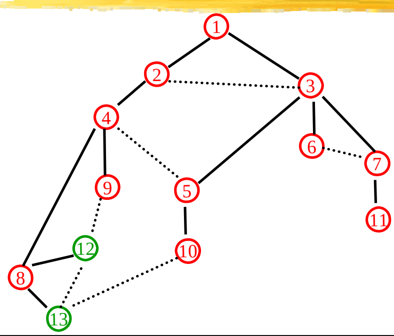
BFS(v)



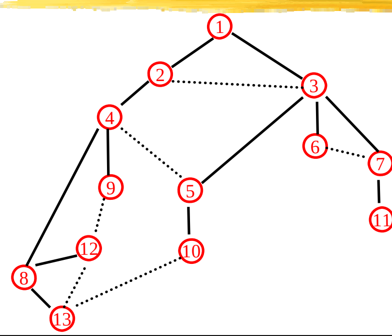
BFS(v)



BFS(v)



BFS(v)



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

Graph Search Application: Connected Components

- Want data structure that allows one 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]?

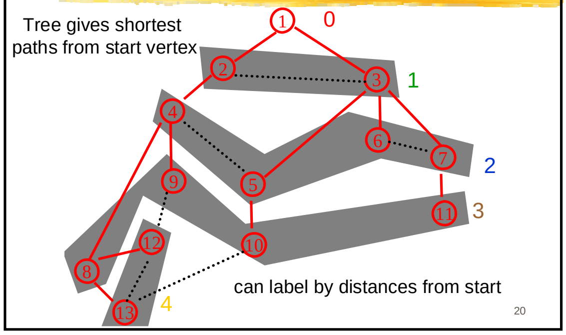
Graph Search Application: Connected Components

- **for** v=1 **to** n **do**
  - **if** state(v)!=fully-explored **then**
    - state(v)←discovered
    - BFS(v): **setting** A[u] ←v **for each** u **found**
  - **endif**
- **endfor**

- Total cost: O(n+m)
  - each vertex an each edge is touched a constant number of times
  - works also with DFS

19

BFS Application: Shortest Paths



20

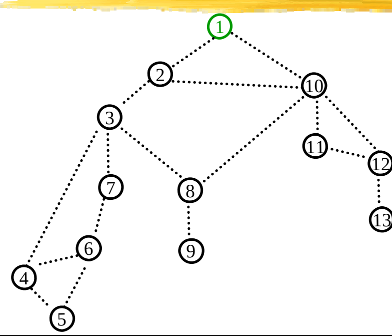
Depth-First Search

- Follow the first path you find as far as you can go, recording all the vertices you will need to explore further as you go.

- Naturally implemented using recursive calls or a stack

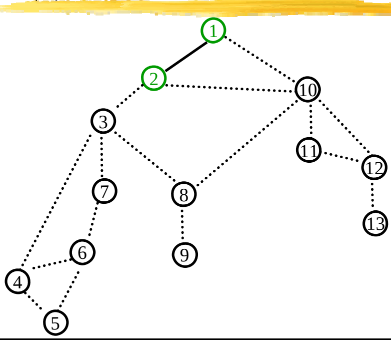
21

DFS(v)



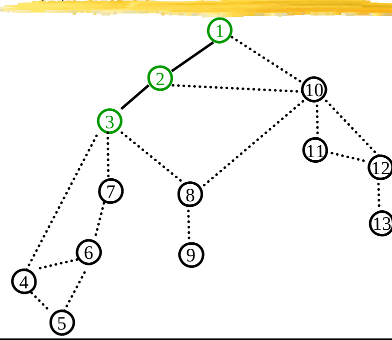
22

DFS(v)



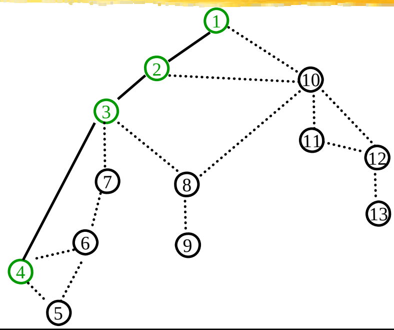
23

DFS(v)



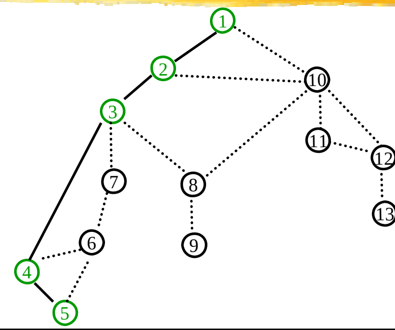
24

DFS(v)



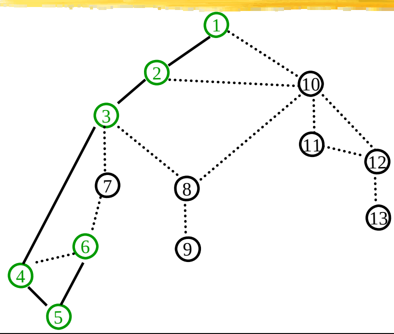
25

DFS(v)



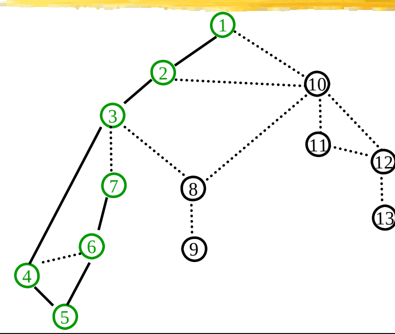
26

DFS(v)



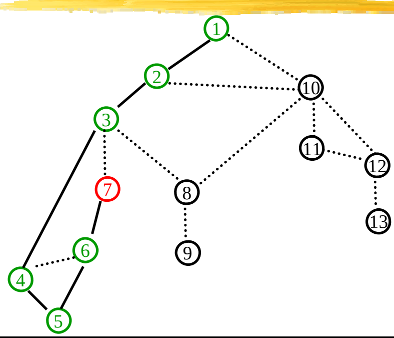
27

DFS(v)



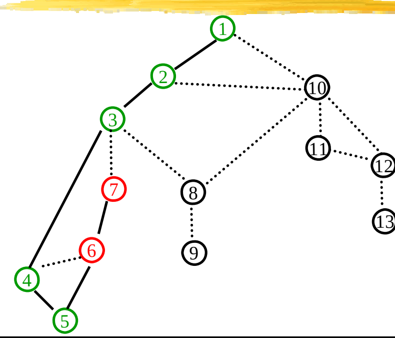
28

DFS(v)



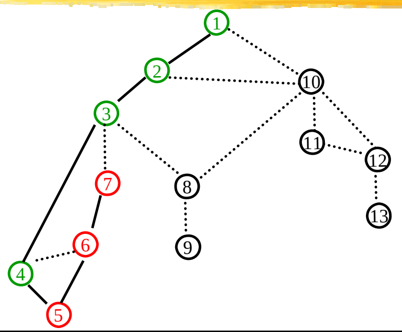
29

DFS(v)



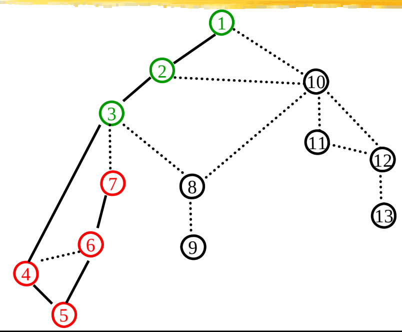
30

DFS(v)



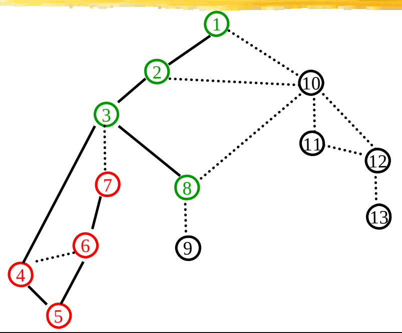
31

DFS(v)



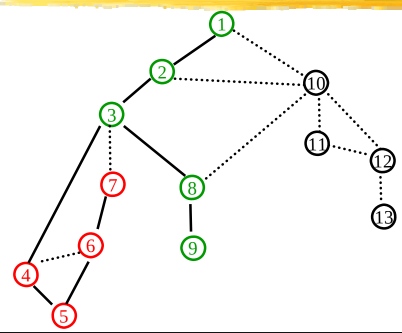
32

DFS(v)



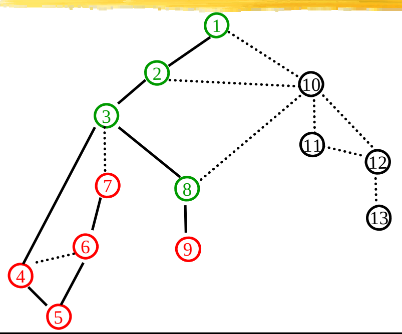
33

DFS(v)



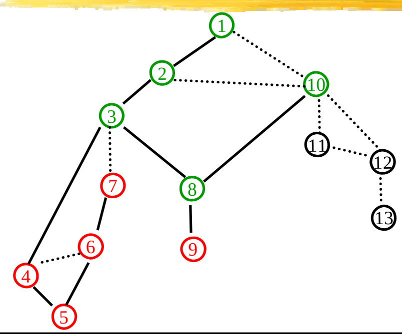
34

DFS(v)



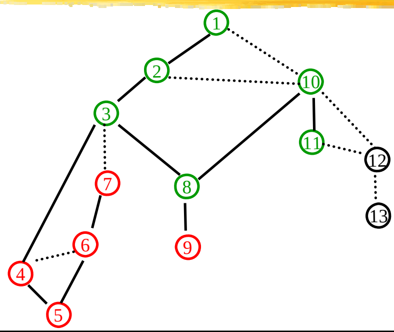
35

DFS(v)



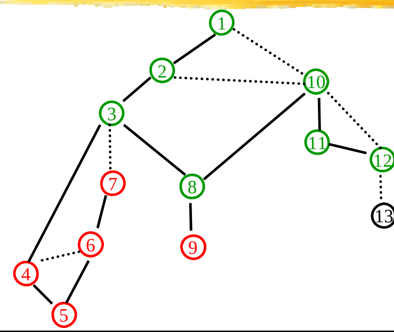
36

DFS(v)



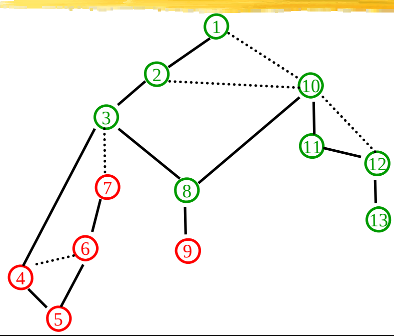
37

DFS(v)



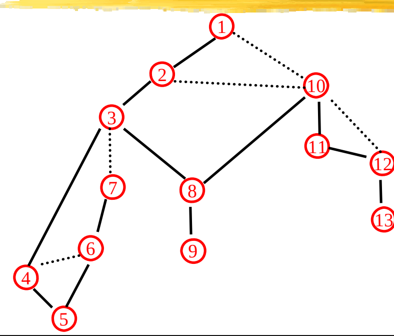
38

DFS(v)



39

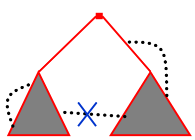
DFS(v)



40

Non-tree edges

- All non-tree edges join a vertex and its descendent in the DFS tree
- No cross edges



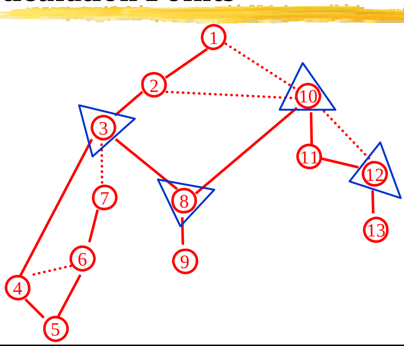
41

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

42

Articulation Points



43

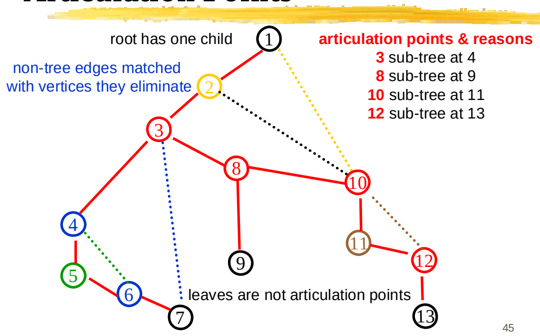
Articulation Points from DFS

- Every interior vertex of a tree is an articulation point
  - Non-tree edges eliminate articulation points
- Root nodes are articulation points iff they have more than one child

non-leaf, non-root node u is an articulation point ⇔ no non-tree edges going from sub-tree below some child of u to above u in the tree

44

DFS Application: Articulation Points



45