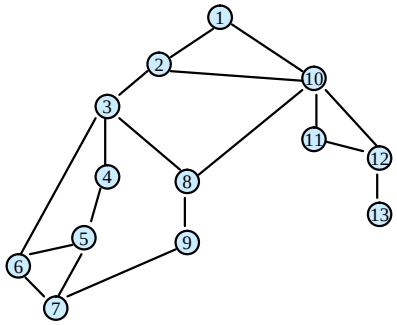
CSE 417: Algorithms and Computational Complexity

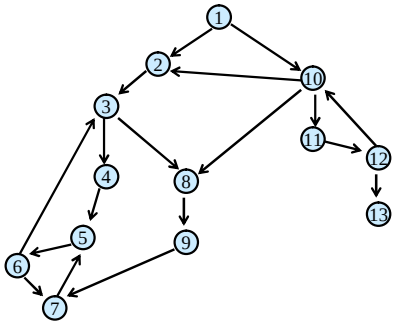
Graphs & Graph Algorithms I

Autumn 2002
Paul Beame

Undirected Graph G = (V,E)



Directed Graph G = (V,E)

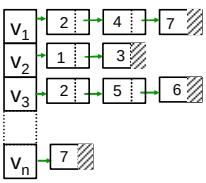


Representing Graph G=(V,E)
n vertices, m edges

- 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

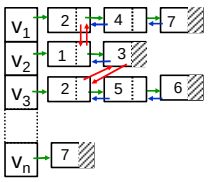
- Adjacency List:
  - **O(n+m)** words
  - **O(log n)** bits each



- Advantages:
  - Compact for sparse graphs

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

- 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**

7

Breadth-First Search

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

8

BFS(v)

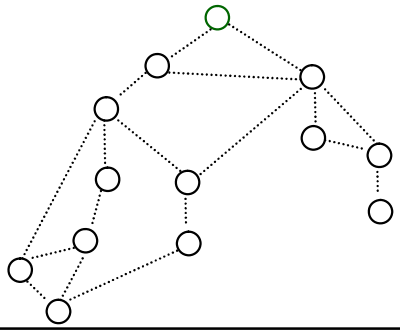
Global initialization: mark all vertices "undiscovered"

```
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 to queue
        mark u "fully-explored"
```

Exercise: modify code to number vertices & compute level numbers

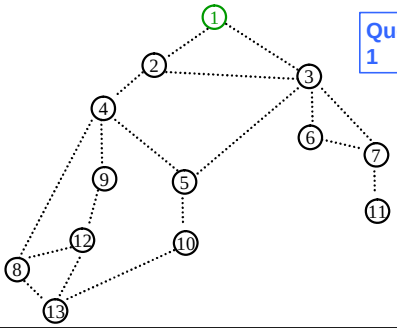
9

BFS(v)



10

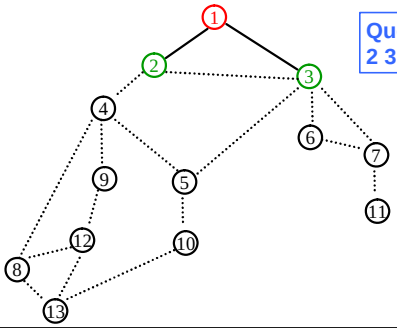
BFS(v)



Queue: 1

11

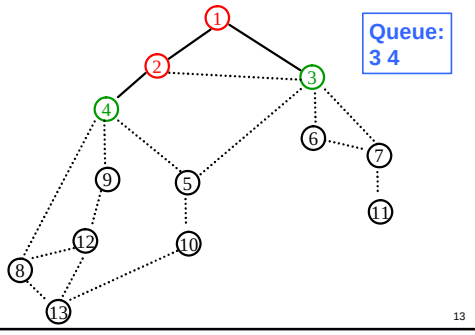
BFS(v)



Queue: 2 3

12

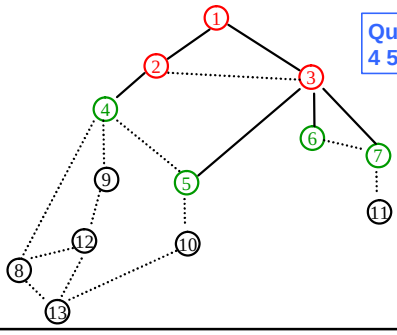
BFS(v)



Queue:
3 4

13

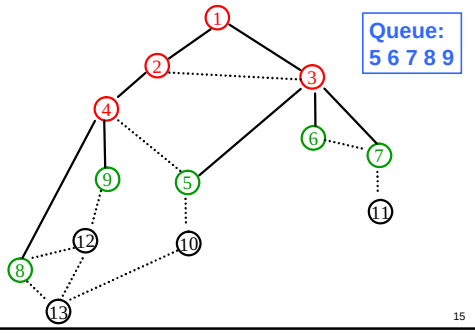
BFS(v)



Queue:
4 5 6 7

14

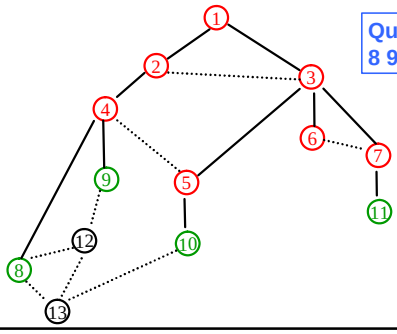
BFS(v)



Queue:
5 6 7 8 9

15

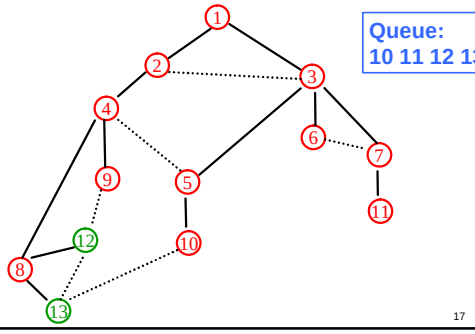
BFS(v)



Queue:
8 9 10 11

16

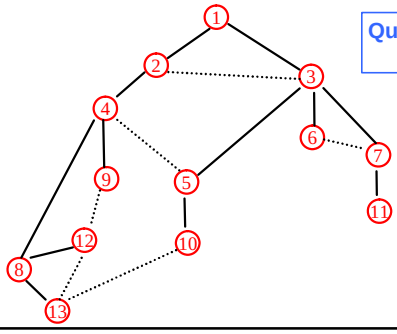
BFS(v)



Queue:
10 11 12 13

17

BFS(v)



Queue:

18

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 to answer questions of the form:
  - Given: vertices **u** and **v** in **G**
  - 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 an array **Path[u,v]**?

22

th in **G** from

define a

tree

- "breadth first spanning tree" of **G**

- Level **i** in this tree
  - those vertices **u** such that the shortest path in **G** from the root **v** is of length **i**.

e same or

20

Graph Search Application: Connected Components

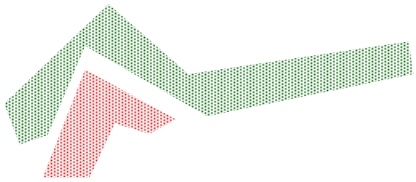
- initial state: all **v** undiscovered
  for **v**←**1** to **n** do
  if state(**v**) != "fully-explored" then
  BFS(**v**): setting **A**[**u**] ←**v** for each **u** found
  (and marking **u** discovered/fully-explored)
  endif
  endfor
- Total cost: **O(n+m)**
  - each vertex an each edge is touched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

23

Depth-First Search

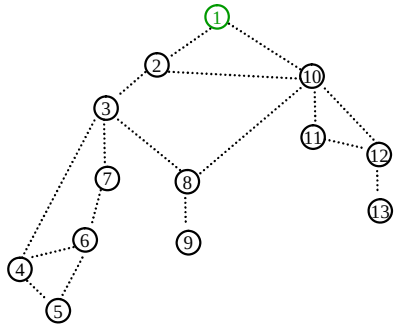
- Follow the first path you find as far as you can go
- Back up to last unexplored edge when you reach a dead end, then go as far you can
- Naturally implemented using recursive calls or a stack

24



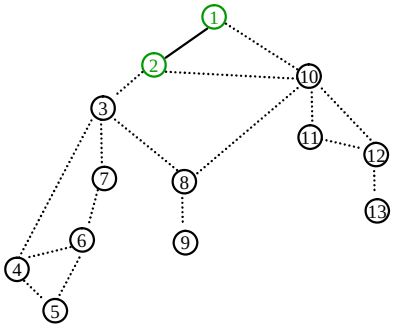
21

DFS(v)



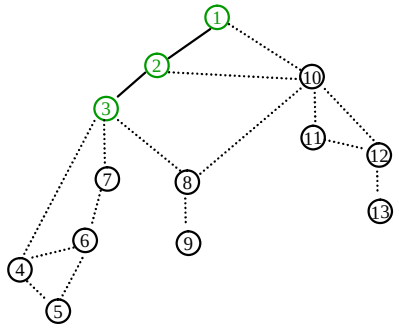
27

DFS(v)



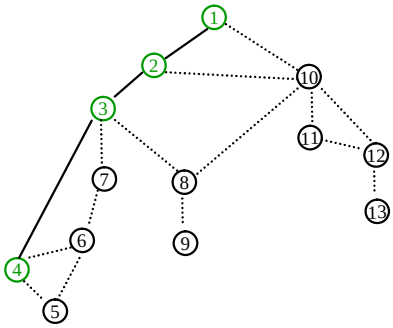
28

DFS(v)

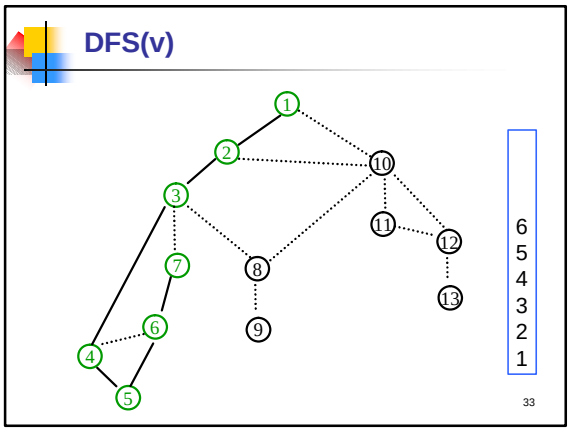


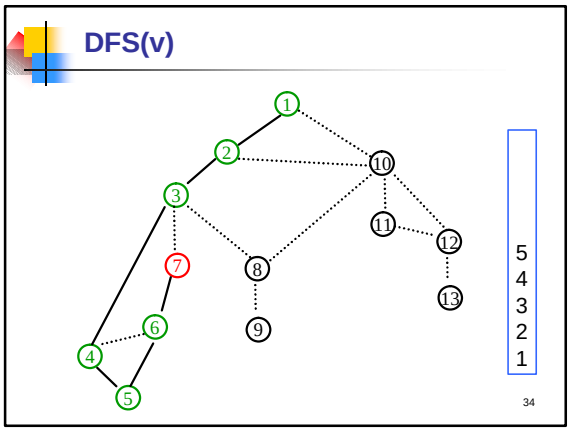
29

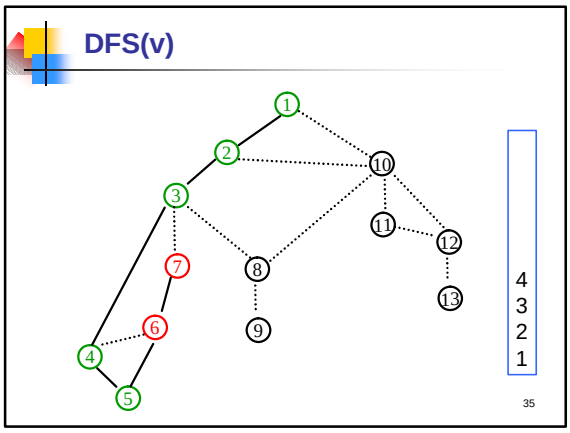
DFS(v)

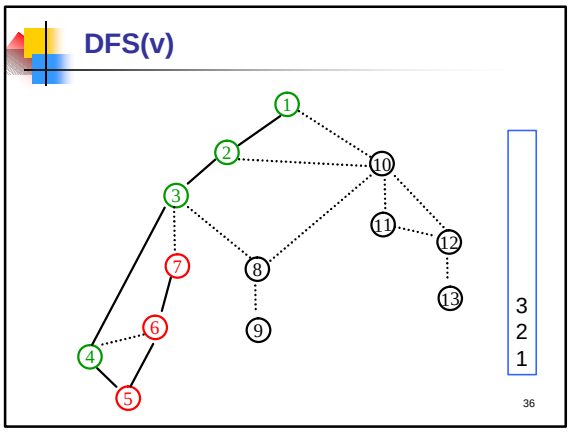


30

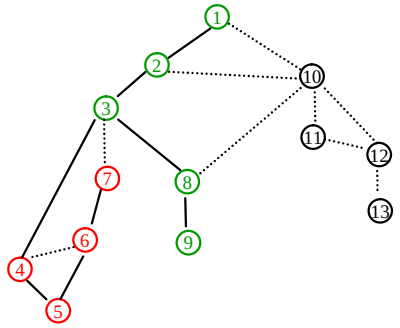

DFS(v)
1
2
10
3
11
12
7
8
13
6
9
4
5
6
5
4
3
2
1
33


DFS(v)
1
2
10
3
11
12
7
8
13
6
9
4
5
5
4
3
2
1
34


DFS(v)
1
2
10
3
11
12
7
8
13
6
9
4
5
4
3
2
1
35


DFS(v)
1
2
10
3
11
12
7
8
13
6
9
4
5
3
2
1
36

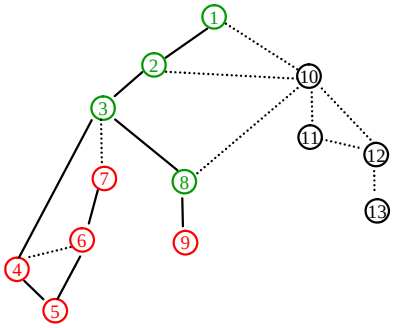
DFS(v)



8
3
2
1

39

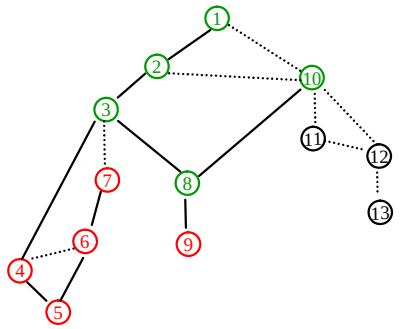
DFS(v)



3
2
1

40

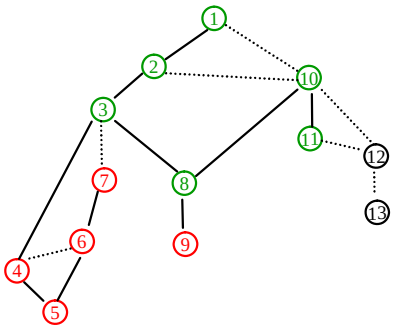
DFS(v)



8
3
2
1

41

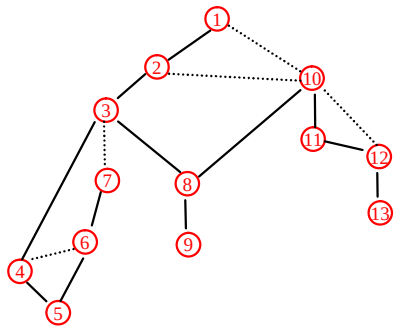
DFS(v)



10
8
3
2
1

42

DFS(v)



45

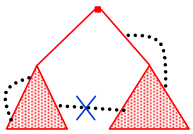
Properties of DFS(v)

- Like BFS(**v**):
  - DFS(**v**) visits **x** if and only if there is a path in **G** from **v** to **x**
  - Edges into undiscovered vertices define a tree
    - "depth first spanning tree" of **G**
- Unlike the BFS tree:
  - the DFS 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…

46

Non-tree edges

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



47

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

48

Articulation Points from DFS

- Non-tree edges eliminate articulation points
- Root node is articulation point ⇔ it has more than one child
- Leaf nodes are never articulation points
- Other nodes **u** are articulation points ⇔
  - no non-tree edges going from the sub-tree rooted at a child of **u** to above **u** in the tree

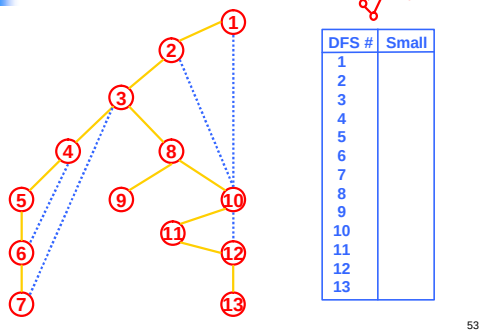
51

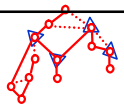
Articulation Points from DFS

- For each vertex **v** compute
  - Small(**v**)
    - the smallest number of a node pointed at by any descendant of **v** in the DFS tree (including **v** itself)
  - Can compute Small(v) for every **v** during DFS at minimal extra cost
- Non-tree, non-root node **u** is an articulation point ⇔ for some child **v** of **u**
  - Small(**v**) = DFSnumber(**u**)
  - Easy to check during DFS

52

Articulation Points

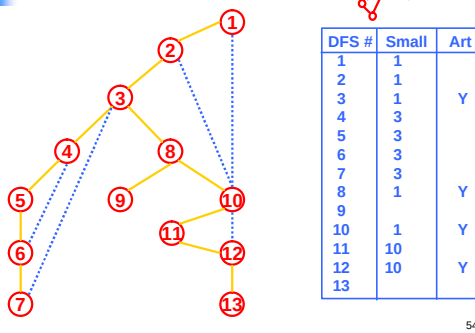


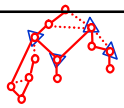


DFS #	Small
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	

53

Articulation Points





DFS #	Small	Art
1	1	
2	1	
3	1	Y
4	3	
5	3	
6	3	
7	3	
8	1	Y
9		
10	1	Y
11	10	
12	10	Y
13		

54