Lecture 26: Graph Algorithms

Agenda:

- DFS application: finding biconnected components
- Greedy algorithms: elements & properties
- Minimum spanning tree

Reading:

- Textbook pages 379 – 384, 558 – 579

Biconnected component:

- Definition — every pair of vertices are connected by two vertex-disjoint paths

- Cut vertex — its removal increases the number of connected components

- Fact: biconnected $\Longleftrightarrow$ no cut vertices

- Biconnected component $\Longleftrightarrow$ maximal connected subgraph containing no cut vertex

- In a DFS tree:
  - root is a cut vertex **iff** it has ≥ 2 child vertices (Why ???)
    $\longrightarrow$ One simplest implementation (assuming connected):
    1. try every vertex v as the start vertex and do the DFS
    2. in the DFS tree, if $\texttt{degree}_{DFS}(v) > 1$, decompose the graph accordingly into $\texttt{degree}_{DFS}(v)$ subgraphs sharing one common vertex v
    3. repeat on subgraphs until for every subgraph the DFS tree with every possible start vertex has root degree 1

    Problem: too time consuming $\Theta(n(n+m))$ ...
  - any other vertex is a cut vertex **iff** vertices in the child subtrees have no back edges to its proper ancestors
    $\longrightarrow$ Idea in the improved implementation — ($\Theta(n+m)$):
    for each vertex v, and each of its child w, keep track of furthest back edge from the w-subtree

DFS application: finding biconnected components

- Idea in the improved implementation — ($\Theta(n+m)$):

  for each vertex v, and each of its child w, keep track of furthest back edge from the w-subtree

- Details:
  - for every vertex v, 1st encounter child w, recur from w
  - last encounter w (just before backing up to v), check whether v cuts off the w-subtree (rooted at w)
  - maintain $\texttt{dtime}[v]$, $b[v]$, $p[v]$ for v:
    1. $\texttt{dtime}[v]$ — discovery time
    2. $b[v]$ — `dtime` of the furthest ancestor of v to which there is back edge from a descendant w of v
       (a) updated when the first back edge is encountered
       (b) updated when last time encounter of v (backing up)
    3. $p[v]$ — parent of v in the DFS tree

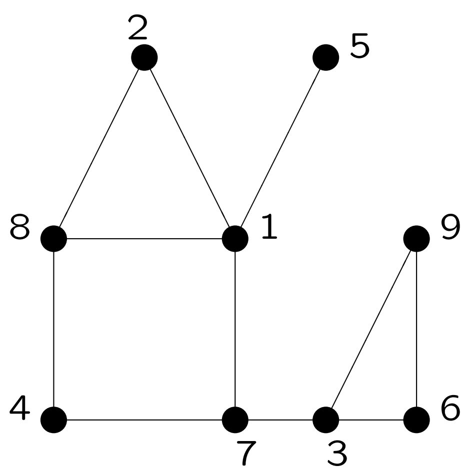
- Reporting biconnected components:
  - recall that biconnected components form a partition of edge set E
  - when edge e first encountered, push into edge stack
  - when a cut vertex discovered, pop necessary edges

Finding biconnected components — pseudocode:

```
procedure bicomponents(G)          **G = (V, E)

S = ∅                              **S is the edge stack
time ← 0
for each v ∈ V do
    p[v] ← 0                       **unknown yet:  NIL
    dtime[v] ← time
    b[v] ← n + 1
for each v ∈ V do
    if dtime[v] = 0 then
        biDFS(v)

procedure biDFS(v)                 **discover v

time ← time +1
dtime[v] ← time
b[v] ← dtime[v]                    **no back edge from descendant yet
for each neighbor w of v do        **first time encounter w
    if dtime[w] = 0 then           **unknown yet
        push(v, w)
        p[w] ← v
        biDFS(w)                   **recursive call
        if b[w] ≥ dtime[v] then
            print ‘‘new biconnected component’’
            repeat
                pop & print
            until (popped edge is (v, w))
        else
            b[v] ← min{b[v], b[w]}
    else if (dtime[w] < dtime[v] and w ≠ p[v]) then
                                   **(v, w) is a back edge from v
        push(v, w)
        b[v] ← min{b[v], dtime[w]}
```

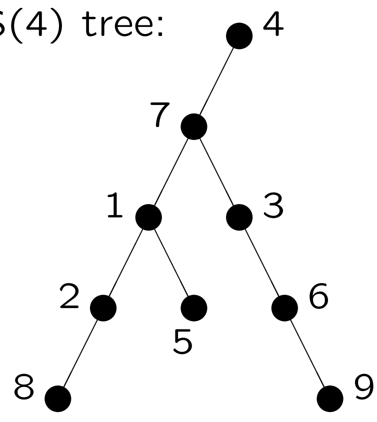
Finding biconnected components — example:

Execute `biDFS(4)` on the following graph, assuming no previous `biDFS()` calls:



1: 2 5 7 8
2: 1 8
3: 6 7 9
4: 7 8
5: 1
6: 3 9
7: 1 3 4
8: 1 2 4
9: 3 6

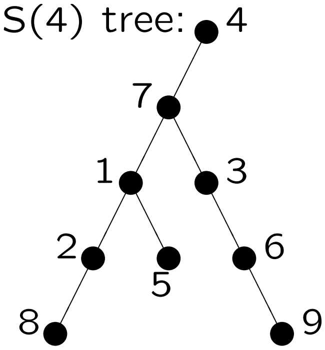
DFS(4) tree:



Finding biconnected components — answer:

1: 2 5 7 8
2: 1 8
3: 6 7 9
4: 7 8
5: 1
6: 3 9
7: 1 3 4
8: 1 2 4
9: 3 6

DFS(4) tree:



dtime	3	4	7	1	6	8	2	5	9
	$b[1]$	$b[2]$	$b[3]$	$b[4]$	$b[5]$	$b[6]$	$b[7]$	$b[8]$	$b[9]$
biDFS(4)	10	10	10	1	10	10	10	10	10
4} biDFS(7)	10	10	10	1	10	10	2	10	10
4, 7} biDFS(1)	3	10	10	1	10	10	2	10	10
4, 7, 1} biDFS(2)	3	4	10	1	10	10	2	10	10
4, 7, 1, 2} (2,1)									
4, 7, 1, 2} biDFS(8)	3	4	10	1	10	10	2	5	10
4, 7, 1, 2, 8} (8,1)	3	4	10	1	10	10	2	*3*	10
4, 7, 1, 2, 8} (8,2)									
4, 7, 1, 2, 8} (8,4)	3	4	10	1	10	10	2	*1*	10
4, 7, 1, 2} biDFS(8) done	3	*1*	10	1	10	10	2	1	10
4, 7, 1} biDFS(2) done	*1*	1	10	1	10	10	2	1	10
4, 7, 1} biDFS(5)	1	1	10	1	6	10	2	1	10
4, 7, 1, 5} (5,1)									
4, 7, 1} biDFS(5) done	new biconnected component: (1, 5)								
4, 7, 1} (1,7)									
4, 7, 1} (1,8)									
4, 7} biDFS(1) done	1	1	10	1	6	10	*1*	1	10
4, 7} biDFS(3)	1	1	7	1	6	10	1	1	10
4, 7, 3} biDFS(6)	1	1	7	1	6	8	1	1	10
4, 7, 3, 6} (6,3)									
4, 7, 3, 6} biDFS(9)	1	1	7	1	6	8	1	1	9
4, 7, 3, 6, 9} (9,3)	1	1	7	1	6	8	1	1	*7*
4, 7, 3, 6, 9} (9,6)									
4, 7, 3, 6} biDFS(9) done	1	1	7	1	6	*7*	1	1	7
4, 7, 3} biDFS(6) done	new biconnected component: (9, 3), (6, 9), (3, 6)								
4, 7, 3} (3,7)									
4, 7, 3} (3,9)									
4, 7} biDFS(3) done	new biconnected component: (7, 3)								
4, 7} (7,4)									
4} biDFS(7) done	new biconnected component: (8, 4), (8, 1), (2, 8), (1, 2), (7, 1), (4, 7)								
biDFS(4) done	1	1	7	1	6	7	1	1	7

Finding biconnected components — analysis:

- Correctness ???

- Complexity — running time and space requirement:

  - running time:
    constant for each vertex encounter and each edge encounter $\longrightarrow$
    $\Theta(c_1 n + c_2 \sum_{v \in V} \texttt{degree}(v)) = \Theta(n+m)$

  - space:
    assume adjacency list representation: space for graph, arrays of size n, edge stack, and runtime stack

    1. space for graph and arrays $\Theta(m+n)$
    2. edge stack requires $O(m)$ — since every edge pushed
    3. runtime stack $O(n)$ — since at most n `biDFS` activations each is of constant size
    4. therefore, $\Theta(n+m)$ in total

Minimum spanning tree (MST) problem:

- Input: edge-weighted (simple, undirected) connected graphs (positive weights)

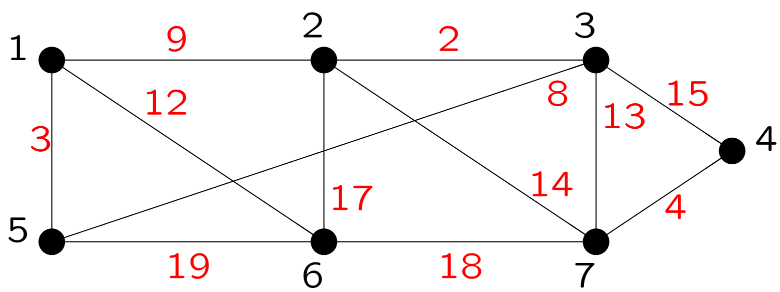
- Notions:
  - subgraph, acyclic, tree
  - spanning subgraph: subgraph including all the vertices
  - spanning tree: spanning subgraph which is a tree — acyclic connected subgraph $T = (V, E')$, where $E' \subset E$

    e.g., BFS/DFS (on a connected input graph) tree is a spanning tree of the graph
  - minimum spanning tree: minimum weight

- The MST Problem:

  Find a minimum spanning tree for the input graph.

  For example:



- The minimum spanning forest problem:

  The given graph is not necessarily connected.

  Find an MST for each connected component.

Greedy algorithms and MST problem:

- Greedy algorithms:
  - greedy — each step makes the best choice (locally maximum)
  - iterative algorithms
  - optimal substructure

    an optimal solution to the original problem contains within it optimal solutions to subproblems

- Greedy solution may NOT be globally optimum

  e.g., matrix-chain multiplication: $A_{6\times5} \times A_{5\times2} \times A_{2\times5} \times A_{5\times6}$

  Greedy: $50 + 150 + 180 = 380$ scalar multiplications

  Dynamic programming: $60+60+72 = 192$ scalar multiplications

- The MST problem:

  Two greedy solutions are globally optimum
  - Prim's (Prim + Dijkstra + Boruvka's)

    growing the tree to include more vertices
  - Kruskal's (Kruskal + Boruvka's)

    growing the forest to become a tree

Have you understood the lecture contents?

well	ok	not-at-all	topic
☐	☐	☐	biconnected component & cut vertex
☐	☐	☐	one simplest implementation via DFS
☐	☐	☐	the improved DFS implementation
☐	☐	☐	execution and correctness
☐	☐	☐	minimum spanning tree
☐	☐	☐	greedy algorithms in general