Lecture 27: Wednesday March 19, 2003

today

- biconnected components
- muddytown

announcements

recall: bicomponent (= biconnected component)

- *cut vertex* — removal increases number of components
- *biconnected graph* — connected and no cut vertex
- *bicomponent* — maximal biconnected subgraph
- v cut vertex iff, w.r.t. dfs tree

root: more than one subtree

not root: child subtree has no back edge to proper v-ancestor

recall: finding bicomponents via depth first search

- algorithm: for each v, for each child w, keep track of furthest back edge from w-subtree
- how to implement algorithm using dfs?
  - 1st encounter of child w of parent v
    - recurse from w
  - last encounter of w, just before backing up to v
    - check whether v cuts off w-subtree
  - maintain **`dfn`**, **`back`**, **`parent`** for each v
    - **`parent`**: parent in DFS tree
    - **`dfn`**: number, by discovery, in DFS
    - **`back`**: dfn of furthest ancestor (of descendant)
    - update **`back`** when backedge 1st encountered
    - update **`back`** when backing up
  - maintain edge stack
    - push edge when edge 1st encountered
    - pop edges when cutpoint discovered

```
bicomponents()    (*version 2001, based on CLRS DFS*)
  empty stack; dfnum <- 0
  for all v do
    parent[v] <- 0; dfn[v] <- 0; back[v] <- n+1
  for all v do
    if dfn[v]=0 then bidfs(v)
end_bicomponents

bidfs(v)
  inc(dfnum); dfn[v] <- dfnum; back[v] <- dfn
  for each neighbour w do
    if dfn[w]=0 then                        (*1st w encounter*)
      push [vw]; parent[w] <- v                   (*tree edge*)
      bidfs(w)
      (* backup up from w to v*)
      if back[w] >= dfn[v] then  (*v root or cuts off w? yes*)
        print ‘new bicomponent’
        repeat: pop and print edge
        until  popped edge is [vw]
      else                       (*v root or cuts off w? no*)
        back[v] <- min {back[v],back[w]}
      (*end backup from w to v*)
    elsif dfn[w]<dfn[v] and w<>parent[v] then  (*back edge*)
      push [vw]; back[v] <- min {dfn[w],back[v]}
end_bidfs
```

- example trace: execute **`bidfs(4)`** on the graph below, assuming no previous **`bidfs()`** calls (answer on the next page)

```
                 1: 2 5 7 8    dfs(4) tree  4
   2     5       2: 1 8                    /
  / \   /        3: 6 7 9                 7
 /   \ /         4: 7 8                  / \
8-----1      9   5: 1                   1   3
|     |     /|   6: 3 9                / \   \
|     |    / |   7: 1 3 4             2   5   6
4-----7---3--6   8: 1 2 4            /         \
                 9: 3 6             8           9
```

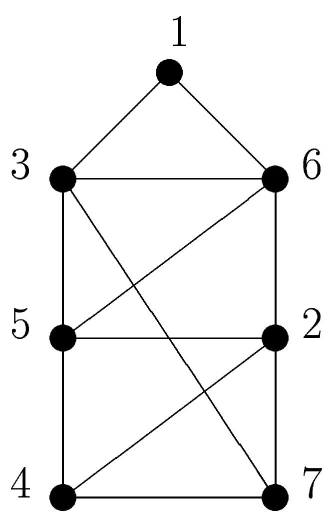
```
                        back[1 2 3 4 5 6 7 8 9]
bidfs(4)                    * * * 1 * * * * *
4} tree[47]
4} bidfs(7)                 * * * 1 * * 2 * *
4} 7} tree[71]
4} 7} bidfs(1)              3 * * 1 * * 2 * *
4} 7} 1} tree[12]
4} 7} 1} bidfs(2)           3 4 * 1 * * 2 * *
4} 7} 1} 2} tree[28]
4} 7} 1} 2} bidfs(8)        3 4 * 1 * * 2 5 *
4} 7} 1} 2} 8} back[81]     3 4 * 1 * * 2 3 *
4} 7} 1} 2} 8} back[84]     3 4 * 1 * * 2 1 *
4} 7} 1} 2} backup noout    3 1 * 1 * * 2 1 *
4} 7} 1} backup noout       1 1 * 1 * * 2 1 *
4} 7} 1} tree[15]
4} 7} 1} bidfs(5)           1 1 * 1 6 * 2 1 *
4} 7} 1} backup             out[15]
4} 7} backup noout          1 1 * 1 6 * 1 1 *
4} 7} tree[73]
4} 7} bidfs(3)              1 1 7 1 6 * 1 1 *
4} 7} 3} tree[36]
4} 7} 3} bidfs(6)           1 1 7 1 6 8 1 1 *
4} 7} 3} 6} tree[69]
4} 7} 3} 6} bidfs(9)        1 1 7 1 6 8 1 1 9
4} 7} 3} 6} 9} back[93]     1 1 7 1 6 8 1 1 7
4} 7} 3} 6} backup noout    1 1 7 1 6 7 1 1 7
4} 7} 3} backup             out[93][69][36]
4} 7} backup                out[73]
4} backup                   out[84][81][28][12][71][47]
```

bicomponent algorithm: analysis

- correctness? the truth is out there ☺
- complexity?
  - time: constant for each vertex/edge encounter
    $\Theta(c_1 n + c_2 \sum_v deg(v) = c_1 n + 2c_2 m) \;=\; \Theta(n+m)$
  - space: assume adjacency list representation
    - graph, arrays of size n, edge stack, runtime stack
    - edge stack: $O(m)$ since each edge pushed
    - runtime stack: $O(n)$ since at most
      n constant size activation records
    - $\Theta(n+m) + \Theta(n) + O(m) + O(n) \;=\; \Theta(n+m)$

muddytown

- problem: dirt streets, muddy when it rains
- goal: to be able to walk without muddying shoes
- idea: pave enough streets to walk anywhere
- problem: given street paving costs, find min cost paving



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

	1	2	3	4	5	6	7
1							
2							
3							
4							
5							
6							
7							