Lecture 25: Graph Algorithms

Agenda:

- Graph traversal — Depth-first search
- DFS application: finding biconnected components

Reading:

- Textbook pages 540 – 549, 558 – 559

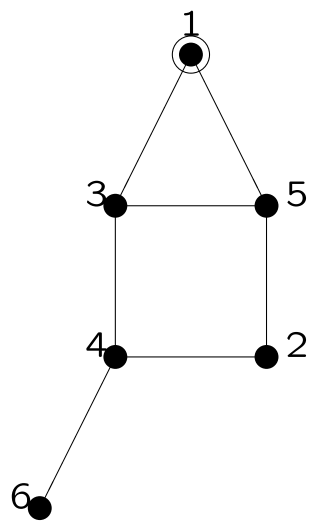
Depth First Search (DFS):

- Input: simple undirected graph $G = (V, E)$
- Output: all vertices discovered (pick one vertex from each component as the start vertex)
- Idea: to search deeper in the graph whenever possible ...
- Pseudocode (recursive version):

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

for each v ∈ V do
    c[v] ← WHITE              **unknown yet
    p[v] ← NIL                **predecessor
time ← 0
for each v ∈ V do
    if c[v] = WHITE then
        DFS-visit(v)

procedure DFS-visit(v)        **any v ∈ V

c[v] ← GRAY                   **start discovering v
time ← time + 1
dtime[v] ← time
for each u ∈ Adj[v] do
    if c[u] = WHITE then
        p[u] ← v
        DFS-visit(u)
c[v] ← BLACK                  **finished discovering
time ← time + 1
ftime[v] ← time
```

DFS example:

- $V = \{1, 2, 3, 4, 5, 6\}$
  $E = \{\{1,3\}, \{1,5\}, \{2,4\}, \{2,5\}, \{3,4\}, \{3,5\}, \{4,6\}\}$
  $s = 2$



Adjacency lists:

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

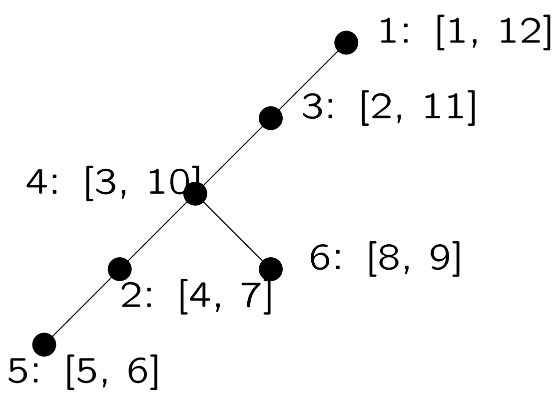
	1	2	3	4	5	6	DFS-visit path
color	W	W	W	W	W	W	
parent	NIL	NIL	NIL	NIL	NIL	NIL	
dtime	∞	∞	∞	∞	∞	∞	`initialization`
ftime	∞	∞	∞	∞	∞	∞	
color	G	W	W	W	W	W	
parent	NIL	NIL	NIL	NIL	NIL	NIL	
dtime	1	∞	∞	∞	∞	∞	`DFS-visit(1)`
ftime	∞	∞	∞	∞	∞	∞	
color	G	W	G	W	W	W	
parent	NIL	NIL	1	NIL	NIL	NIL	
dtime	1	∞	2	∞	∞	∞	`DFS-visit(1-3)`
ftime	∞	∞	∞	∞	∞	∞	
color	G	W	G	G	W	W	
parent	NIL	NIL	1	3	NIL	NIL	
dtime	1	∞	2	3	∞	∞	`DFS-visit(1-3-4)`
ftime	∞	∞	∞	∞	∞	∞	
color	G	G	G	G	W	W	
parent	NIL	4	1	3	NIL	NIL	
dtime	1	4	2	3	∞	∞	`DFS-visit(1-3-4-2)`
ftime	∞	∞	∞	∞	∞	∞	
color	G	G	G	G	G	W	
parent	NIL	4	1	3	2	NIL	
dtime	1	4	2	3	5	∞	`DFS-visit(1-3-4-2-5)`
ftime	∞	∞	∞	∞	∞	∞	
color	G	G	G	G	B	W	
parent	NIL	4	1	3	2	NIL	
dtime	1	4	2	3	5	∞	`DFS-visit(1-3-4-2-5)`
ftime	∞	∞	∞	∞	6	∞	
color	G	B	G	G	B	W	
parent	NIL	4	1	3	2	NIL	
dtime	1	4	2	3	5	∞	`DFS-visit(1-3-4-2)`
ftime	∞	7	∞	∞	6	∞	
color	G	B	G	G	B	G	
parent	NIL	4	1	3	2	4	
dtime	1	4	2	3	5	8	`DFS-visit(1-3-4-6)`
ftime	∞	7	∞	∞	6	∞	
color	G	B	G	G	B	B	
parent	NIL	4	1	3	2	4	
dtime	1	4	2	3	5	8	`DFS-visit(1-3-4-6)`
ftime	∞	7	∞	∞	6	9	
color	G	B	G	B	B	B	
parent	NIL	4	1	3	2	4	
dtime	1	4	2	3	5	8	`DFS-visit(1-3-4)`
ftime	∞	7	∞	10	6	9	
color	G	B	B	B	B	B	
parent	NIL	4	1	3	2	4	
dtime	1	4	2	3	5	8	`DFS-visit(1-3)`
ftime	∞	7	11	10	6	9	
color	B	B	B	B	B	B	
parent	NIL	4	1	3	2	4	
dtime	1	4	2	3	5	8	`DFS-visit(1)`
ftime	12	7	11	10	6	9	

DFS example:

- Adjacency lists:

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

- DFS tree: [`dtime`, `ftime`]



Notes:

- the result would be a forest of rooted trees
- the root of each tree is up to the selection (ordering of the vertices)
- parent of x is predecessor $p[x]$
- different orderings of adjacency lists might result in different trees
- nested structure of [`dtime, ftime`]
  — they don't intersect each other

DFS analysis:

- $n = |V|$, $m = |E|$
- Handshaking Lemma: $\sum_{v \in V} \texttt{degree}(v) = 2m$
- Analysis:
  - each vertex is discovered exactly once (WHITE $\rightarrow$ GRAY $\rightarrow$ BLACK)

    each edge is examined exactly twice
  - running time:
    1. adjacency list representation:
       $\Theta(n + 2m) = \Theta(n + m)$
    2. adjacency matrix representation:
       $\Theta(n + n^2) = \Theta(n^2)$
  - space complexity:
    1. adjacency list representation:
       $\Theta(n + m)$
    2. adjacency matrix representation:
       $\Theta(n^2)$

Classifying graph edges with BFS/DFS:

- During the traversal, all vertices and edges are examined
- Given a BFS/DFS traversal forest:
  - tree root — start vertex for that component
  - tree edge — child discovered while processing the parent
  - each edge in the original graph is examined twice
- Question:

  Where are the other possible edges, besides tree edges ???
- Answer:

  With respect to the traversal forest, categorize graph edges by their first time encounter:
  - tree edges
  - back edges: to ancestor
  - forward edges: to descendant
  - cross edges: to non-ancestor, non-descendant

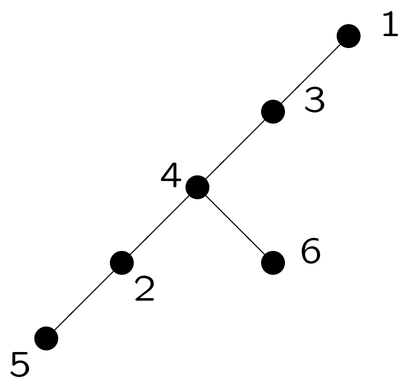
  Note: in undirected graphs, "back" = "forward"
- Examples:

An example:

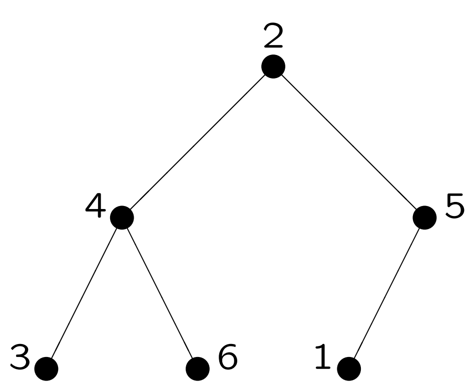
- Adjacency lists:

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

- DFS tree (start vertex 1):



- BFS Tree (start vertex 2):

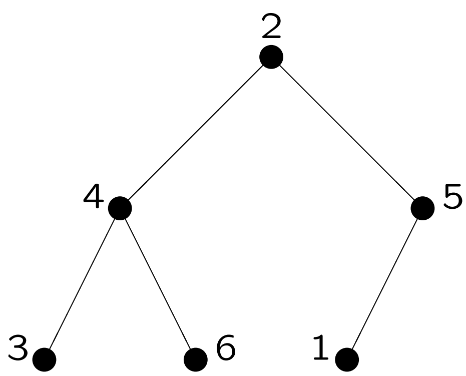


Properties of BFS/DFS:

- BFS:
  - each graph edge connects two vertices with level-difference ≤ 1
    Proof.
  - no back / forward edges
- DFS:
  - each non-tree edge is a back edge
    Proof.
  - no forward edges
  - no cross edges
  - vertex processing time intervals $[\texttt{dtime}[v],\ \texttt{ftime}[v]]$ and $[\texttt{dtime}[w],\ \texttt{ftime}[w]]$:
    $[\texttt{dtime}[v],\ \texttt{ftime}[v]] \subset [\texttt{dtime}[w],\ \texttt{ftime}[w]]$ — v is a descendant of w in the DFS forest
    $[\texttt{dtime}[v],\ \texttt{ftime}[v]] \cap [\texttt{dtime}[w],\ \texttt{ftime}[w]] = \emptyset$ — no ancestor-descendant relationship between v and w
- BFS vertex order:
  level-order of each tree in the BFS forest
- DFS vertex order:
  pre-order of each tree in the DFS forest
- Some other vertex order associated with rooted trees:
  - in-order (for binary trees only)
  - post-order

Vertex order with respect to a binary rooted tree:

- Tree:



- Vertex orders:
  - level-order: level by level (each level: left to right)
    $(2, 4, 5, 3, 6, 1)$
  - pre-order: parent - child one - child two - . . . - last child
    $(2, 4, 3, 6, 5, 1)$
  - in-order: left child - parent - right child
    $(3, 4, 6, 2, 1, 5)$
  - post-order: child one - child two - . . . - last child - parent
    $(3, 6, 4, 1, 5, 2)$

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
  - any other vertex is a cut vertex **iff** vertices in the child subtrees have no back edges to its proper ancestors
- 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 with only one common vertex v
  3. repeat on subgraphs until for every subgraph the DFS tree with every possible start vertex has a root degree 1

  Problem: too time consuming $\Theta(n(n+m))$ ...
- Idea in finding biconnected components via DFS tree

  — $(\Theta(n+m))$:

  for each vertex v, and each of its child w, keep track of furthest back edge from the w-subtree (detail next lecture)

Have you understood the lecture contents?

well	ok	not-at-all	topic
☐	☐	☐	depth first search execution
☐	☐	☐	depth first search analysis
☐	☐	☐	graph edge categorization
☐	☐	☐	BFS/DFS vertex order
☐	☐	☐	biconnected component & cut vertex
☐	☐	☐	one simplest implementation