CPCFI

Lecture: 2: GRAPH TRAVERSAL II

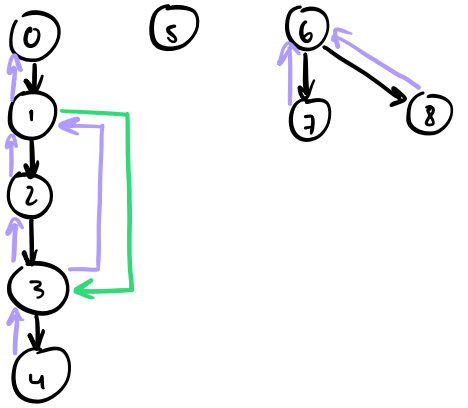
Unit: 6

Instructor: WESTON

I. GRAPH EDGE CLASSIFICATION VIA DFS SPANNING TREE (REMAINDER)

- RUNNING DFS ON A GRAPH GENERATES A SPANNING TREE
- USING ONE MORE VERTEX STATE: [UNVISITED, VISITED, EXITED] → color = 0, 1, 2
  WE HAVE THE FOLLOWING VERTEX CLASSIFICATION:

1. TREE EDGE: VISITED → UNVISITED
2. BACK EDGE: VISITED → VISITED
3. FORWARD EDGE: COMPLETED → VISITED



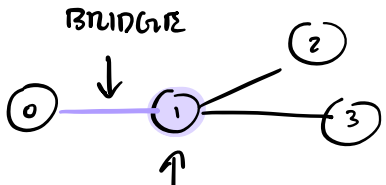
TREE EDGE
BACK EDGE
FWD EDGE

II ARTICULATION POINTS AND BRIDGES (CODEFORCES BLOG ENTRY 71146)

SEANLEE97

ARTICULATION → VERTEX
BRIDGES → EDGE

WHEN REMOVED, THE NUMBER OF CONNECTED COMPONENTS INCREASES



ARTICULATION POINT

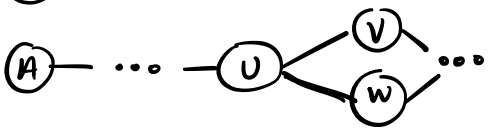
[IDEA] - ARTICULATION POINT (AP)

VERTEX U CAN BE AN ARTICULATION POINT (AP) IF:

(1) ALL PATHS FROM VERTEX A TO V PASS THROUGH U

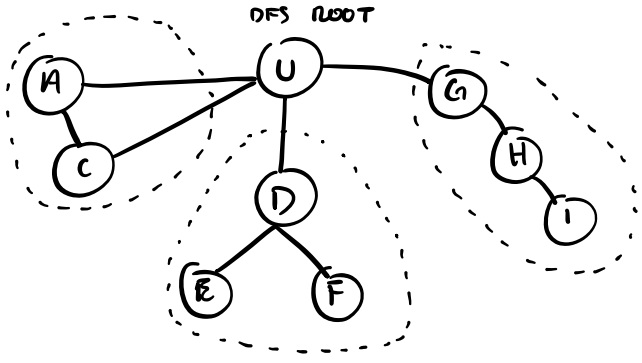
.) 

(V CANNOT REACH A WITHOUT U)

..) 

(PART IN SOME CYCLE OF DFS TRAVERSAL)

(2) U IS THE ROOT OF DFS SPANNING TREE WITH AT LEAST 2 SUBGRAPHS DISCONNECTED FROM EACH OTHER



[IMPLEMENTATION] ⇒ TARJAN'S ALGORITHM IN O(n+m)

1. FIND IF A IS ANCESTOR OF SOME NODE V → WE USE TIME OF ENTRY ARRAY BASED ON DFS TRAVERSAL
   - ENTRY[A] < ENTRY[V] : A IS ANCESTOR OF V

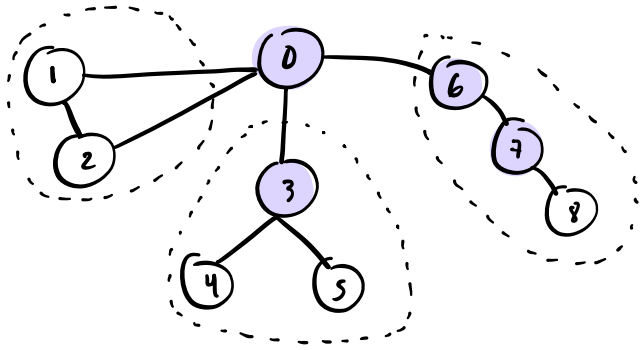
2. FIND IF U IS AN A.P., CHECK THE FOLLOWING CONDITIONS:
   - IF WE CANNOT REACH V WITH STRICTLY SMALLER T.O. ENTRY THAN U, THEN U IS AN AP
   - IF U IS THE ROOT OF DFS TREE AND HAS AT LEAST 2 CHILDREN SUBGRAPHS DISCONNECTED FROM EACH OTHER, THEN U IS AN AP

→ FOR EVERY NODE U, WE HAVE TO FIND NODE LOW WHICH IS THE NODE THAT CAN BE REACHED FROM U WITH THE LEAST TIME OF ENTRY

→ U WILL BE AN AP IF: $\boxed{\text{ENTRY}[U] \leq \text{LOW}[V]}$

→ U WILL BE AN AP IF: $\boxed{\text{tentry}[v] \leq \text{low}[v]}$

↳ V IS AN ADJACENT NODE TO U



0, 3, 6, 7 ARE ARTICULATION POINTS

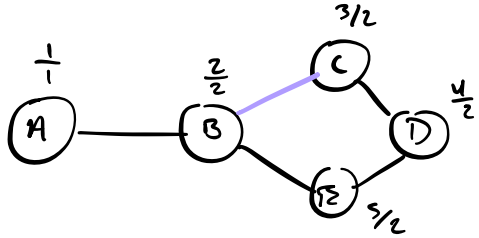
[NOW FOR BRIDGES]

- THE ONLY CHANGE: $\underbrace{\text{tentry}[u] < \text{low}[v]}$

V IS ADJACENT TO U

WHY?

↳ IF $\text{tentry}[u] = \text{low}[v]$ IT MEANS THERE IS A CYCLE ROOTED AT U AND REMOVING EDGE UV WILL NOT INCREASE THE # OF CONNECTED COMPONENTS



ENTRY: i
LOW: j

III. STRONGLY CONNECTED COMPONENTS

[WITH MULTIPLE EDGES AND LOOPS]

- GIVEN A DIRECTED GRAPH, A S.C.C. IS A SUBSET OF G SUCH THAT:

$$\forall\, u, v \in G, \quad \underbrace{u \mapsto v,\ v \mapsto u}$$

REACHABILITY (PATH BETWEEN u, v)

- CONDENSATION GRAPH G^{scc}: GRAPH CONTAINING EVERY S.C.C. AS A NODE

↳ ACYCLIC

↳ A GRAPH CAN CONTAIN MULTIPLE S.C.C.'s

↳ THEY DON'T INTERSECT WITH ONE ANOTHER

[ALGORITHM FOR FINDING S.C.C.'s] ⇒ KOSARAJU AND SHARIR ALGORITHM

("TWO DFS's ALGORITHM")

O(n+m)

1. RUN A SEQUENCE OF DFS's ON G AND BUILD A LIST OF VERTICES SORTED BY DECREASING ORDER OF FEXIT (TIME OF FEXIT)
2. BUILD G^T (TRANSPOSE OF G)
3. RUN A SEQUENCE OF DFS's IN THAT ORDER DETERMINED BY THE LIST OF EXIT.
   ↳ EVERY SET OF VERTICES REACHED AFTER THE NEXT SEARCH WILL BE THE NEXT S.C.C.

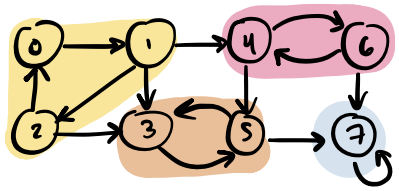
THEOREM: LET C AND C' ARE TWO S.C.C's OF GRAPH G AND THERE IS AN EDGE (C, C') IN A CONDENSATION GRAPH BETWEEN THESE TWO VERTICES. THEN:

$$FEXIT[C] > FEXIT[C']$$

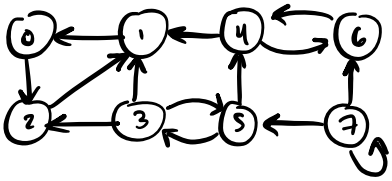
↓

MAXIMUM TIME OF EXIT $\forall v \in C$ VIA DFS

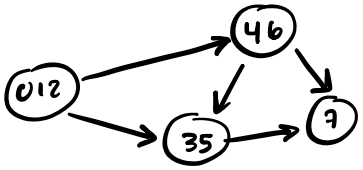
IMPLICATION: ANY EDGE (C, C') IN A CONDENSATION GRAPH COMES FROM A COMPONENT WITH LARGER FEXIT VALUE TO A COMPONENT WITH SMALLER VALUE



GRAPH G WITH 4 S.C.C.



G^T: TRANSPOSE OF G



CONDENSATION GRAPH

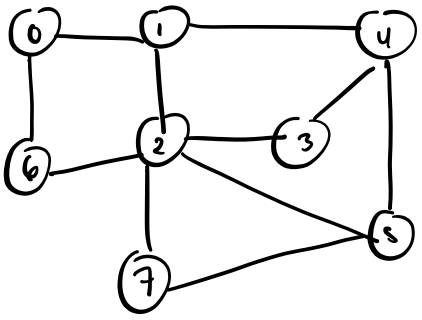
IV. STRONG ORIENTATION

↳ ASSIGNMENT OF A DIRECTION TO EVERY VERTEX IN AN UNDIRECTED GRAPH G SO THAT G ENDS UP BEING A STRONGLY CONNECTED GRAPH

↳ BRIDGELESS CONNECTED GRAPH

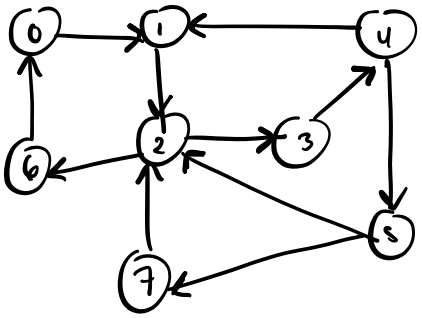
- IF G CONTAINED A BRIDGE, THEN WE WON'T BE ABLE TO ASSIGN A STRONG ORIENTATION

→ VISIT ALL EDGES FROM ANY EDGE AFTER THAT STRONG ORIENTATION



GRAPH G

$\Rightarrow$



STRONG ORIENTATION OF G

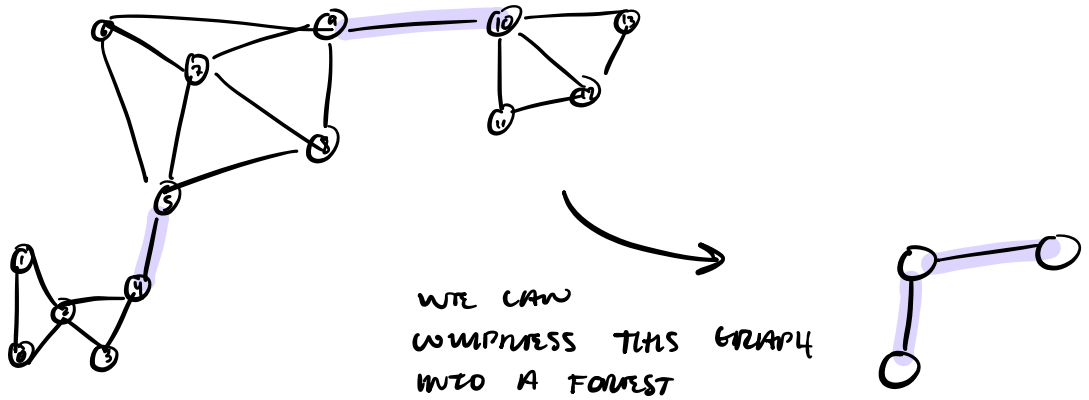
V. FINDING BRIDGES ONLINE

→ THE GRAPH DOESN'T HAVE TO BE KNOWN IN ADVANCE

- PROBLEM: OUTPUT THE # OF BRIDGES AFTER ADDING EACH RECEIVED EDGE TO THE GRAPH
- $O(n \log n + m)$
- BASED ON DISJOINT SET UNION D.S.

[ALGORITHM]

- k - EDGE - CONNECTED COMPONENT: CONNECTED COMPONENT THAT REMAINS CONNECTED WHEN REMOVING FEWER THAN k EDGES
- A BRIDGE PARTITIONS THE GRAPH INTO 2 - EDGE CONN. COMPS.:



- THE ALGORITHM MAINTAINS BOTH 2-EDGE C.C. AND COMPRESSED GRAPH

1. INITIALLY THE GRAPH IS EMPTY

2. WHEN ADDING A NEW EDGE (A,B):

   (1) BOTH VERTICES A,B ARE IN THE SAME 2-EDGE C.C. ∴ (A,B) IS NOT A BRIDGE
   (# OF BRIDGES UNMODIFIED)

   (2) VERTICES A AND B ARE IN DIFFERENT C.C. ∴ (A,B) BECOMES A NEW BRIDGE
   (# OF BRIDGES +1)

   (3) VERTICES A AND B IN THE SAME C.C. BUT DIFFERENT 2-EDGE C.C. ∴ (A,B) FORMS A CYCLE AND BRIDGES STOP BEING BRIDGES AND THE CYCLE MUST BE COMPRESSED INTO A NEW 2-EDGE COMPONENT
   (# OF BRIDGES -2 OR MORE)

[IMPLEMENTATION]

- TWO DISJOINT SET UNIONS → STORES UNCONNECTED COMPONENTS
  → STORES 2-EDGE C.C.'s
- ALSO, STORE THE FOREST OF 2-EDGE C.C.'s

==== MAIN OPERATIONS ====

1.- CHECK WHETHER TWO VERTICES LIE IN THE SAME C.C. / 2-EDGE C.C. ⇒ FINDSET(v) DSU'S OPERATION

2.- MERGE TWO TREES (WE NEED TO RE-ROOT THEM IF NEITHER A NOR B ARE THE ROOTS OF THEIR C.C/2-EDGE C.C.) ⇒ RE-ROOT HAVE SMALLER OF THE TWO TREES

3.- SEARCHING FOR A CYCLE FORMED BY ADDING EDGE (A,B) ⇒ LOWEST COMMON ANCESTOR, CHECK IF THE FOLLOWING PATH EXISTS:

A → LCA → B → (A,B)