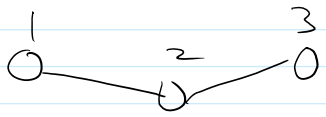
COMPSCI 330 Lecture 9 Graph Traversal Algorithms

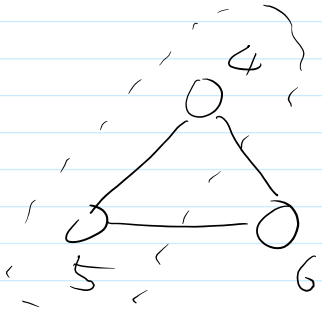
Monday, September 26, 2016 1:59 PM

- Connected Components
- Strongly Connected Components
- Bipartite Graphs

– Connected component (undirected graph)

– Connected component: subset of vertices that are mutually connected by paths, and is not connected to any other vertex.





– Goal: find all the connected components of the graph.

– Algorithm: BFS or DFS

– Claim: If no vertices were visited, DFS_visit(u), BFS_visit(u) will visit every vertex in the same connected component as u.

Proof: induction on the length of path.

if (u,v) is an edge

DFS-visit / BFS_visit will visit v ✓

assume DFS/BFS will visit v if the path between u, v have length ≤ k.

if path between u, v has length k+1

take w, one step before v

path between u, w has length k

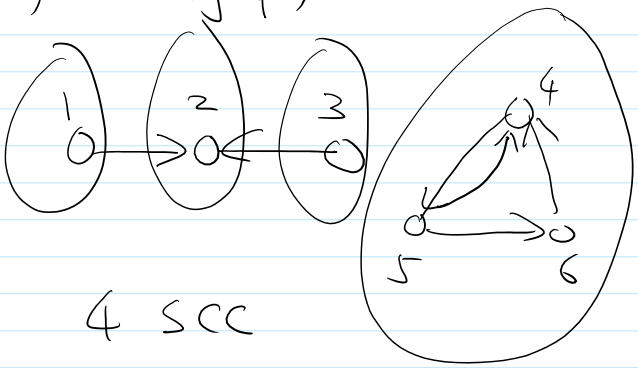
w must be visited (by induction hypothesis)

DFS_visit / BFS_visit will visit all neighbors of w □

– Strongly connected Components (SCC, directed graph)

– SCC: subset of vertices that are mutually connected (by directed path) Cannot add more vertices and presevre connectivity.

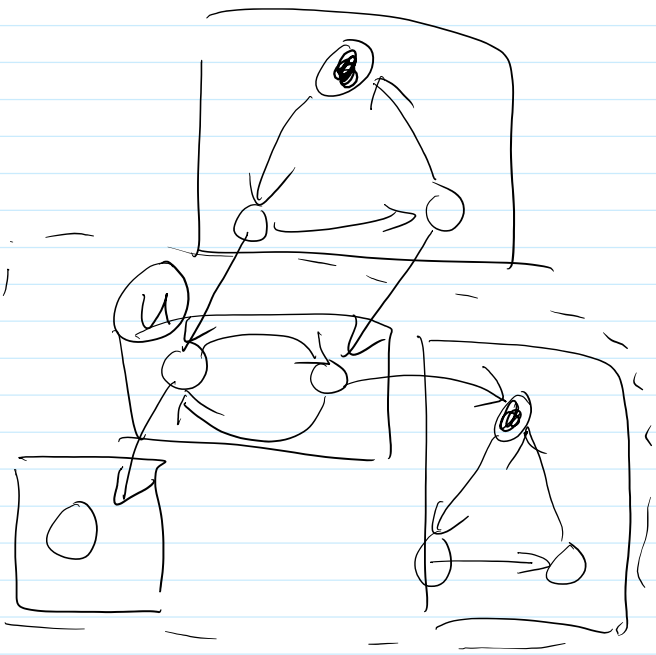
– Goal: find strongly connected components in graph G.



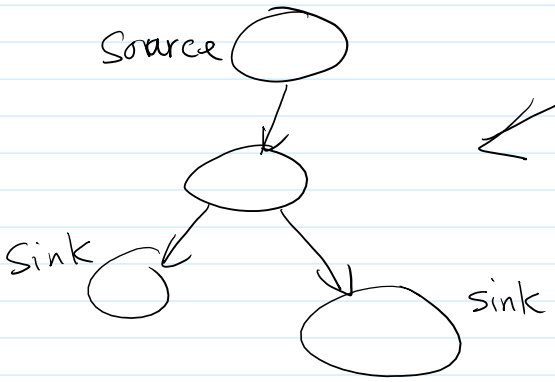
4 SCC

– Idea: use DFS / BFS

- Idea: use DFS/BFS
  - Q: if start from u, what vertices do we visit?
    A: SCC of u, and possibly some other SCC.



- contraction
  - group vertices to "big" vertex



- Observation: contracted graph has no cycles.

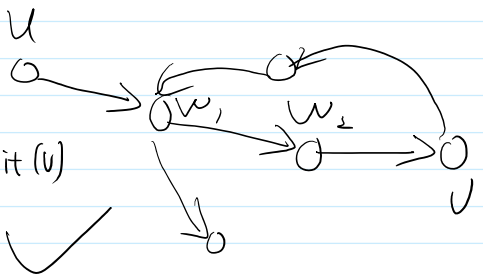
Sink: a vertex (SCC) that do not have any outgoing edges.

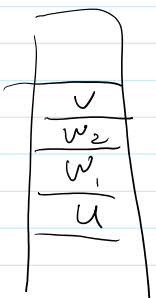
- If we start at a vertex in a sink component, will only visit points that are in the same SCC.

Question: How to find a vertex in sink component?
A: DFS on inverse graph, last one in post order will be in sink.

Claim: If $u \rightsquigarrow v$, but $v \not\rightsquigarrow u$, then u is later than v in post order of DFS.

Proof: If DFS_visit(v) is first.
it will return after all vertices reachable from v are visited.
u is not visited even after DFSvisit(v) returns. ✓





If DFS_visit(u) is first
v will be a children of the DFS_tree routed at u.
u is after v in post order ✓

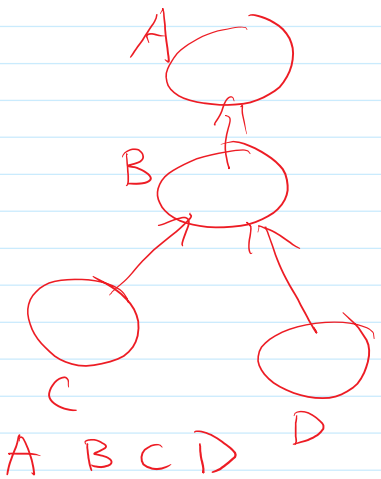
Claim: If v is last vertex in post order. if $u \rightsquigarrow v$

Claim: If v is last vertex in post order. if $u \rightsquigarrow v$ then $v \rightsquigarrow u$.

$\Rightarrow$ last vertex in post order is in a source!

we can invert direction of edges to swap source/sink.

Let G' be G with edges reversed
Do DFS on G', keep the post order
While graph G is not empty
  let u be the remaining vertex last in post order.
  DFS_visit(u) (in original graph G)
  Claim all vertices visited are in same SCC.
  Delete all visited vertices.

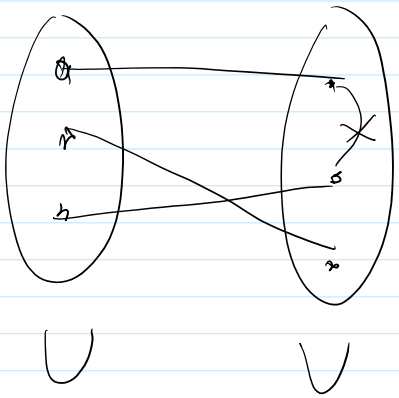


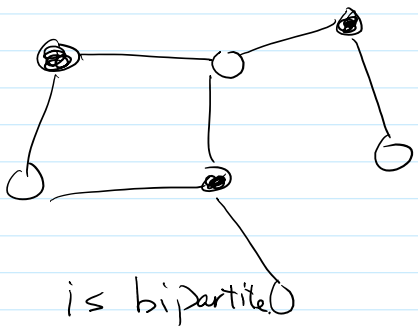
running time: $O(n+m)$ first DFS
$O(n+m)$ while loop

$O(n+m)$

- bipartite graph

A graph is bipartite if vertices can be partitioned into U, V every edge connects to one vertex in U, one in V

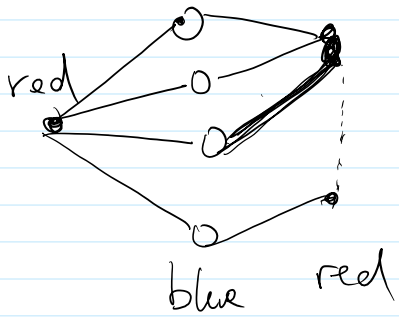




- Goal: decide whether graph G is bipartite.

$\Leftrightarrow$ can we color vertices using two colors (red/blue), s.t. every edge connect two different colors.

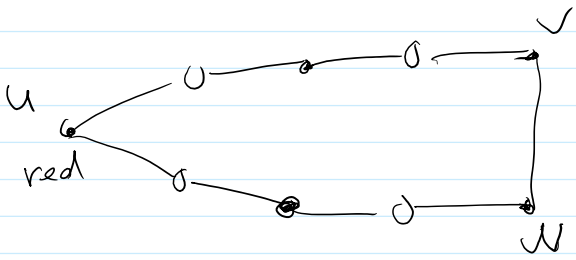
```
BFS_color(v)
  put v into queue
  color v red. mark v visited.
  while queue is not empty
     u = dequeue
     for all edges (u,w)
        if w is not visited
            color(w) = − color(u)
            put w into queue
            mark w visited
        if color(w) = color(u)
            Fail.
```



Theorem: Connected Component is bipartite iff BFS_color does not fail.

Proof: BFS_color succeeds: output a coloring
all edges are checked ✓

BFS_color fails
look at BFS tree (v,w) is the failed edge



u ~~~ v — w ~~~ u
is a cycle of odd length.
no way to color an odd cycle □