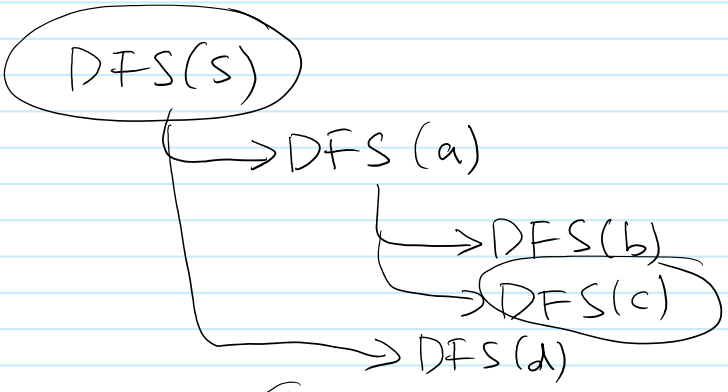
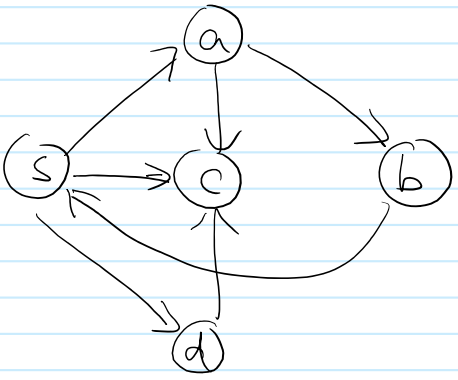
Lecture 11 DFS/BFS

Monday, February 24, 2020 2:53 PM

- DFS and type of edges





s a b b c c a d d s

enter enter enter leave enter leave leave enter leave leave

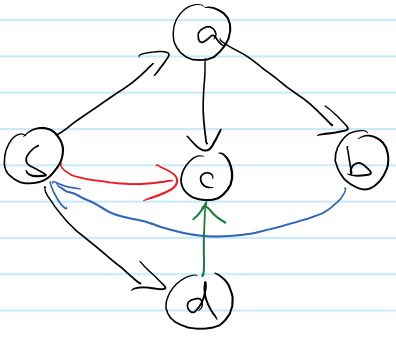
pre-order s a b c d

post-order b c a d s

[( ])

a b a b

enter enter leave leave



- type of edges

① tree edges (s,a), (a,b), (a,c), (s,d)

② forward edge (s,c)

s is an ancestor for vertex c

first vertex

s ... c ... c ... s

enter enter leave leave

③ backward edge (b,s)

s is an ancestor for vertex b

second vertex

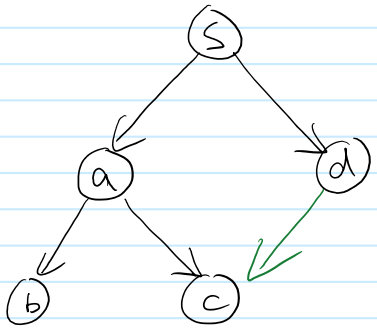
s ... b ... b ... s

enter enter leave leave

④ cross edge (d,c)

(c) d, c do not have ancestor/descendant

(4) cross edge (d, c)



d, c do not have ancestor/descendant relationships on the tree

c ... c ... d - - d
enter leave enter leave

- cycle finding

Proof of correctness:

① if alg claims to find a cycle, there is indeed a cycle

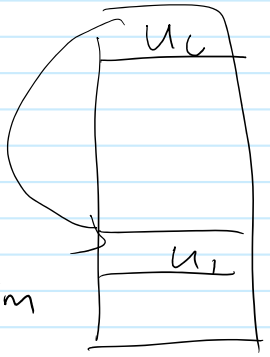
② if alg claims the graph has no cycle, there is no cycle

assume towards contradiction that there is a cycle in graph: $u_1, u_2, \cdots, u_L$

$(u_1, u_2), (u_2, u_3) \cdots, (u_{L-1}, u_L), (u_L, u_1)$ are edges in the graph.

let u_p be the first vertex visited by DFS on this cycle. w.l.o.g assume $u_p = u_1$

want: when u_L visited, u_1 is on the stack.



Claim: when every other vertex $(u_2, u_3, \cdots, u_L)$ is visited, u_1 is on the stack

assume towards contradiction that claim is false. there is a first vertex u_p s.t. when u_p is visited, u_1 is not on the stack.

know: when u_{p-1} is visited, u_1 is on the stack when considering the edge u_{p-1} to u_p

case ① u_p has not been visited

alg will call DFS_visit(u_p)

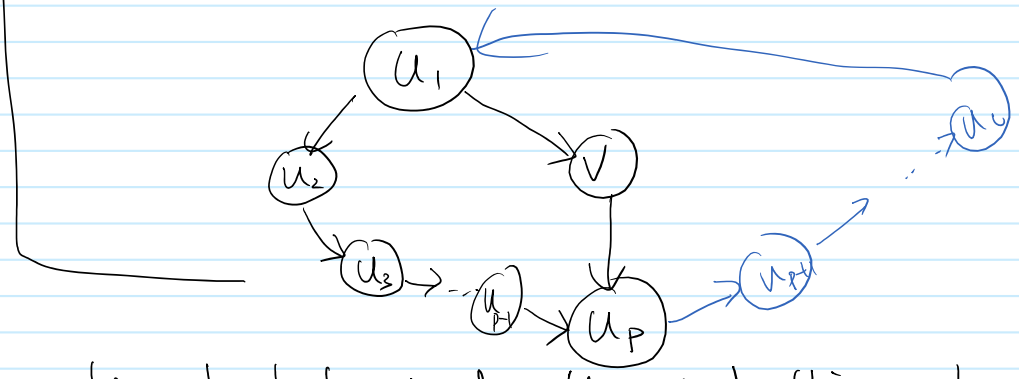
case ② u_p has been visited.

alg will call DFS_visit(u_p)

case (2) u_p has been visited.

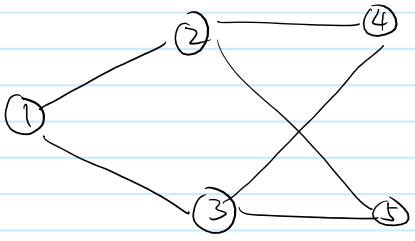
Know u_p were not visited when u_1 enters stack

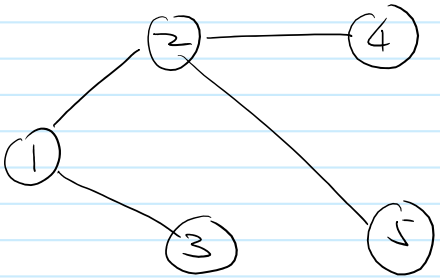
Now u_p has been visited when u_1 is still on stack

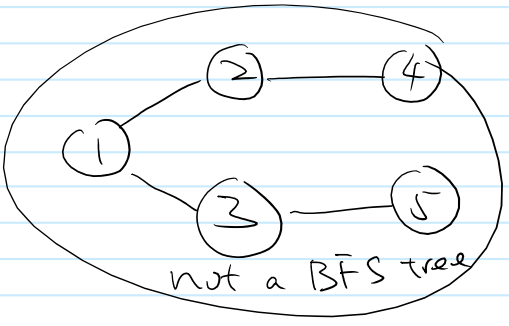


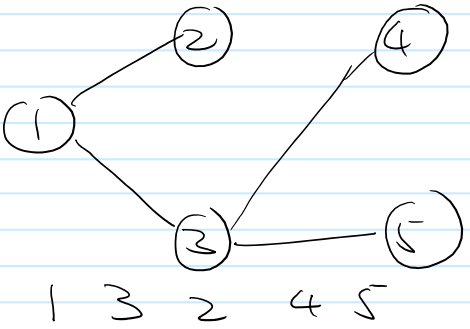
when alg looks at edge (u_v, u_1), this a backward edge alg can always find a cycle.

- BFS tree









- topological sort
  - algorithm: output the inverse of post-order
  - proof of correctness: assume towards contradiction that the algorithm is not correct. then there must be an edge (u, v) where v comes later than u in

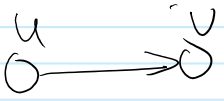
post order.

consider the DFS procedure

case ① if u is visited before v

when the edge (u,v) is considered

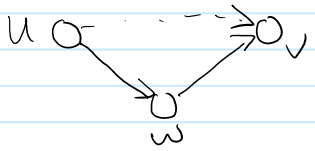
case (1.1) if v is not visited, DFS will visit v,



(u,v) is a tree edge, and DFS(u) returns after DFS(v)

this contradicts with assumption that v comes after u in post-order.

case (1.2) if v is already visited, then v must be visited between u is visited and edge (u,v) is considered,



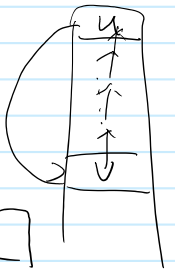
so u is on the stack when v is visited,

this contradicts with the same assumption

case ② if v is visited before u, since v is after u in post-order,

the only possible sequence of events is

v enter u enter u leave v leave



therefore when u is visited v is on the stack

in this case (u,v) is a backward edge and forms a cycle. This contradicts with the fact that the input graph is acyclic.

□