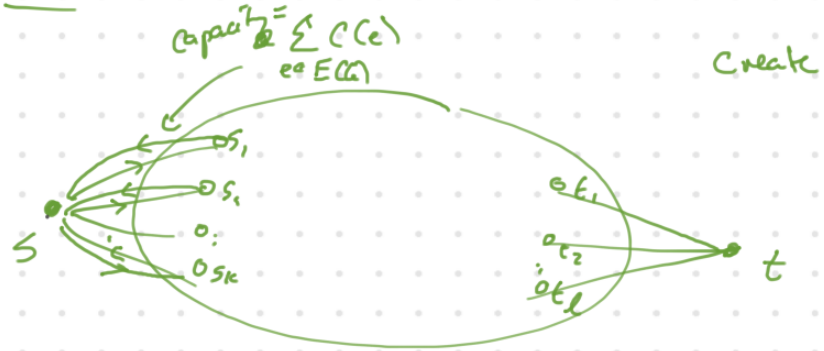
1) Network with sources $s_1, \dots, s_k$, & sinks $t_1, \dots, t_\ell$ & we want to find a flow maximizing

$$\sum_{\substack{(s_i, x) \in E(G) \\ i = 1, \dots, k}} f(s_i, x)$$

Sol

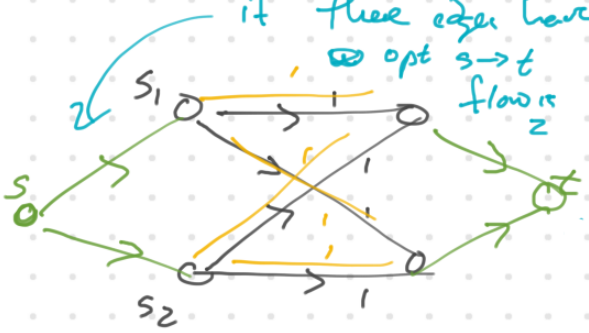


add $s \sim s_i$ $\forall i$ w/ (s, s_i) having capacity $\sum_{e \in E} c(e)$

& similarly, $t \sim t_j$ $\forall j$ w/ (t_j, t) having capacity $\sum_{e \in E(G)} c(e)$

Now find an $s \to t$ flow w/ Edmonds Karp.

Note: we can't use $\max_{e \in E(t_i)} c(e)$ for the capacity of edges from $s \to s_i$



opt flow $(s_1, s_2) \to (t, t_2)$

Flow has value 4

2) Network G w/ s, t & capacities $c(e)$ We want to find a min capacity s-t cut w/ minimum # of edges

we know that in the Edmonds Karp alg., when we can't find an s-t path in the residual graph that the set U of vertices reachable from s will define a min cut of capacity equal to the value of the flow.

Hint try to modify the capacities to ensure the min cut is a min cut w/ a minimum # of edges.

No class tomorrow

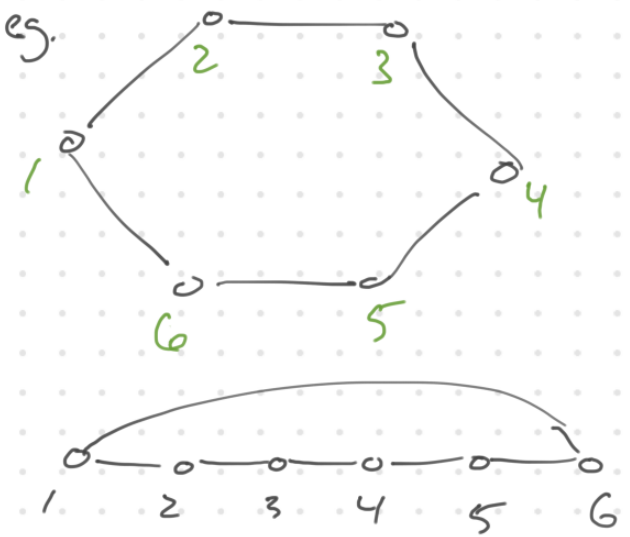
(class as normal on Wed 6)

Recall an s-t labeling is an ordering of vertices $x_1, \dots, x_n$ ST

- $x_1 \sim x_n$
- $\forall i$, $2 \le i \le n-1$, the vertex x_i has neighbor in each of

The sets

$\{x_1, \dots, x_{i-1}\}$ & $\{x_{i+1}, \dots, x_n\}$



Prop if G is 2-conn, Then

$\forall$ edges $xy \in E(G)$,

$\exists$ an s-t labeling of

G st $x = s$, & $y = t$. And

we can find it in time $O(n+m)$

we will need a modification of DFS

Let T be a DFS-tree with vertices labeled by their order of discovery in DFS-tree. let $Low[v]$

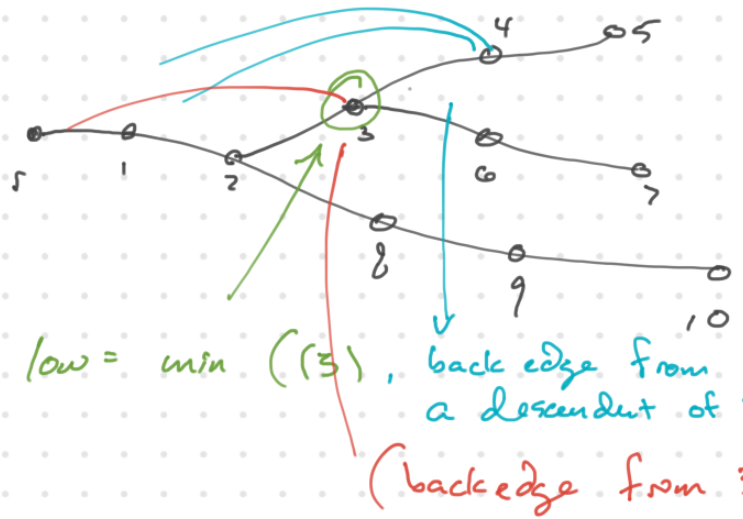
$= \min\Big(\{v\}, \{w : (y,w)$ is a backedge of search tree & y is a descendant of $v\}$,

$\{w : (v,w)$ is a back-edge of the search tree.$\}\Big)$

Obs if G is 2-conn,

Then $low(v) < v \quad \forall\, v$ distinct from the root.

pf



$\overline{X}$:= {descendants of v} since by 2-connectivity, v is not a cut vertex separating $\overline{X}$ from the root, $\exists$ a back edge from a vertex of $\overline{X}$ to an ancestor of v $\Rightarrow$ $low(v) < v$.

This argument works except in the case when $\overline{X} = \emptyset$ ie v is a leaf. But if v is a leaf, by 2-connectivity, v has a nbhr w distinct from parent $[v]$ $\Rightarrow$ $low(v) < v$

∎

```
Procedure DFS(G)
  procedure Search(v)
    begin
      mark v as "old"
      DFS_Num(v) = count
      count ++
      Low(v) = DFS_Num(v)
      For each w ∈ V Do
        if w is "new" Do
          add (v,w) to tree
          parent[w] = v
          Search(w)
          low(v) = min(low v, low(w))
        else if w ≠ father(v)
          low(v) = min (low(v),
                        DFS_num(w))
    end
  Begin T = ∅ , T our DFS tree
    count = 1
    mark each vertex of G as "new"
    v a vertex
    Search(v)
end.
```

Given the DFS_num, low(v) for each v of DFS tree

We define a function Path(v) for each vertex v.

Path(v) is a path from a vertex v to a vertex marked "old" in a graph w/ vertices & edges marked either "new" or "old"

CASE 1 ∃ a back edge (v, w) which is marked "new"

mark (v, w) as "old" &

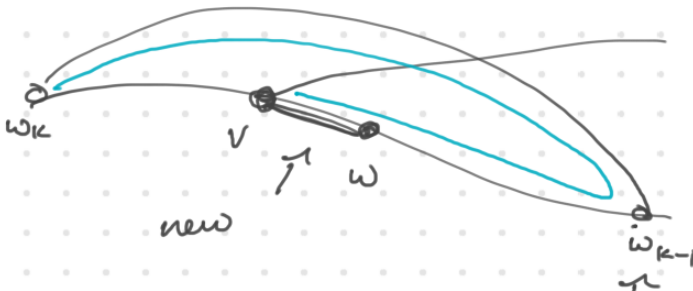
Path(v) returns :" v, w"

CASE 2 ∃ a tree-edge (v, w) (w a descendant of v) which is marked new

$w_0, w_1, \ldots, w_k$ be path

$w_0 = w$

$w_k = low(w)$



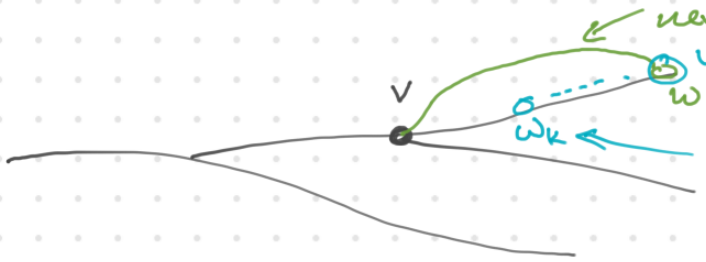
descendant of w defining low(w)

~~Path~~ return

$v\ w_1, \ldots, w_k$ as path(v)

& mark all edges & vertices on the path as "old"

CASE: $\exists$ a back edge (w, v) with $DFS\_num(w) > DFS\_num(v)$



consider the path

w_0 w_1 ... w_k
$\|$ $\uparrow$
w first "old" vertex we find walking from w up to v

return $v\ w_0\ w_1 \dots w_k$

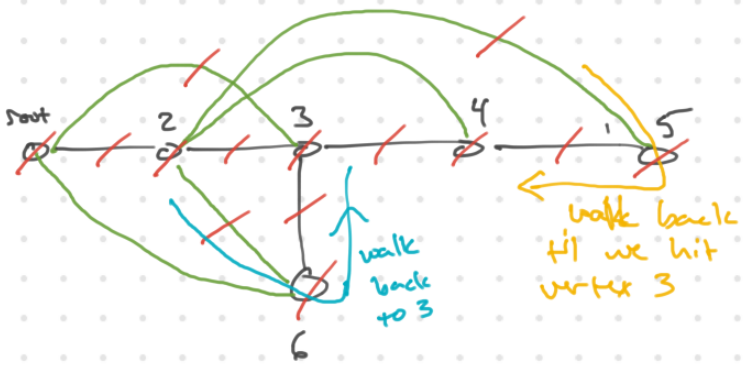
mark all edges & vertices as "old"

CASE ow all edges incident to v are old & return $\emptyset$ for the path.

Obs if we repeatedly apply this function Path at a vertex, $\Rightarrow$ it will eventually return $\emptyset$

because as we repeatedly apply the function, we're always marking an edge incident to v as old

until no such edge exists



everything marked new except root + (r, 2)

Path (2) - we need an edge giving the certificate of low (3) = root

we're in CASE 2

w/ path @ 3, root certificate of low(3) = root

& Path returns 2, 3, root + updates dd edges

apply path (2)

Case 3 : ∃ a back edge landing at 2 which is new

- pick (6, 2)

walk back in tree from 6 til first time we encounter a "old" vertex - hit (3)

path returns 2, ~~6~~, 3 & marks as old 2, 6, 3, 26, 36

apply Path(2) - pick edge (2,5)
walk up T from 5 to first
~~unmarked~~ "old" vertex which is 3
& path will be
2, 5, 4, 3 & we mark as "old"
5, 4, 3, 25, 54, 43

apply Path(2) - one final back
edge landing at 2 - namely
(2,4)
Path(2) returns 2, 4 &
marks as "old" 2, 4, 24

now all edges incident to
2 are marked "old" so
when we apply Path(2) = ∅
keep going & apply path(3) = ∅
path(4) = ∅
path(5) = ∅
path(6) = - (we're in
CASE 1)
= ~~(6, T)~~ 6, root
& mark 6 1 as "old"

ST-num (G, s, t)

run modified DFS search to get tree, DFS_num, & low(v) w/ root = t & s w/ DFS_num = 2

mark every vertex & edge as new except s, t & st which are old

S a stack; S.push(t), S.push(s)

Count = 1

v = S.pop()

while v = t DO

if Path(v) = ∅

St.num(v) = Count; count++

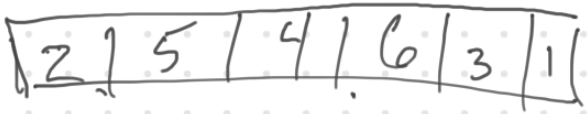
else let v u₁ u₂ ⋯ uₖ w = Path(v)

S.push(uₖ), S.push(uₖ₋₁), ⋯, S.push(u₁), S.push(v)

st_num[t] = count

v = S.pop()

look at example we just did.



ST-num

S = [2, 1] → path(2) = (2, 3, 1)

[2, 3, 1] → path(2) = 2, 6, 3

[2, 6, 3, 1]

→ path(2) = 2, 5, 4, 3

[2, 5, 4, 6, 3, 1] → path(2) = 2, 4

[2, 5, 4, 6, 3, 1] → path(2) = ∅

[5, 4, 6, 3, 1]

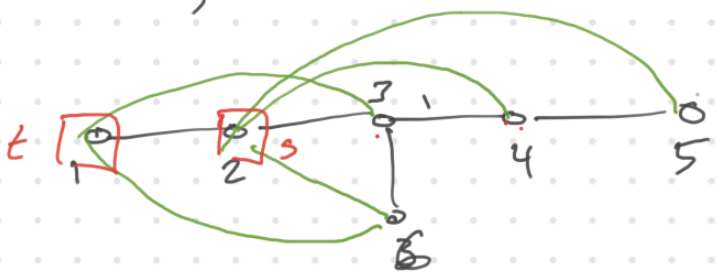
[4, 6, 3, 1] → path(5) = ∅

[4, 6, 3, 1] → path(4) = ∅

[6, 3, 1] → path(6) = 6, 1

[6, 3, 1]

2, 5, 4, 6, 3, 1 is an s-t ordering of G



Thm alg returns an s-t ordering (and does so in time $O(n+m)$)

pf t will always be last in ordering because it's the bottom vertex in the stack S, it is an st-ordering

s starts at top of stack & keeps being reinserted at top of stack

until path(s) = $\emptyset$ =>
s first in the ordering.
every time a vertex is inserted for first time in S, it is an internal vertex of a path, so it has a neighbor above & below in the stack, so it will have a neighbor before & after in the ordering. ie