CS3000: Algorithms & Data
Drew van der Poel

Recitation 6:

- DFS, Topological Ordering, Dijkstra’s

June 14, 2021

Problem 1

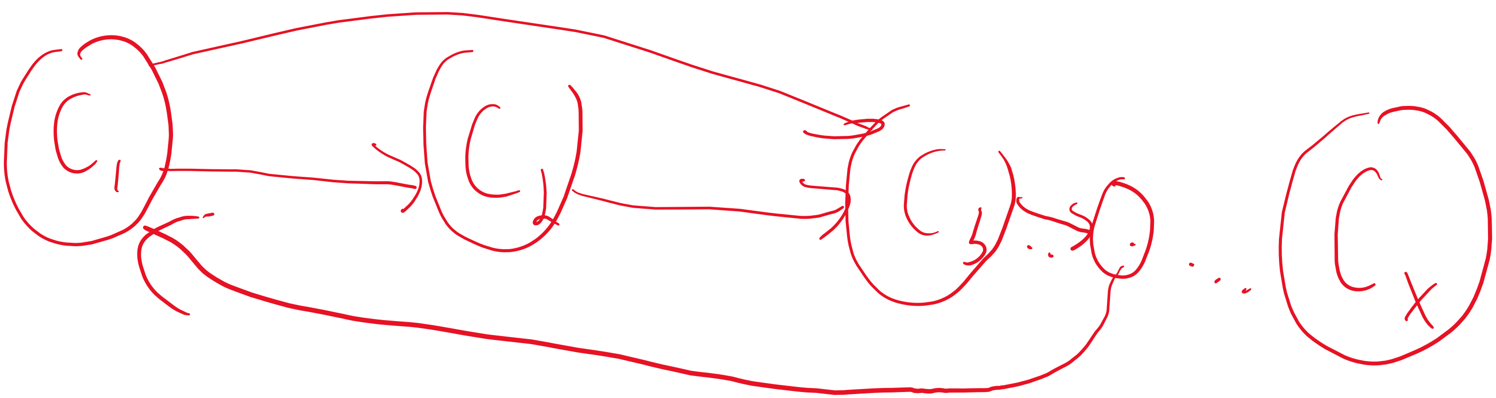
Recall that the strongly connected components algorithm takes as input a directed graph $G = (V, E)$ and computes all the strongly connected components of G in $O(n+m)$ time where n is the number of vertices and m the number of edges of G.

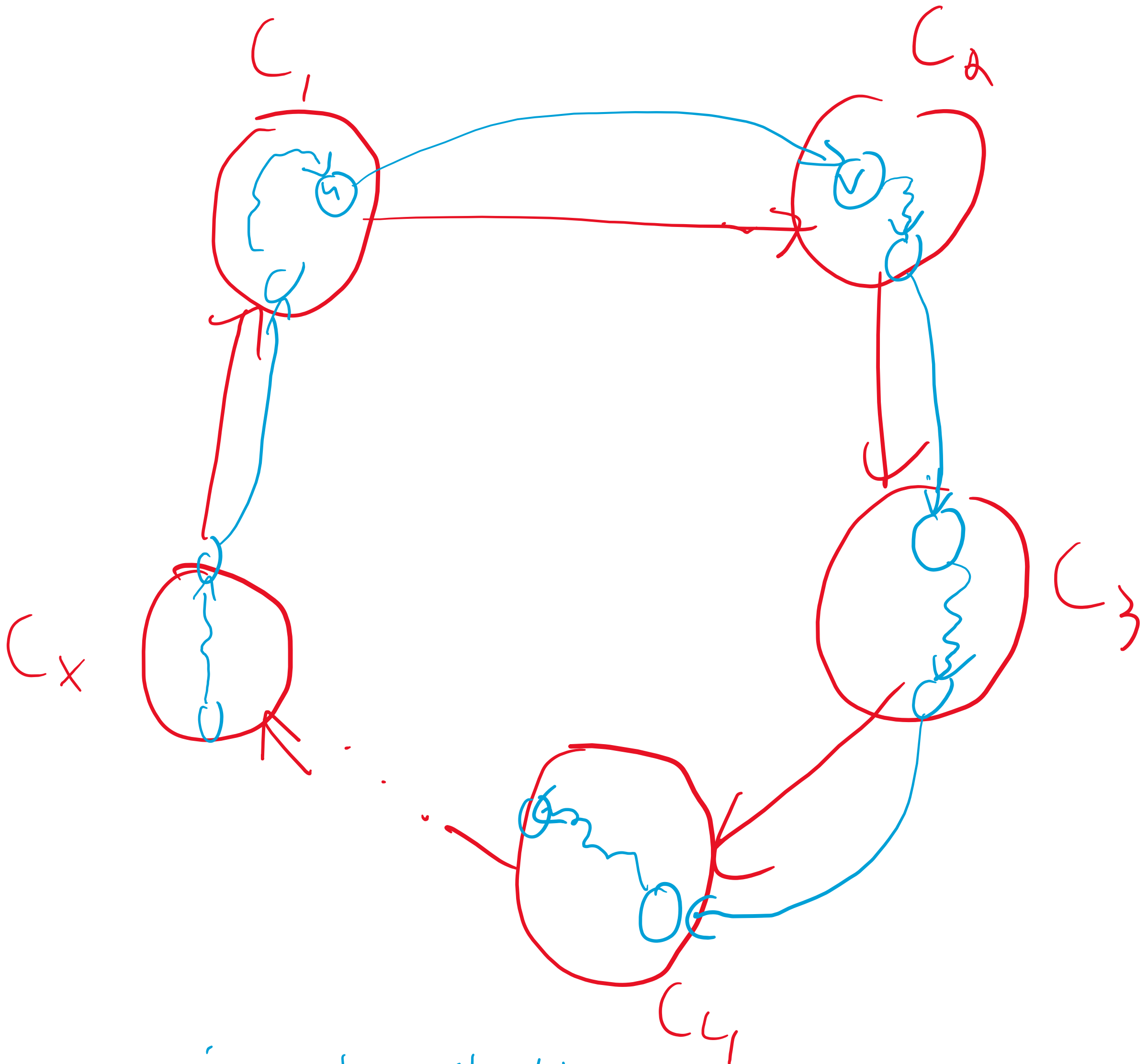
Define the SCC graph $G^{\mathrm{SCC}} = (V^{\mathrm{SCC}}, E^{\mathrm{SCC}})$ as follows. V^{SCC} contains a vertex for each SCC of G. E^{SCC} contains an edge from vertex representing SCC C_1 to a vertex representing SCC C_2 if there is a vertex $u \in C_1$ and a vertex $v \in C_2$ with an edge $(u, v) \in E$. Prove that G^{SCC} is a DAG.

"SCC graph"

Assume false, G^{SCC} is not acyclic

$\rightarrow \exists$ cycle in G^{SCC}





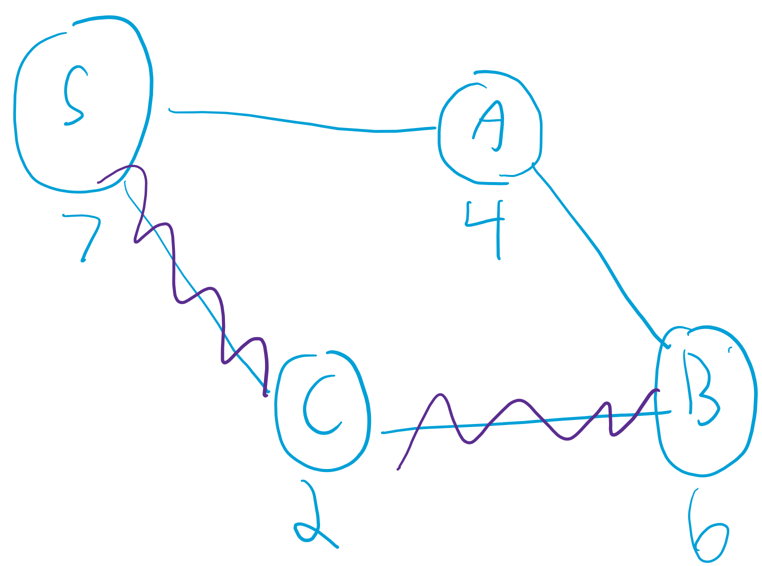
u is reachable from v

v is reachable from u

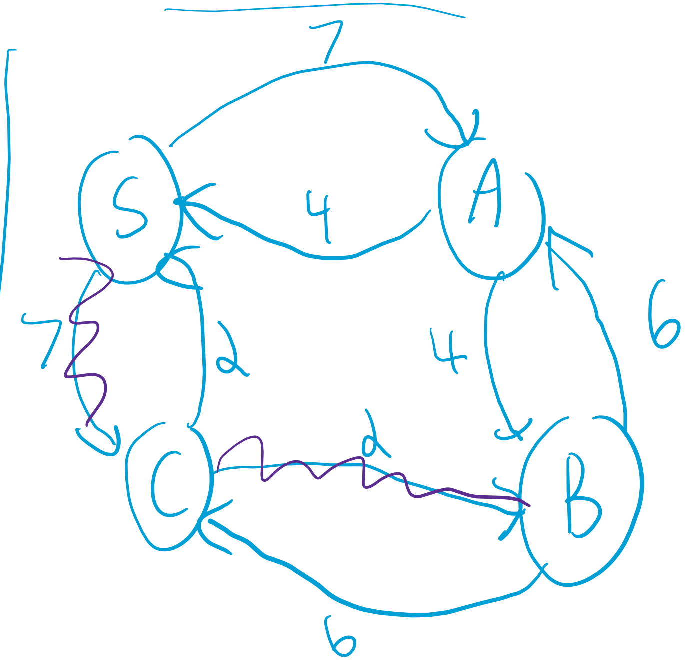
$\therefore$ u and v should be in same SCC $\rightarrow\leftarrow$

Problem 2

Suppose you are given a connected undirected graph with weights on vertices (rather than on edges) and you are asked to compute the single-source shortest paths from a given source vertex. Here, the length of a path is defined as the sum of the weights on the vertices comprising the path. Show how to solve this problem by creating an equivalent instance of the standard single-source shortest paths problem on directed graphs with *weights on edges*.



Vertex weighted



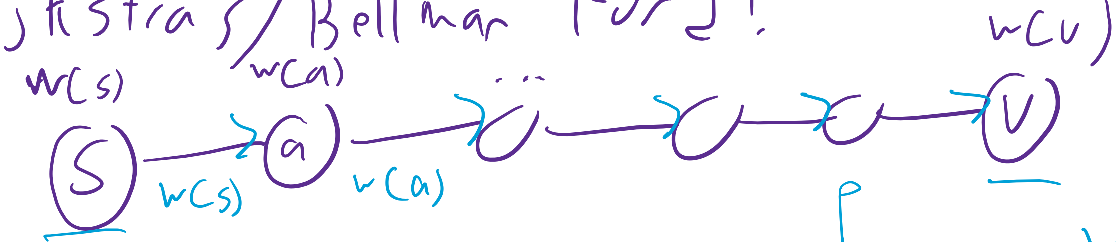
Edge weighted

Mapping:

for vertex v of weight $w(v)$,

$\forall u \in NEIGH[v]$ add an edge from v to u of weight $w(v)$

Solve w/ Dijkstras/Bellman Ford!



Length of shortest s-v path in V SSSP:

$$\ell(P) = \sum_{x \in P} w(x) = \underline{\underline{OPT_v}}$$

||

$$\ell(P) = OPT_v - w(v)$$

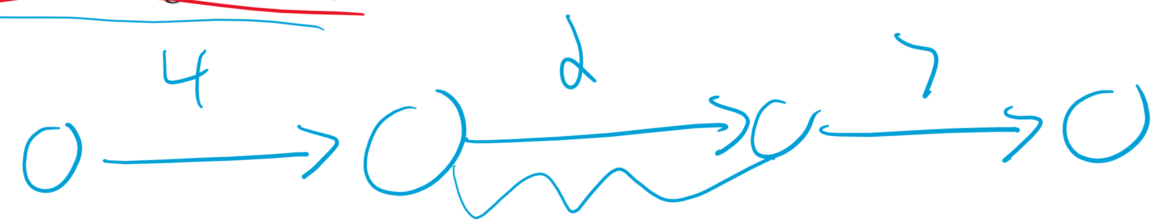
" (E) SSSP

Problem 3

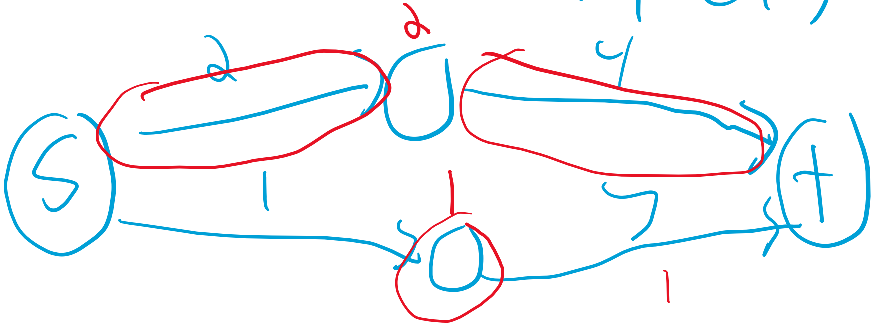
A communication network is often modeled by a weighted directed graph G, in which the vertices of G represent the nodes of the network, the edges of G represent the communication links of the network and the weight of an edge is the bandwidth of the link. We define the *bottleneck bandwidth* of a path p as the bandwidth of the minimum-bandwidth link in p.

Design and analyze a polynomial-time algorithm to determine a path with the *largest bottleneck bandwidth* from a given node s to a given node t. If no path from s to t exists, then your algorithm must indicate so. You may assume that the bandwidth of every link is positive. (*Hint:* Modify Dijkstra's algorithm.)

p



bottleneck-bandwidth(p) = 2



bb(top) = 2

bb(bottom) = 1

Bandwidth

$s \xrightarrow{d[u]} u \xrightarrow{w(u,v)} v$

bottleneck bandwidth = min(d[u], w(u,v))

~~Dijkstra~~(G = (V,E,{ℓ(e)}, s):

Let Q be a new heap (Max Heap)

Let parent[u] ← ⊥ for every u

Insert(Q,s,~~0~~) ∞ , Insert(Q,u,~~∞~~) 0 for every u != s

While (Q is not empty):

(u,d[u]) ← Extract~~Min~~(Q) Max

For (v in out[u]):

d[v] ← Lookup(Q,v)

If (d[v] ~~>~~ < ~~d[u] + ℓ(u,v)~~): min(d[u], w(u,v))

Increase **~~Decrease~~Key(Q,v,~~d[u] + ℓ(u,v)~~)** min(d[u], w(u,v))

parent[v] ← u

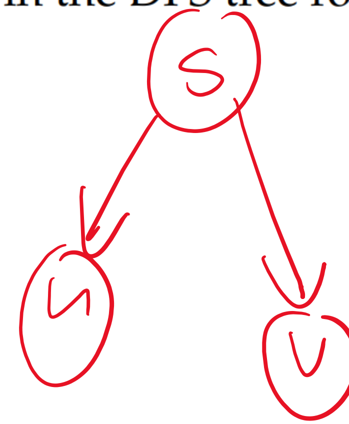
Return (d, parent)

In Bandwidth, keys - nodes

values - largest bottleneck bandwidth of a path to that node (lower bound)

Problem 4

(a) Give an example of a directed graph G, including vertices s, u and v such that (i) there is a path from u to v in G, and (ii) there is a DFS traversal starting from s such that u is discovered before v, yet v is not a descendant of u in the DFS tree rooted at s.



(b) Let G be a directed graph, and s be a vertex in G. Suppose a DFS traversal from s visits all vertices of G and has the property that there are no cross or forward edges. Prove that for every vertex v in G, there is exactly one simple path from s to v in G.

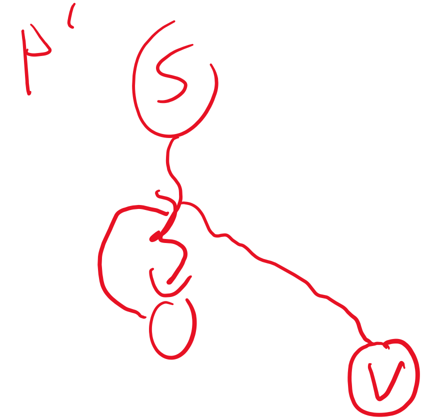
All edges are either tree or backward,

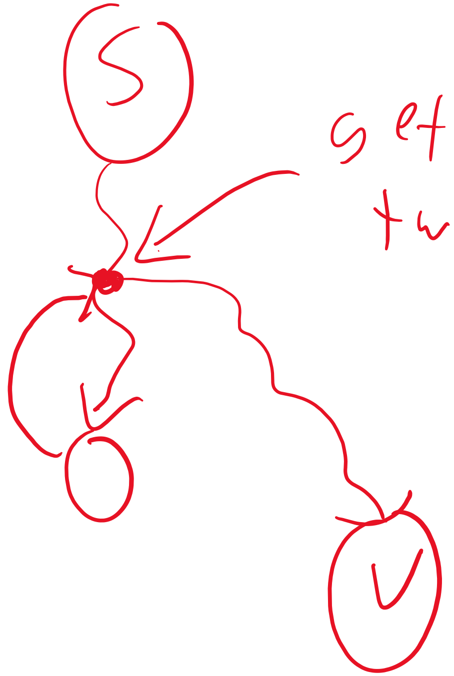
$> 0:$ $\forall v$, $\exists$ s-v path consisting only of tree edges!

S1: Assume false, there are two simple s-v paths in G.

Let p be the one using tree edges,

Let p' be the other one, → uses back edges





gets visited twice ∴ P' is not a simple path

$\rightarrow\!\!\lightning\!\!\leftarrow$