

CS 331, Fall 2026

Lecture 11 (10/5)

Today: - Graph search

- BFS

- DFS

Graph Search (Part V, Section 1)

We have already seen some graph algos.

- SSSP on DAGs
- APSP (Floyd-Warshall)
- MST (Kruskal)

Basic theme:

Graph structure + algos bag

(lots to come!)

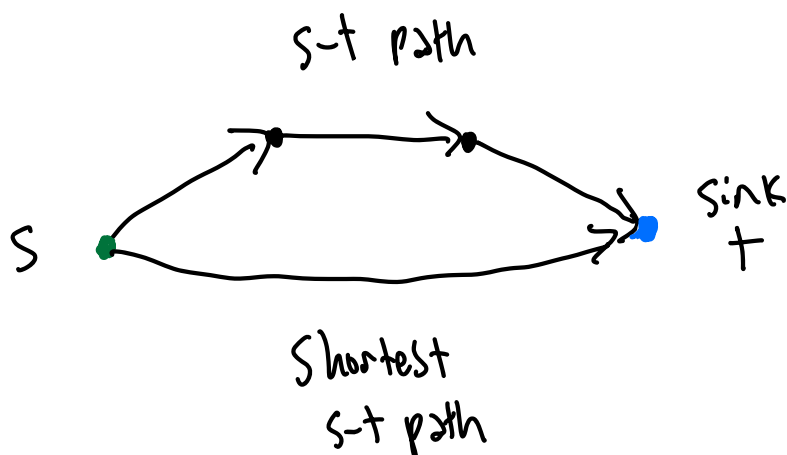
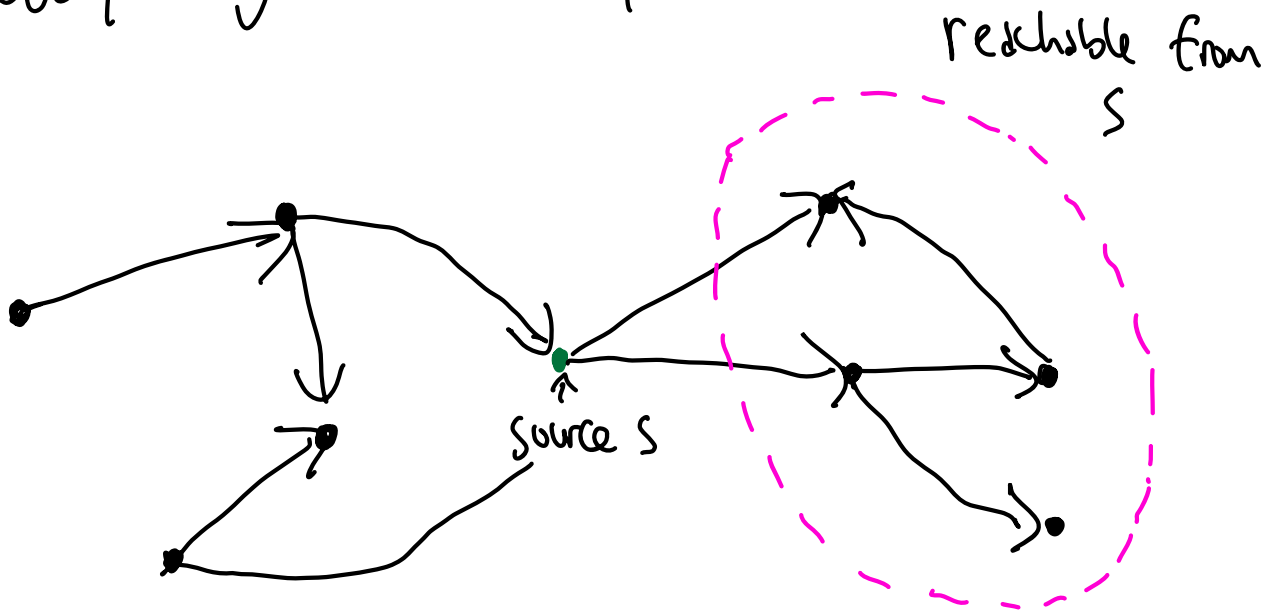
(recursion, DP,
greedy, data structures)

Example

MST: "greedy stays ahead"

- exchange lemma: k CC's
 $n-k$ edges
- data structure for maintaining CC's

Today: graph search



Graph Search ($G = (V, E), s$):

$S \leftarrow \{s\}$

$R \leftarrow [\text{False for } v \in V]$

While $S \neq \emptyset$:

$v \leftarrow \text{any element of } S$

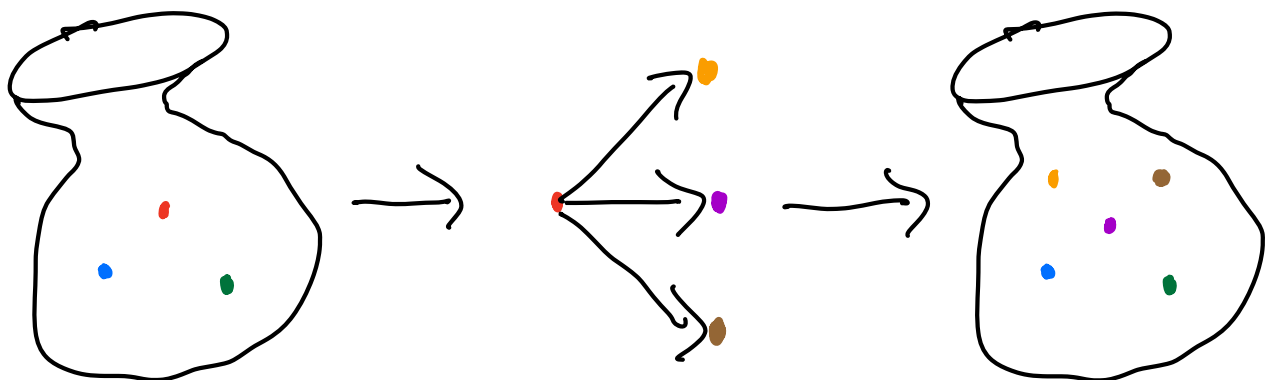
$S \leftarrow S \setminus v$

If $R[v] == \text{False}$:

$R[v] \leftarrow \text{True}$

For $(v, u) \in E$: $S \leftarrow S \cup \{u\}$

Return R



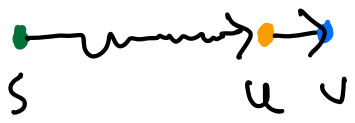
Claim: Graph Search solves reachability.

Proof: Every vertex set to True ≤ 1 time.

$R(v) = \text{True}$
 $\Downarrow$
 v reachable

Induct on when set to True.
Base Case: s first.
Induct: If $R(u) = \text{True}$,
 v was added b/c $R(u) = \text{True}$.
By assumption, u reachable $\Rightarrow$ so is v .

v reachable
 $\Downarrow$
 $R(v) = \text{True}$

Induct on shortest path distance.
If 0: $R(s) = \text{True}$.
If k : let path be 
 $R(u) = \text{True}$
Then, v added to S . Will become True

Breadth-First Search (Part V, Section 2.1)

How to implement Graph Search?

Need data structure for S.

- Insert an element
- Remove an element

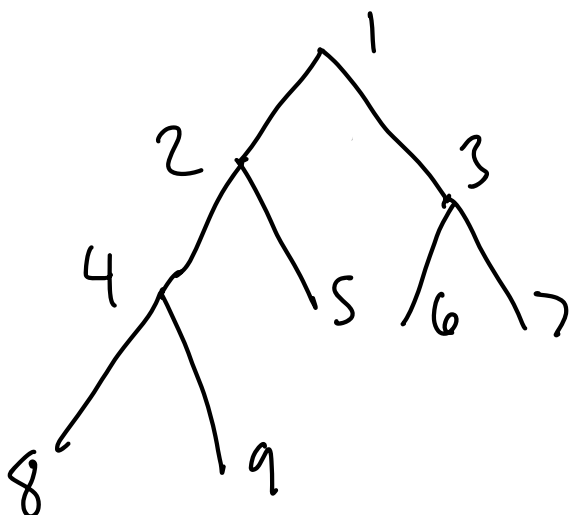
(O2: Linked List does both in $O(1)$ time.

Queue = BFS

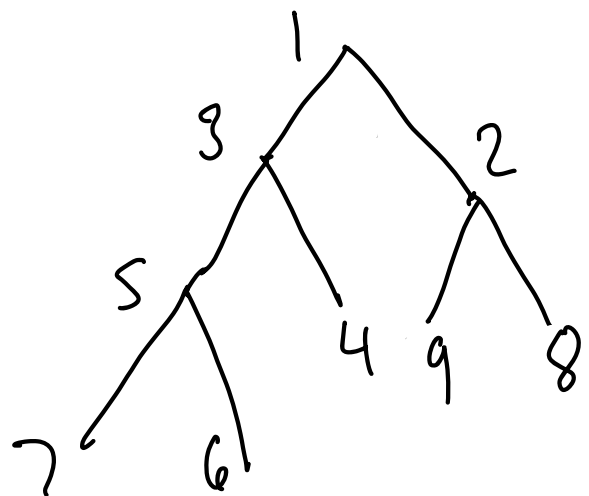
Push() :    

Stack = DFS

Push() :    



BFS



DFS

Connected Components

Say $S \sim T$ if connected in undirected graph

Then $\sim$ is equivalence relation

- reflexive



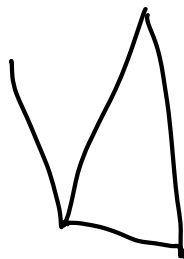
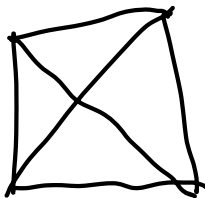
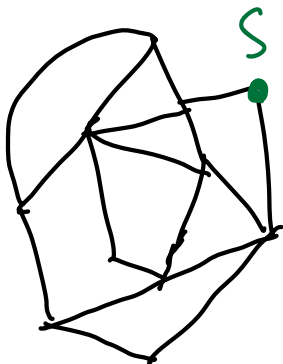
- symmetric



- transitive



Partition into Connected Components (CCs)



reachable from S

Runtime of BFS: - every edge used $\leq 2x$.

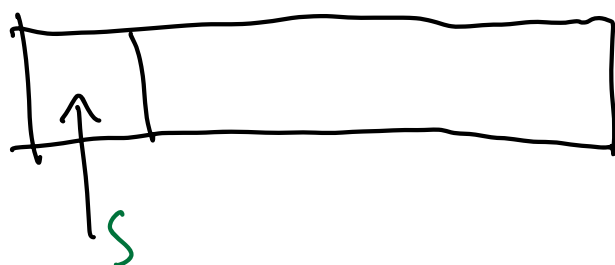
- $2M_{C_s}$ total vertices

- $M_{C_s} = \#$ edges in C_s : CC of s .

$O(M_{C_s})$

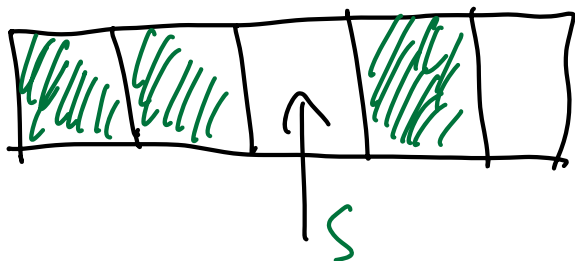
Using Queue
(or Stack)

CC algo:



Vertex list:
total move $O(n)$

Graph Search(s) ... $O(M_{C_s})$



$$O(n) + \underbrace{\sum O(M_{C_s})}_{O(n)}$$

Unweighted SSSP

Let $p(v)$ be the parent of v :

v added for the first time

due to the edge $(p(v), v)$

Claim: $d(s, v) = 1 + d(s, p(v))$

With Claim, simple mods compute SSSP!

- For $(u, v) \in E$: S.Push((u, v))
include parent info

- If $R[v] == \text{False}$:

$$D[v] = D[u] + 1$$

memoized

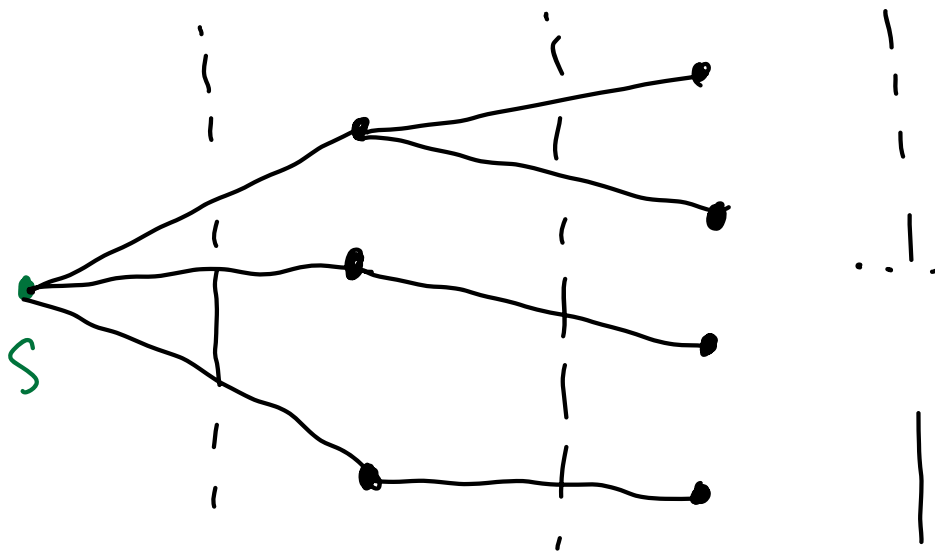
Proof of Claim: Let $R_0 = \{s\}$

$$R_1 = \{d(s, \cdot) = 1\}$$

$$R_2 = \{d(s, \cdot) = 2\}$$

$\vdots$

Claim is that they form "frontiers" in S :



$R_0 \quad \dots \quad R_1 \quad \dots \quad R_2 \quad \dots$

Queue, restricted to first visits

Base case: $R_0 \checkmark$

Induct:

$P_0 P_1 P_2 \dots$	P_i
---------------------	-------

current queue

let $v \in P_i$ be dequeued

All (v, u) have $d(s, u) \leq i + 1$

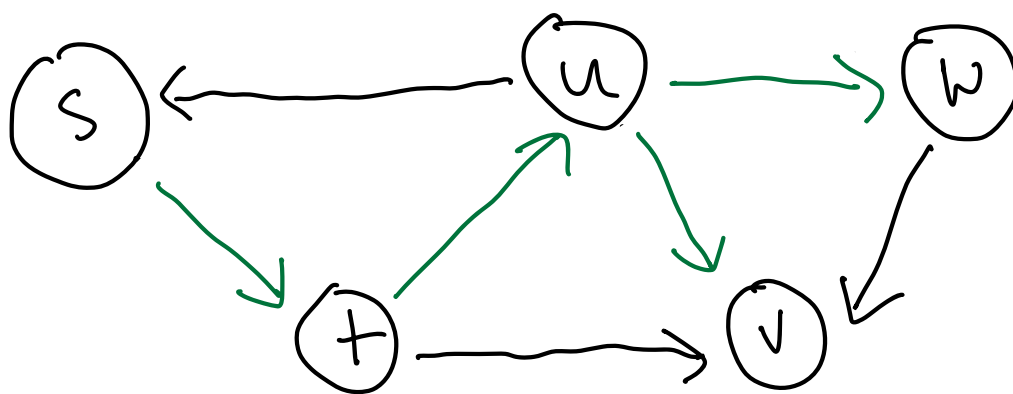
If $d(s, u) \leq i$ then reached (induction). $\checkmark$

Depth-First Search (Part V, Section 2.2)

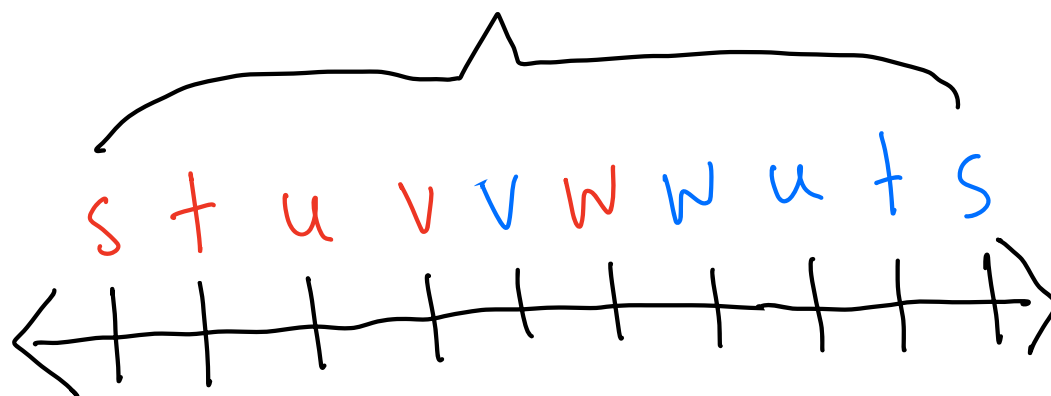
Each vertex has:

- start time (enter the stack)
- end time (leave the stack)
 - when all children done executing
 - implementable at no overhead (see notes)

Example



Stack: Duration of s on the stack



Preorder: s + u v w

Postorder: v w u + s

Key claim:

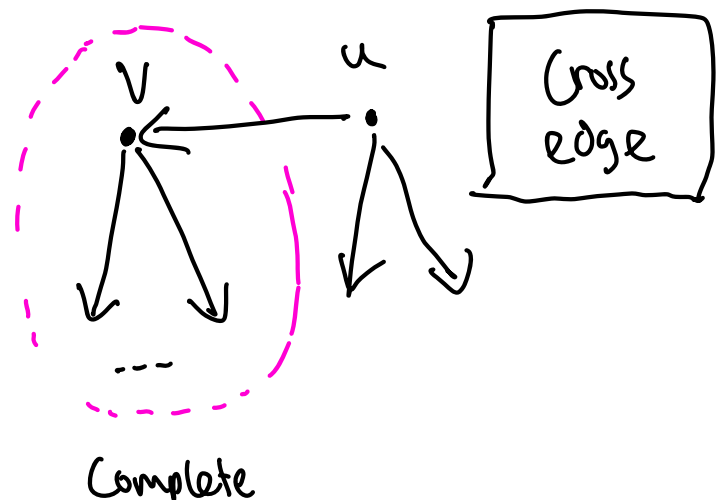
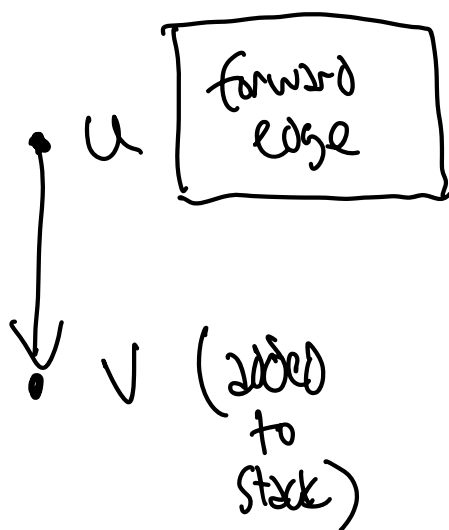
If input is DAG,
then postorder reversed
= topological order!

Lemma: Suppose $u \rightarrow v$,
 u, v reachable from s .

If $u.\text{finish} < v.\text{finish}$, $\exists$ cycle

Proof: three cases when u reached

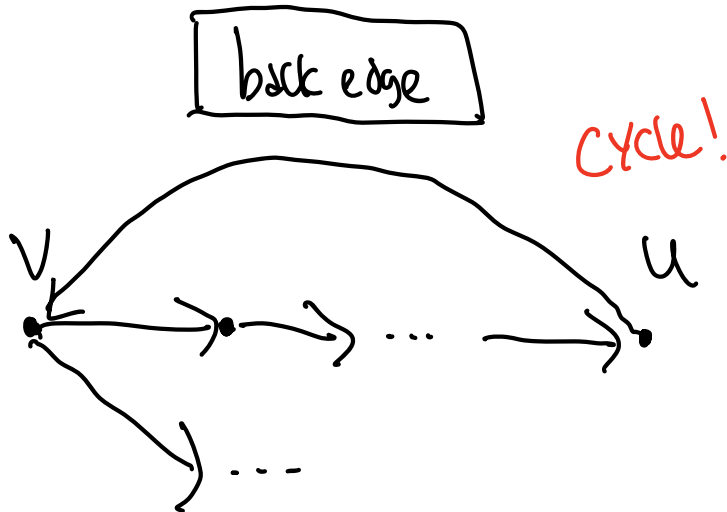
- ① • v not visited ($u.\text{start} < v.\text{start}$)
- ② • v finished ($v.\text{finish} < u.\text{start}$)
- ③ • v currently active ($v.\text{start} < u.\text{start} < v.\text{finish}$)



①: $v.\text{finish} < u.\text{finish}$

② same

③ u belongs to v 's recursive subtree



Only case with $u.\text{finish} < v.\text{finish}$ 

Punchline: Suppose $u \rightarrow v$

$u.\text{finish} < v.\text{finish}$ (top order failed?)

No! It's not a DAG by lemma.

If DAG: run DFS, reverse postorder

If not DAG: just check the edges!

$O(m+n)$ time.