CS 331, Fall 2025

Lecture 11 (10/6)

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)

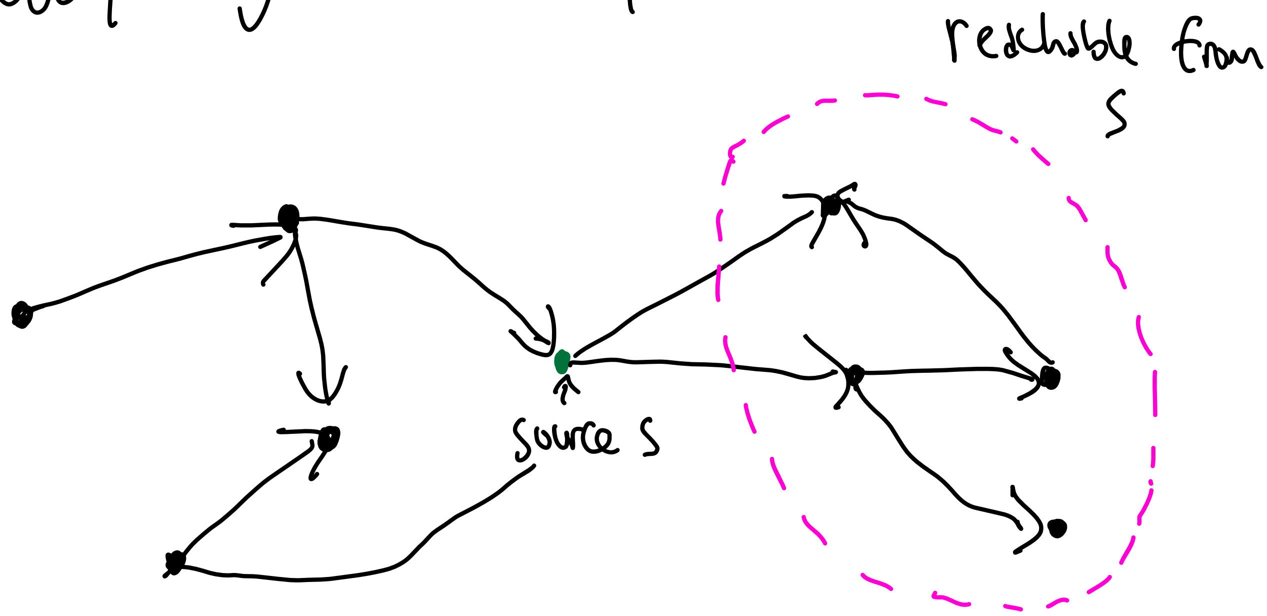
Example

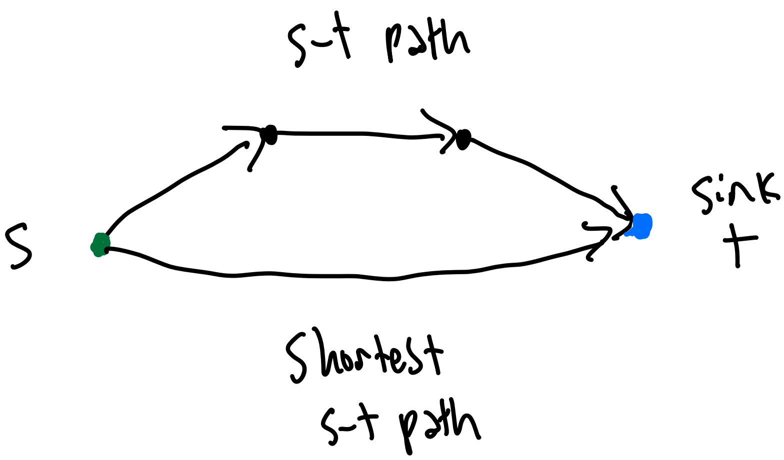
MST: "greedy stays ahead"

- exchange lemma: k CC's
  n-k edges

- data structure for maintaining CC's

Today: graph search





GraphSearch$(G=(V,E), s)$:

$S \leftarrow \{s\}$

$R \leftarrow$ [False for $v \in V$]

While $S \neq \emptyset$:

  $v \leftarrow$ any element of S

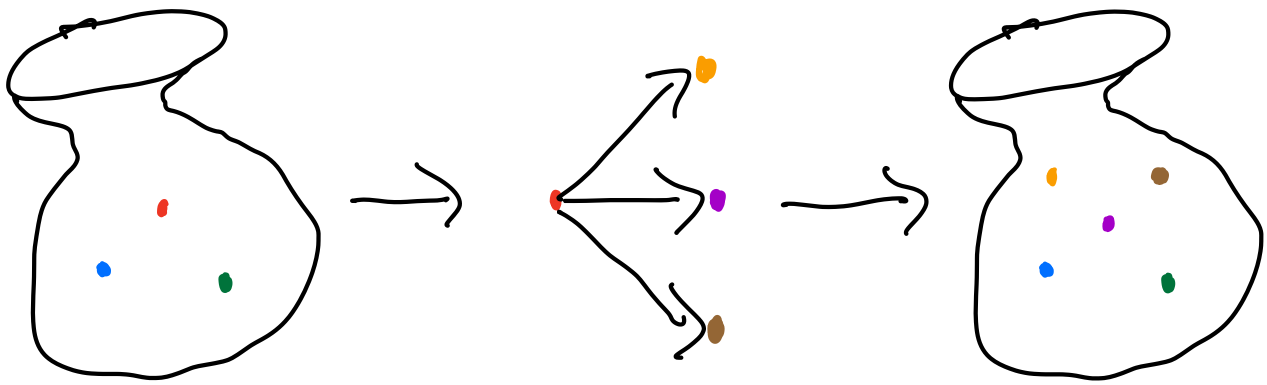
  $S \leftarrow S \setminus v$

  If $R[v]$ == False:

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

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 $s \rightsquigarrow u \to v$

$R[u] = \text{True}$

Then, v added to S. Will become True

Breadth-First Search (Part V, Section 2.1)

How to implement Graph Search?

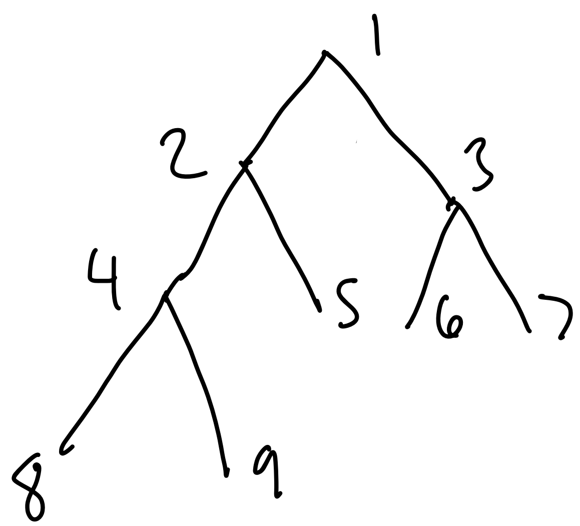
Need <u>data structure</u> for S.

- Insert an element
- Remove an element

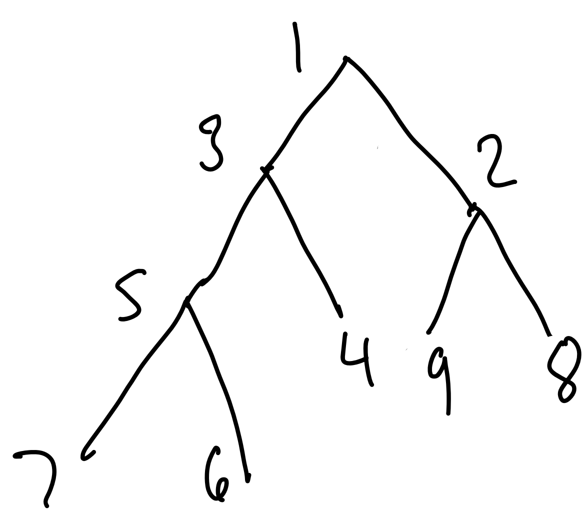
Idea: Linked List does both in $O(1)$ time.

Queue = BFS Push( ):

Stack = DFS Push( ):



BFS



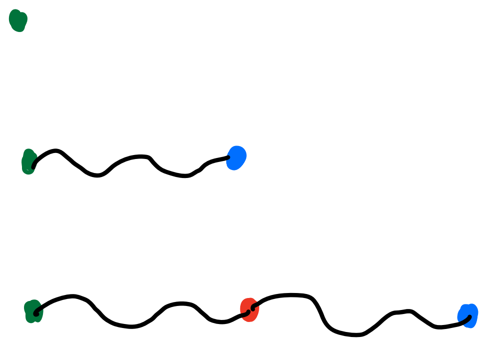
DFS

Connected components

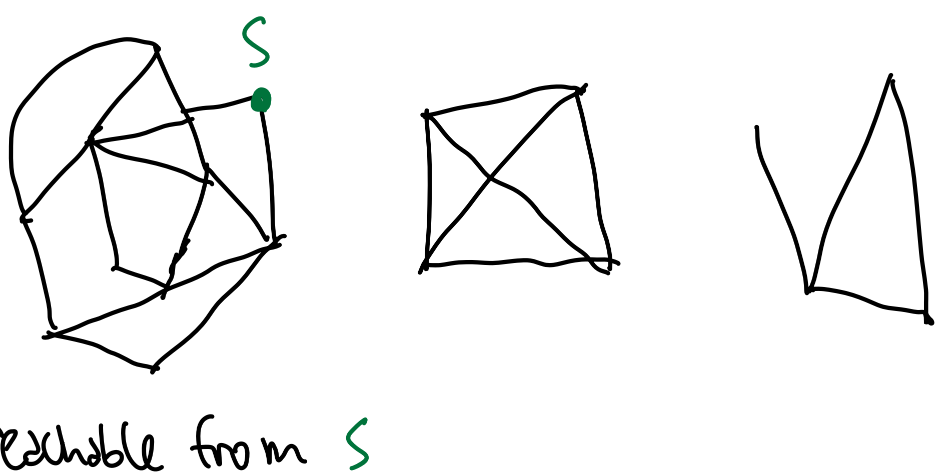
Say $s \sim t$ if connected in undirected graph

Then $\sim$ is equivalence relation

- reflexive
- symmetric
- transitive



Partition into <u>connected components</u> (CCs)



reachable from s

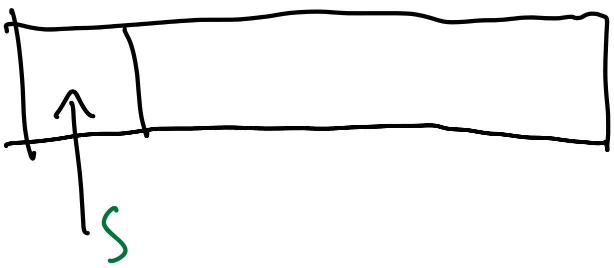
Runtime of BFS:

- every edge used $\leq 2x$.
- $2 m_{C_s}$ total vertices
- m_{C_s} = # edges in C_s : CC of s.

$O(m_{C_s})$

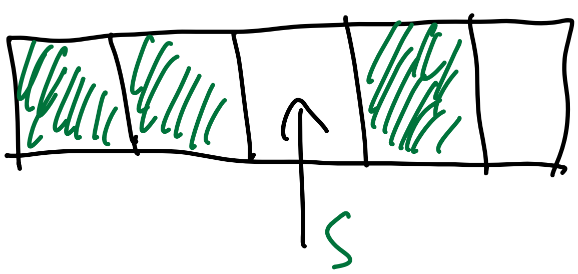
Using Queue (or Stack)

CC algo:



Vertex list: total move $O(n)$

GraphSearch(s) ... $O(m_{C_s})$



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

Unweighted SSSP

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

v added for the first time

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

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

With Claim, simple mods compute SSSP!

- For $(u,v) \in E$: $S.\text{Push}((u,v))$

include parent info

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

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

Memoized

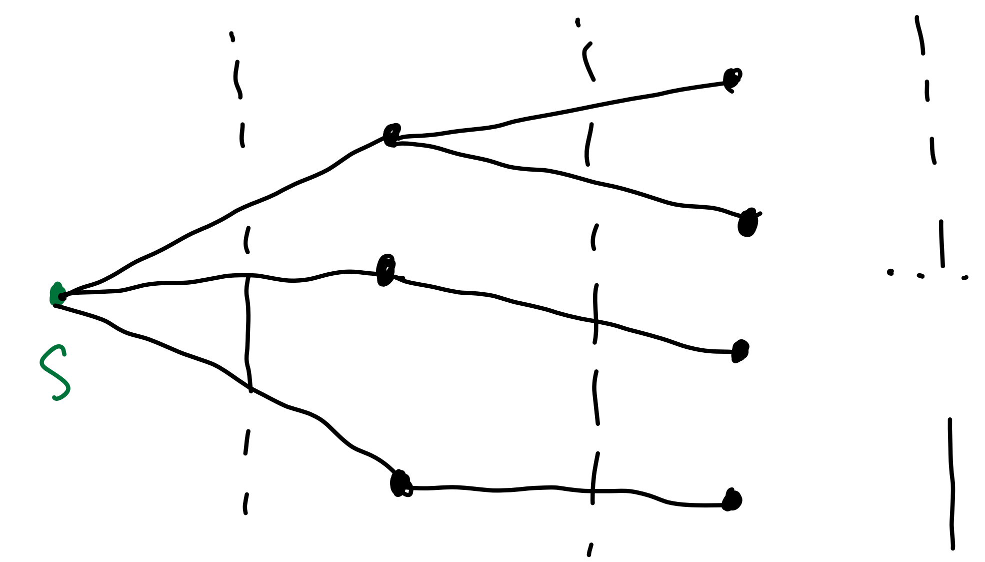
Proof of Claim: Let $R_0 = \{s\}$

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

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

$$\vdots$$

Claim is that they form "frontiers" in S:



$$\underbrace{R_0 \;\dots\; R_1 \;\dots\; R_2 \;\dots}_{\text{Queue, restricted to \underline{first visits}}}$$

Base case: R_0 ✓

Input: | $R_0 \ R_1 \ R_2 \dots$ | R_i

current queue

let $v \in R_i$ be dequeued

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

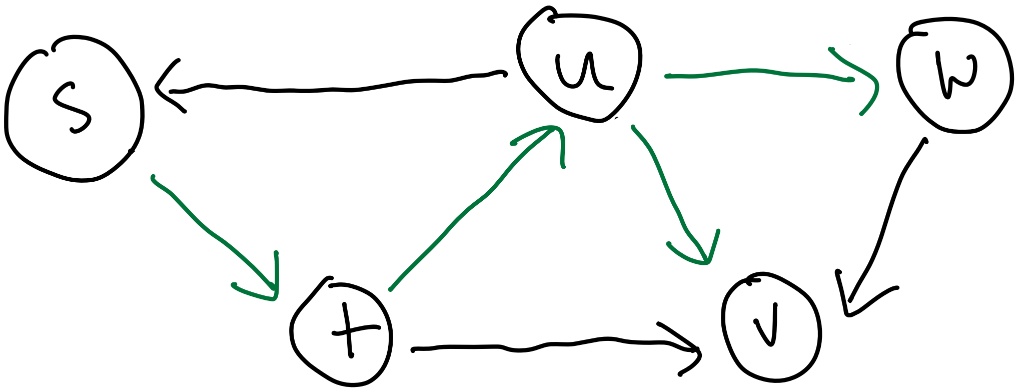
If $d(s, u) \leq i$ then reached (induction). ✓

Depth-First Search (Part V, Section 2.2)

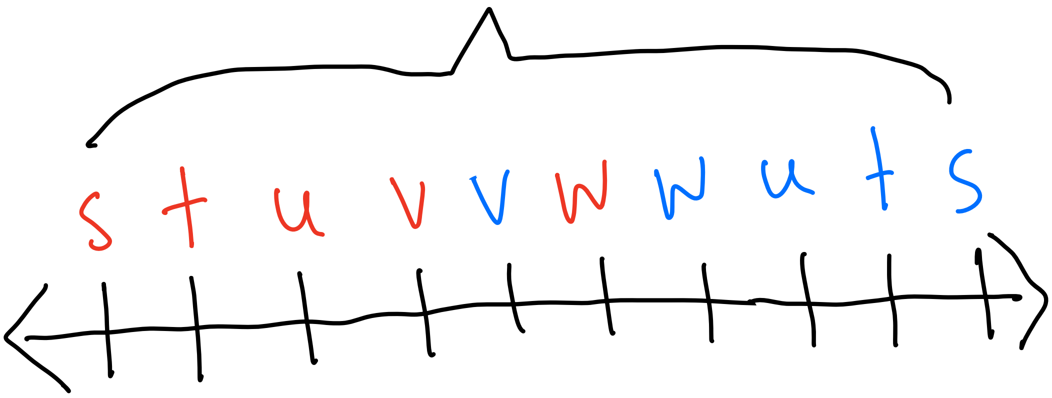
Each vertex has:

- 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 t u v w

Postorder: v w u t s

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

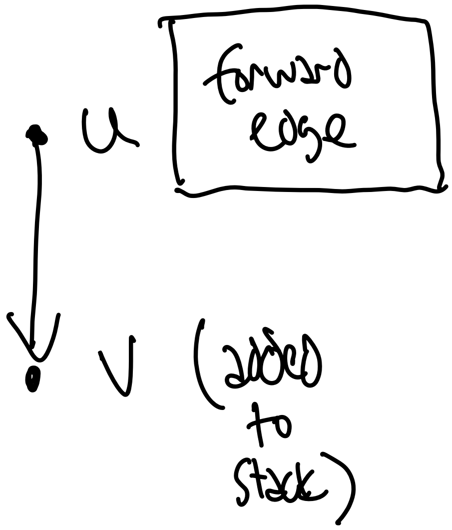
Lemma: Suppose $u \to v$,

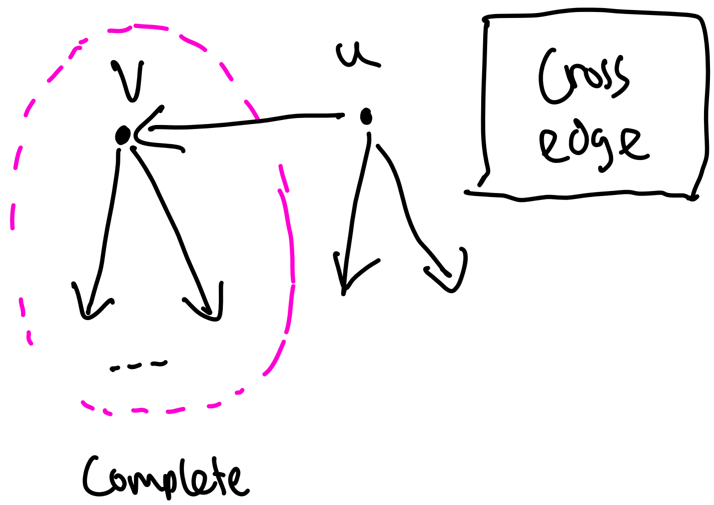
u, v reachable from s.

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 $(v.\text{start} < u.\text{start} < v.\text{finish})$

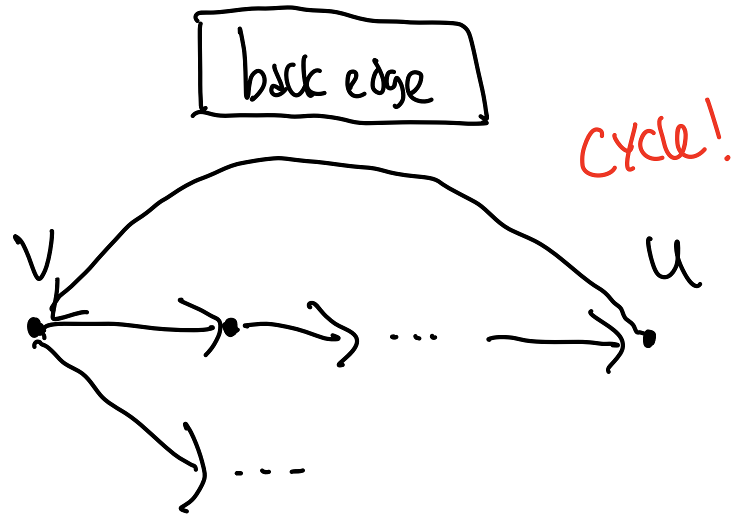




1: $v.\text{finish} < u.\text{finish}$

2 same

③ u belongs to v's recursive subtree



Only case with u.finish < v.finish ∎

Punchline: Suppose $u \to v$

$u.\text{finish} < v.\text{finish}$ (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.