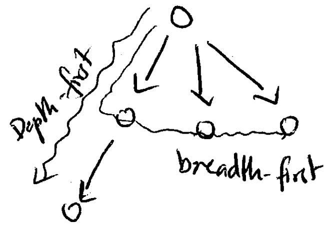
Wed Nov 15:

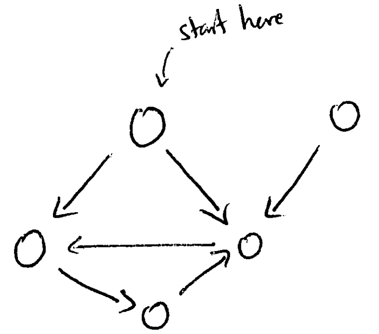


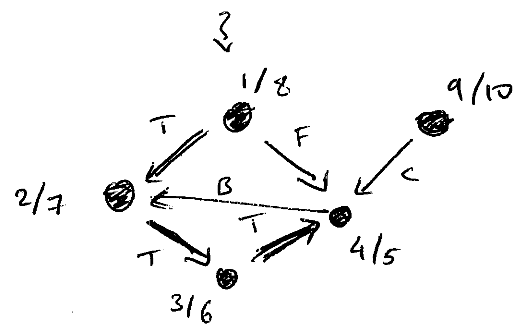
Today:

- Depth-first search (DFS)
- Topological sort
- Minimum spanning tree

DFS(G): ↗ directed or undirected

```
for each u ∈ G.V:
    u.colour = white        } Θ(V)
    u.π = NIL
t = 0
for each u ∈ G.V:
    if u.colour == white:
        DFS-visit(G, u)
```






```
DFS-visit(G, u):
    t = t + 1
    u.d = t                       // discovery time
    u.colour = gray
    for each v ∈ G.Adj[u]:        } runs once for
        if v.colour == white          each edge (u,v) ∈ G.E
            v.π = u                   Θ(E)
            DFS-visit(G, v)
    u.colour = black
    t = t + 1
    u.f = t                       // finish time
```

result: depth-first forest.

Property: at any time, gray vertices are exactly the set of ancestors of u.

Total time = $\Theta(V + E)$

Thm: u is a proper descendent of v in depth first forest iff $[v.d, v.f] \subsetneq [u.d, u.f]$.

$(u.d < v.d < v.f < u.f)$

Wed Nov 15: (2)

Types of edges:

		When (u,v) first explored $v.colour$ is:
1) Tree edges : $(u.\pi, u)$	———	white
2) Back edge : (u,v) v ancestor of u in a depth-first tree	———	gray
3) Forward edge : (u,v) u ancestor of v but $u \neq v.\pi$.	———	black
4) Cross edge : all other edges (between same tree or between trees)	———	black.

Thm 22.10 : G undirected $\Rightarrow$ every edge is either a tree or back edge.

Corollary: Undirected graph is cyclic $\Leftrightarrow$ there are no back edges.

What about directed graphs?

Another useful Thm:

Thm 22.9 : v is a descendent of u iff at time $u.d$ $\exists$ a path from u to v with only white vertices.

Proof idea: $(\Rightarrow)$ if v proper descendent then $u.d < v.d$ so v still white. Holds for all v on path.

$(\Leftarrow)$ by contradiction.

Wed Nov 15: (3)

G: directed graph. A topological sort is an ordering of vertices $V = \langle v_1, v_2, \ldots v_n \rangle$
such that for every $(v_i, v_j) \in E$. $i < j$. (all edges go "forward")

Only possible for directed acyclic graphs (DAGs).

Very useful to do DP on graphs!

How to compute?

Sort vertices by reverse order of finishing times $v.f$. (after running DFS)

Lem 22.11: G directed, is acyclic iff DFS finds no back edges.

Thm 22.12: Reverse order of finishing times is a topological sort if G is a DAG.

Proof sketch: take any edge (u,v)

when explored, v gray $\Rightarrow$ back edge $\Rightarrow$ cycle ($\Rightarrow\Leftarrow$)

v white $\Rightarrow$ descendent of u $\Rightarrow$ $v.f < u.f$

v black $\Rightarrow$ already finished, ~~so~~ and u unfinished $\Rightarrow$ $v.f < u.f$.

Other applications of DFS:

- finding connected components / strongly connected components
- finding bridges / articulation points
- testing planarity
- ...

Wed Nov 15: (4) **Minimum Spanning Trees**

$G=(V,E)$ undirected. $w: E \to \mathbb{R}$

Spanning tree is $T \subseteq E$ s.t. (V,T) is connected and acyclic (i.e. a tree)

want to find T with minimum weight $w(T) = \sum_{(u,v)\in T} w(u,v)$

How? Greedy!

Generic-MST(G,w):

$A = \emptyset$

while A is not a spanning tree: → loop invariant: A is a subset of some MST.

find $(u,v) \in E$ safe for A

$A = A \cup \{(u,v)\}$

return A

→ (u,v) safe for A if $A \cup \{(u,v)\}$ also subset of some MST.

Note: loop inv $\Rightarrow \exists$ some safe edge.

Definitions:

A cut of $G=(V,E)$ is $(S, V\setminus S)$. → same as $V-S$ in CLRS.

Edge (u,v) crosses cut $(S, V\setminus S)$ if $u \in S$ and $v \in V\setminus S$.

Cut respects A if no edge in A crosses the cut.

(u,v) is a light edge crossing a cut if it has minimum weight of any edge crossing the cut.

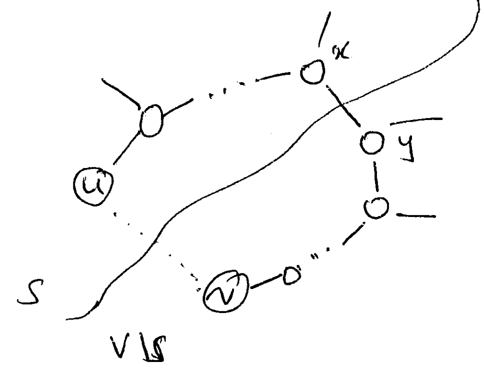
Thm: (23.1) $A \subseteq E$ included in some MST, $(S, V\setminus S)$ any cut respecting A, (u,v) light edge crossing $(S, V\setminus S)$ $\Rightarrow$ (u,v) safe for A.

Pf: Let $A \subseteq T$ some MST. Assume $(u,v) \notin T$.

$\therefore$ $T \cup \{(u,v)\}$ has a cycle.

$u \in S$, $v \in V\setminus S$ $\Rightarrow$ $\exists$ edge (x,y) on path connecting u & v in T that crosses $(S, V\setminus S)$

Wed Nov 15: (5)



as (u,v) light edge, $w(u,v) \le w(x,y)$

look at $T' \coloneqq T \setminus \{(x,y)\} \cup \{(u,v)\}$

$$w(T') = w(T) - w(x,y) + w(u,v)$$
$$\le w(T)$$

but T was MST, so T' also MST, and contains (u,v), so it is <u>safe</u>.

<u>Cor</u>: (23.2) $A \subseteq E$ included in some MST, $C = (V_c, E_c)$ is a connected component in $G_A = (V, A)$,

(u,v) light edge from C to any other component $\Rightarrow$ (u,v) safe for A.

<u>Kruskal's algo</u>:

```
A = ∅
for each v ∈ G.V:
    Make-Set(v)                          ⟶ each vertex is in its own connected component
for each (u,v) ∈ G.E in ascending order: ⟶ sorting takes O(E log E) = O(E log V)
                                            as |E| ≤ |V|^2
    if Find-Set(u) ≠ Find-Set(v):        ⟶ if u and v are in
        A = A ∪ {(u,v)}                      different connected components
        Union(u,v)                       ⟶ join those components together.
return A
```

Needs a <u>disjoint</u> <u>set</u> data structure.

Example: union-find / disjoint-set-forest (CLRS 21.3)

- Make-Set : $O(1)$
- Find-Set : $O(\log V)$
- Union : $O(\log V)$

So ~~total~~ total time:

$$O(\underbrace{V}_{\text{init}} + \underbrace{E \log V}_{\text{sorting}} + \underbrace{E \log V}_{\text{loop body}})$$
$$= O(E \log V)$$

Wed Nov 15: (6)

Prim's algo: uses a min-priority queue. (on attribute $u.key$)

MST-Prim (G, w, r): ($r \to$ some root node)

$O(V)$:

- for each $u \in G.V$:
  - $u.key = \infty$
  - $u.\pi = NIL$
- $r.key = 0$
- $Q = G.V$

$|V|$ iterations $\leftarrow$ while $Q \neq \emptyset$:

$O(\log V) \leftarrow$ $u =$ Extract-Min(Q) $\longrightarrow$ u is always the end of a light edge crossing $(V_A, V \setminus V_A)$.

overall $|E|$ iterations $\leftarrow$ for each $v \in G.Adj[u]$:

if $v \in Q$ and $w(u,v) < v.key$:

$v.\pi = u$

$O(\log V) \leftarrow$ $v.key = w(u,v)$ $\longrightarrow$ this is the min weight of any edge from V_A to v.

Implicitly builds

$$A = \{ (u.\pi, u) \mid u.\pi \neq NIL \}$$

is always a tree (connected component)

Using a binary min-heap: all Q operations are $O(\log V)$, building is $O(V)$.

Running time $= O(V + V \log V + E \log V) = O(E \log V)$

Can do faster with Fibonacci heaps!