DFS on graphs

Non tree edges

Undirected G : Back

Directed G : Back, Forward, Cross

Path : v_i to v_j $v_0 - v_1 - v_2 \cdots v_\ell$

s.t. $(v_k, v_{k+1}) \in E \ \forall k$

but no vertex repeats

Walk = Path with repetitions

Loop or cycle

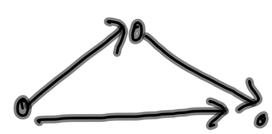
Graphs without loops = acyclic

Undirected: Acyclic connected graph ≡ Tree

n vertices, $n-1$ edges

Each pair (v_i, v_j) connected by a unique path

Directed :



directed acyclic graphs

DAGs

Claim: G has a cycle iff DFS reveals a back edge

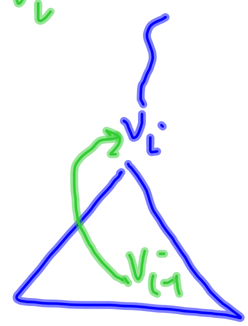
If cycle then back edge

$v_1 v_2 \ldots v_k v_1$

DFS enters cycle at v_i

$\underbrace{v_i v_{i+1} \ldots v_k v_1 v_2 \ldots}\, v_{i-1} v_i$

is in subtree of v_i in DFS tree



DAGs

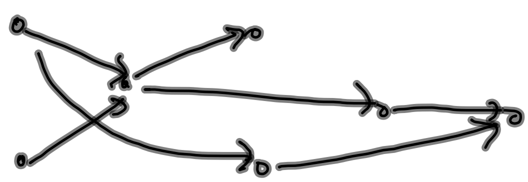
Models for practical problems

e.g. Scheduling

Vertices are tasks

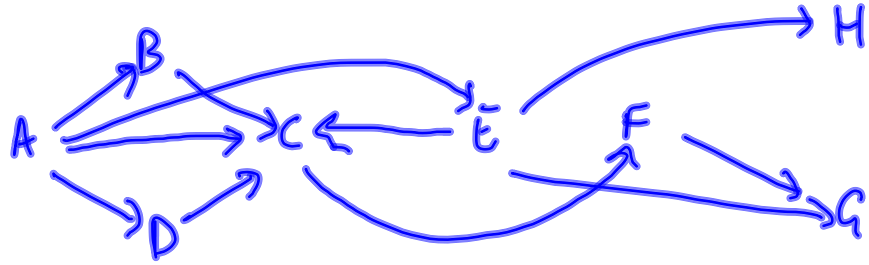
$i \rightarrow j$ j cannot start till i ends

Must be acyclic to be feasible



Given a DAG, enumerate the vertices respecting the edge reln

edge $v_i \to v_j \Rightarrow v_i$ is enumerated before v_j



A B E D H C F G

Not unique

TOPOLOGICAL SORT

Does a minimal vertex (no incoming edge) always exist in a DAG?

Suppose not: Pick v_0 – not minimal

$$v_n \to v_{n-1} \to \cdots \to v_2 \to v_1 \to v_0$$

By Pigeonhole Principle $\to \exists$ cycle!

Symmetrically, $\exists$ at least one maximal vertex.

Algorithm for topological sort:

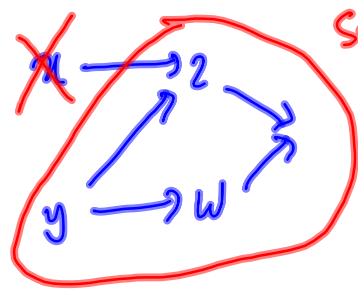
$X = V$

while $X \neq \phi$

choose x minimal in X and print x

$X = X \setminus \{x\}$

How to identify minimal vertices efficiently?



Still a DAG

Precompute indegree(v) for each v

- one scan of adj matrix / adj list

$O(n^2)$ $O(m)$

Minimal vertex has indegree = 0 $O(n)$

$\parallel$

x

For all $(x,v) \in E$, indegree[v] reduces by 1 $O(\text{outdeg}(x))$

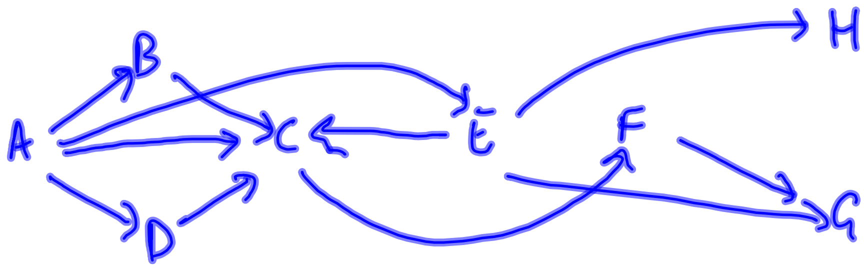
Preprocessing: $O(n^2)$ / $O(m)$

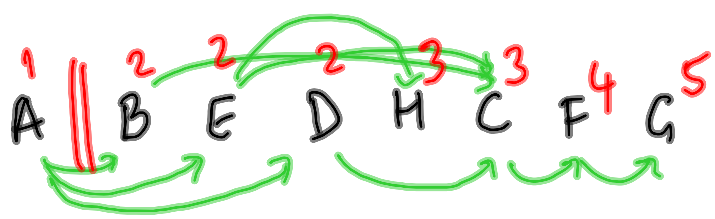
Loop: n times - each iteration $O(n)$

$\Rightarrow O(n^2)$

If each task takes exactly 1 day and I can do any number of independent tasks in parallel, find minimum no. of days to complete all tasks

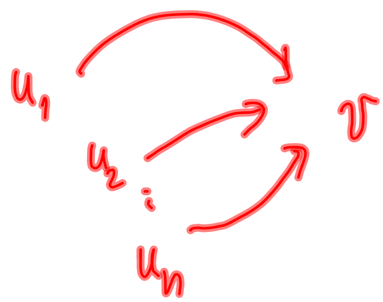
e.g. 5 days for the example below





Want to compute earliest[v]

earliest day when v can be performed



$$\text{earliest}[v] = 1 + \max_{u \in \{u_1, \dots u_n\}} \text{earliest}[u]$$

If we enumerate V in topological order, we can directly compute earliest[v] when we enumerate v

Earliest [v] corresponds to length of longest path to v from a minimal vertex

Computing longest path in a dag is as efficient as topological sort

In general directed graphs, longest path is NP complete