Depth-first Search (DFS).

A recursive algorithm to search a graph.

Preprocessing algorithm for graphs with many applications.

<u>Idea</u>: Go from node to node, visiting each node only once, recursively.

Global var: time

Attributes of vertex: color, fin (finish time), dis (discovery time), parent

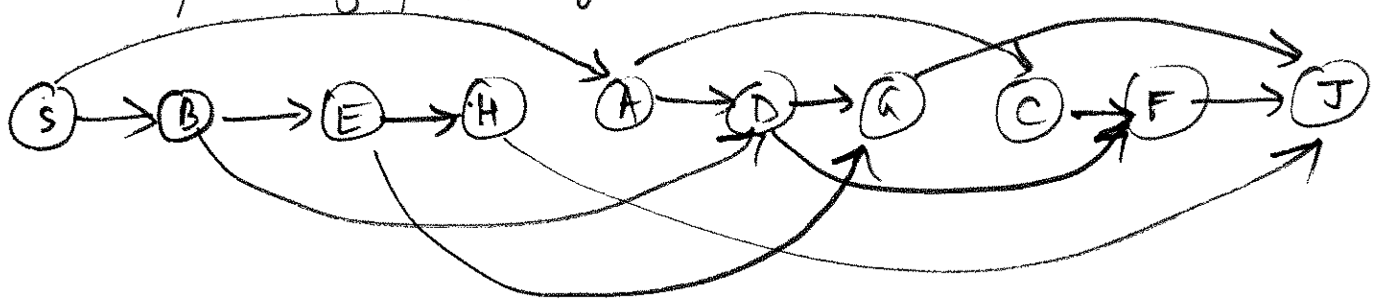
```
// Main loop:
dfs(g):
    // Initialize
    time ← 0
    for u ∈ g do
        u.color ← white
        u.parent ← null

    // Loop:
    for u ∈ g do
        if u.color = white then
            dfsVisit(u)
```

```
// Visit a node
dfsVisit(u):   // Pre: u.color = white
    u.color ← gray
    u.dis ← ++time

    for each edge e = (u,v) do
        if v.color = white then
            v.parent ← u
            dfsVisit(v)

    u.fin ← ++time
    u.color ← black
```

Application of DFS: Topological order.

<u>Def</u>: Topological order: $T : V \to \mathbb{Z}^+$ such that for all $(u,v) \in E$, $T(u) < T(v)$.

Informally, topological order is a linear ordering of the vertices of a directed graph such that all edges of graph go from left to right.



Theorem: G has a topological ordering

$\Longleftrightarrow$ G is a DAG (directed, acyclic graph).

Computing top ordering with DFS

In main loop: global var topNum $\leftarrow |V|$.

At the end of dfsVisit(u): u.top $\leftarrow$ topNum--

RT of DFS = $O\left(|V|+|E|\right)$ because each vertex is visited once, each edge is examined at most twice (once for dir graphs, twice for undir graphs).

Applications of topological ordering: PERT

(Program evaluation and review technique).

Problem: Consider a project, consisting of many tasks $1, 2, \ldots, n$. Task i has duration d_i.
Precedence constraints: (i, j) – task i must complete before j can begin.

Q: What is the fastest time in which project can be completed, given infinite resources?

Model: DAG: Vertices = tasks.
Precedence constraints = directed edges,

Let $EC(u)$ = earliest time at which task u can be completed.

Recursion for EC:

$$EC(u) = d_u \text{ if } u \text{ has no predecessors}$$
(duration of u)

$$EC(v) = \max_{(u,v) \in E} \{EC(u)\} + d_v$$

Let $LC(u)$ = latest time at which u can be completed without delaying the time at which project is completed.

Recursion for LC:

$$LC(\text{project}) = \max_{u \in V} \{ EC(u) \} = \text{Critical path length}$$

Time to complete project.

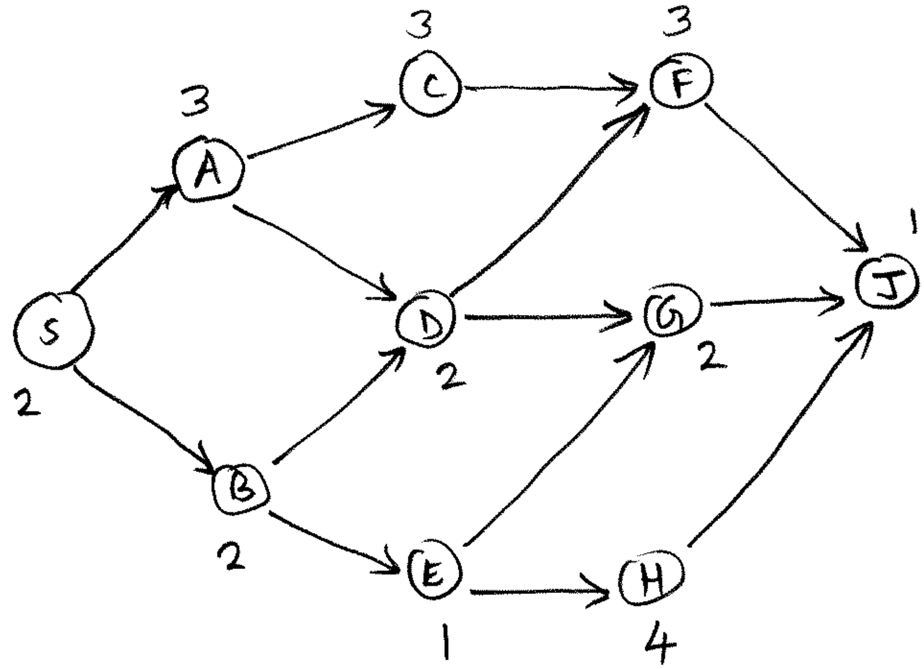
$$LC(u) = LC(\text{Project}) \text{ if } u \text{ has no successors.}$$

$$LC(u) = \min_{(u,v) \in E} \{ LC(v) - d_v \}$$

$$\text{slack}(u) = LC(u) - EC(u).$$

<u>Algorithm Pert</u> (g):

1. Call dfs(g) and get a topological ordering of its nodes.
2. for $u \in g$ do $u.ec \leftarrow d_u$
3. for u in topological order do // LI: $u.ec = EC(u)$.
   // Propogate it to successors of u
   for each $e = (u,v)$ out of u do
   if $v.ec < u.ec + d_v$ then
   $v.ec \leftarrow u.ec + d_v$
4. $CPL \leftarrow \max_{u \in g} \{u.ec\}$
5. for $u \in g$ do $u.lc \leftarrow CPL$
6. for u in reverse topological order do // LI: For successors v of u, $v.lc = LC(v)$
   for $e = (u,v)$ out of u do
   if $u.lc > v.lc - d_v$ then
   $u.lc \leftarrow v.lc - d_v$
   $u.slack \leftarrow u.lc - u.ec$



Node	Neighbors	dis	fin	parent	top	EC	LC	slack
S	A, B	1	20	—	1	2	min(5-3, 6-2) = 2	0
A	C, D	2	13	S	5	2+3=5	min(8-3, 8-2)=5	0
B	D, E	14	19	S	2	2+2=4	min(8-2, 7-1) = 6	6-4=2
C	F	3	8	A	8	5+3=8	11-3=8	0
D	F, G	9	12	A	6	max(4,5) +2 = 7	min(11-3, 11-2)=8	8-7=1
E	G, H	15	18	B	3	4+1=5	min(11-2, 11-4) =7	7-5=2
F	J	4	7	C	9	max(8,7)+3 =11	12-1=11	0
G	J	10	11	D	7	max(7,5) +2 = 9	12-1 =11	2
H	J	16	17	E	4	5+4=9	12-1=11	2
J	—	5	6	F	10	max(11,9,9) +1 = 12	12	0

PERT (Program evaluation and review technique):

Given n tasks 1..n, and a set of precedence constraints of the form (i,j), which means that task i must complete before task j can be initiated. This problem is modeled as a Project graph $G = (V, E)$, where V is the set of tasks, and the precedence constraints form the edge set E. G is a directed, acyclic graph (DAG). The tasks have durations, $d_1..d_n$. Suppose you have infinite resources available to execute the project. In such a case, all tasks whose predecessors have completed, can be executed in parallel.

Q: How much time is needed to complete the project, i.e., what is the length of a critical path? How much slack is available for each task? Slack(u) is the maximum time by which the duration of u can be extended without delaying the completion of the project. What are the critical tasks (i.e., tasks with zero slack)?

Terms:
EC(u): earliest completion time of task u.
LC(u): latest time at which task u can be completed, without delaying the completion of the project.

Minimum time needed to complete project (Critical path length), CPL = max { EC(u) }, for u in V.
Recursion for EC:

EC(u) = d_u if u does not have any predecessors,
EC(v) = max { EC(u) } + d_v for (u,v) in E, otherwise.

Recursion for LC:

LC(u) = CPL if u does not have any successors,
LC(u) = min { LC(v) − d_v }, for (u,v) in E.

Slack of task u, Slack(u) = LC(u) − EC(u).

The following algorithm implements the above recurrence.

```
Algorithm PERT(g):  // u.duration stores duration of u (d_u)
   find a topological ordering of the vertices of g (topList)
   for u in g do
      u.ec ← u.duration
   for u in topList do  // LI:  u.ec = LC(u). Propagate it to successors of u
      for each edge e=(u,v) out of u do
         if v.ec < u.ec + v.duration then
            v.ec ← u.ec + v.duration
   CPL ← max { u.ec }, for u in V
   for u in g do
      u.lc ← CPL
   for u in reverse topList do  // use descendingIterator of lists.  LI: for all successors v of u, v.lc = LC(v).
      for each edge e=(u,v) out of u do
         if u.lc > v.lc − v.duration then
            u.lc ← v.lc − v.duration
      u.slack ← u.lc − u.ec
```