Topological Sort

and Lowest Common Ancestor

Mohammed Yaseen Mowzer

17 April 2015

Outline

Outline

Outline

Directed Acyclic Graphs (DAGs)

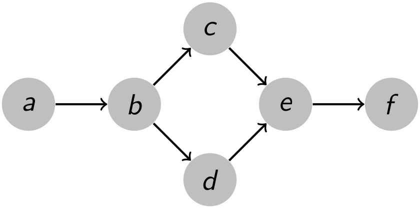
Definition

A Directed Acyclic Graph (DAG) is a graph such that

- all of its edges are directed
- there exist no cycles

A DAG is not a Forest

Forest	DAG
Edges are undirected	Edges are directed
Each node has one parent	Each node can have multiple parents
At most one path between any two points	Multiple paths between any two points.
No cycles	No cycles



What do DAGs represent

A DAG can be used to represent any transitive relation.

Definition

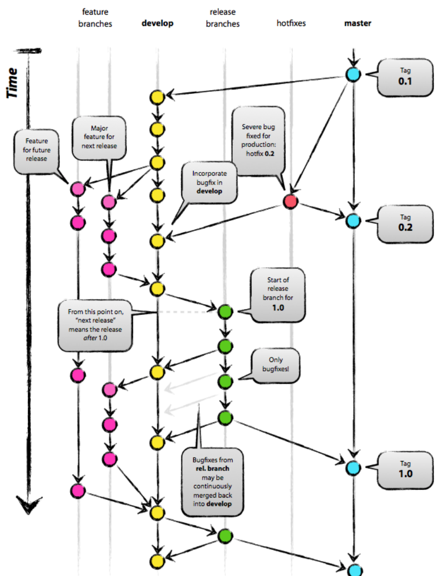
An operation, $\circ$ is transitive if for any a, b, c, if $a \circ b$ and $b \circ c$ then $a \circ c$.

For example

- An ordering $a < b$ and $b < c$ then $a < c$.
- If a requires b and b requires c then a requires c

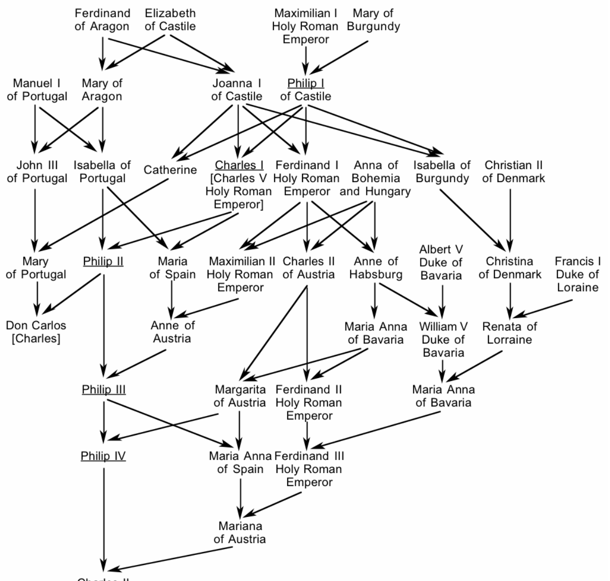
Outline

Git

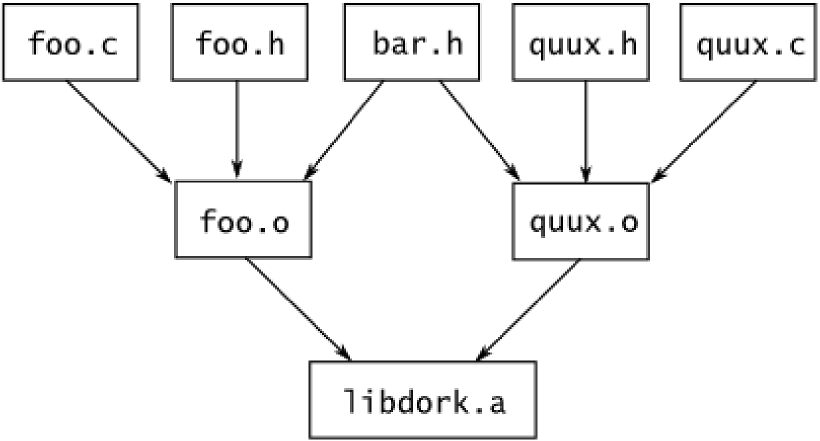


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Family ~~Tree~~ DAG



Compilation dependencies



Outline

What is topological sort?

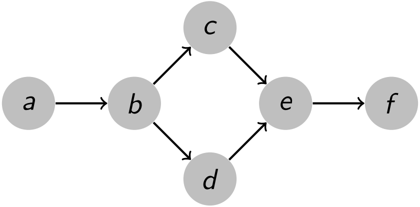
Definition

A topological ordering of a directed graph is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering — Wikipedia

What is topological sort?

Definition

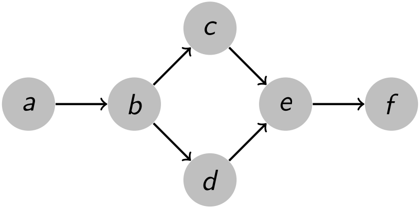
A topological ordering of a directed graph is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering — Wikipedia

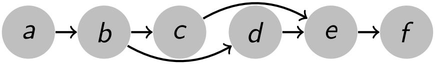


What is topological sort?

Definition

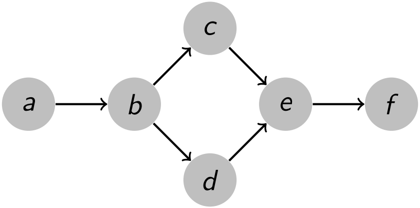
A topological ordering of a directed graph is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering — Wikipedia

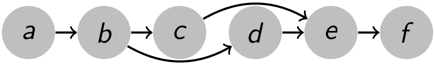




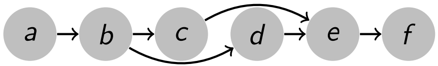
What is topological sort?

The topological ordering is the sequence in which tasks need to be completed so that all dependencies are satisfied.

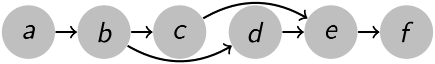




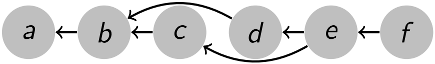
Properties of a Topological ordering



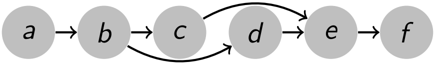
Properties of a Topological ordering



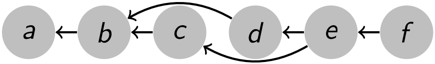
- It is trivially reversible.



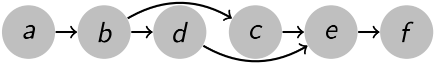
Properties of a Topological ordering



- It is trivially reversible.



- There may be multiple orderings.



Outline

Outline

Iterative (Khan's?) algorithm

```
L = List (will contain topological ordering)
S = List of nodes with no incoming edges

while S is non-empty do
  remove a node n from S
  add n to tail of L
  for each node m with an edge e from n to m do
      remove edge e from the graph
      if m has no other incoming edges then
          insert m into S

if graph has edges then
  return error (graph has at least one cycle)
else
  return L (a topologically sorted order)
```

Explanation

1. Find a node *n* with no unsatisfied dependencies (incoming edges).
2. “Compile” *n* and “remove” it from it’s dependents.
3. If nodes have not been “compiled“ goto 1.

Outline

Recursive (DFS) algorithm

```
L = List (will contain topological ordering)
Mark all nodes white.

for each node n
  if n is white
    visit(n)

function visit(node n)
  mark n grey
  for each node m with an edge from n to m do
    if m is grey
      error # There is a cycle
    if m is white
      visit(m)
  mark n black
  add n to head of L
```

C++ Topological sort (DFS)

```
for (int i = 0; i < N; ++i)
  if (color[i] == WHITE)
    visit(i);

void visit(int v)
{
  color[v] = GREY;
  for (int u : graph[v])
    if (color == GREY)
      exit(1);
    else if (color[u] == WHITE)
      visit(u);
  color[v] = BLACK;
  L.push_back(v);
}
```

Explanation

Visit:

- If a node has no dependencies (outgoing edges) “compile“ it.
- Otherwise visit all it’s dependents (neighbours) then “compile“ it.

Outline

Comparison between iterative and recursive algorithms

Iterative algorithm

- Need to store number of incoming edges.
- Has an explicit stack.
- Will not cause stack overflow.
- Check for cycles occurs after algorithm.

Recursive algorithm

- Needs a color array.
- Has an implicit stack.
- Might cause stack overflow.
- Check for cycles during occurs during algorithm.

Time Complexity

Time Complexity is $\Theta(V + E)$

- Every vertex is visited once.

```
for (int i = 0; i < N; ++i)
  if (color[i] == WHITE)
    visit(i);
```

- Each edge of every vertex checked once.

```
  for (int u : graph[v])
```

Hamiltonian Path

Definition

A Hamiltonian Path is a path that traverses every vertex in a graph.

- Finding a Hamiltonian Path is an NP-Complete problem: there is no known polynomial time solution, but

Hamiltonian Path

Definition

A Hamiltonian Path is a path that traverses every vertex in a graph.

- Finding a Hamiltonian Path is an NP-Complete problem: there is no known polynomial time solution, but
- Hamiltonian Path exists if and only if every adjacent pair of a topological ordering has an edge between them.
- Finding a Hamiltonian Path in a DAG is in P.

Outline

Example Problem

Example (Codeforces Round 290 div. 1 Problem A)

A list of names are written in lexicographical order, but not in a normal sense. Some modification to the order of letters in alphabet is needed so that the order of the names becomes lexicographical. Given a list of names, does there exist an order of letters in Latin alphabet such that the names are following in the lexicographical order. If so, you should find out any such order.

Sample Input Output

Input

```
3
rivest
shamir
adleman
```

Output

```
bcdefghijklmnopqrsatuvwxyz
```

Solution

Between every consecutive pair of words, draw and edge between the first two different letters. Output the topological ordering of that graph.