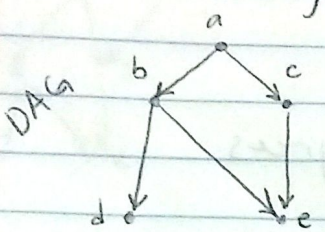
1/21/16

Breadth first Search will tell you the number of connected components

## Directionality

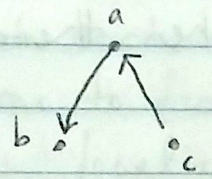



DAG

in a directed graph, every edge has a direction

a directed graph may or may not have cycles

BFS + DFS algorithms are similar to those for undirected graphs but the definition of neighbor is different

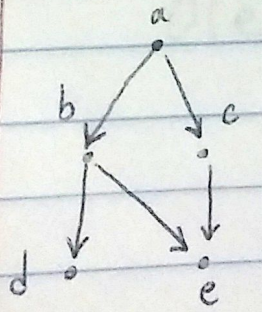



c is NOT a neighbor of a

DAG : directed acyclic graph

## Sorting Graphs

each vertex may represent a task and some tasks may have to be completed before others (precedence ordering)




possible sorting: a c b e d

b + c depend on a

e depends on b and c

Topological Sorting of the DAG

directed graph
- in-degree: number of edges going to a vertex
- out-degree: number of edges leaving a vertex

if in-degree = 0 : source
if out-degree = 0 : sink

undirected graphs just have degrees

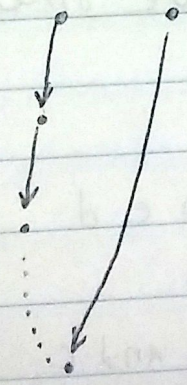
n-1 sources possible for a graph
n-1 sinks possible for a graph

a topological sort doesn't make sense for a graph that isn't DAG b/c that would mean sorting vertexes when there is a cycle

in a topological sort the order matters b/c it implies there is an edge between an earlier vertex to one further in the list

always check if there is a cycle using BFS

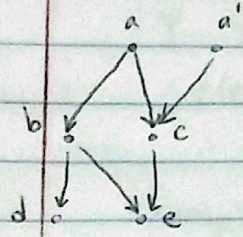
ex:



(n-1)! → possible topological sort outputs
not possible to do all of them in linear time

efficient algorithm for topological sort:

first find all the sources and generate a list




output an arbitrary ~~source~~ source like a

now ignore the fact that a exists

↳ update the source list

↳ output an arbitrary source

~~delete an arbitrary edge~~

first assume we have a set of vertexes with no edges (in-deg = 0, out-deg = 0)

start adding edges and update in-deg, out-deg for each vertex ⇒ for every vertex we spend n-1 time to find all in-degree, out-degree

localized analysis $O(n^2)$ -- maybe linear in e

$O(e)$ : e is number of edges

↳ always linear in e

linear means linear in ⊛ input size

generating list of sources $O(n)$

output sources $O(1) \rightarrow O(n)$

update source list $O(n) \rightarrow O(n^2) \Rightarrow O(e)$

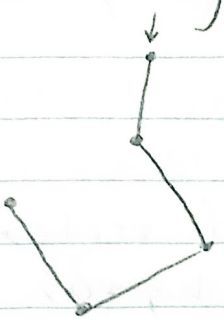
whenever we output a source, I look at the neighbors and decrement their in-degrees

$O(e+n)$ → time complexity

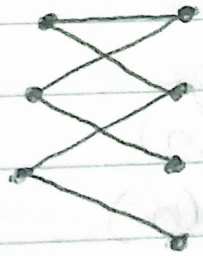
np-complete: an efficient solution has not yet been found
common for graphs not necessarily DAGs

np-hard similar to np-complete

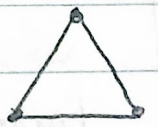
ex: Traveling Salesman



Bipartite Graph (undirected)
if vertices can be partitioned into 2 groups and the edges run between the two groups



not every graph is bipartite



bipartites can't have odd cycles

Algorithm to determine bipartite:

do BFS