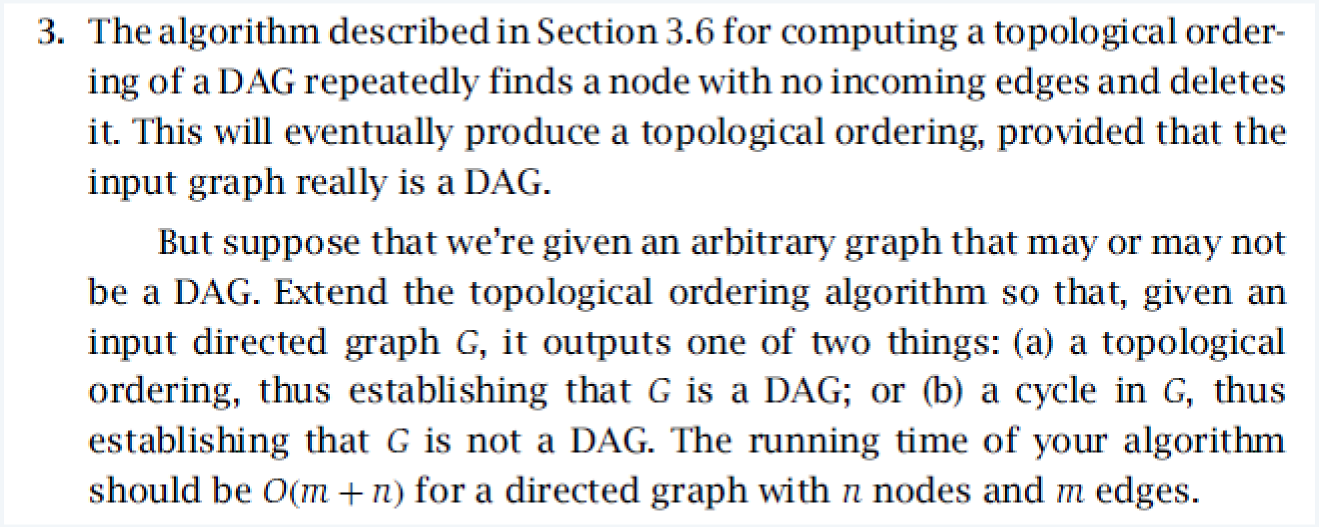
CS 3330 Algorithms

2. Graph

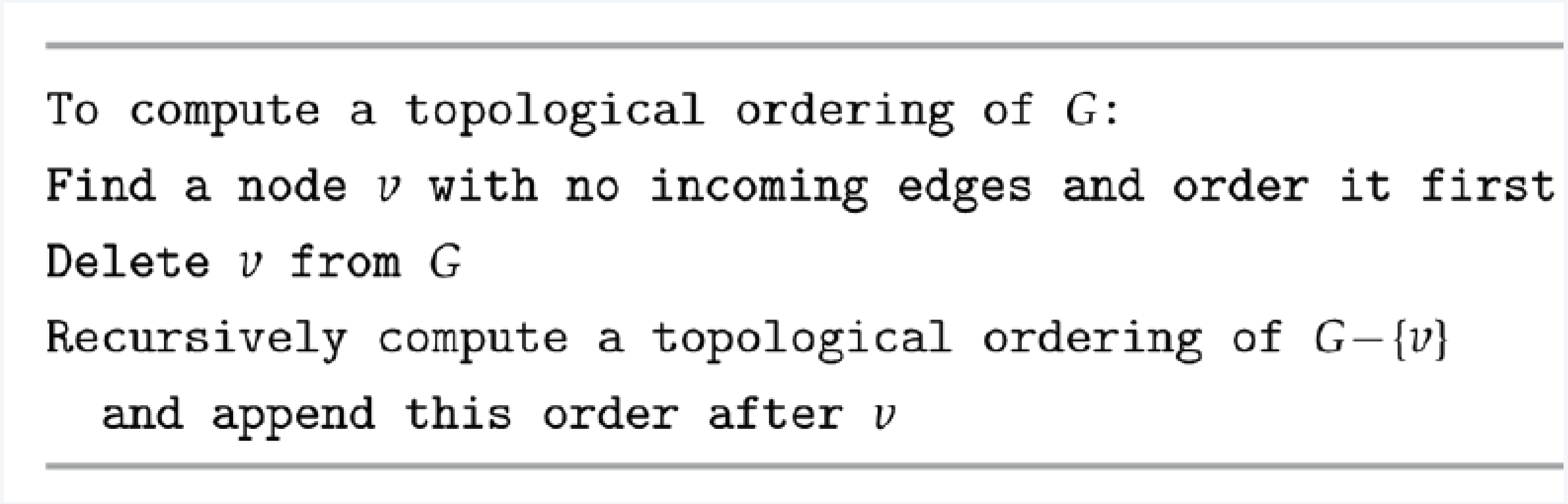
- *graph algorithms*

Graph algorithms



3. The algorithm described in Section 3.6 for computing a topological ordering of a DAG repeatedly finds a node with no incoming edges and deletes it. This will eventually produce a topological ordering, provided that the input graph really is a DAG.

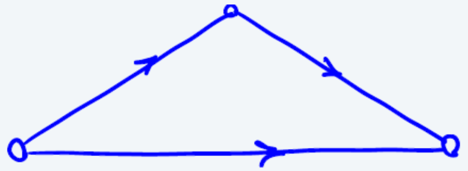
   But suppose that we're given an arbitrary graph that may or may not be a DAG. Extend the topological ordering algorithm so that, given an input directed graph G, it outputs one of two things: (a) a topological ordering, thus establishing that G is a DAG; or (b) a cycle in G, thus establishing that G is not a DAG. The running time of your algorithm should be $O(m + n)$ for a directed graph with n nodes and m edges.



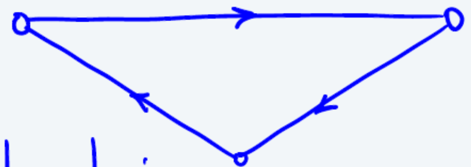
```
To compute a topological ordering of $G$:
Find a node $v$ with no incoming edges and order it first
Delete $v$ from $G$
Recursively compute a topological ordering of $G-\{v\}$
  and append this order after $v$
```

Graph algorithms

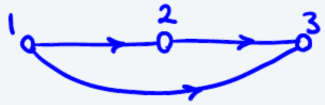
DAG:



Not a DAG:



: Topological ordering:



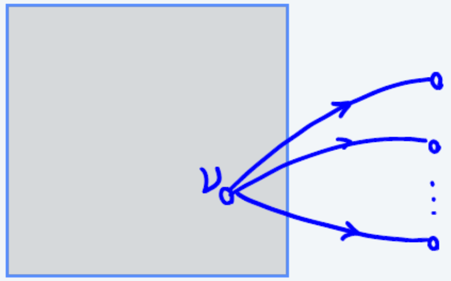
$i \rightsquigarrow j$ then $i < j$.

How to find topological ordering:

S: set of vertices with no incomming edge

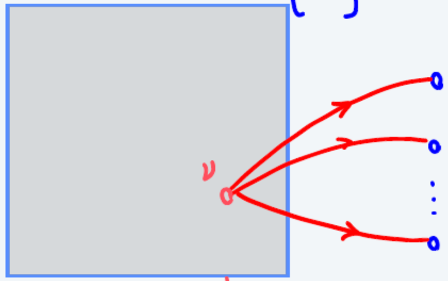
$S \longleftarrow S \setminus \{v\}$

Step 1:



pick a vertex v & remove it.

Step 2:



remove all red edges

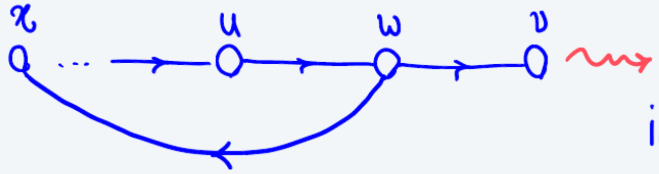
update S & repeat

Graph algorithms

If the input graph is not a DAG, at some iteration $S = \emptyset$ & all the remaining vertices have incomming edges.

Since all the remaining vertices have incoming edges we can run the following algorithm:

```
s ← v ; v is some vertex
While s has not been visited before
    s ← pred[s] ; since s has incomming edge.
Return the cycle.
```

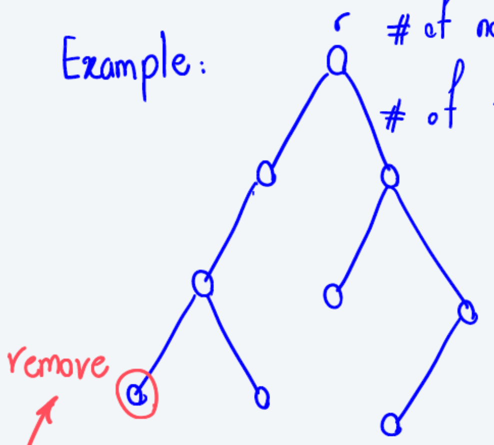


we explore a path in reverse dir.

Graph algorithms

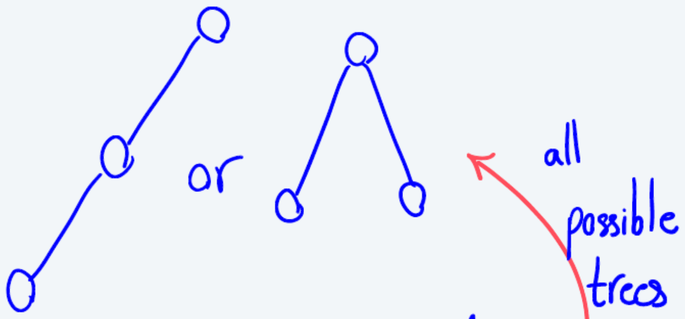
5. A binary tree is a rooted tree in which each node has at most two children. Show by induction that in any binary tree the number of nodes with two children is exactly one less than the number of leaves.

Example:



Proof by induction.

n=3:



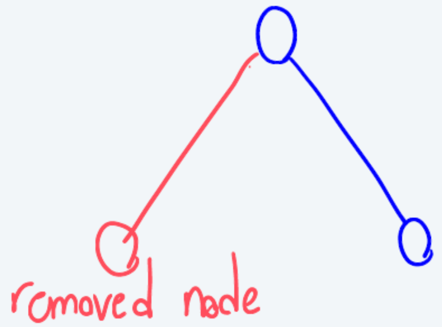
① induction basis: n=3 there are two leaves.

Induction step: suppose its true for n=k, prove it for n= k+1. Pick a leaf & remove it. We are left with k vertices & we can use induction hypothesis: # of leaves in the remaining graph −1 =

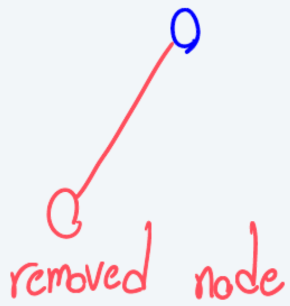
# of vertices with degree 2 in the remaing graph

Graph algorithms

Let's bring back the vertex that we removed:



OR

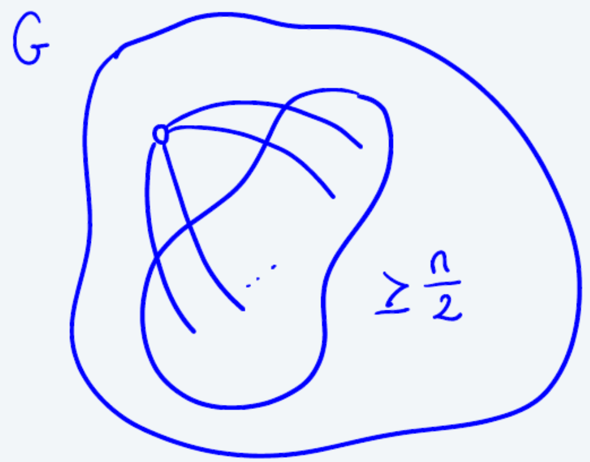


case 1: After bringing it back both sides of equality goes up with 1.

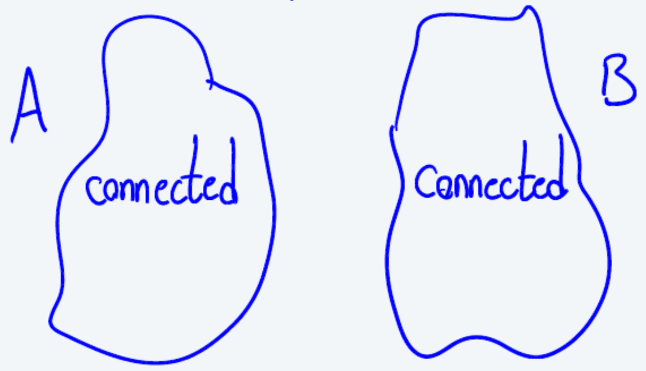
Case 2: the both sides remain the same. Since we are removing leaf and adding another leaf.

Graph algorithms

Claim: Let G be a graph on n nodes, where n is an even number. If every node of G has degree at least n/2, then G is connected.



If it is not connected, then it at least 2 connected components:



and prob. more

① $|A| + |B| \leq n$, so

$\min(|A|, |B|) \leq \frac{n}{2}$

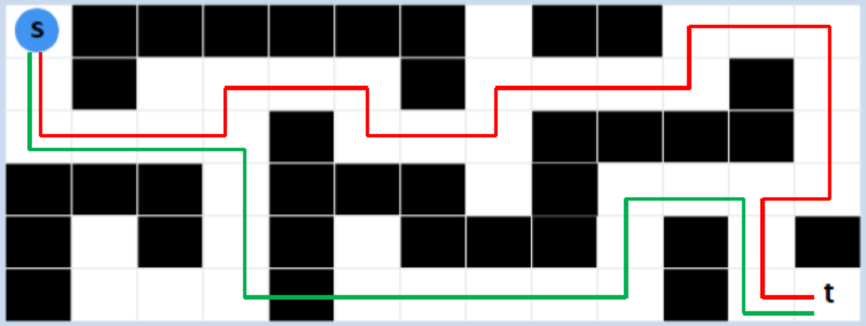
By $|A|$ we mean the number of vertices of A. Suppose $|A| \leq |B|$

Pick a vertex in A, call it v. $\text{degree}(v) \geq \frac{n}{2}$. Contradiction!

Graph algorithms

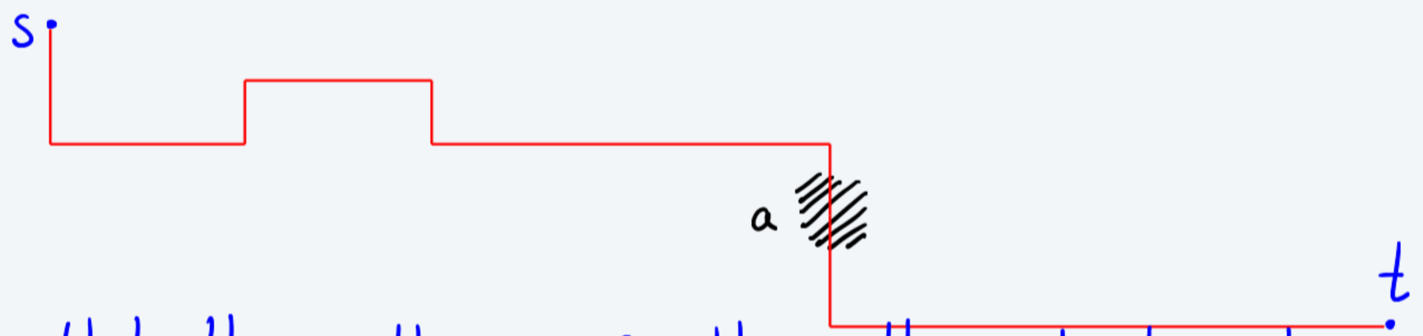
You have maps of parts of the space station, each starting at a prison exit and ending at the door to an escape pod. The map is represented as a matrix of 0s and 1s, where 0s are passable space and 1s are impassable walls. The door out of the prison is at the top left $(0, 0)$ and the door into an escape pod is at the bottom right $(w-1, h-1)$.

Write a function that generates the length of a shortest path from the prison door to the escape pod, where you are allowed to remove one wall as part of your remodeling plans.



Graph algorithms

Suppose that the shortest path, passes through a:



Notice that the path s-a & the path a-t does not cross any black cube, & the have to be the shortest ones. Hence, we do the following:

Step 1: find shortest path from s to each black cube. Call the resulted distance d(s,u)

Graph algorithms

Step 2: find shortest path from t to each black cube. Call the resulted distance $d(t,u)$

Now the vertex a that we are searching for, minimizes the value of $d(s,a)+d(t,a)$.