CS 341 – Algorithms

Lecture 6 – Depth First Search

28 May 2021

Today’s Plan

1. Depth First Search

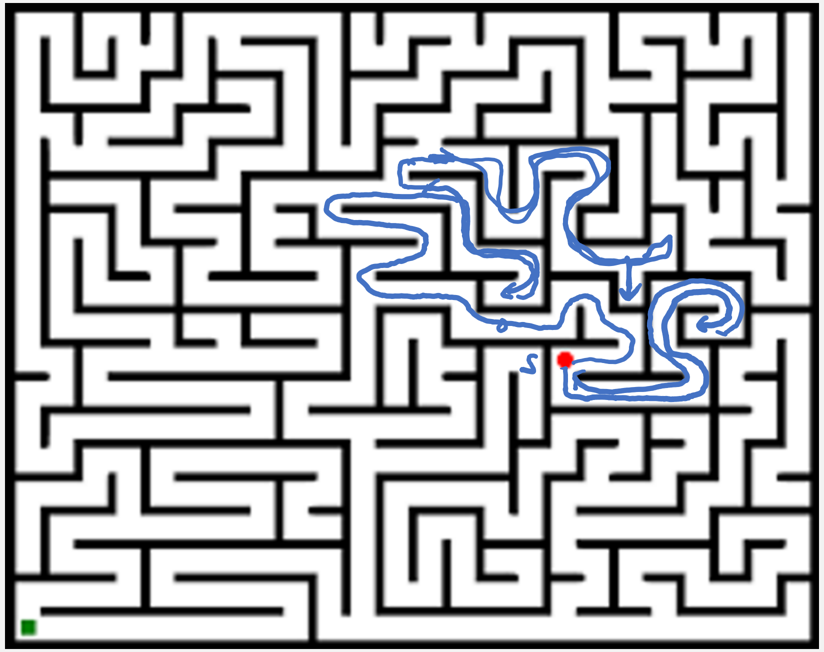
2. DFS Tree, Back Edges, Starting and Finishing Time

3. Cut Vertices and Cut Edges Next time

Motivating Example

Imagine we are in a maze searching for the exit.

This can be modeled as an s-t connectivity problem.



t

How would you search for a path in the maze?

no friends. BFS. not efficient. back and forth

chalk. marks on the floor random walk $O(n^2)$ 466

Depth First Search

Input : an undirected graph $G=(V,E)$, a vertex $s \in V$.

Output : all vertices reachable from s.

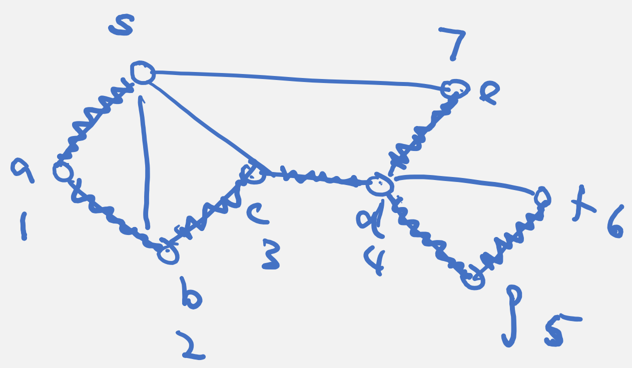
[Main program] visited[v] = false $\forall v \in V$, visited [s] = true. explore(s).

explore (u) // recursive function explore.

for each neighbor v of u

if visited [v] = false

visited[v] = true . explore(v).



DFS and BFS

Time Complexity:

each vertex v is called explore(v) at most once

for loop. deg(v) operations

total time complexity: $O(n + \sum_{v \in V} \deg(v)) = O(n+m)$.

Stack and Queue:

DFS recursive

[KT] Alex

non-recursive implementation of DFS, by using a stack (exercise)

syntatically the same as BFS   BFS FIFO   DFS LIFO

Graph Connectivity:

1. graph connected? ✓
2. components ✓
3. u,v connected ✓
4. shortest path X

visited[v] = true iff $\exists$ a path from s to v.

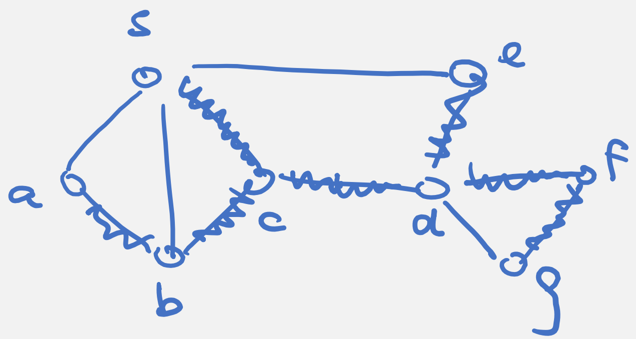
Today's Plan

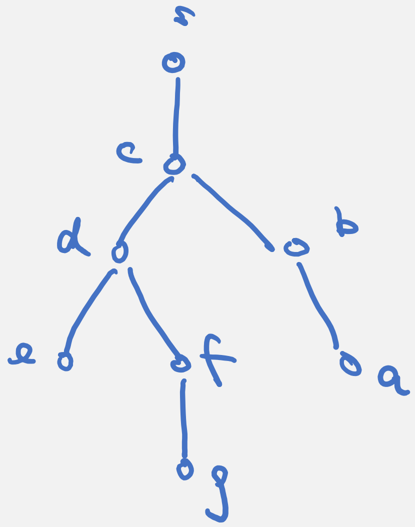
1. Depth First Search
2. DFS Tree, Back Edges, Starting and Finishing Time
3. Cut Vertices and Cut Edges

DFS Tree

As for BFS, we can construct a DFS tree to trace out the paths from s.

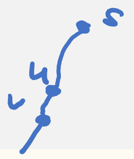
DFS tree : $(v, \text{parent}[v])$ $\forall v$





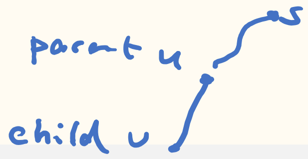
Definitions/Terminology for DFS Tree

- The starting vertex s is regarded as the <u>root</u> of the DFS tree.



- A vertex u is called the <u>parent</u> of a vertex v if the edge uv is in the DFS tree.

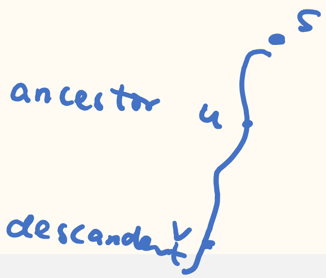
  and u is closer to the root than v is to the root.

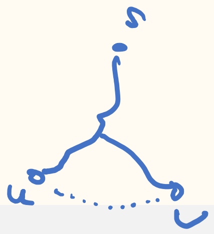


- A vertex u is called an <u>ancestor</u> of a vertex v if u is closer to the root than v.

  and u is on the path from v to the root.

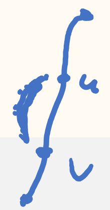
In this situation, we also say v is a <u>descendant</u> of vertex u.





- A non-tree edge uv is called a **<u>back edge</u>** if either u is an ancestor or descendant of v.

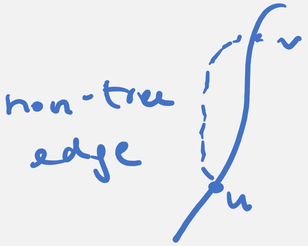
It is called a back edge because this edge from the descendant to the ancestor.

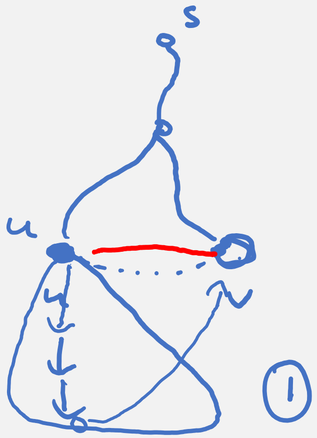


Back Edges

Property (back edges). In an undirected graphs, all non-tree edges are back edges.

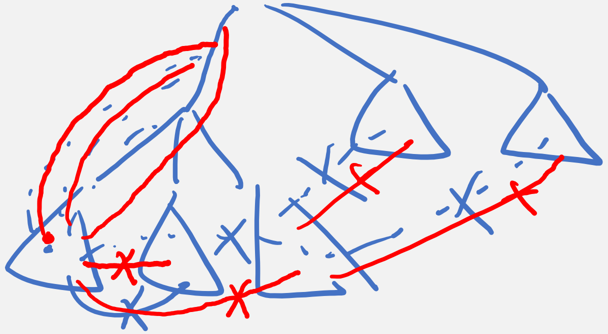
proof

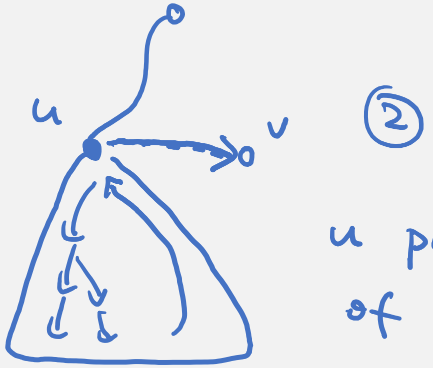




this case is not possible

WLOG explore u first before v





u parent of v

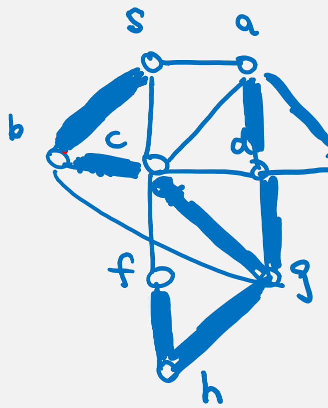
□

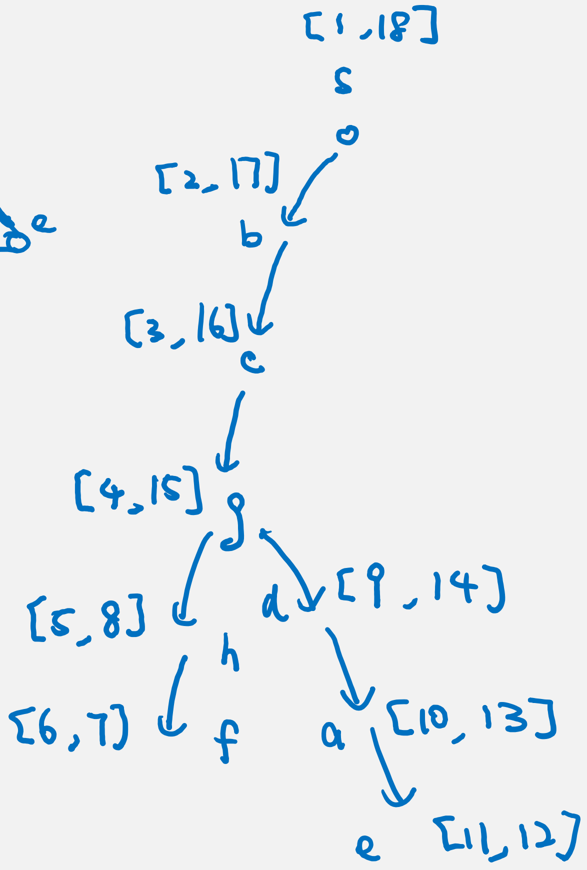
Starting Time and Finishing Time

We record the time when a vertex is first visited and when a vertex finished exploring.

[Main program]   visited[v] = false $\forall v \in V$.   time = 1.   visited[s] = true.   explore(s).

```
explore(u)          // recursive function explore.
    start[u] = time.     time ← time + 1.
    for each neighbor v of u
        if visited[v] = false
            visited[v] = true.    explore(v).
    finish[u] = time.    time ← time + 1.
```





Parenthesis Property

Property (parenthesis). The intervals [start[u],finish[u]] and [start[v],finish[v]] are either disjoint ② or one interval contains ① the other.

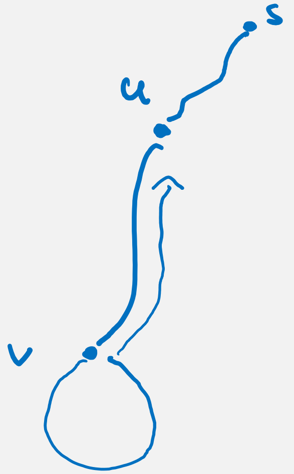
[ ] [ ] ①
u u v v

[[ ]] ②
u v v u

[[ ]] ③ ✕
s[u] s[v] f[u] f[v]

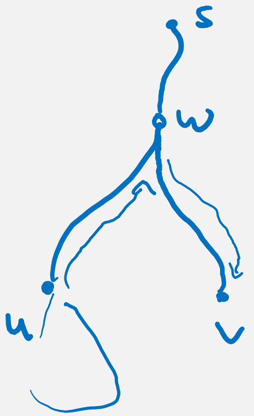
Proof

①



ancestor
descendant
pair

②



not
ancestor
descendant
pair

start[u] < start[v] < finish[v] < finish[u]

[[ ]]
u v v u

start[u] < finish[u] < start[v] < finish[v]

3 10 17 25
[ ] [ ]
u u v v

□

Today’s Plan

1. Depth First Search

2. DFS Tree, Back Edges, Starting and Finishing Time

3. Cut Vertices and Cut Edges

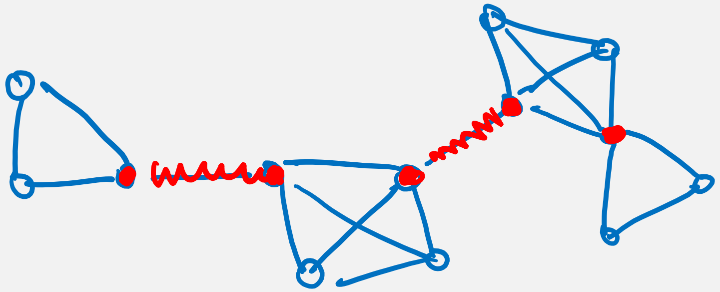
Cut Vertices and Cut Edges

A vertex v is a cut vertex if $G - v$ is not a connected graph.

G connected

An edge e is a cut edge if $G - e$ is not a connected graph.

We would like to design an algorithm to identify all cut vertices and cut edges of a given graph.

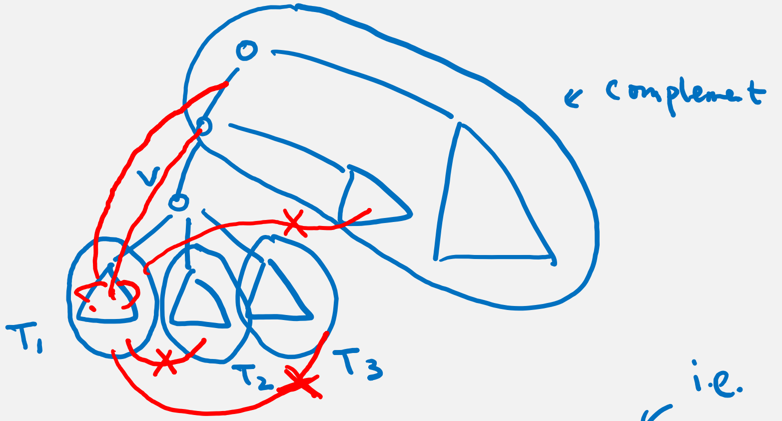


naive : compute G-v , check whether G-v connected or not , for all v

time: $O(n(n+m))$

Idea and Observation

The idea is to use a DFS tree to help us identify all cut vertices and cut edges.

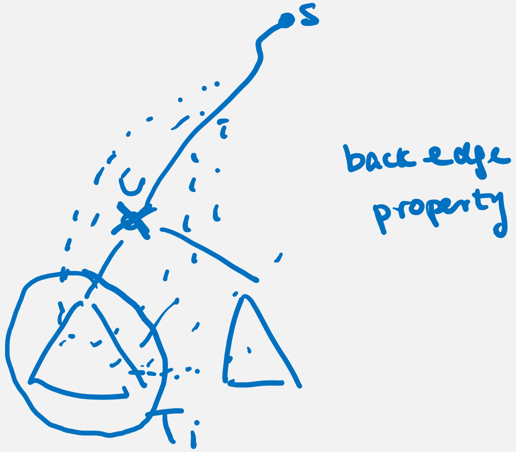


i.e. T_i is disconnected

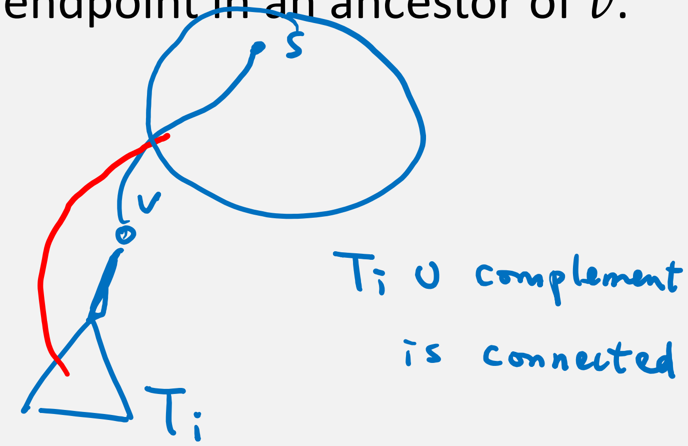
Claim. A subtree T_i below v is a connected component in $G - v$ if and only if

there are no edges with one endpoint in T_i and another endpoint in an ancestor of v.

$\Leftarrow$



$\Rightarrow$



Characterization for a Non-Root Vertex

The argument applies to every subtree gives a characterization for a non-root vertex to be a cut vertex.

Lemma. For a non-root vertex v in a DFS tree, v is a cut vertex if and only if

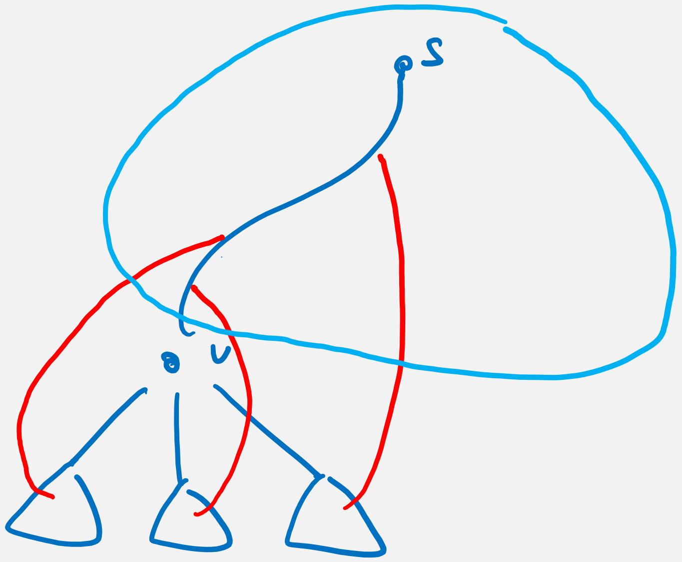
there is a subtree below v with no edges going to an ancestor of v.

proof

$\Leftarrow$

then T_i will be disconnected
by the previous claim

$\Rightarrow$



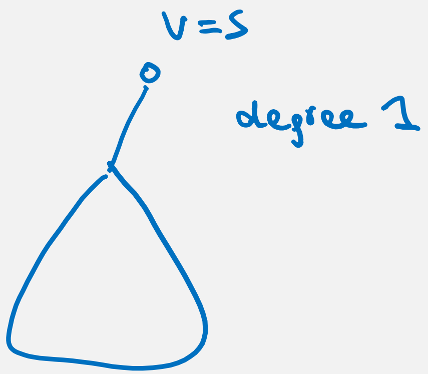
then every T_i is connected to
the complement, and so $G-v$ is connected

Characterization for the Root Vertex

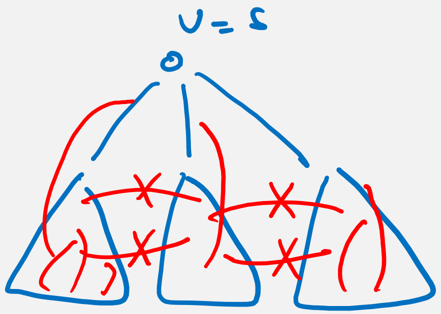
The characterization for the root vertex is simple.

Lemma. For the root vertex v in a DFS tree, v is a cut vertex if and only if v has at least two children.

proof

① 

v is not a cut vertex

② 

v is a cut vertex

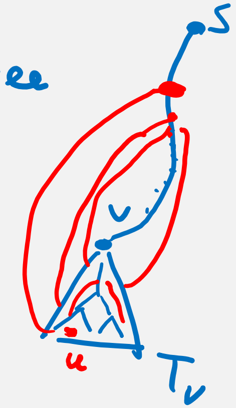
Algorithm

need to check : for each subtree rooted at v, how far up we can go?

need to come up a ~~me~~ parameter to measure how close to the root

e.g. starting time $\quad start[u] < start[v]$

depth $\quad$ depth = distance to the root in DFS tree



Key parameter : $low[v] = \min \left\{ \begin{array}{l} start[w] \\ \text{or } depth[w] \end{array} \middle| \begin{array}{l} \text{for all edges } uw \\ \text{with } u \in T_v \end{array} \right\}$

1. $low[v]$ $\forall v$ can be computed in $O(n+m)$ time

2. identify all cut vertices using $low[\,]$ in $O(n+m)$ time

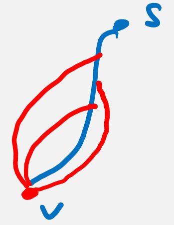
Computing the low[] in Linear Time

To have an efficient implementation, we process the DFS tree using a bottom-up ordering.

base case : v is a leaf

$\mathrm{low}[v] = \min \{ \mathrm{start}[w] \mid vw \in E \}$

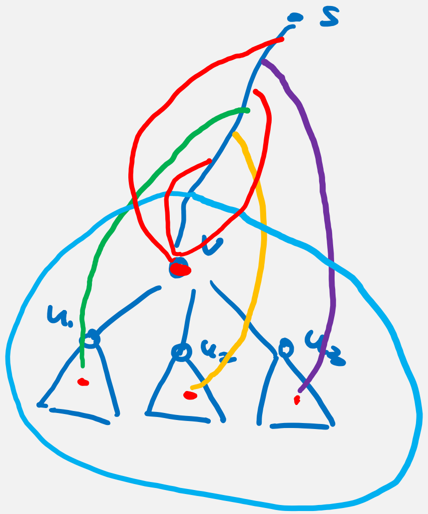
time : $O(\deg(v))$



induction step : by IH, computed $\mathrm{low}(u_i)$ correctly

$\mathrm{low}[v] = \min \{ \mathrm{low}[u_1], \mathrm{low}[u_2], \mathrm{low}[u_3],$

$\min \{ \mathrm{start}[w] \mid vw \in E \} \}$



time : $O(\deg(v))$

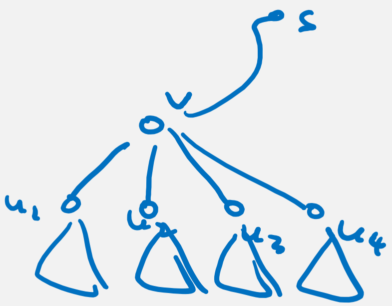
$\Rightarrow$ total time : $O(\sum_{v \in V} \deg(v)) = O(n+m)$.

Identify Cut Vertices Using low[]

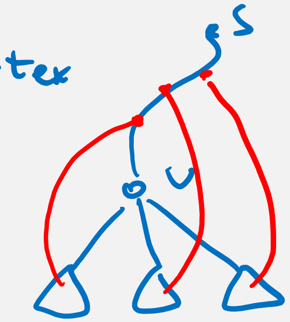
non-root

to check whether v is a cut vertex

we check whether $low[u_i] < start[v]$ $\forall i$

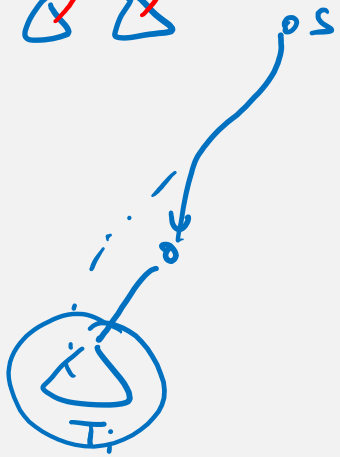


① If yes, then v is not a cut vertex



② If no, then v is a cut vertex

because T_i is disconnected in $G-v$.



Exercise: Identify all cut edges in linear time.