Overview

- Depth-First Search
  - Basic Idea
  - Algorithm
  - DFS Tree
  - Start Time and Finish Time
  - Cuts

- Acknowledgements
Basic Idea

- Exploring a maze
- Would like to explore a full path of the maze, before backtracking and trying the other paths
Depth-First Search Algorithm

- **Input:** Graph $G(V, E)$, vertex $s \in V$
- **Output:** connected component of $s$

Main algorithm:
1. Initialize $\text{visited}[v] = 0$ for all $v \in V$
2. Set $\text{visited}[s] = 1$
3. EXPLORE($s$, visited)

Runtime analysis:
- Initialization takes $O(n)$ time.
- We call EXPLORE at most once per vertex $u \in V$, and once called, we will run through a loop of length $\deg(u)$ and perform $O(1)$ operations before we call EXPLORE on another vertex.

$O(n) + \sum_{u \in V} \deg(u) = O(n + m)$
Depth-First Search Algorithm

- **Input:** Graph $G(V, E)$, vertex $s \in V$
- **Output:** connected component of $s$
- Easiest way to describe algorithm is recursively.
- Subroutine given by
- **EXPLORE**(u, visited):
  1. for each $v \in N(u)$:
     - If visited[v] = 0, then visited[v] = 1 and **EXPLORE**(v, visited).
Depth-First Search Algorithm

- **Input:** Graph $G(V, E)$, vertex $s \in V$
- **Output:** connected component of $s$

**EXPLORE($u$, visited):**
- for each $v \in N(u)$:
  - If visited[$v$] = 0, then visited[$v$] = 1 and EXPLORE($v$, visited).

**Main algorithm:**
- initialize visited[$v$] = 0 for all $v \in V$
- set visited[$s$] = 1
- EXPLORE($s$, visited)

**Runtime analysis:**
Initialization takes $O(n)$ time. We call EXPLORE at most once per vertex $u \in V$, and once called, we will run through a loop of length $\deg(u)$ and perform $O(1)$ operations before we call EXPLORE on another vertex.

$O(n) + \sum_{u \in V} \deg(u) = O(n + m)$
Depth-First Search Algorithm

- **Input:** Graph \( G(V, E) \), vertex \( s \in V \)
- **Output:** connected component of \( s \)
- \( \text{EXPLORE}(u, \text{visited}) \):
  1. for each \( v \in N(u) \):
     - If \( \text{visited}[v] = 0 \), then \( \text{visited}[v] = 1 \) and \( \text{EXPLORE}(v, \text{visited}) \).
- **Main algorithm:**
  1. initialize \( \text{visited}[v] = 0 \) for all \( v \in V \)
  2. set \( \text{visited}[s] = 1 \)
  3. \( \text{EXPLORE}(s, \text{visited}) \)
- **Runtime analysis:** initialization takes \( O(n) \) time. We call \( \text{EXPLORE} \) at most once per vertex \( u \in V \), and once called, we will run through a loop of length \( \deg(u) \) and perform \( O(1) \) operations before we call \( \text{EXPLORE} \) on another vertex.

\[
O \left( n + \sum_{u \in V} \deg u \right) = O(n + m)
\]
Connectivity

Lemma (Connectivity)

There is an \( s - t \) path in \( G \) \iff \( \text{visited}[t] = 1 \) at the end of DFS.

- Same proof idea as we did in BFS

(exercise)
(Augmented) Depth-First Search Algorithm

- **EXPLORE**(\(u\), visited, \(p\)):
  1. for each \(v \in N(u)\):
     - If visited[\(v\)] = 0, then
       - visited[\(v\)] = 1, \(p[v] = u\)
       - and EXPLORE(\(v\), visited, \(p\)).

- **Main algorithm**:
  1. initialize visited[\(v\)] = 0 and \(p[v] = NULL\) for all \(v \in V\)
  2. set visited[\(s\)] = 1
  3. EXPLORE(\(s\), visited, \(p\))
DFS Tree

- In the same way that BFS gave us a tree, DFS will also give us a tree \( T \), with edges \((u, p(u))\) for all \( u \) in the connected component of \( s \).
- This tree has different properties than the BFS tree
  
  In particular, **NO** shortest paths.
DFS Tree

- In the same way that BFS gave us a tree, DFS will also give us a tree $T$, with edges $(u, p(u))$ for all $u$ in the connected component of $s$.
- This tree has different properties than the BFS tree.
  - In particular, NO shortest paths.
- What can we use it for?
DFS Tree

- In the same way that BFS gave us a tree, DFS will also give us a tree $T$, with edges $(u, p(u))$ for all $u$ in the connected component of $s$.
- This tree has different properties than the BFS tree
  - In particular, NO shortest paths.

- Helpful to think of this tree as giving an “orientation” of the edges of the graph
  - Starting vertex $s$ is the root of $T$
  - A vertex $u \in V$ is the parent of $v$ if the edge $\{u, v\} \in T$ and $u$ closer to the root
  - Vertex $u$ is the ancestor of $v$ if $u$ closer to root and $u$ is in the $s – v$ path in $T$. We say $v$ is a descendant of $u$ and that $u, v$ are related.
  - A non-tree edge $\{u, v\}$ will be called back edge if $u$ is the ancestor of $v$ (or vice-versa).
DFS Tree

- In the same way that BFS gave us a tree, DFS will also give us a tree \( T \), with edges \((u, p(u))\) for all \( u \) in the connected component of \( s \).
- This tree has different properties than the BFS tree
  
  In particular, NO shortest paths.

- Helpful to think of this tree as giving an “orientation” of the edges of the graph

  - Starting vertex \( s \) is the root of \( T \)
  - A vertex \( u \in V \) is the parent of \( v \) if the edge \( \{u, v\} \in T \) and \( u \) closer to the root
  - Vertex \( u \) is the ancestor of \( v \) if \( u \) closer to root and \( u \) is in the \( s \rightarrow v \) path in \( T \). We say \( v \) is a descendant of \( u \) and that \( u, v \) are related.
  - A non-tree edge \( \{u, v\} \) will be called back edge if \( u \) is the ancestor of \( v \) (or vice-versa).

- What are relationships between related vertices in this tree?
(Augmented) Depth-First Search Algorithm (again)

EXPLORE(\(u, \text{ visited}, p, S, F, \tau\)):

1. \(S[u] = \tau\), and \(\tau \leftarrow \tau + 1\)
2. for each \(v \in N(u)\):
   - If \(\text{visited}[v] = 0\), then
     \(\text{visited}[v] = 1\), \(p[v] = u\)
     and EXPLORE(\(v, \text{ visited}, p, S, F, \tau\)).
3. \(F[u] = \tau\) and \(\tau \leftarrow \tau + 1\)

Main algorithm:

1. initialize \(\text{visited}[v] = 0\), \(S[v] = F[v] = \infty\) and \(p[v] = \text{NULL}\) for all \(v \in V\)
2. set \(\text{visited}[s] = 1\) and \(\tau = 1\)
3. EXPLORE(\(s, \text{ visited}, p, S, F, \tau\))
Lemma (Parenthesis lemma)

For any pair $u, v \in V$, the intervals $[S(u), F(u)]$ and $[S(v), F(v)]$ are either disjoint or one is contained in the other (the descendant is contained in the ancestor).

- Follows easily from augmented algorithm, as we only finish an ancestor after going through all its descendants.
DFS Tree Properties

A corollary of the parenthesis lemma is the following:

**Lemma (Back edge lemma)**

In an undirected graph $G$, all non-DFS-tree edges are back edges.
A corollary of the parenthesis lemma is the following:

**Lemma (Back edge lemma)**

In an undirected graph $G$, all non-DFS-tree edges are back edges.

- Suppose there is edge $\{u, v\} \in E$
- W.l.o.g. can assume $u$ visited by DFS before $v$. Thus, $S[u] < S[v]$
A corollary of the parenthesis lemma is the following:

**Lemma (Back edge lemma)**

*In an undirected graph $G$, all non-DFS-tree edges are back edges.*

- Suppose there is edge $\{u, v\} \in E$
- W.l.o.g. can assume $u$ visited by DFS before $v$. Thus, $S[u] < S[v]$
- Since $v \in N(u)$, $v$ will be explored before EXPLORE($u$) is finished, thus $S[v] < F[u]$
A corollary of the parenthesis lemma is the following:

**Lemma (Back edge lemma)**

*In an undirected graph G, all non-DFS-tree edges are back edges.*

- Suppose there is edge \( \{u, v\} \in E \)
- W.l.o.g. can assume \( u \) visited by DFS before \( v \). Thus, \( S[u] < S[v] \)
- Since \( v \in N(u) \), \( v \) will be explored before \( \text{EXPLORE}(u) \) is finished, thus \( S[v] < F[u] \)
- By parenthesis lemma, we must have \( F[v] < F[u] \). Hence \( v \) is descendent of \( u \).
Depth-First Search
- Basic Idea
- Algorithm
- DFS Tree
- Start Time and Finish Time
- Cuts

Acknowledgements
Definitions

- A vertex $u \in V$ is a **cut vertex**, if removing $u$ from $G$ (and its edges) we disconnect $G$ (also known as articulation point/separating vertex).

- An edge $\{u, v\}$ is a **cut edge** if removing this edge we disconnect the graph (also known as a bridge).

We will use the DFS tree to identify all cut vertices and edges.

Observation: the only way vertex $u$ is a cut vertex is if there are no back edges from a subtree rooted at a child of $u$ to an ancestor of $u$.

One way to compute the above is to keep track of “earliest” vertex in $T$ connected by a back edge to subtree $T_u$: $E[u] = \min S[u], \min w \in T_u S[z]$ s.t. $\{w, z\}$ back edge & $u$ descendant of $z$. 


Definitions

- A vertex $u \in V$ is a **cut vertex**, if removing $u$ from $G$ (and its edges) we disconnect $G$ (also known as articulation point/separating vertex).

- An edge $\{u, v\}$ is a **cut edge** if removing this edge we disconnect the graph (also known as a bridge).

- We will use the DFS tree to identify all cut vertices and edges.
Definitions

- A vertex \( u \in V \) is a **cut vertex**, if removing \( u \) from \( G \) (and its edges) we disconnect \( G \) (also known as articulation point/separating vertex).
- An edge \( \{u, v\} \) is a **cut edge** if removing this edge we disconnect the graph (also known as a bridge).

- We will use the DFS tree to identify all cut vertices and edges.
- **Observation:** only way vertex \( u \) is a cut vertex is if there are no back edges from a subtree rooted at a child of \( u \) to an **ancestor** of \( u \).
Definitions

- A vertex $u \in V$ is a **cut vertex**, if removing $u$ from $G$ (and its edges) we disconnect $G$ (also known as articulation point/separating vertex).
- An edge $\{u, v\}$ is a **cut edge** if removing this edge we disconnect the graph (also known as a bridge).

We will use the DFS tree to identify all cut vertices and edges.

**Observation:** only way vertex $u$ is a cut vertex is if there are no back edges from a subtree rooted at a child of $u$ to an ancestor of $u$.

(One way to) compute the above is to keep track of “earliest” vertex in $T$ connected by a back edge to subtree $T_u$

$$E[u] = \min \left\{ S[u], \min_{w \in T_u} \left( S[z] \text{ s.t. } \{w, z\} \text{ back edge } \& \ u \text{ descendant of } z \right) \right\}$$
Let $T$ be our DFS tree and $T_u$ be the subtree rooted at $u$.

**Lemma (Connected Components)**

Given two vertices $u, v \in T$ such that $u$ is an ancestor of $v$, then a subtree $T_v$ of $T_u$ is a connected component of $G \setminus \{v\}$ iff there are no back edges from $T_v$ to an ancestor of $u$ in $T$. 
Cut vertex lemmas

Let $T$ be our DFS tree and $T_u$ be the subtree rooted at $u$.

**Lemma (Connected Components)**

*Given two vertices $u, v \in T$ such that $u$ is an ancestor of $v$, then a subtree $T_v$ of $T_u$ is a connected component of $G \setminus \{v\}$ iff there are no back edges from $T_v$ to an ancestor of $u$ in $T$.*

**Lemma (Cut vertex - non-root)**

*For non-root vertex $u \in T$, $u$ is a cut vertex iff there is subtree $T_v \subset T_u$ with $v$ descendant of $u$, with no back edges to an ancestor of $u$.*
Cut vertex lemmas

Let $T$ be our DFS tree and $T_u$ be the subtree rooted at $u$.

**Lemma (Connected Components)**

Given two vertices $u, v \in T$ such that $u$ is an ancestor of $v$, then a subtree $T_v$ of $T_u$ is a connected component of $G \setminus \{v\}$ iff there are no back edges from $T_v$ to an ancestor of $u$ in $T$.

**Lemma (Cut vertex - non-root)**

For non-root vertex $u \in T$, $u$ is a cut vertex iff there is subtree $T_v \subset T_u$ with $v$ descendant of $u$, with no back edges to an ancestor of $u$.

**Lemma (Cut vertex - root)**

If $s \in T$ is the root of $T$, then $s$ is a cut vertex iff $s$ has two children.
(Augmented) DFS Algorithm (again, for real?)

- **EXPLORE**(*u*, visited, *p*, *S*, *F*, *τ*, *E*):
  1. *S*[u] = *τ*, and *τ* ← *τ* + 1
  2. for each *v* ∈ *N*(u):
     - If visited[*v*] = 0, then
       visited[*v*] = 1, *p*[v] = *u*
       and EXPLORE(*v*, visited, *p*, *S*, *F*, *τ*, *E*).
  3. *F*[u] = *τ*, *τ* ← *τ* + 1 and

     \[E[u] = \min \left\{ S[u], \min_{\{uw\} \text{back edge}} S[w], \min_{\text{v child of } u} E[v] \right\}\]

- **Main algorithm**:
  1. initialize visited[*v*] = 0, *S*[v] = *F*[v] = *E*[v] = ∞ and *p*[v] = NULL for all *v* ∈ *V*
  2. set visited[s] = 1 and *τ* = 1
  3. EXPLORE(s, visited, *p*, *S*, *F*, *τ*, *E*)
Correctness of augmented algorithm

- All that is left to prove is that above algorithm computes $E[u]$ correctly for each $u \in V$
- Can prove this by induction on depth of the tree, starting from the leaves. We will make sure to prove that $E[u]$ computes the starting time of the earliest direct neighbor of $T_u$.
- Inductive step: if have computed $E[v]$ correctly for every non-root of $T_u$, then step 3 of the EXPLORE algorithm will correctly compute $E[u]$
Finding cut vertices

**Lemma**

Vertex $u \in T$ is not a cut vertex iff $S[u] > E[v]$ for all children $v$ of $u$ in $T$.

- $E[v]$ captures the start time of the earliest vertex which directly connects to $T_v$ (via a back edge).
- $w \in T_v$ and $w, v \in T_u \Rightarrow E[w] \geq E[v]$, as back edge from $T_w$ to ancestor of $u$ is a back edge from $T_v$ to ancestor of $u$ hence
- $S[u] > E[v] \Rightarrow$ there is a back edge from $T_v$ is an ancestor of $u$
- By previous bullet, enough to focus on children of $u$
- If every children $v$ of $u$ has $E[v] < S[u]$, then $T_v$ is connected in $G \setminus \{u\}$. Thus $u$ not cut vertex.
Finding cut vertices

**Lemma**

Vertex $u \in T$ is not a cut vertex iff $S[u] > E[v]$ for all children $v$ of $u$ in $T$.

- $E[v]$ captures the start time of the earliest vertex which directly connects to $T_v$ (via a back edge).
- $w \in T_v$ and $w, v \in T_u \Rightarrow E[w] \geq E[v]$, as back edge from $T_w$ to ancestor of $u$ is a back edge from $T_v$ to ancestor of $u$ hence
- $S[u] > E[v] \Rightarrow$ there is a back edge from $T_v$ is an ancestor of $u$.
- By previous bullet, enough to focus on children of $u$.
- If every children $v$ of $u$ has $E[v] < S[u]$, then $T_v$ is connected in $G \setminus \{u\}$. Thus $u$ not cut vertex.
- Other direction analogous.
Acknowledgement

- Based on Prof. Lau's lecture 06
  
  https://cs.uwaterloo.ca/~lapchi/cs341/notes/L06.pdf

- For non-recursive version of DFS, see [Kleinberg Tardos 2006]
References I

Cormen, Thomas and Leiserson, Charles and Rivest, Ronald and Stein, Clifford. (2009)
MIT Press

Kleinberg, Jon and Tardos, Eva (2006)
Algorithm Design.
Addison Wesley