Explore.

Graph $G = (V, E)$.
Size of adjacency list representation? $O(|V| + |E|)$.

Explore$(v)$:
2. For each edge $(v,w)$ in $E$
3. if not visited[w]: Explore(w)

Property:
All and only nodes reachable from A are reached by explore.
Idea:
Reachable means there is a path, and there is no first unexplored node since the previous node would explore it.

Runtime: $O(|V| + |E|)$.
The time is proportional to total size of the adjacency lists.

Explore.

Connected Components.

explore$(v)$:
2. previsit$(v)$
3. For each edge $(v,w)$ in $E$
4. if not visited[w]: explore(w).
5. postvisit$(v)$

Previsit$(v)$:

DFS$(G)$:
0. Set cc := 0.
1. for each $v$ in $V$:
2. if not visited[v]:
3. explore$(v)$
4. ccnum = ccnum + 1

Each node will be labelled with connected component number.
Runtime: $O(|V| + |E|)$.

Explore.

Depth First Search.
Call explore until explore the whole graph.
Connected Components.
Tree/back edges.
Pre/Post Ordering.
Interval of time “on stack”.
Quick cycle test.
Directed Graphs.
Tree/Back/Forward/Cross edges.
From pre/post!
Back Edge $\iff$ cycle!
Topological Sort.
Alg 1: Inverse Post order number.
Inverse order of “stack” pop.
Alg 2: Peeling off sources.

Explore.

Depth first search.
Process whole graph.
DFS$(G)$
1. For each node $u$,
2. visited[$u$] = false.
3. For each node $u$,
4. if not visited[u]: explore(u)

Running time: $O(|V| + |E|)$.
Intuitively: tree for each “connected component”.
Several trees or Forest! Output connected components?

Explore.

DFS and connected components.

Change explore a bit:
explore$(v)$:
2. previsit$(v)$
3. For each edge $(v,w)$ in $E$
4. if not visited[w]: explore(w).
5. postvisit$(v)$

DFS$(G)$:
0. Set cc := 0.
1. for each $v$ in $V$:
2. if not visited[v]:
3. explore$(v)$
4. ccnum = ccnum + 1

Each node will be labelled with connected component number.
Runtime: $O(|V| + |E|)$.

Explore.

Connected Components.

explore$(v)$:
2. previsit(v)
3. For each edge (v,w) in E
4. if not visited[w]: explore(w).
5. postvisit(v)

Previsit(v):

DFS(G):
0. Set cc := 0.
1. for each v in V:
2. if not visited[v]:
3. explore(v)
4. ccnum = ccnum + 1

Each node will be labelled with connected component number.
Runtime: $O(|V| + |E|)$.
Introspection: pre/post.

\[
\text{Previsit}(v): \quad \begin{align*}
1. & \quad \text{Set } \text{pre}[v] := \text{clock}. \\
2. & \quad \text{clock} := \text{clock} + 1 
\end{align*}
\]

\[
\text{Postvisit}(v): \quad \begin{align*}
1. & \quad \text{Set } \text{post}[v] := \text{clock}. \\
2. & \quad \text{clock} := \text{clock} + 1 
\end{align*}
\]

DFS(G):

\[
\begin{align*}
0. & \quad \text{Set } \text{clock} := 0. \\
\cdots 
\end{align*}
\]

Clock: goes up to 2 times number of tree edges.

First pre: 0

Property: For any two nodes, \( u \) and \( v \), \([\text{pre}(u), \text{post}(u)]\) and \([\text{pre}(v), \text{post}(v)]\) are either disjoint or one is contained in other.

Interval is “clock interval on stack.”

\[
\begin{align*}
u & \quad \vdots \\
v & \quad \vdots 
\end{align*}
\]

Either both on stack at some point (contained) or not (disjoint.)

Let’s just watch it work!

Example: Pre/Post numbering.

\[
\begin{array}{cccccc}
A & B & C & D & E & \ \ \\
0 & 1 & 2 & 3 & 4 & \\
A & E & D & C & B & G \\
\end{array}
\]

Explored edge \((u, v)\) first from \( u \).

Tree edge iff \([\text{pre}(v), \text{post}(v)]\) \(\subseteq\) \([\text{pre}(u), \text{post}(u)]\).

\( u \) on stack before \( v \).

Back edge iff \([\text{pre}(u), \text{post}(u)]\) \(\subseteq\) \([\text{pre}(v), \text{post}(v)]\).

\( v \) on stack when \( v \) on stack. Path from \( v \) to \( u \)! Cycle!

No edge between \( u \) and \( v \) if disjoint intervals.

Directed graphs.

\[
G = (V, E)
\]

vertices \( V \).

edges \( E \subseteq V \times V \).

Edge: \((u, v)\)

From \( u \) to \( v \).

Tail – \( u \)

Head – \( v \)

Depth first search: directed.

Terminology:

Root: Starting point.

\( v \) is ancestor of \( u \):

\( v \) on path from/to root.

\( v \) is descendant of \( u \):

\( u \) is an ancestor of \( v \).

Tree/forward edge \((u, v)\): \(\text{int}(v)\) in \(\text{int}(u)\).

Forward \((A, F)\): \([10, 11] \) in \([0, 13] \) or \([0, 10, 11, 13] \)

Back edge \((u, v)\): \(\text{int}(v)\) contains \(\text{int}(u)\).

\((C, B)\): \([0, 4] \) in \([1, 8] \) or \([1, 3, 4, 8] \)

Cross edge \((u, v)\): \(\text{int}(v)\) before \(\text{int}(u)\).

\((F, D)\): \([5, 5] \) before \([10, 11] \)

Directed Acyclic Graphs: Depth First Search

\[
\begin{align*}
0 & \quad \text{A} \quad 12 \quad 13 \\
1 & \quad \text{B} \quad 6 \\
2 & \quad \text{C} \quad 7 \\
3 & \quad \text{D} \quad 5 \\
4 & \quad \text{E} \quad 2 \\
5 & \quad \text{F} \quad 11 \\
6 & \quad \text{G} \quad 1
\end{align*}
\]

Edge: \((u, v)\)

From \( u \) to \( v \).

Tail – \( u \)

Head – \( v \)

Tree edge – “Direct call tree of explore.”

Forward edge – “Edge to descendant (not in tree)”

Back edge – “Edge to ancestor”

Cross edge – None of the above.

\( v \) already explored before \( u \) is visited.
**Directed Acyclic Graph**

Directed Graph ... without cycles. Cycle: \( v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_k \rightarrow v_0 \).

Why?

```
Hello
```

```
Goodbye
```

"Hello" before "Goodbye"

---

**Example.**

```
Acyclic Graph?
```

```
C
E
D
A
F
B
G
```

---

**Fast checking algorithm.**

**Thm:** A graph has a cycle if and only if there is back edge.

**Proof:**

- Back edge \( \rightarrow \) cycle!
- There is a cycle
  \( v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_k \rightarrow v_0 \)
  Assume that \( v_0 \) is the first node explored.
  (without loss of generality since can renumber vertices.)
- All nodes on cycle explored when \texttt{explore}(v_0) returns
  For each \( v_i \): \texttt{int}(v_i) \in \texttt{int}(v_0)
  \( \rightarrow \) \( (v_k, v_0) \) is a back edge.
- Cycle \( \rightarrow \) back edge! 

---

**Depth first search: directed.**

```
Back edge (u, v): int(v) contains int(u).
int(C) = [3, 4] and int(B) = [1, 8].
```

```
Back edge (u, v)
--- edge to ancestor
tree edges from v to u.
Back edge means cycle! \( \Rightarrow \) not acyclic!
```

---

**Testing for cycle.**

**Thm:** A graph has a cycle if and only if there is back edge.

**Proof:**

- Back edge \( \Rightarrow \) cycle!
- There is a cycle
  \( v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_k \rightarrow v_0 \)
  Assume that \( v_0 \) is the first node explored.
  (without loss of generality since can renumber vertices.)
- All nodes on cycle explored when \texttt{explore}(v_0) returns
  For each \( v_i \): \texttt{int}(v_i) \in \texttt{int}(v_0)
  \( \Rightarrow \) \( (v_k, v_0) \) is a back edge.
- Cycle \( \Rightarrow \) back edge!

---

**Directed Acyclic Graph**

```
Hello
```

```
Goodbye
```

"Hello" before "Goodbye"

---

**Testing for cycle.**

**Thm:** A graph has a cycle if and only if there is back edge.

- Run DFS. \( O(|V| + |E|) \) time.
- For each edge \( (u, v) \): is \texttt{int}(u) in \texttt{int}(v).
- \( O(|E|) \) time.
- \( O(|V| + |E|) \) time algorithm for checking if graph is acyclic.

---

**Directed Acyclic Graph**

```
Hello
```

```
Goodbye
```

"Hello" before "Goodbye"

---

No cycles! Can tell in linear time!

Ohhh... Kayyy...
Really want to find ordering for build!
Where things are cool!
Linearize.

**Topological Sort**: For $G = (V, E)$, find ordering where each edge goes from earlier vertex to later in acyclic graph.

Source/sinks in a DAG.

**Source** is node with no incoming arcs.
**Sink** is node with no outgoing arcs.

Highest post order node is source.
Lowest post order node is sink.

**Property**: Every DAG has at least one source and sink.

Topological Sort Algorithm: Find source, output, repeat.
Naively: $O(nm)$, there is a better implementation.
Useful on Monday.

Topological Sort Example.

A linear order:
A, E, F, B, G, D, C

In DFS: When is A popped off stack?
Last! E second to last. ...

Topological Sort: DFS

**Property**: Every edge in a DAG $(u, v)$ has $\text{post}(u) > \text{post}(v)$.
No back edges!

Tree and Forward edge $(u, v)$:
\[ \text{int}(u) \text{ contains } \text{int}(v): \text{ pre}(u), \text{pre}[v], \text{post}[v], \text{post}[u] \]

Cross edge $(u, v)$: $\text{int}(u) > \text{int}(v)$

Top Sort: output in reverse post order number.
Runtime: $O(|V| + |E|)$.

..procedure PostVisit outputs during DFS
..reverse.

Lecture in a Minute

Depth First Search.
Call explore until explore the whole graph.
Connected Components.
Tree/back edges.
Back edge $\iff$ cycle

Pre/Post Ordering.
Interval of time “on stack”.
Quick cycle test.

Directed Graphs.
Tree/Back/Forward/Cross edges.
From pre/post!
Back Edge $\iff$ cycle!

Topological Sort.
Alg 1: Inverse Post order number.
Inverse order of “stack” pop.
Alg 2: Peeling off sources.