Analysis of Algorithms

Graph Traversals

Graph Traversals

Input: A simple, undirected, and connected graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges.

Objective: Find a traversal path that

- $\star$ visits all the vertices of the graph and
- $\star$ traverses all the edges of a graph.

Remark: Vertices can be visited more than once and edges can be traversed more than once.

Graph Traversals – General Scheme

⋆ Start with one of the vertices and traverse one of its incident edges to reach another vertex.

– Connectivity implies that this is always possible.

⋆ Using traversed edges, go to one of the visited vertices that has an incident untraversed edge.

– Connectivity implies that this is always possible.

⋆ Traverse this untraversed edge.

⋆ Continue until all the edges are traversed.

– Connectivity implies that all vertices are visited.

Graph Traversals

Efficiency objective:

- ⋆ Apply simple rules to find an untraversed edge.
- ⋆ Implement with an efficient data structure.
- ⋆ Finish fast in $O(m)$ running time.

Example: An Euler path is the shortest possible traversal path if exists – no edge is traversed more than once.

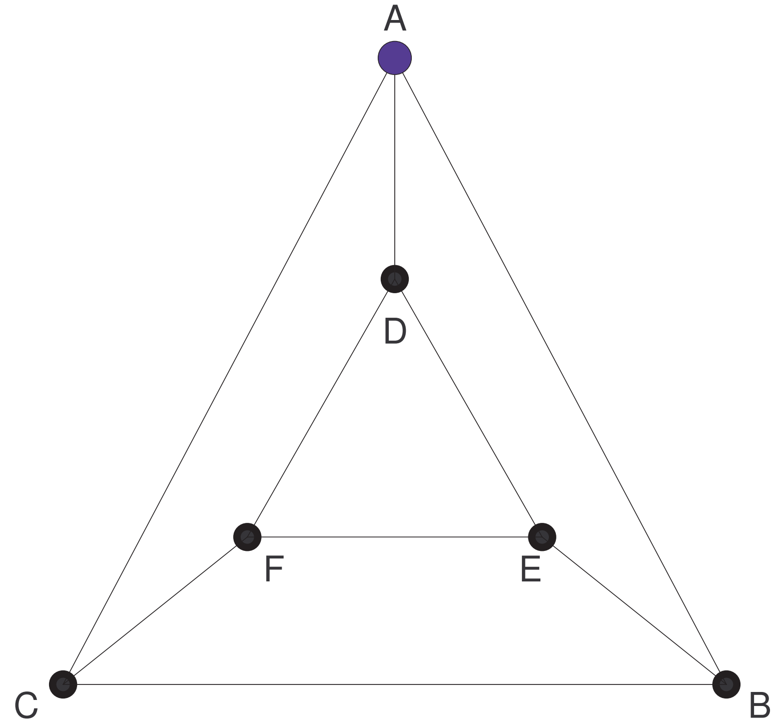
Variants

Directed graphs: An edge is traversed only from its origin to its destination.

Disconnected graphs:

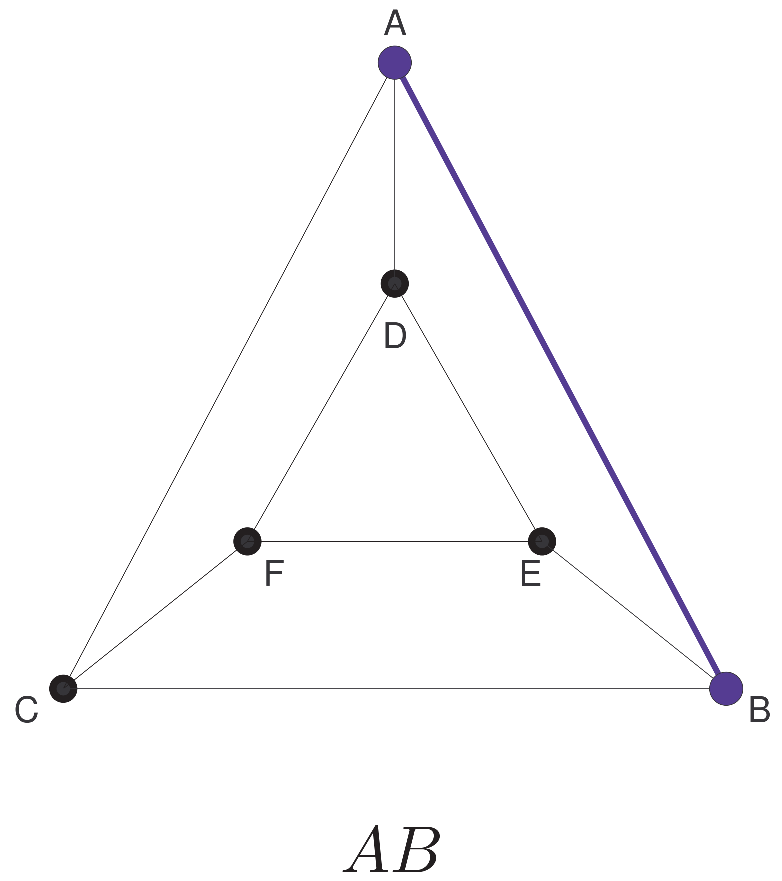
- ⋆ When stuck, jump to an unvisited vertex.
- ⋆ Continue until all vertices are visited and all edges are traversed.
- ⋆ Finish fast in $O(n + m)$ running time.

Example



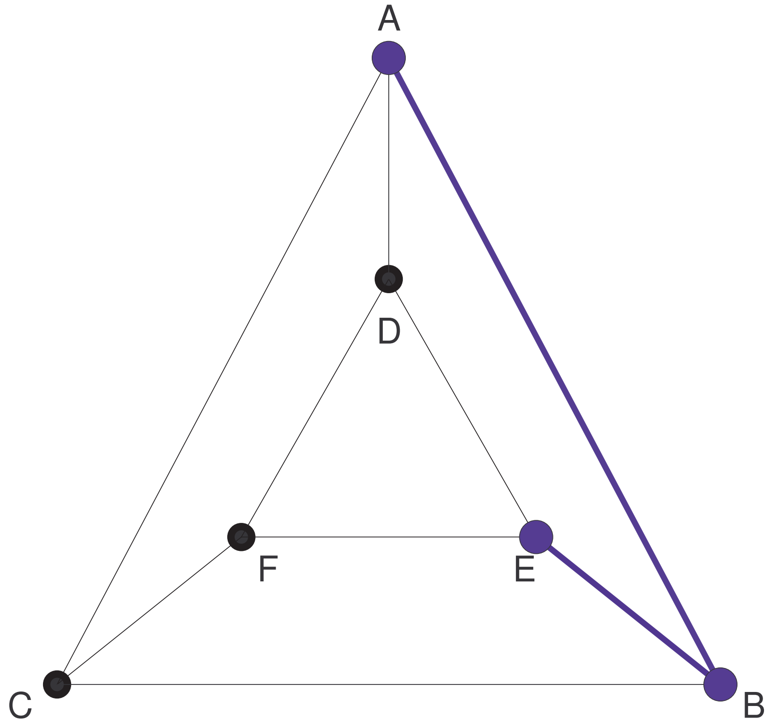
A

Example



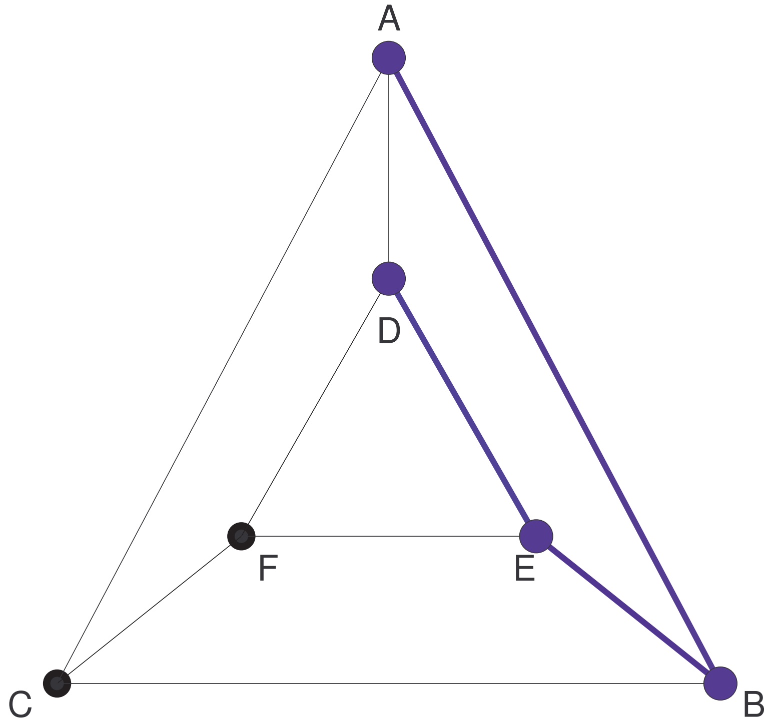
$$AB$$

Example



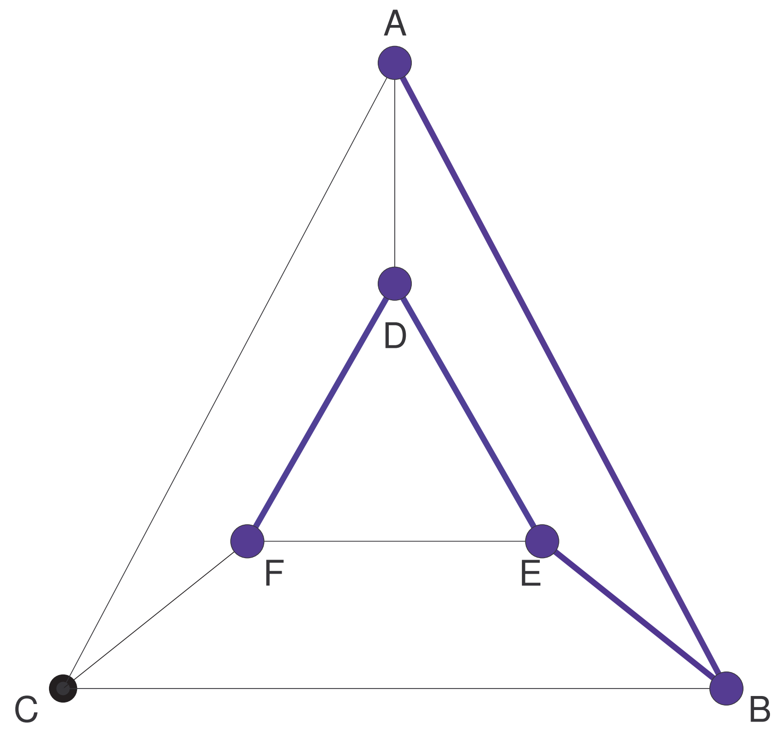
$$ABE$$

Example



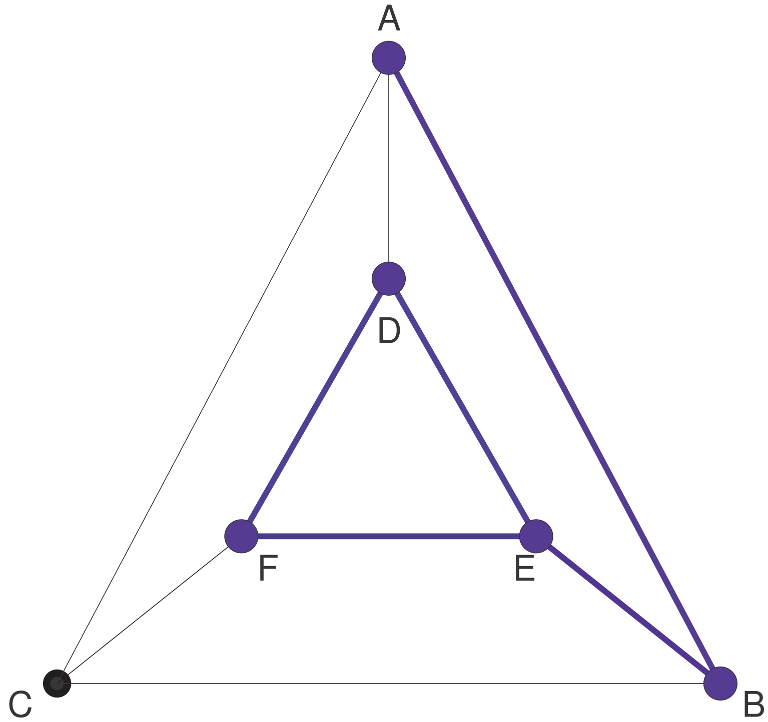
$$ABED$$

Example



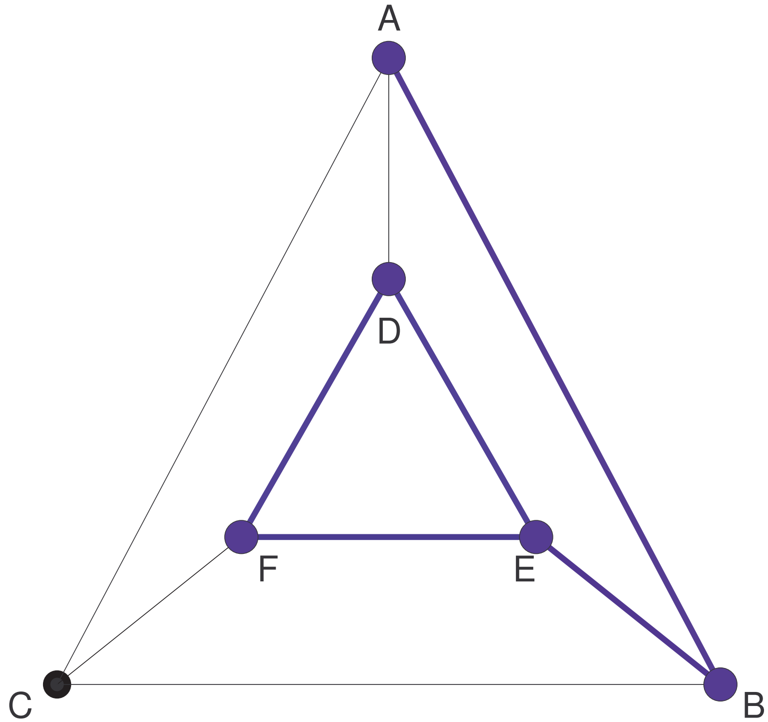
$$ABEDF$$

Example



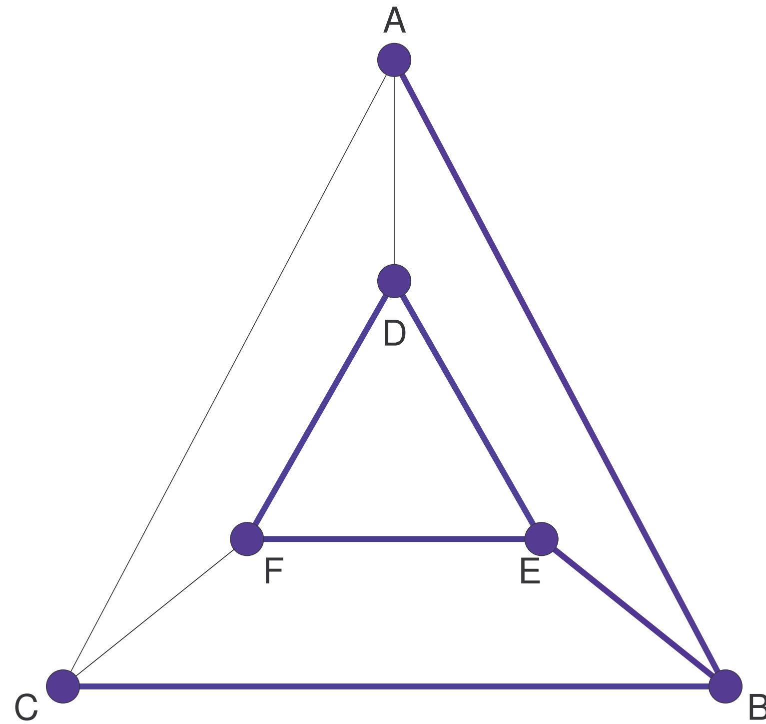
$$ABEDFE$$

Example



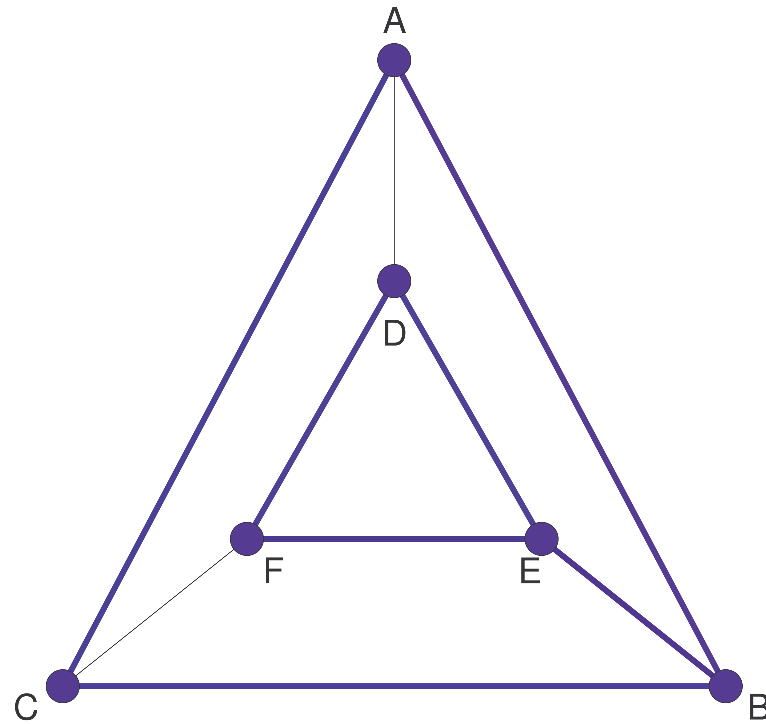
$$ABEDFEB$$

Example



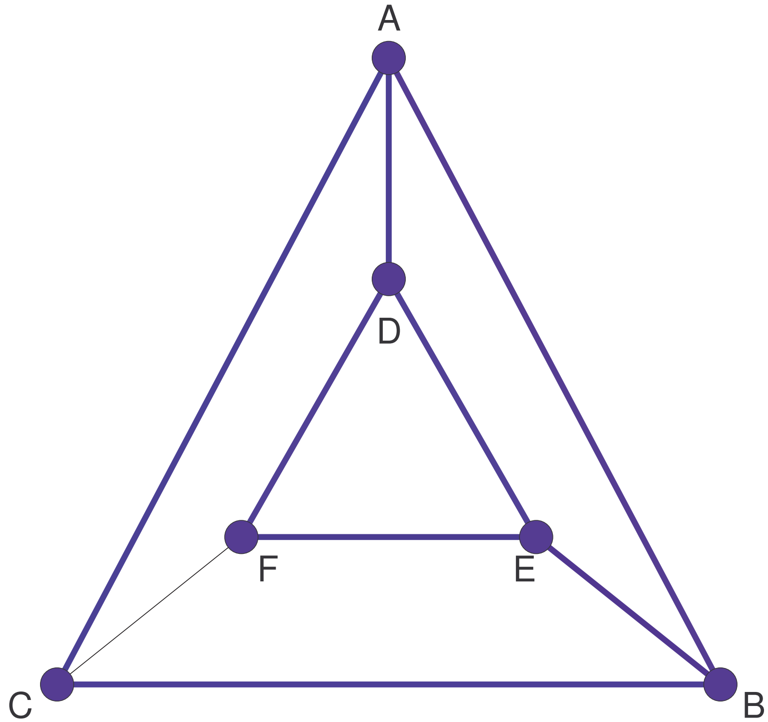
$$ABEDFEBC$$

Example



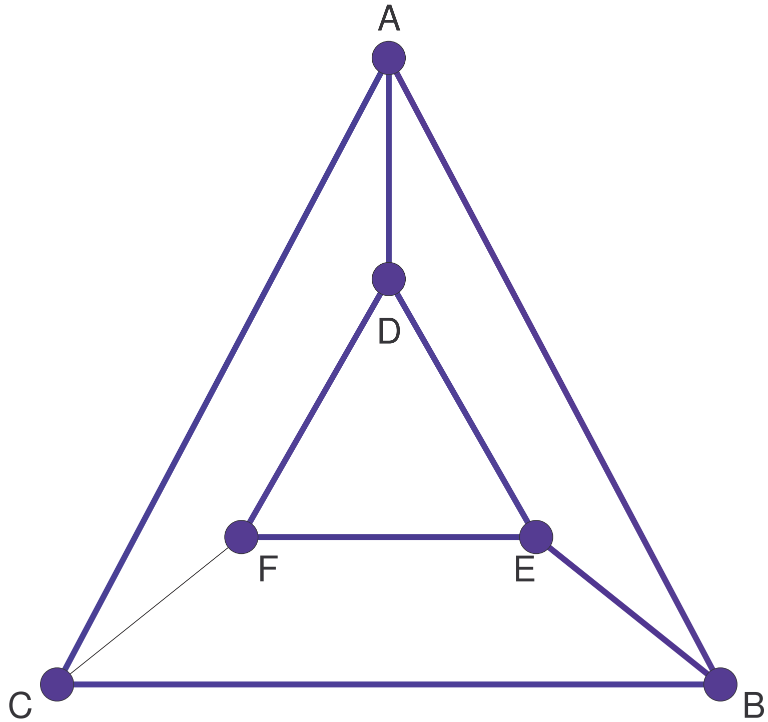
$$ABEDFEBCA$$

Example



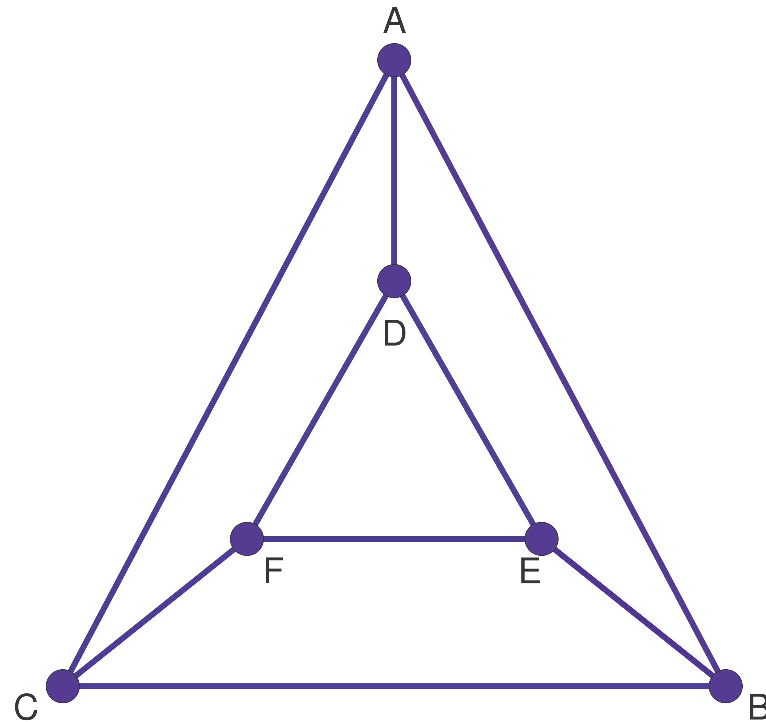
$$ABEDFEBCAD$$

Example



$$ABEDFEBCADF$$

Example



$$ABEDFEBCADFC$$

Traversal Trees

The tree structure: a rooted, ordered, and directed tree.

⋆ The first visited vertex is the root of the tree.

⋆ Vertex u is the parent of v if the first visit to v happened after traversing the edge (u, v).

⋆ The children of a vertex u are ordered according to the time they were first visited from u.

Traversal Trees – Edge Classification

Tree edge: an edge from a vertex to one of its children.

Back edge: an edge from a vertex to one of its ancestors.

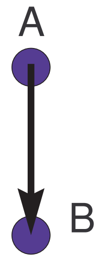
Forward edge: an edge from a vertex to one of its descendants which is not its child.

Cross edge: an edge from a vertex to another vertex that is neither one of its ancestors nor one of its descendants in the traversal tree.

Example

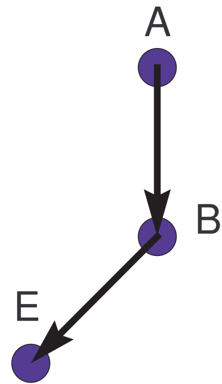
$$A$$

Example



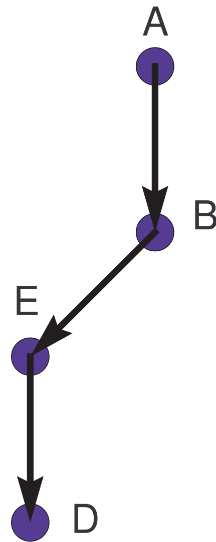
$$AB$$

Example



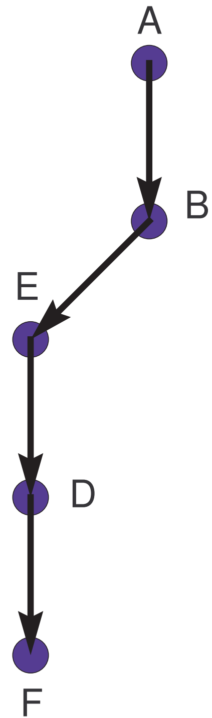
$$ABE$$

Example



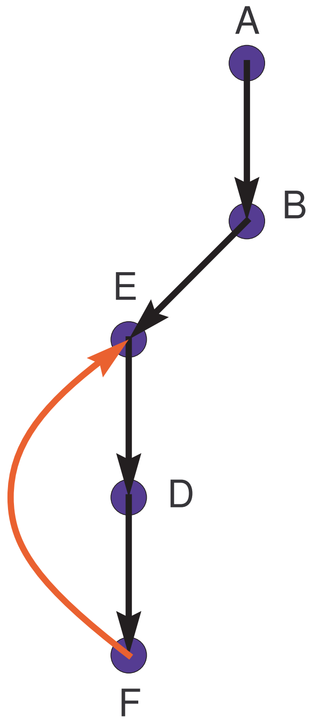
$$ABED$$

Example



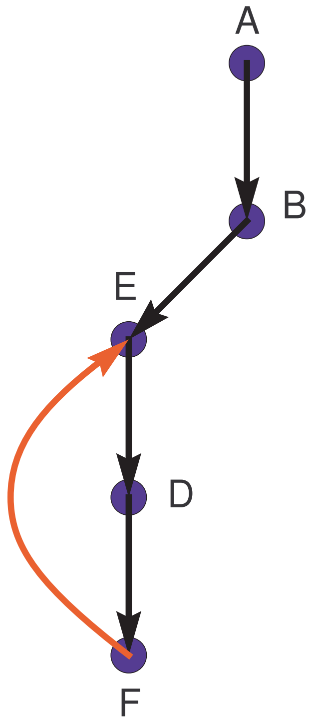
$$ABEDF$$

Example



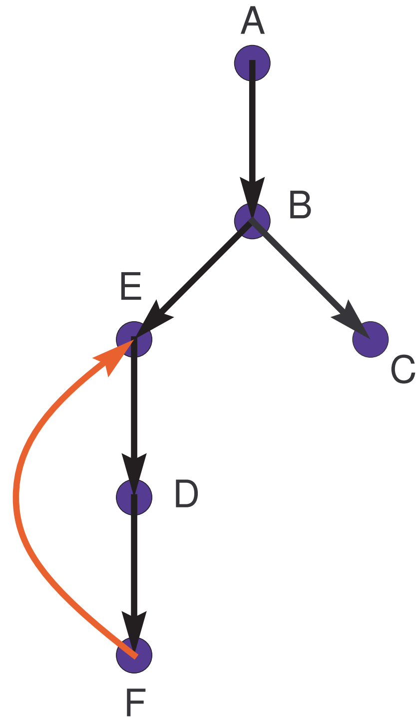
$$ABEDFE$$

Example



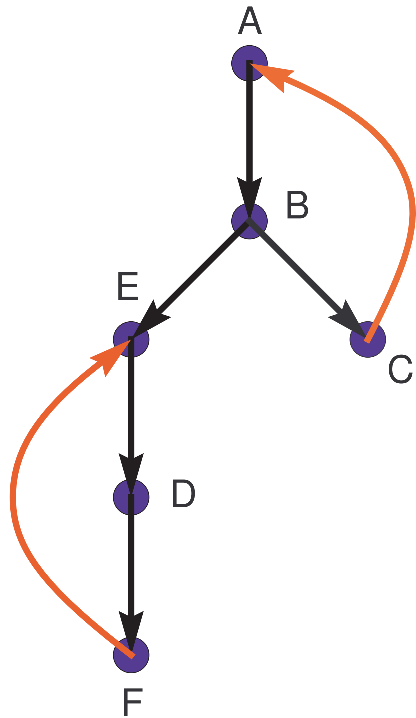
$$ABEDFEB$$

Example



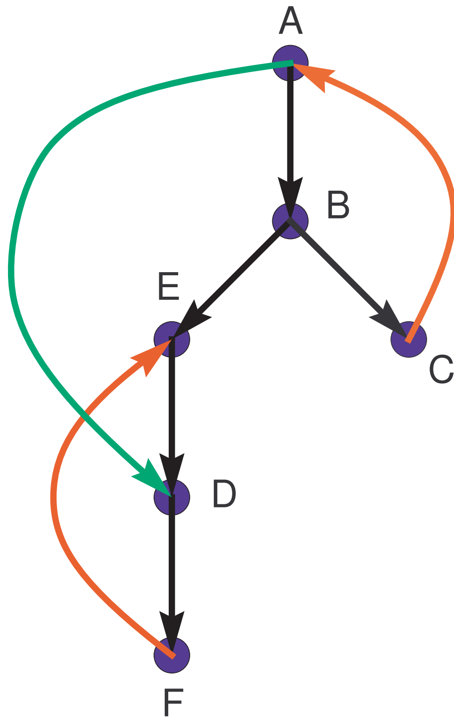
$$ABEDFEBC$$

Example



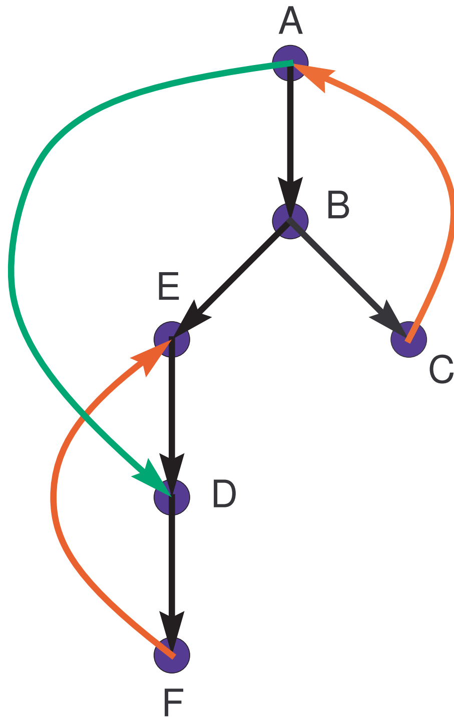
$$ABEDFEBCA$$

Example



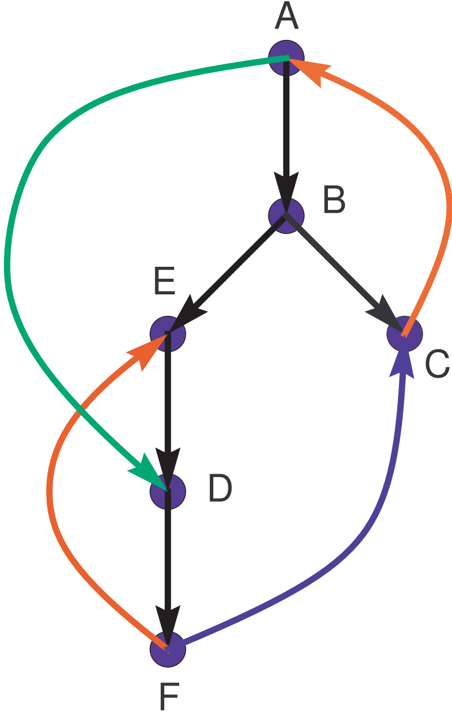
$$ABEDFEBCAD$$

Example



$$ABEDFEBCADF$$

Example



$$ABEDFEBCADFC$$

DFS - Depth First Search

Visiting order: Visit a vertex, then recursively visit all of its neighbors in order.

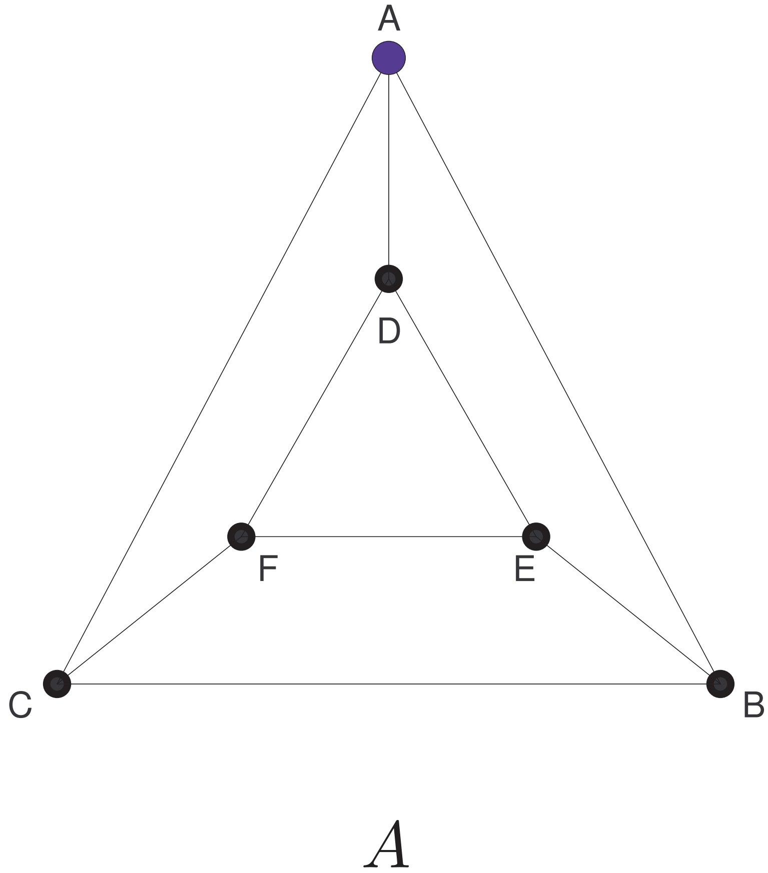
Input: An undirected graph $G = (V, E)$ and a global order on the n vertices.

Output: A traversal forest that contains a traversal tree for each connected component of the graph.

Directed graphs:

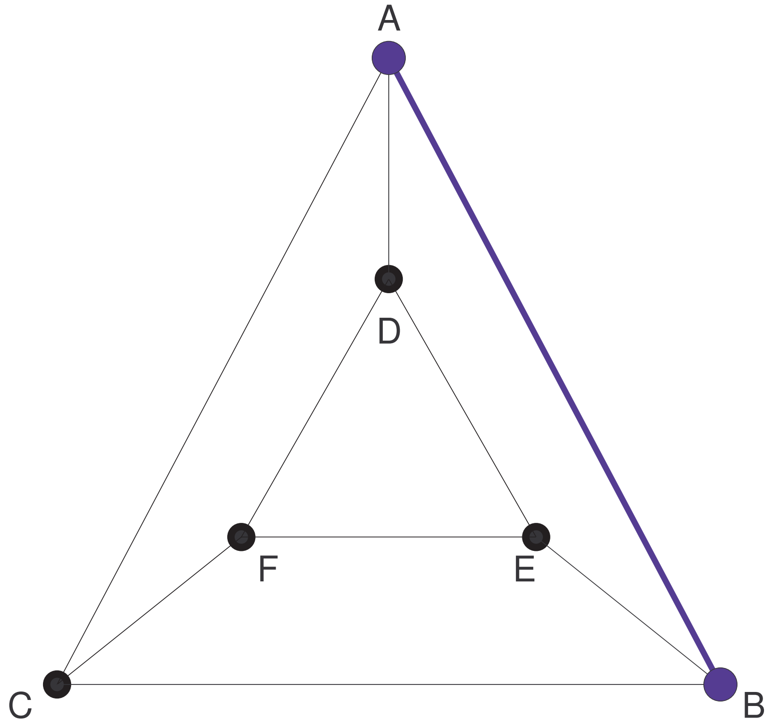
- ⋆ In the traversal forest, each tree is a directed tree.
- ⋆ In each tree, there is a directed path from the root to any other vertex.

Example – a DFS traversal Path



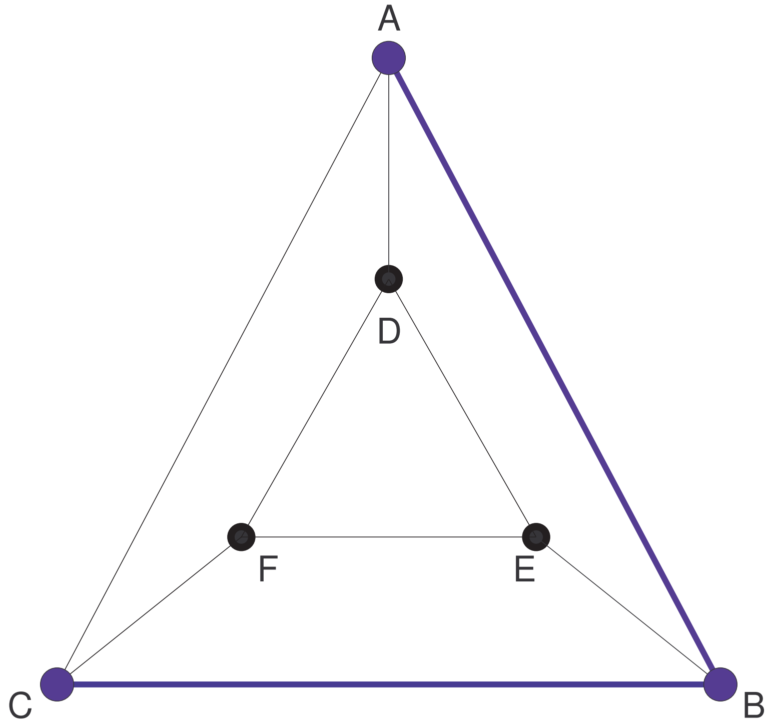
A

Example – a DFS traversal Path



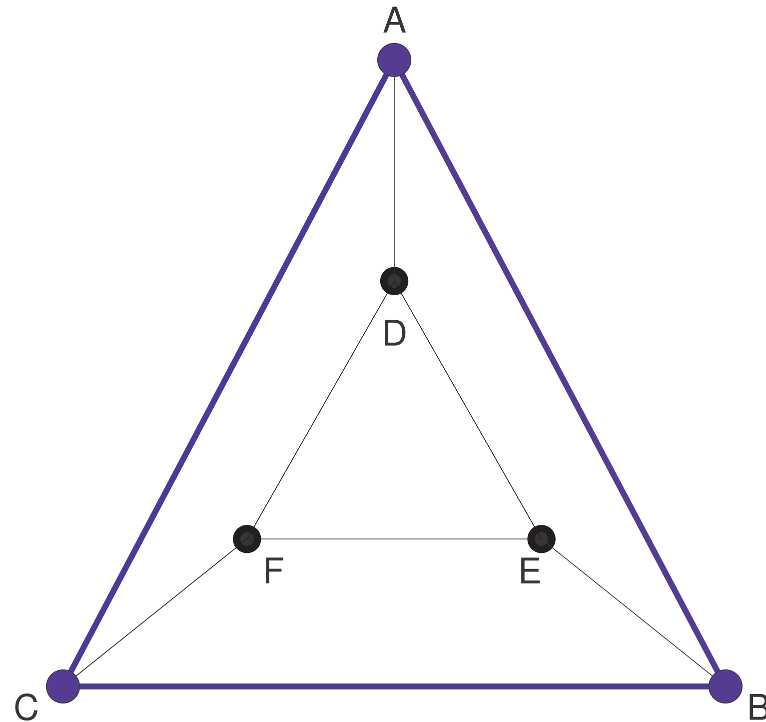
$$AB$$

Example – a DFS traversal Path



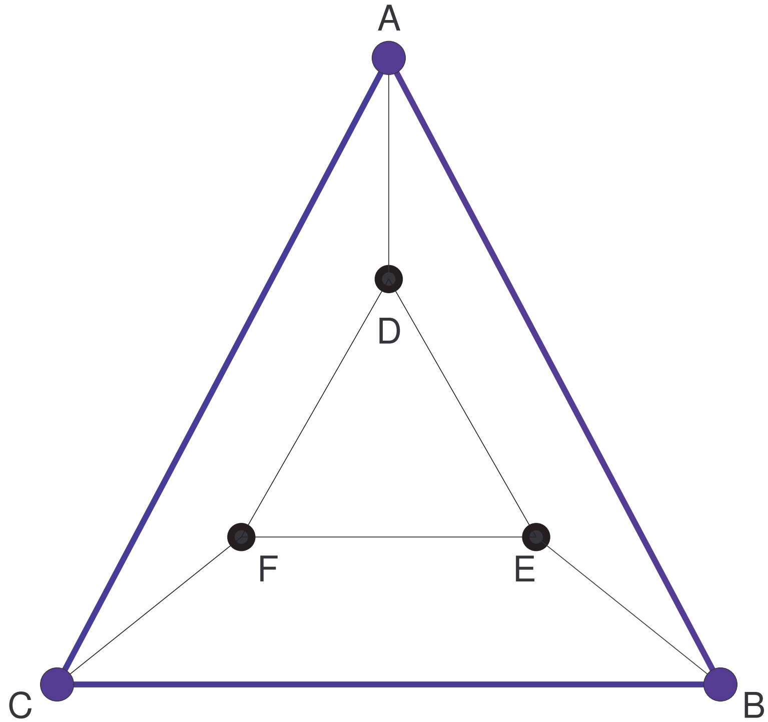
$$ABC$$

Example – a DFS traversal Path



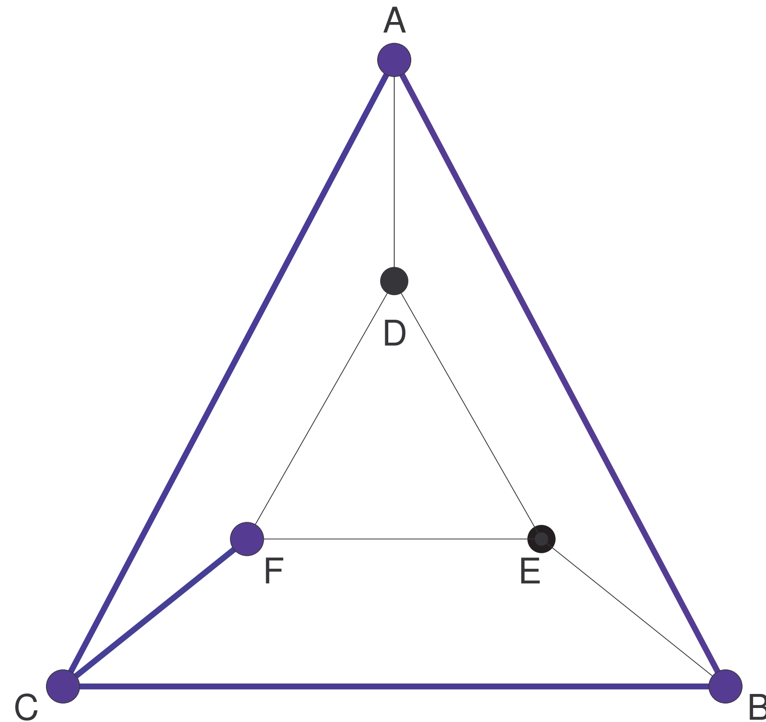
$$ABCA$$

Example – a DFS traversal Path



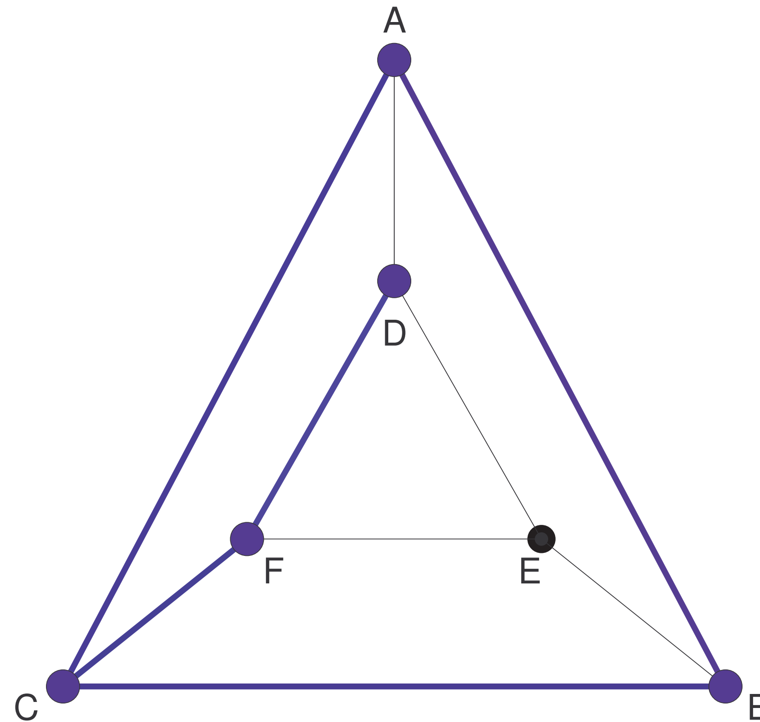
$$ABCAC$$

Example – a DFS traversal Path



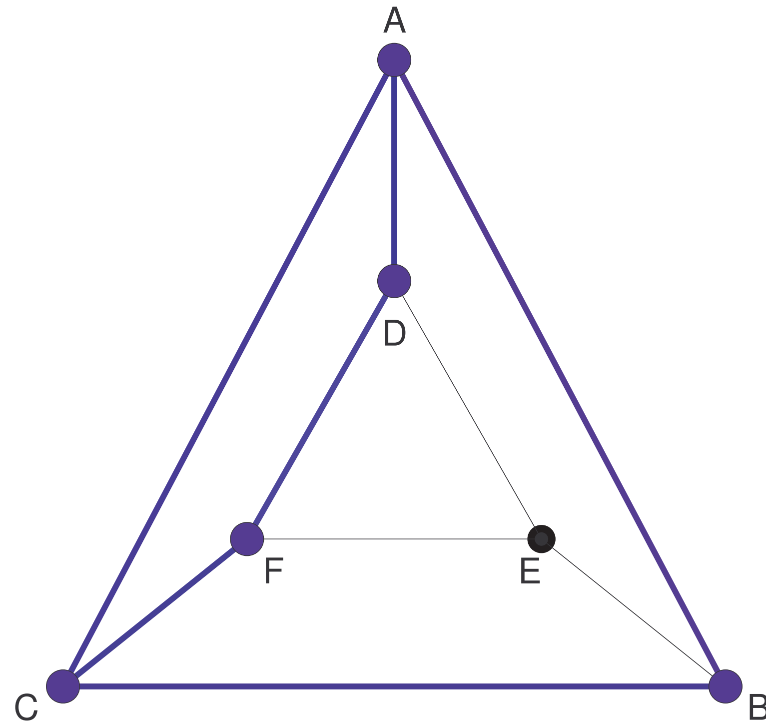
$$ABCACF$$

Example – a DFS traversal Path



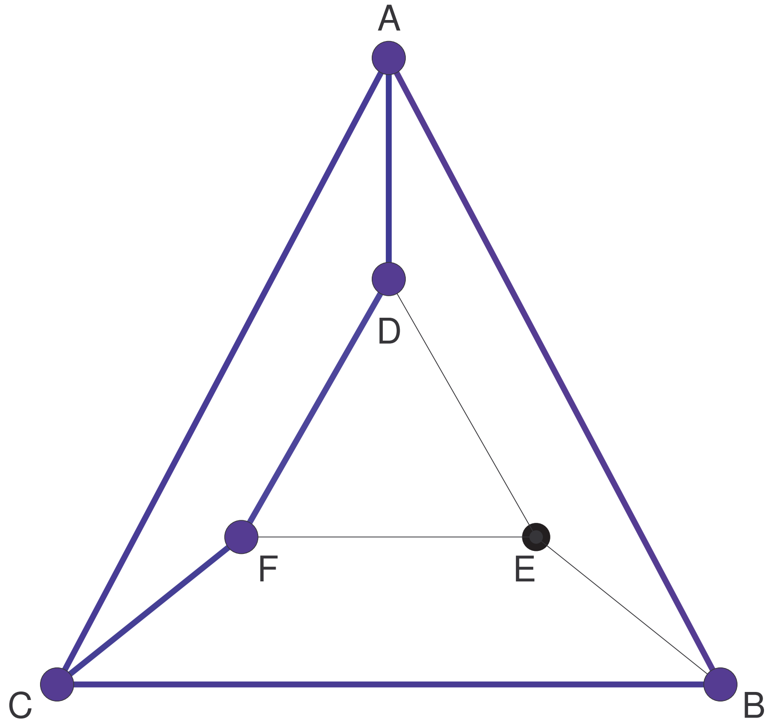
$$ABCACFD$$

Example – a DFS traversal Path



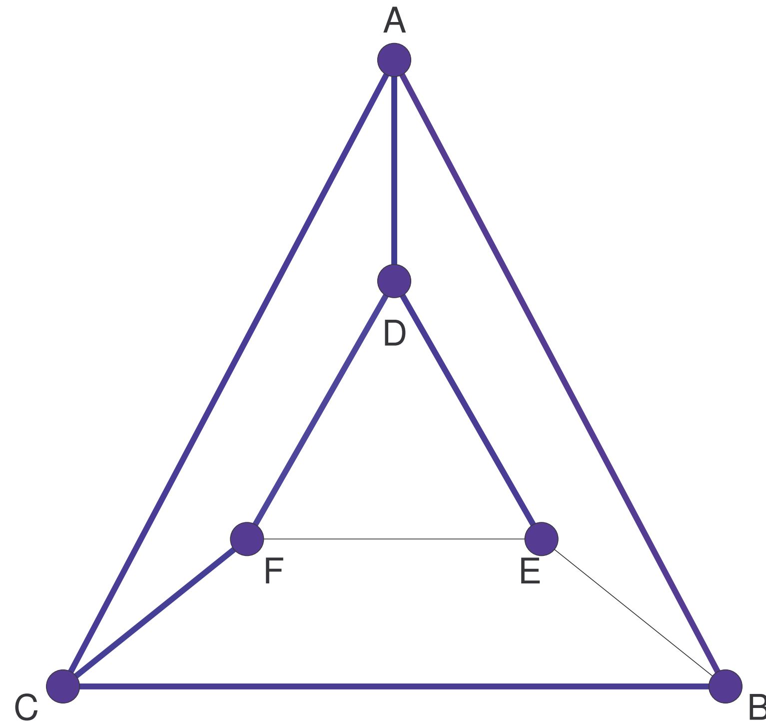
$$ABCACFDA$$

Example – a DFS traversal Path



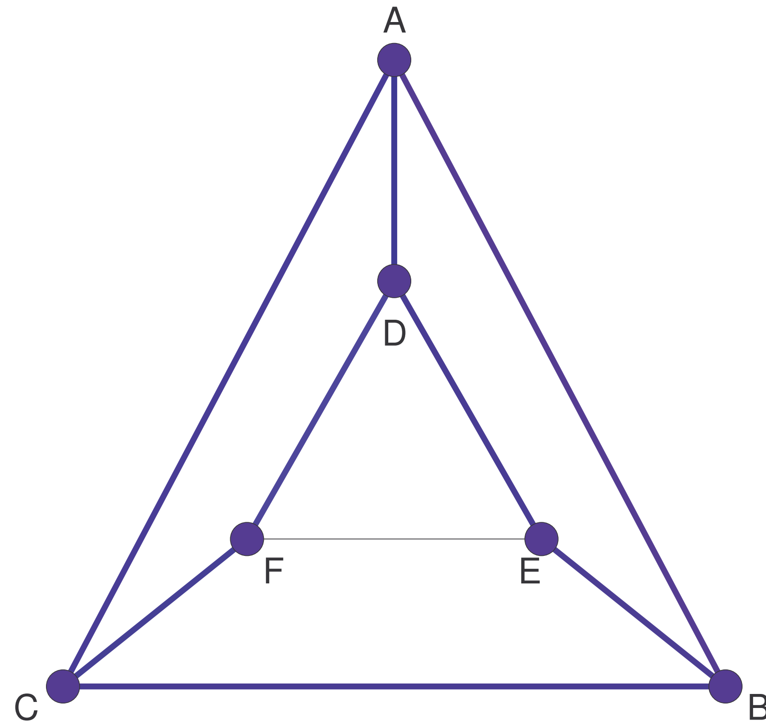
$$ABCACFDAD$$

Example – a DFS traversal Path



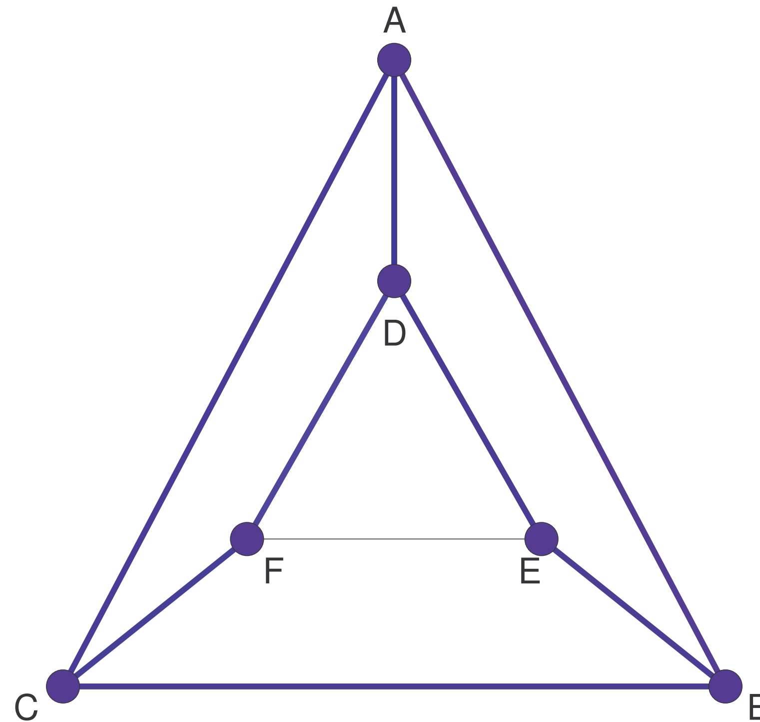
$$ABCACFDADE$$

Example – a DFS traversal Path



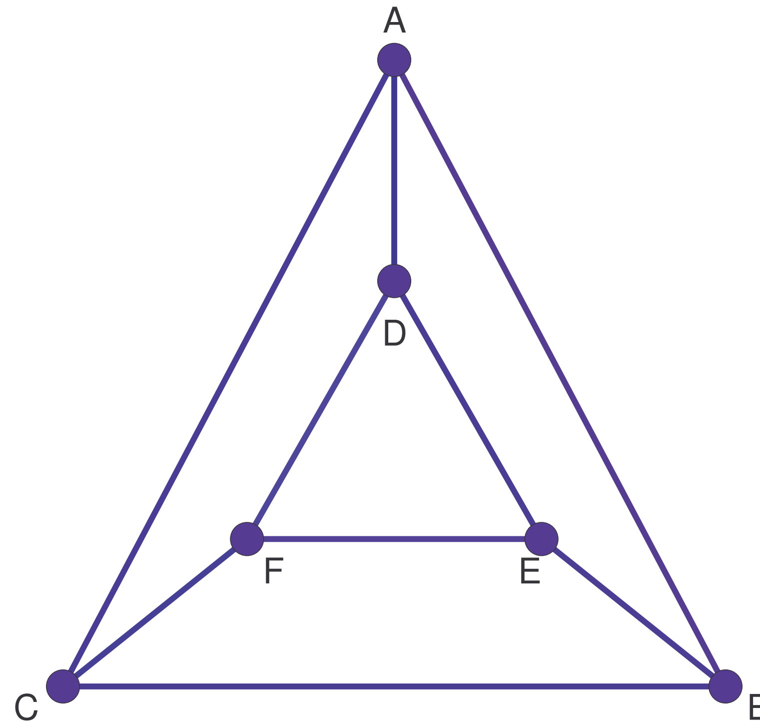
$$ABCACFDADEB$$

Example – a DFS traversal Path



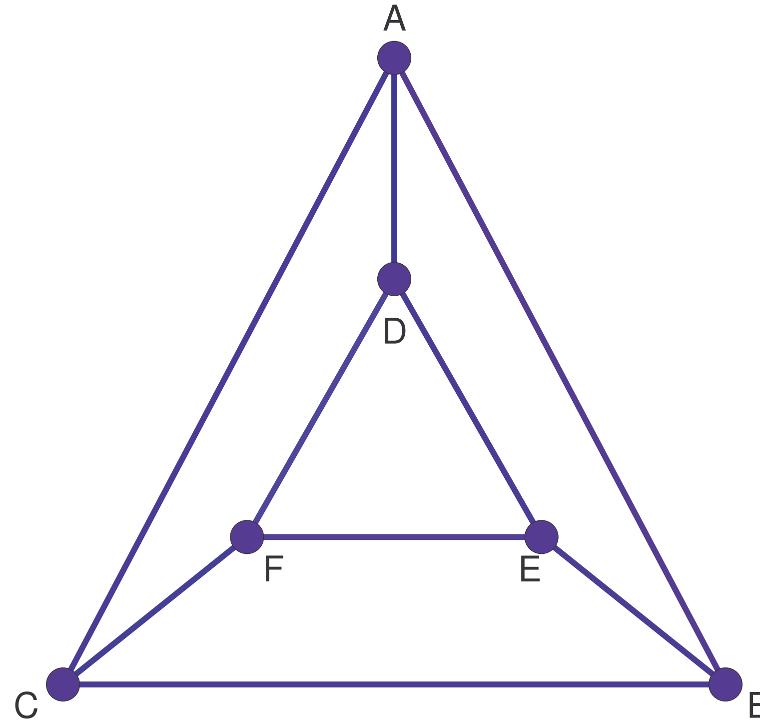
$$ABCACFDADEBE$$

Example – a DFS traversal Path



$$ABCACFDADEBEF$$

Example – a DFS traversal Path

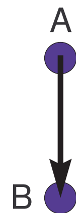


$$ABCACFDADEBEF - EDFCBA$$

Example – the DFS traversal Tree

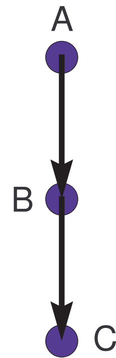
$$A$$

Example – the DFS traversal Tree



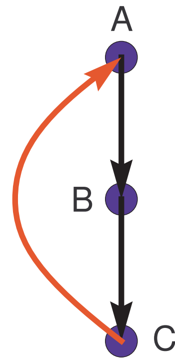
$$AB$$

Example – the DFS traversal Tree



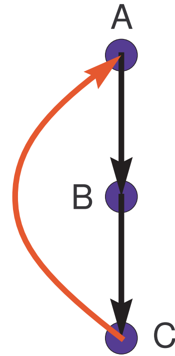
$$ABC$$

Example – the DFS traversal Tree



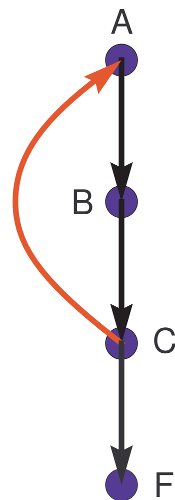
$$ABCA$$

Example – the DFS traversal Tree



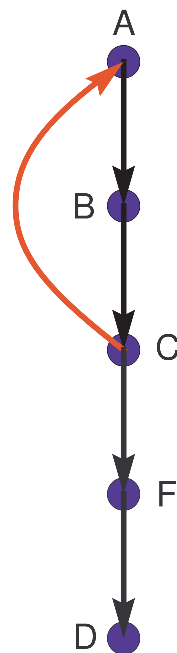
$$ABCAC$$

Example – the DFS traversal Tree



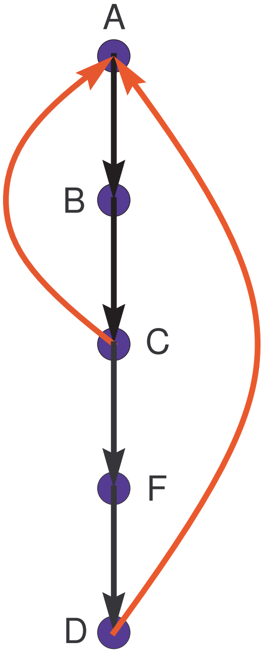
$$ABCACF$$

Example – the DFS traversal Tree



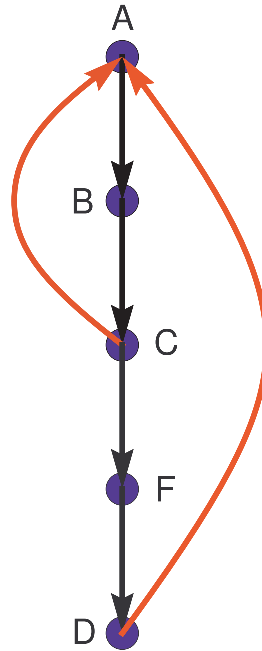
$$ABCACFD$$

Example – the DFS traversal Tree



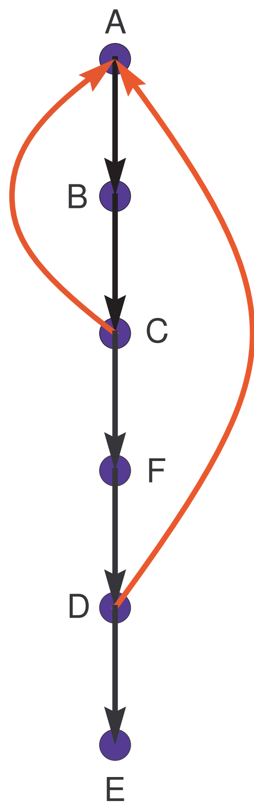
$$ABCACFDA$$

Example – the DFS traversal Tree



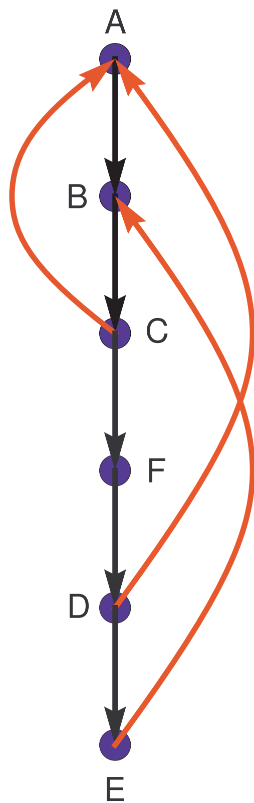
$$ABCACFDAD$$

Example – the DFS traversal Tree



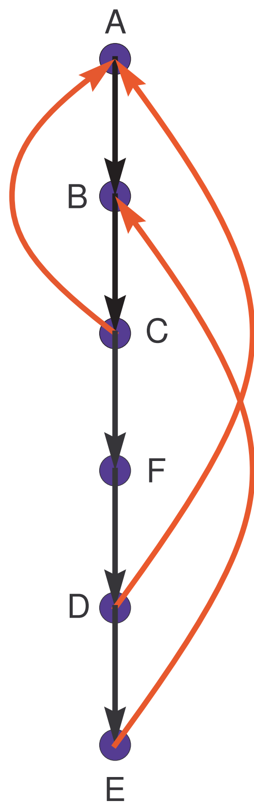
$$ABCACFDADE$$

Example – the DFS traversal Tree



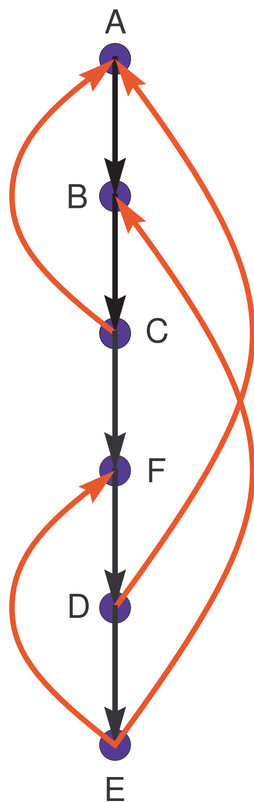
$$ABCACFDADEB$$

Example – the DFS traversal Tree



$$ABCACFDADEBE$$

Example – the DFS traversal Tree



$$ABCACFDADEBEF$$

Variables for the DFS Procedure

- ⋆ Each vertex is colored by one of the following colors:
  - $White$: The recursive visit has not started.
  - $Gray$: The recursive visit started but not finished.
  - $Black$: The recursive visit finished.
- ⋆ A global discrete time variable $time$ that is updated when a vertex becomes $Gray$ and when a vertex becomes $Black$.
  - $d(v)$: The time vertex v becomes $Gray$.
  - $f(v)$: The time vertex v becomes $Black$.
- ⋆ A parenthood function $\Pi(v)$ in the traversal forest:
  - $\Pi(v) = nil$ if v is a root of a tree in the forest.
  - $\Pi(v)$ is v's parent in the traversal tree that contains v.

The DFS Procedure – Initial Call

DFS(G)

for each vertex $v \in V$ do

$Color(v) = White$

$\Pi(v) = nil$

$time = 0$

for each vertex $v \in V$ do

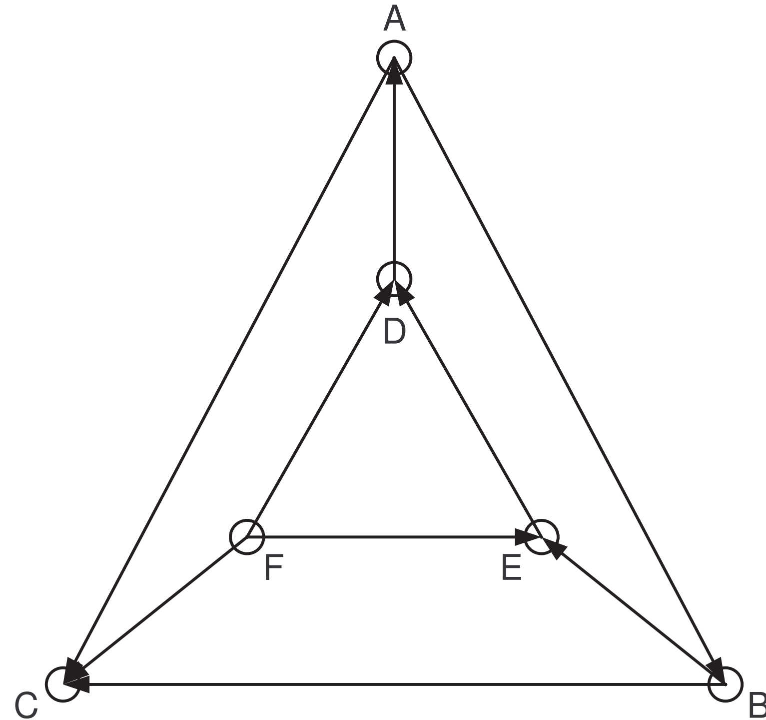
if $Color(v) = White$ then

DFS-Visit(v)

The DFS Procedure – Recursive Call

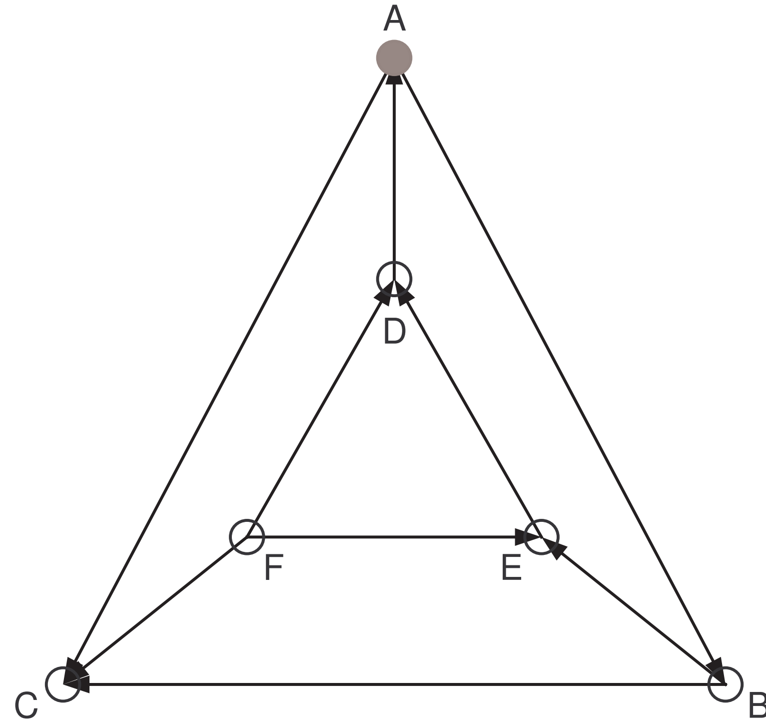
```
DFS-Visit(v)
   Color(v) = Gray
   time++
   d(v)=time
   for each neighbor u of v do
       if Color(u) =white then
           Π(u)= v
           DFS-Visit(u)
   Color(v) =Black
   time++
   f(v)=time
```

Example – Directed DFS



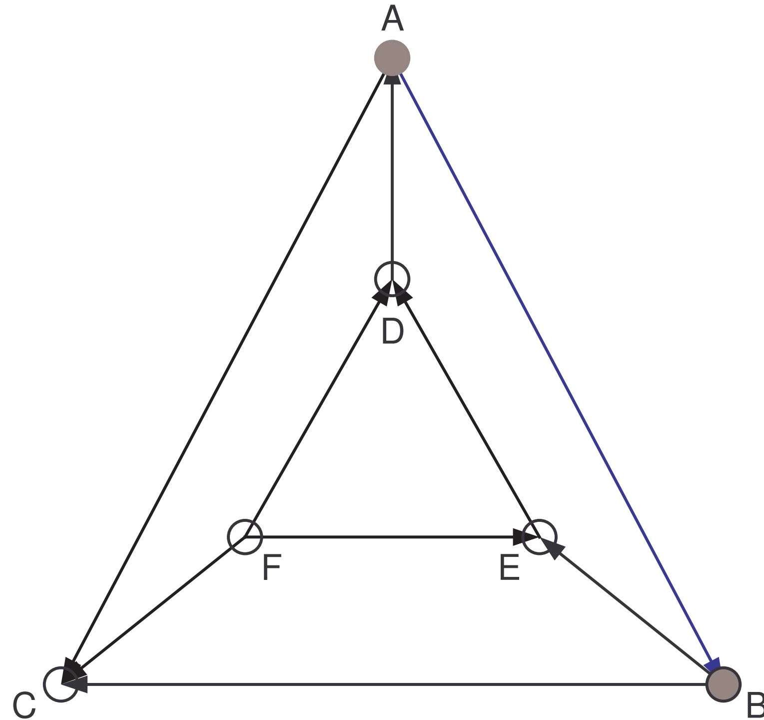
$time = 0$

Example – Directed DFS



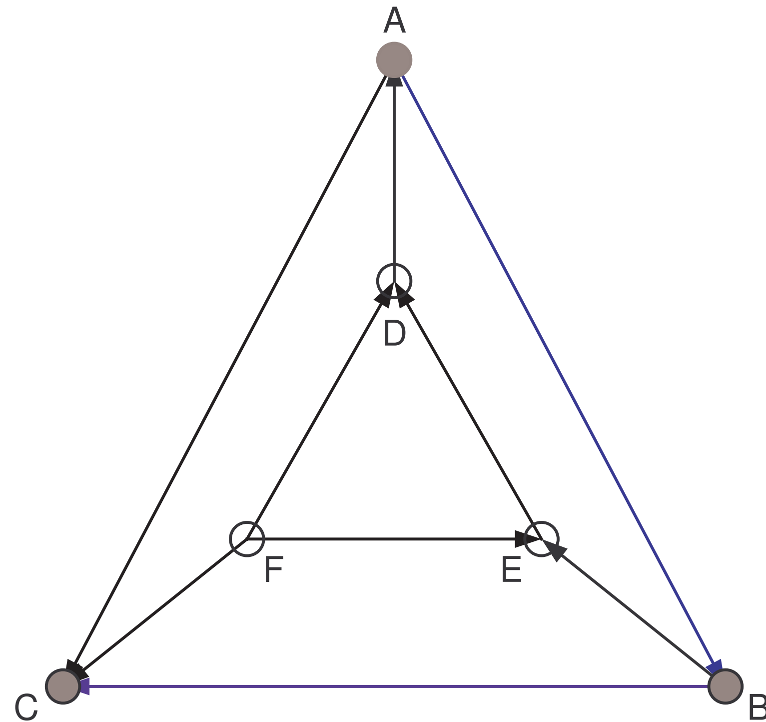
$time = 1$

Example – Directed DFS



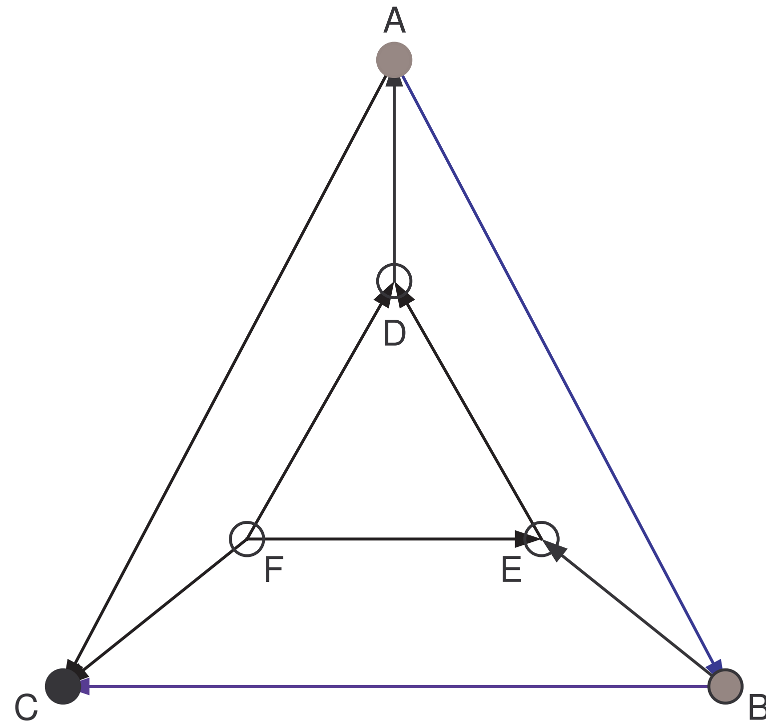
$$time = 2$$

Example – Directed DFS



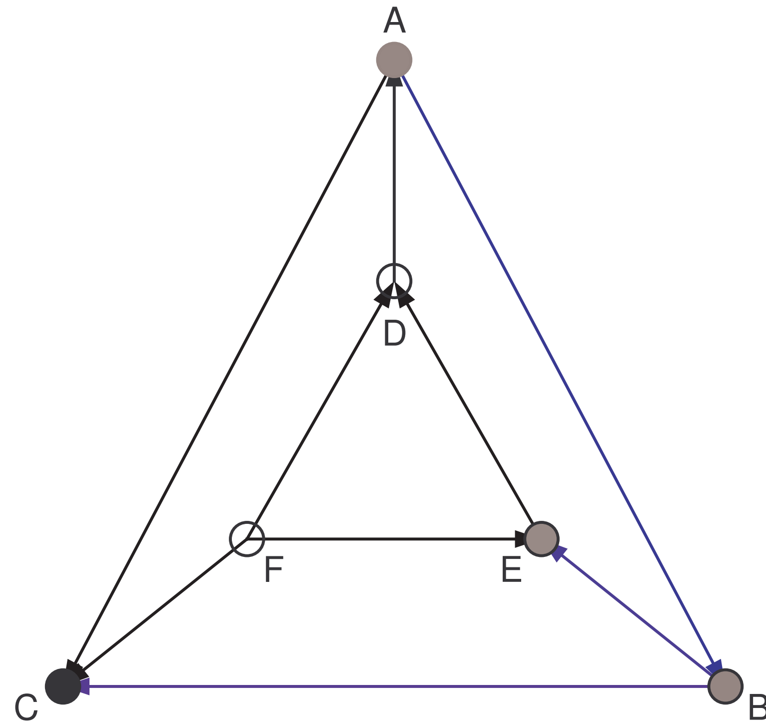
$$time = 3$$

Example – Directed DFS



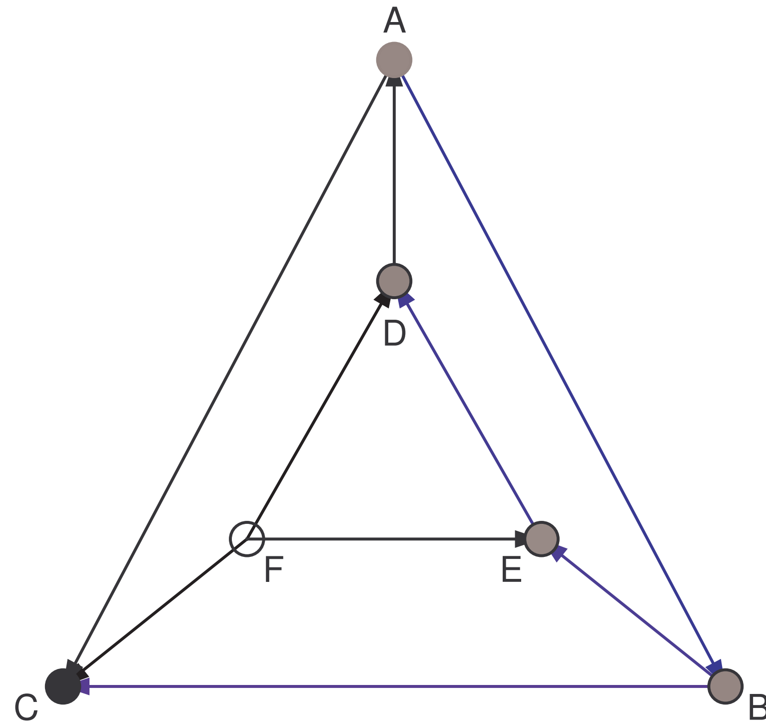
$time = 4$

Example – Directed DFS



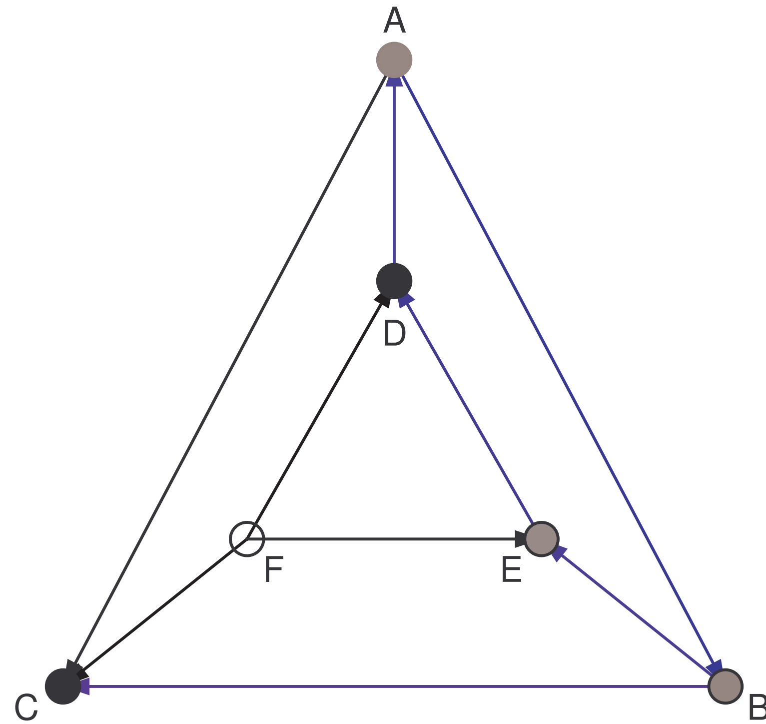
$time = 5$

Example – Directed DFS



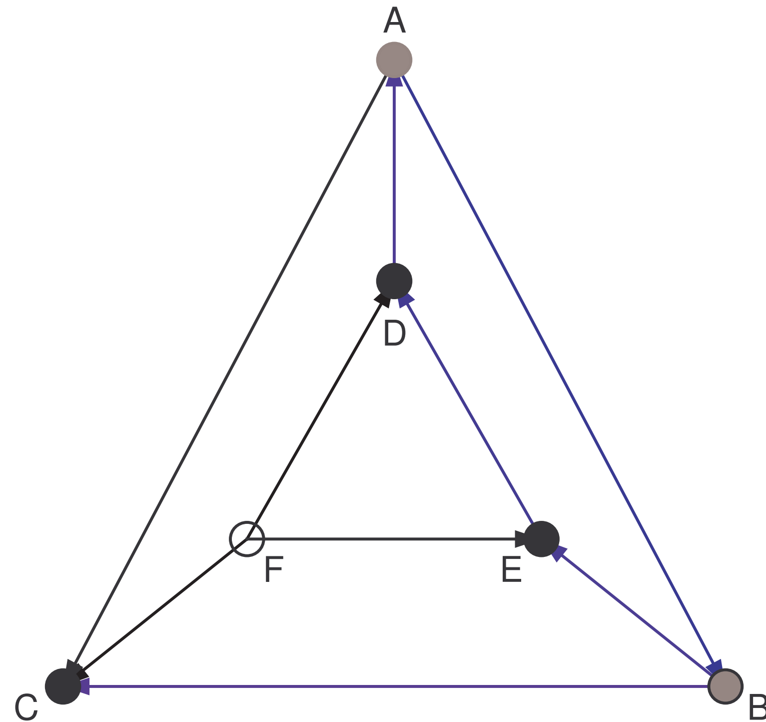
$$time = 6$$

Example – Directed DFS



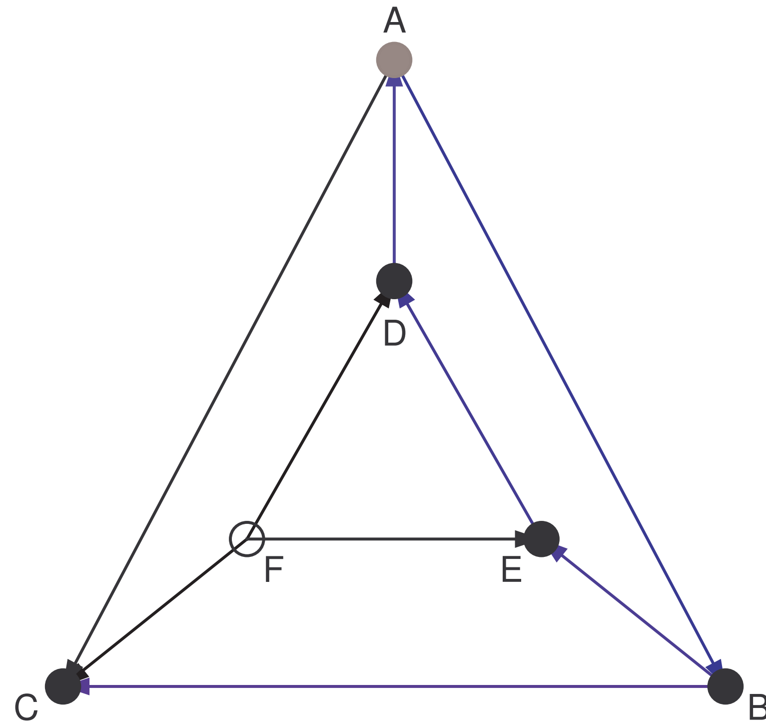
$time = 7$

Example – Directed DFS



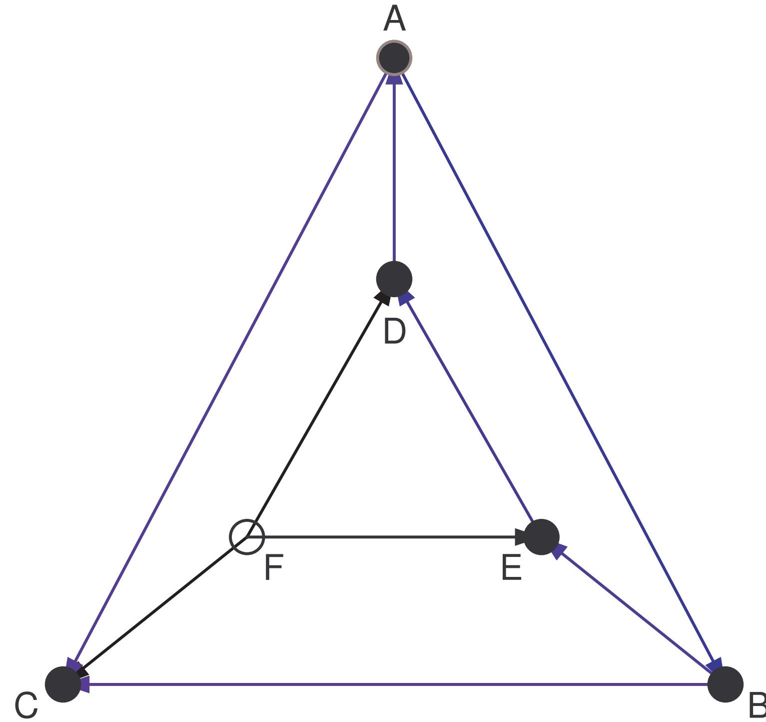
$$time = 8$$

Example – Directed DFS



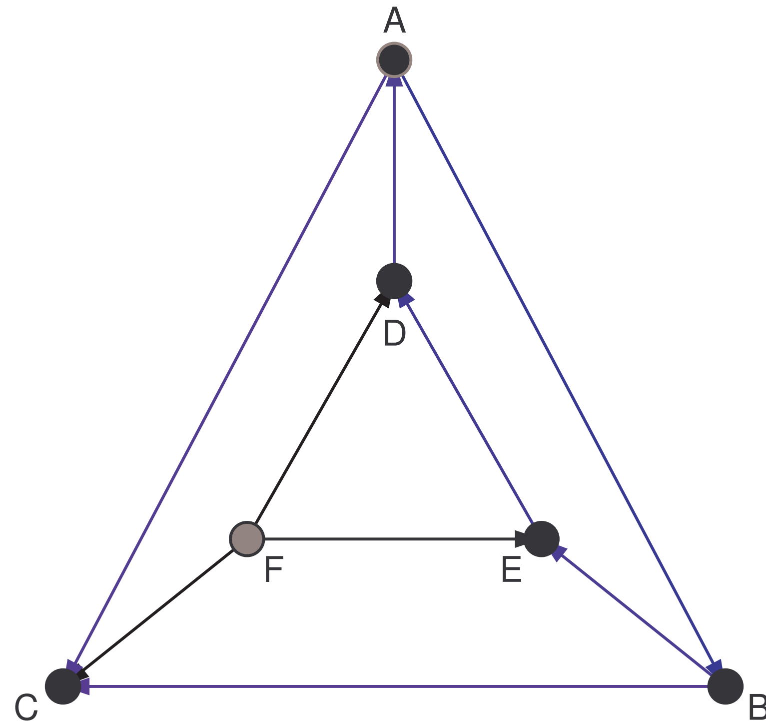
$$time = 9$$

Example – Directed DFS



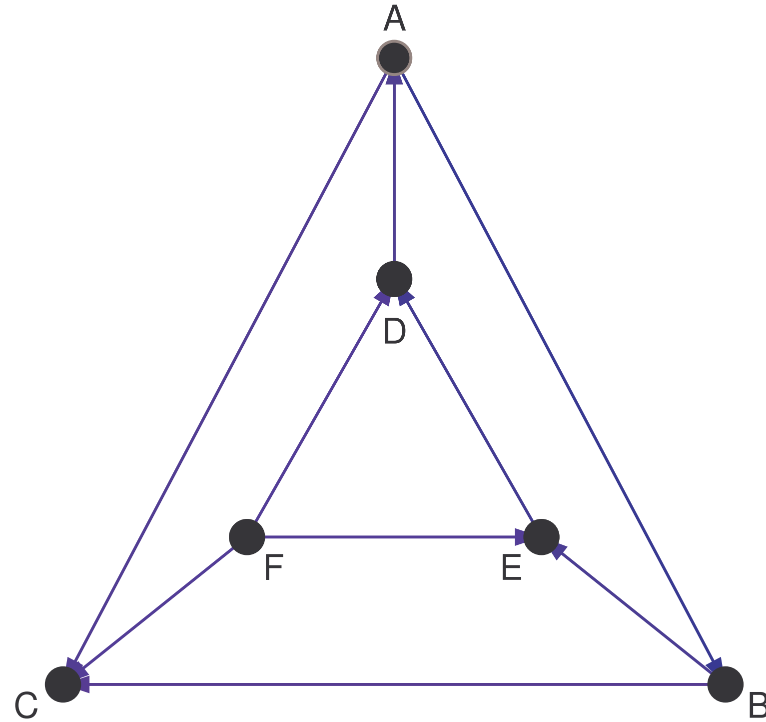
$$time = 10$$

Example – Directed DFS



$$time = 11$$

Example – Directed DFS



$$time = 12$$

DFS – Correctness

Lemma: DFS visits all the vertices.

Proof: Each vertex changes colors as follows:

$$White \rightarrow Gray \rightarrow Black$$

Lemma: DFS traverses all the edges.

Proof: For each vertex, DFS examines all of its incident edges in undirected graphs and all of its outgoing edges in directed graphs.

DFS – Traversal Path

Undirected graphs: The traversal path is not explicit. Traversing back the edges is done implicitly due to the recursive calls.

Directed graphs: The traversal path sometimes uses the wrong direction due to the recursion.

DFS – Time Complexity

Adjacency lists: $\Theta(n+m)$.

- $\star$ $\Theta(m)$: Each edge is examined once in directed graphs and twice in undirected graphs.
- $\star$ $\Theta(n)$: Each vertex is colored three times.

Adjacency matrix: $\Theta(n^2)$.

- $\star$ $\Theta(n)$: For each vertex, for examining all of its incident edges in undirected graphs and all of its outgoing edges in directed graphs.

DFS – Time Intervals

Definition: The DFS interval of vertex v is $[d(v), f(v)]$.

Lemma: Assume $d(u) < d(v)$ for an edge (u, v):

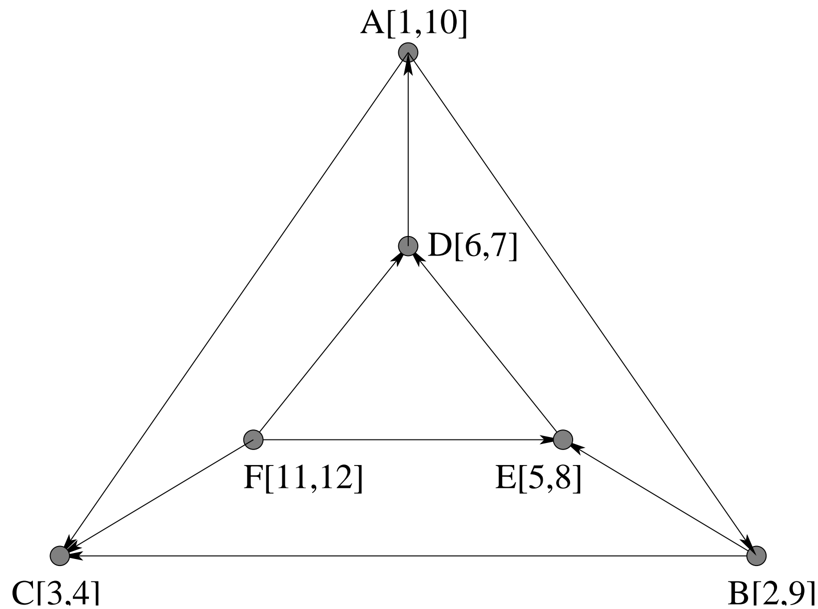
- ⋆ Either the intervals of u and v are disjoint

$$d(u) < f(u) < d(v) < f(v).$$

- ⋆ Or u's interval contains v's interval

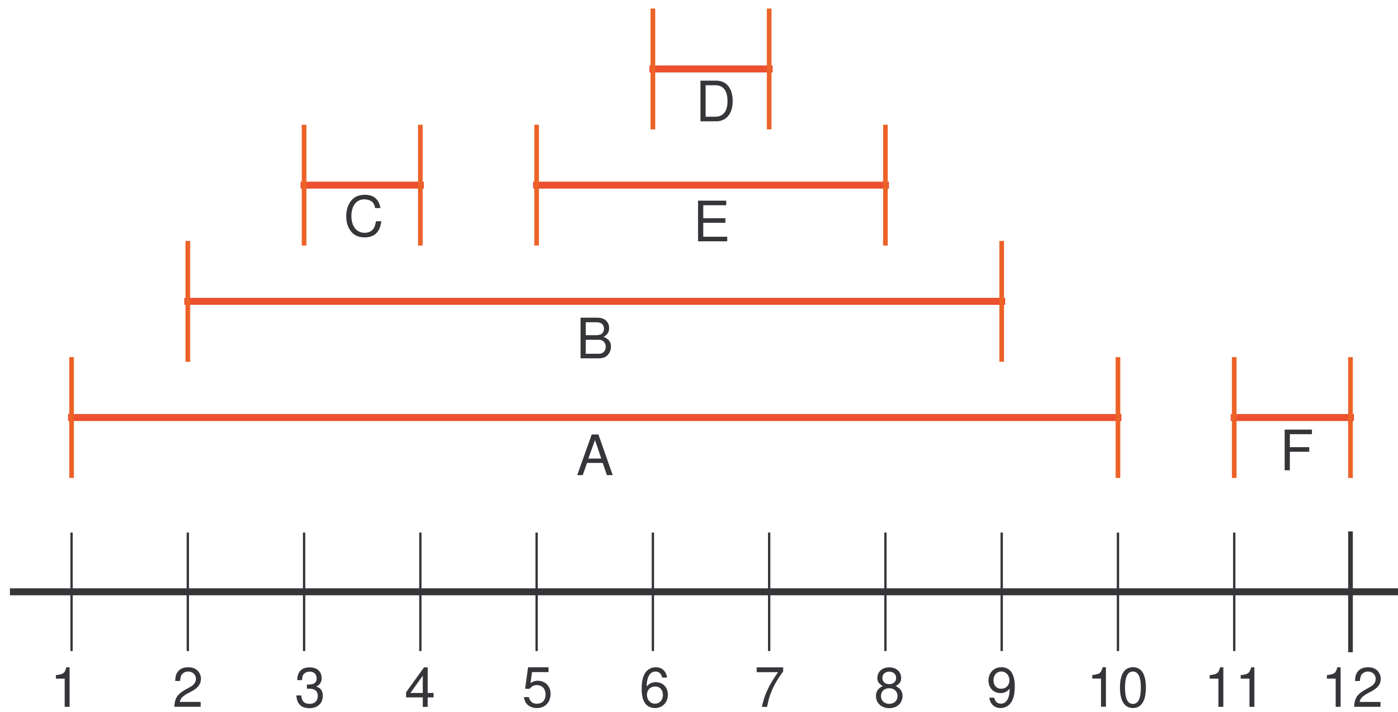
$$d(u) < d(v) < f(v) < f(u).$$

Example – Time Intervals



$$
\begin{array}{lll}
time = 1 & A = Gray & d(A) = 1 \\
time = 2 & B = Gray & d(B) = 2 \\
time = 3 & C = Gray & d(C) = 3 \\
time = 4 & C = Black & f(C) = 4 \\
time = 5 & E = Gray & d(E) = 5 \\
time = 6 & D = Gray & d(D) = 6 \\
time = 7 & D = Black & f(D) = 7 \\
time = 8 & E = Black & f(E) = 8 \\
time = 9 & B = Black & f(B) = 9 \\
time = 10 & A = Black & f(A) = 10 \\
time = 11 & F = Gray & d(F) = 11 \\
time = 12 & F = Black & f(F) = 12
\end{array}
$$

Example – Time Intervals



DFS – Traversal Tree Edge Classification

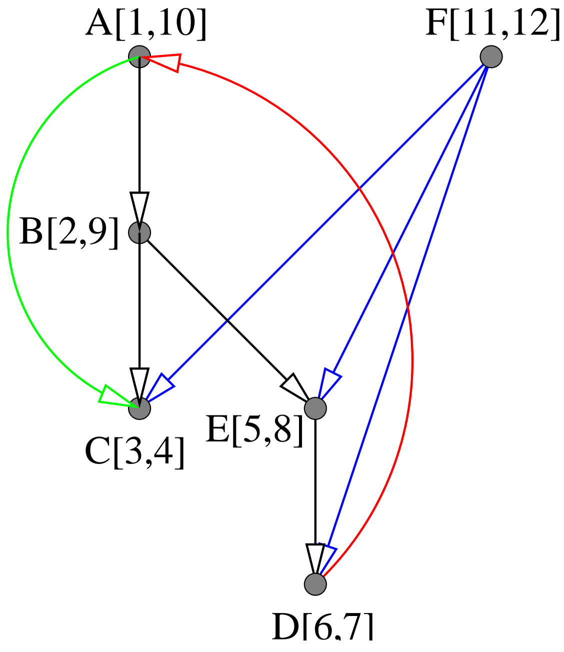
A tree edge (T): An edge from a $Gray$ vertex to a $White$ vertex.

A back edge (B): An edge from a $Gray$ vertex to a $Gray$ vertex.

A forward edge (F): An edge from a $Gray$ vertex to a $Black$ vertex whose interval is nested.

A cross edge (C): An edge from a $Gray$ vertex to a $Black$ vertex whose interval finished.

Example – Edge Classification



(A,B)	$Gray \rightarrow White$	**T**
(B,C)	$Gray \rightarrow White$	**T**
(B,E)	$Gray \rightarrow White$	**T**
(E,D)	$Gray \rightarrow White$	**T**
(D,A)	$Gray \rightarrow Gray$	**B**
(A,C)	$Gray \rightarrow Black(\text{nested})$	**F**
(F,C)	$Gray \rightarrow Black(\text{finished})$	**C**
(F,D)	$Gray \rightarrow Black(\text{finished})$	**C**
(F,E)	$Gray \rightarrow Black(\text{finished})$	**C**

DFS – Undirected Graphs

Lemma: DFS has no **forward** and no **cross** edges.

Proof:

- ⋆ There is no edge from a $Gray$ vertex to a $Black$ vertex.
- ⋆ Otherwise, the $Black$ vertex, before becoming black, would examine its incident edge to the $Gray$ vertex in the other direction.

Cycles in Undirected Graphs

Theorem: An undirected graph has a cycle iff the DFS creates at least one **back** edge.

Proof $\Leftarrow$

- $\star$ Suppose that (u, v) is a **back** edge examined at u.
- $\star$ Then v is an ancestor of u.
- $\star$ Therefore, there exists a path from v to u.
- $\star$ Adding (u, v) to the path creates a cycle.

Cycles in Undirected Graphs

Theorem: An undirected graph has a cycle iff the DFS creates at least one **back** edge.

Proof $\Rightarrow$

- $\star$ Suppose that $\mathcal{C}$ is a cycle in G.
- $\star$ Let v be the first $Gray$ vertex in $\mathcal{C}$.
- $\star$ Let (u, v) be the preceding edge in $\mathcal{C}$.
- $\star$ After time $d(v)$, DFS explores the v to u path.
- $\star$ When (u, v) is traversed, both v and u are $Gray$.
- $\star$ Hence, (u, v) is a **back** edge.

DFS – Application

Problem:

- ⋆ Is an undirected graph a forest or a tree?
- ⋆ Does an undirected graph have a cycle?

Algorithm:

- ⋆ Run the DFS algorithm.
- ⋆ Terminate if a **back** edge is found.

Time complexity: $O(n)$.

- ⋆ If there are no cycles there are $O(n)$ edges. Therefore, the DFS time complexity $O(m+n)$ becomes $O(n)$.
- ⋆ If there exists a cycle, the algorithm terminates the first time a $Gray$ vertex is visited for a second time.

Directed Acyclic Graphs

DAG: A directed graph without cycles.

Theorem: A directed graph G is DAG iff running DFS on G does not produce a **back** edge.

Proof $\Rightarrow$

- $\star$ Suppose that $(u \to v)$ is a **back** edge.
- $\star$ Then v is an ancestor of u.
- $\star$ Therefore, there exists a directed path from v to u.
- $\star$ Adding $(u \to v)$ to the path creates a directed cycle.

Directed Acyclic Graphs

Theorem: A directed graph G is DAG iff running DFS on G does not produce a **back** edge.

Proof $\Leftarrow$

- $\star$ Suppose that $\mathcal{C}$ is a directed cycle in G.
- $\star$ Let v be the first $Gray$ vertex in $\mathcal{C}$.
- $\star$ Let $(u \rightarrow v)$ be the preceding edge in $\mathcal{C}$.
- $\star$ After time $d(v)$, DFS explores the v to u directed path.
- $\star$ When $(u \rightarrow v)$ is traversed, both v and u are $Gray$.
- $\star$ Hence, $(u \rightarrow v)$ is a **back** edge.

Topological Sort of a DAG

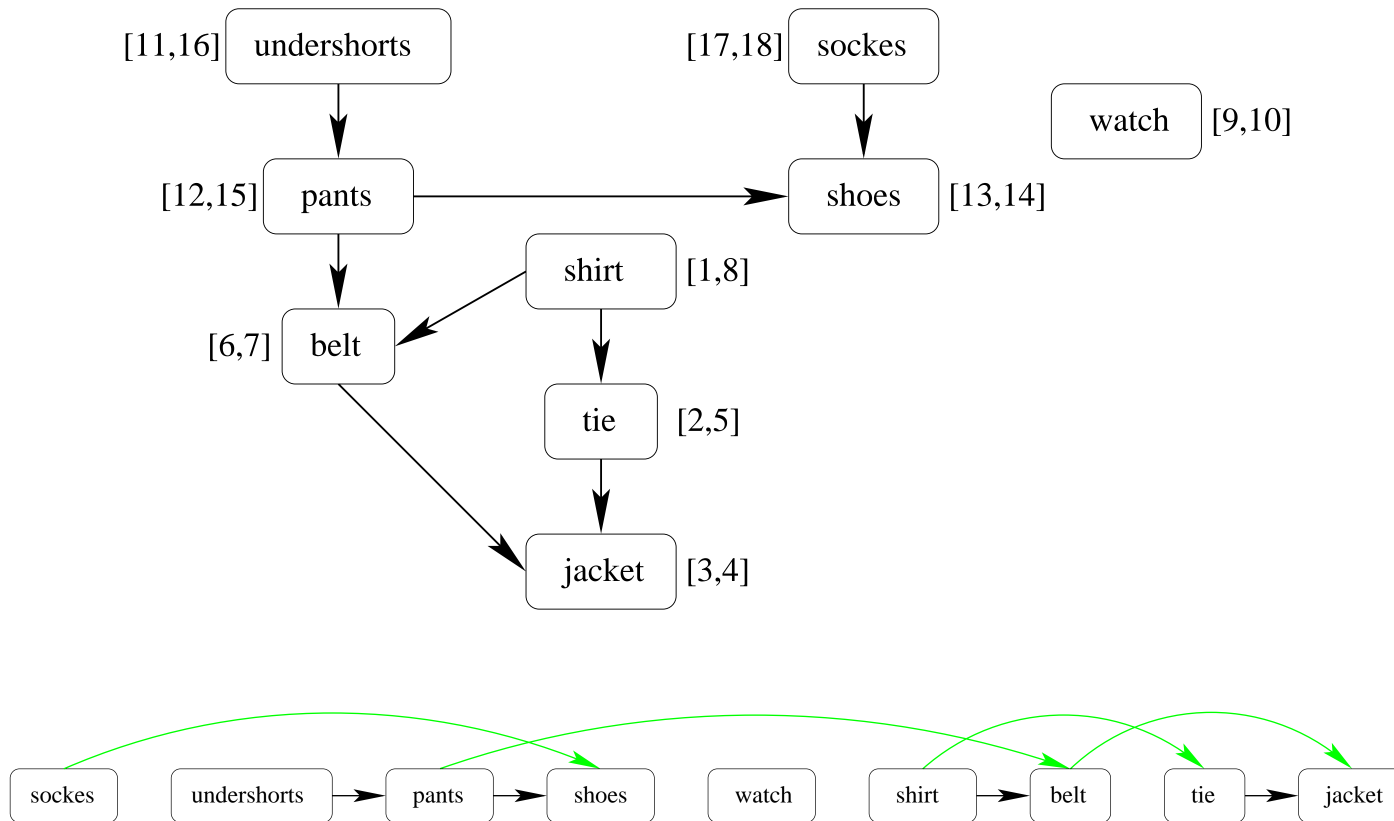
Definition: A topological sort of a DAG is a linear ordering of its vertices such that vertex u appears before vertex v for any directed edge $(u \rightarrow v)$.

Problem: Find one of the topological sorts of a DAG.

Algorithm:

- ⋆ Create an empty linked list.
- ⋆ Run DFS on the DAG.
- ⋆ A vertex becomes $Black$: add it to the front of the list.
- ⋆ Return the linked list as a topological sort.

Example



The Algorithm Finds a Topological Sort for a DAG

- ⋆ Consider examining the edge $(u \longrightarrow v)$.
- ⋆ At this time u is $Gray$.
- ⋆ v is not $Gray$ since then $(u \longrightarrow v)$ is a **back** edge.
- ⋆ If v is $White$ then $f(v) < f(u)$ since v becomes $Black$ before u.
- ⋆ If v is $Black$ then $f(v) < f(u)$ since u is still $Gray$.
- ⋆ $f(v) < f(u)$ implies that u appears before v in the list.

Complexity of the Topological Sort Algorithm

Running time: $\Theta(n + m)$.

Proof: The same complexity as DFS.

Breadth First Search - BFS

Visiting order: Visit a vertex, then visit all of its neighbors, then visit all of the neighbors of its neighbors, . . .

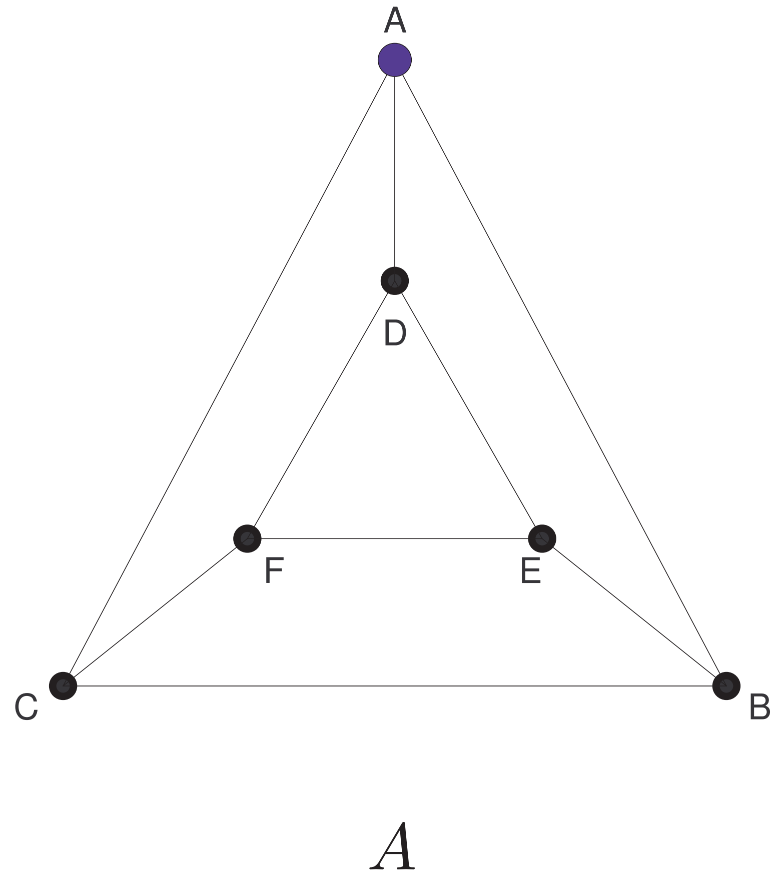
Input: An undirected graph $G = (V, E)$ and a global order on the n vertices.

Output: A traversal forest that contains a traversal tree for each connected component of the graph.

Directed graphs:

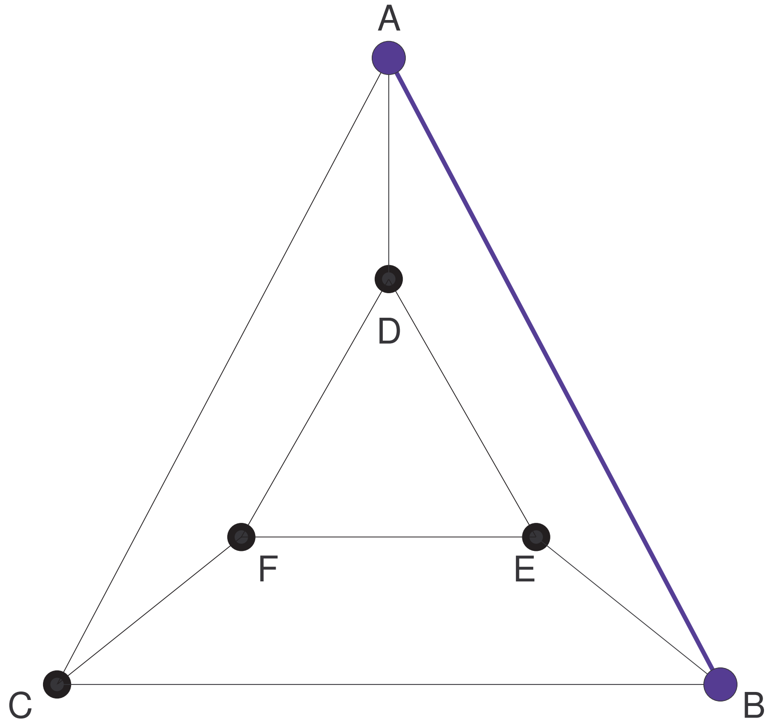
- ⋆ In the traversal forest, each tree is a directed tree.
- ⋆ In each tree, there is a directed path from the root to any other vertex.

Example – a BFS traversal Path



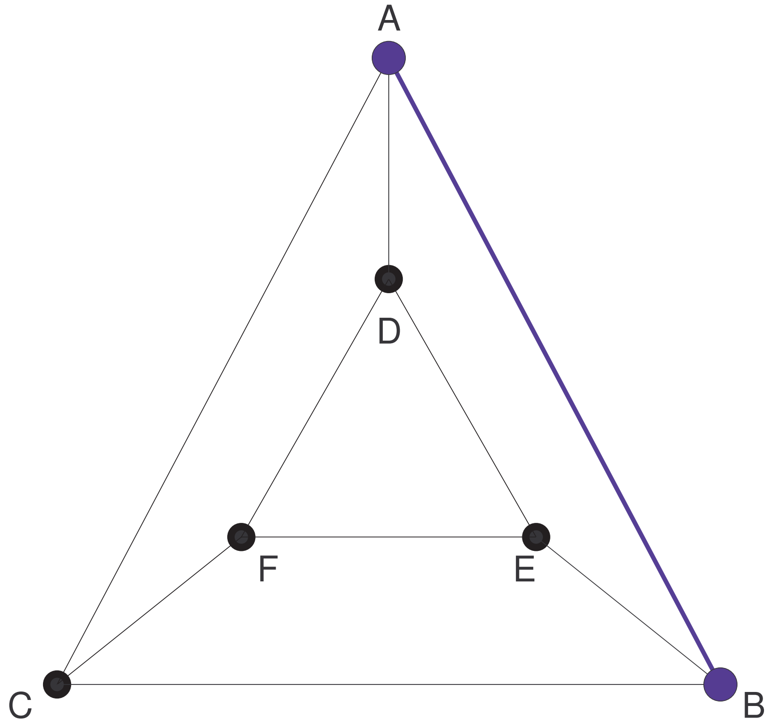
A

Example – a BFS traversal Path



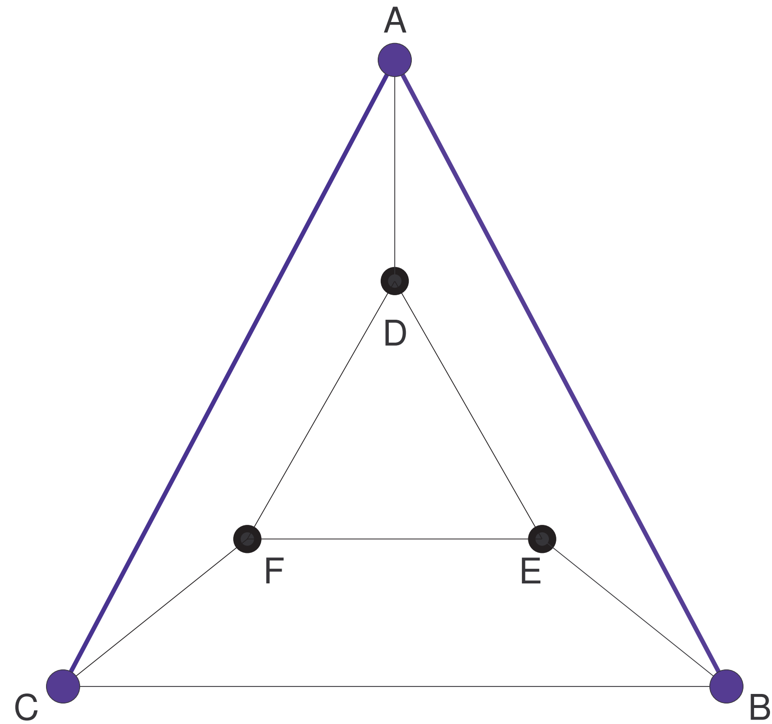
$$AB$$

Example – a BFS traversal Path



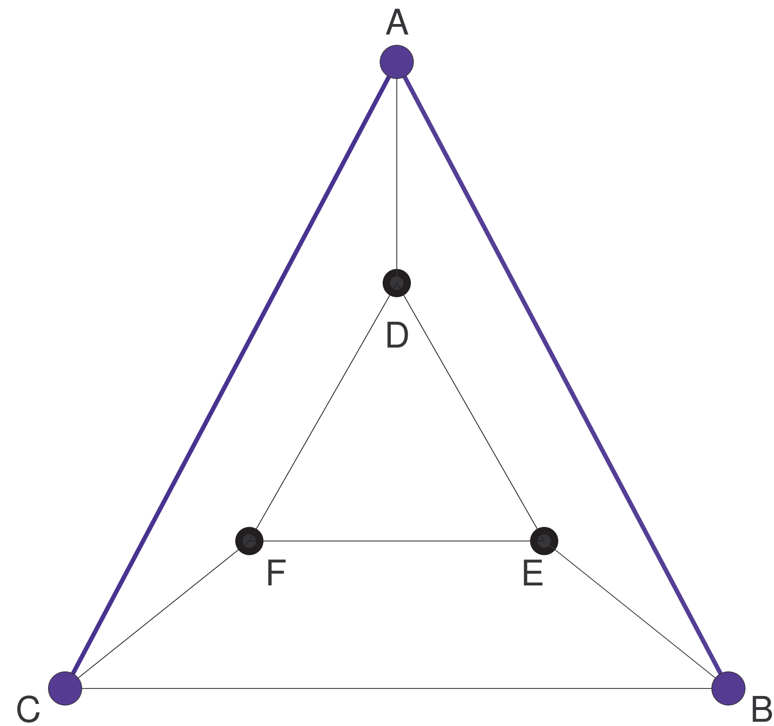
$$ABA$$

Example – a BFS traversal Path



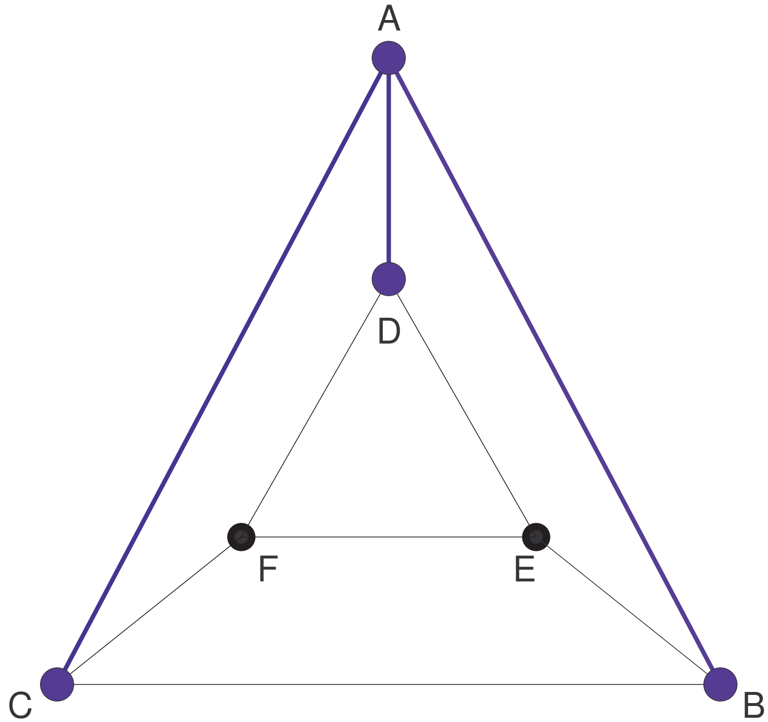
$$ABAC$$

Example – a BFS traversal Path



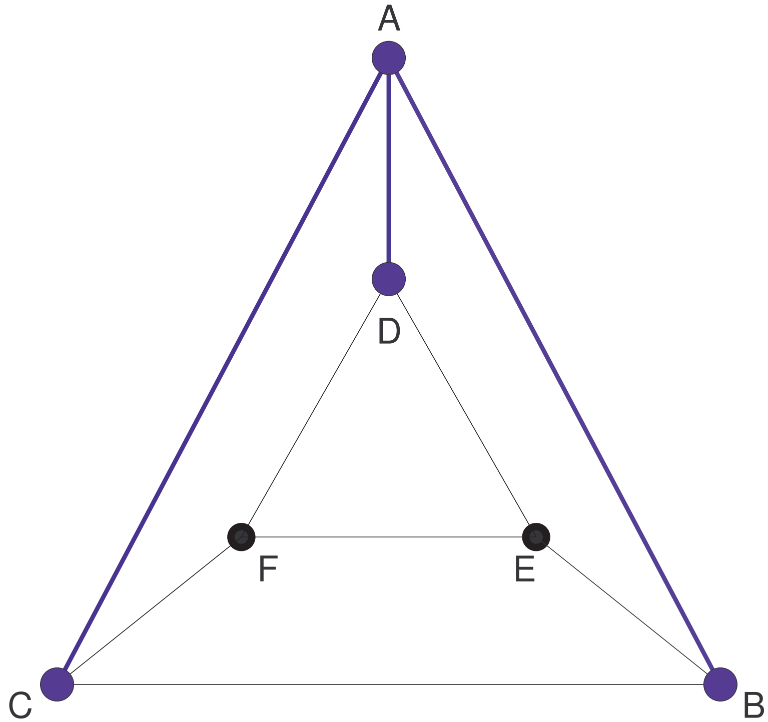
$$ABACA$$

Example – a BFS traversal Path



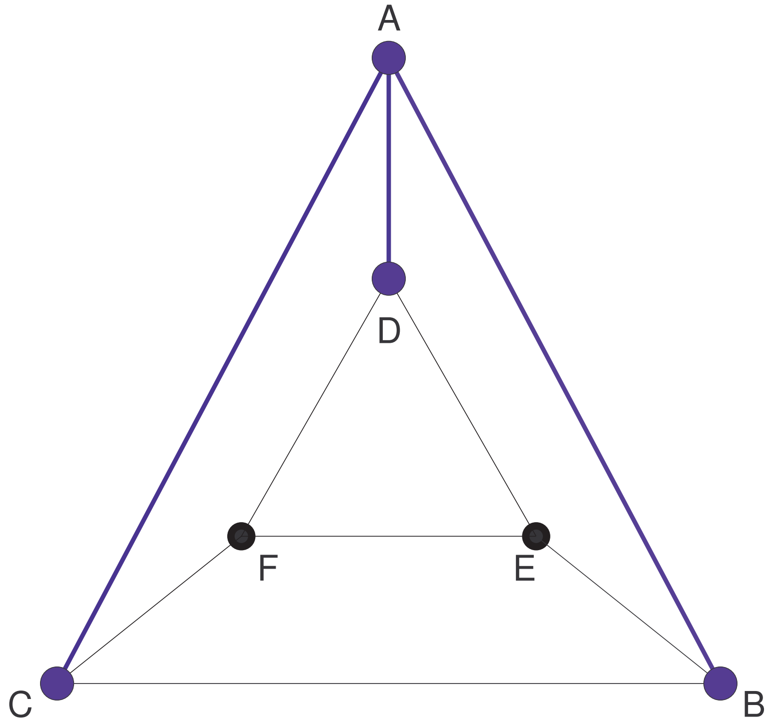
$$ABACD$$

Example – a BFS traversal Path



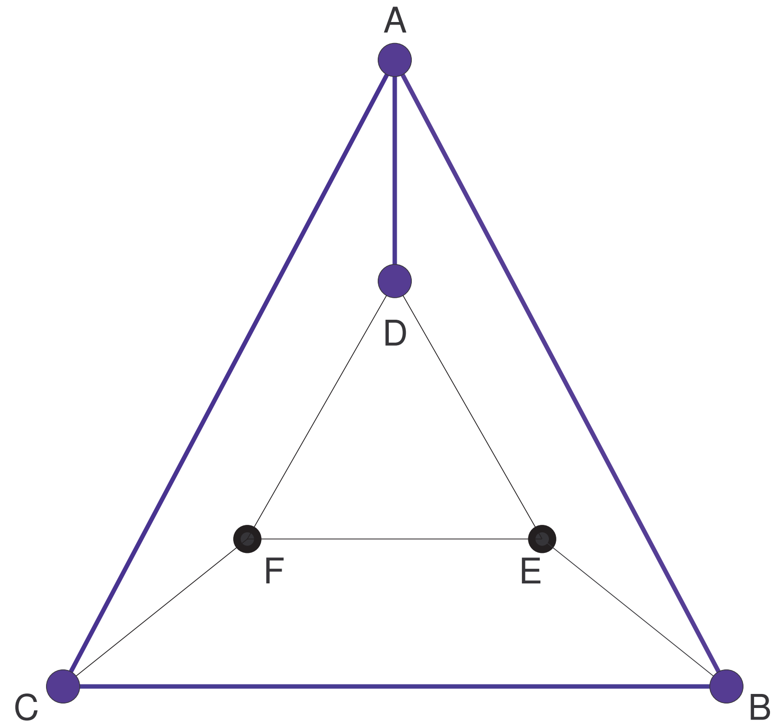
$$ABACDA$$

Example – a BFS traversal Path



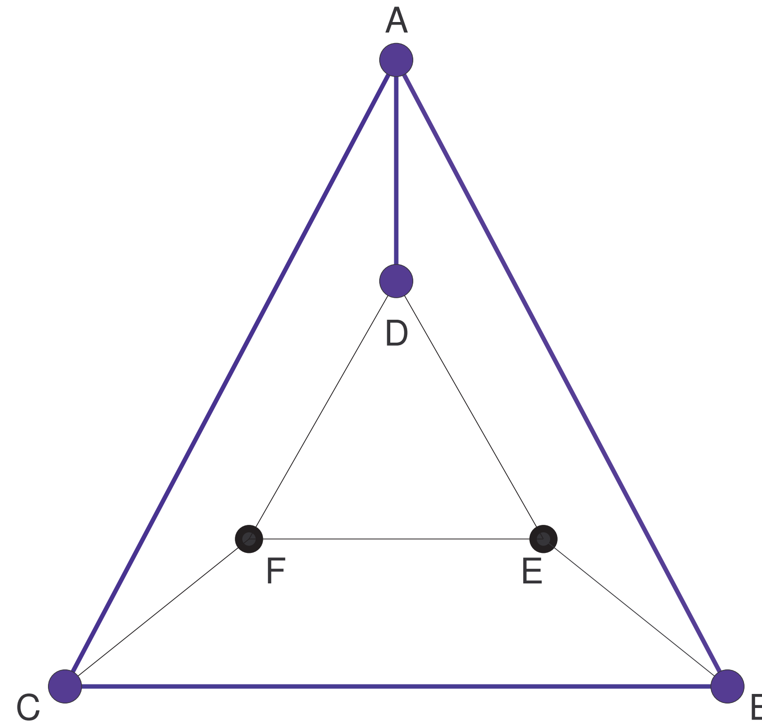
$$ABACDAB$$

Example – a BFS traversal Path



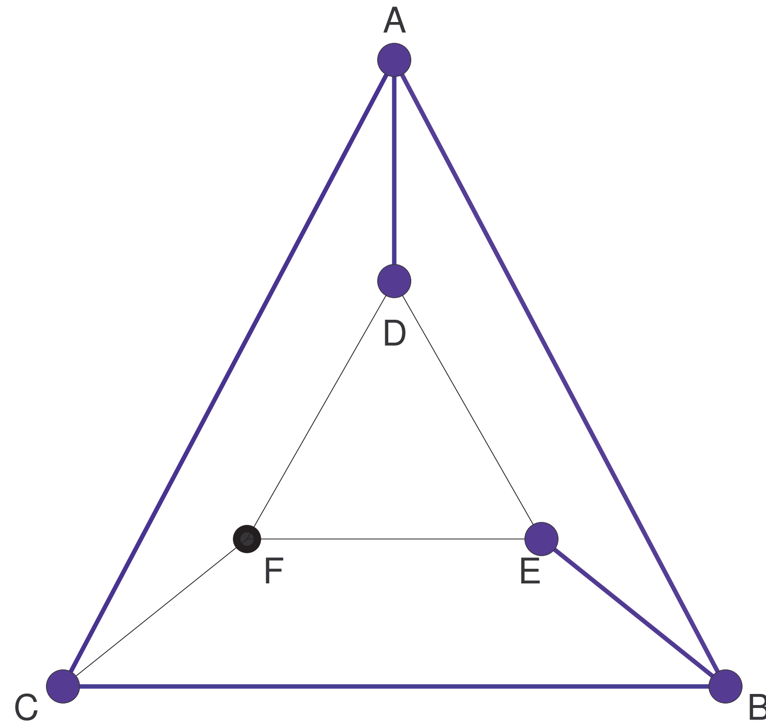
$$ABACDABC$$

Example – a BFS traversal Path



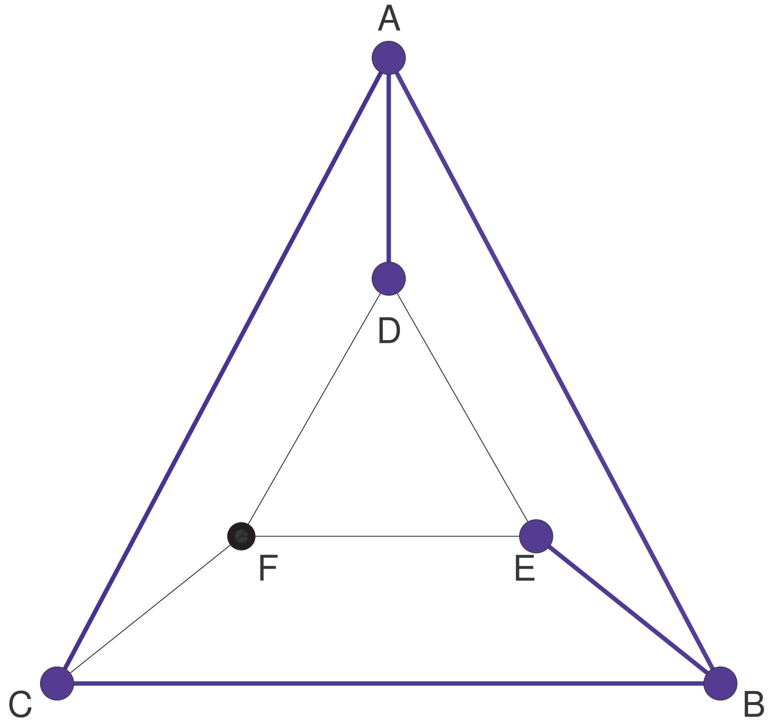
$$ABACDABCB$$

Example – a BFS traversal Path



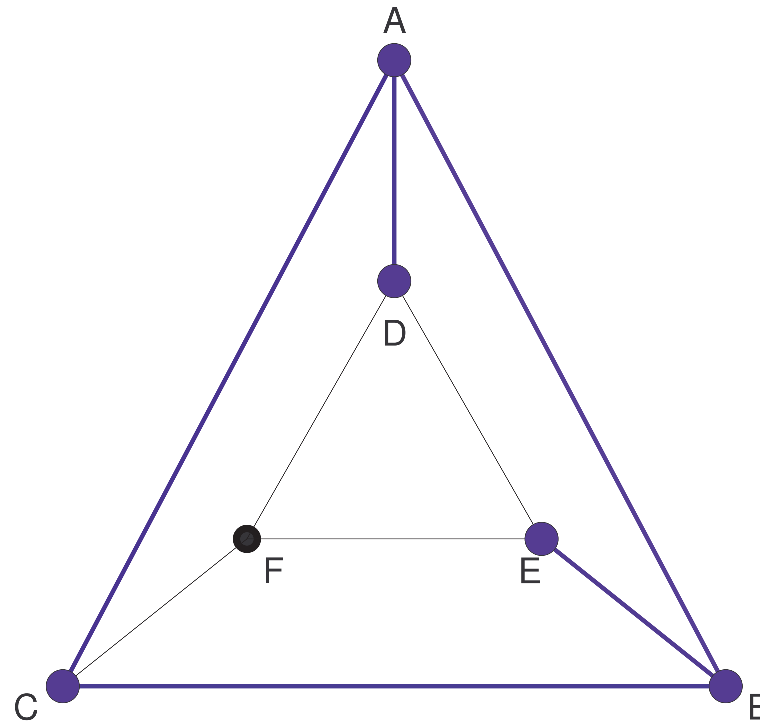
$$ABACDABCBE$$

Example – a BFS traversal Path



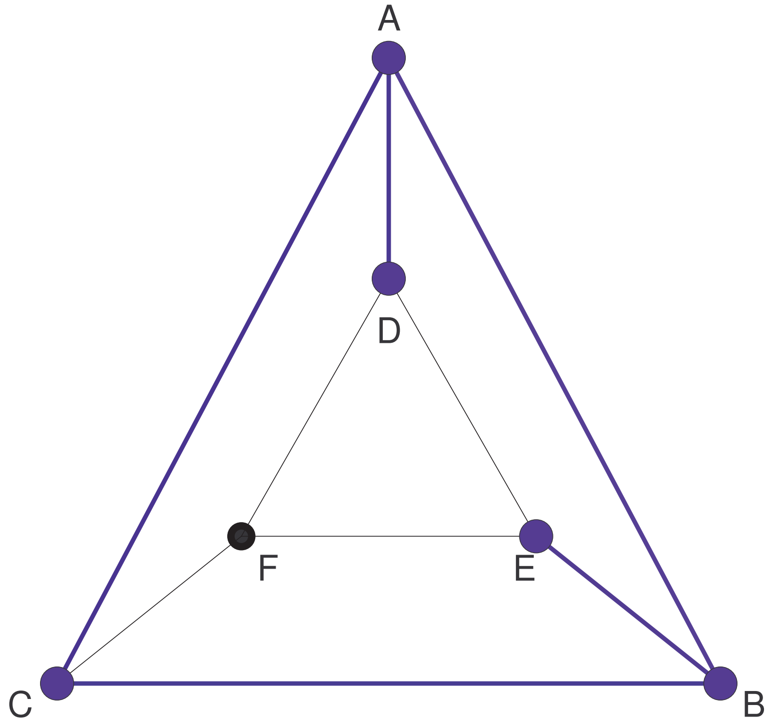
$$ABACDABCBEB$$

Example – a BFS traversal Path



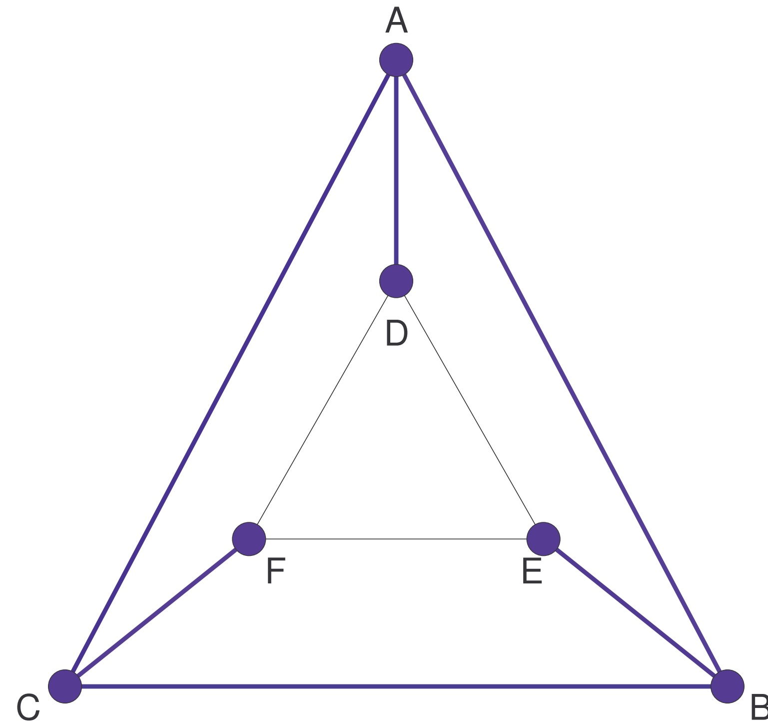
$$ABACDABCBEBA$$

Example – a BFS traversal Path



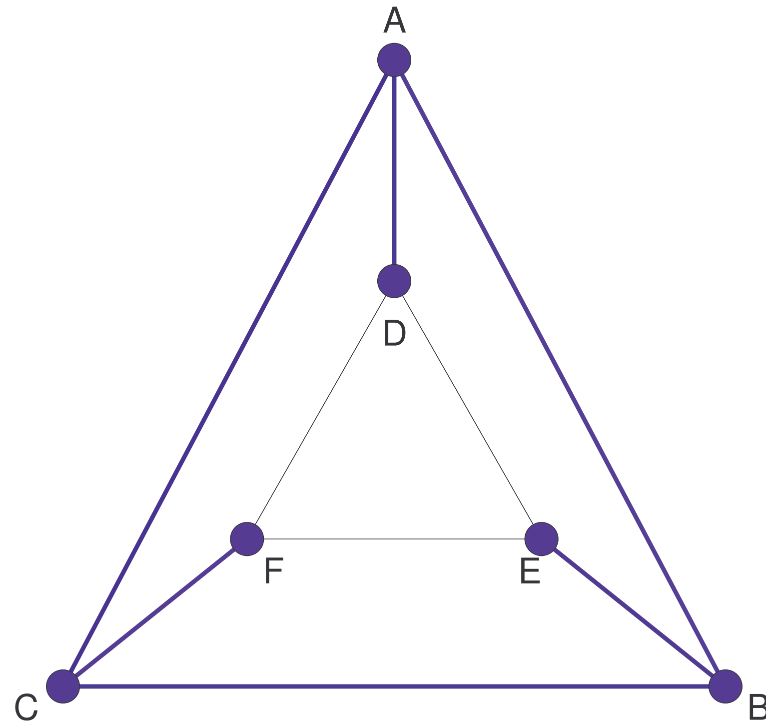
$$ABACDABCBEBAC$$

Example – a BFS traversal Path



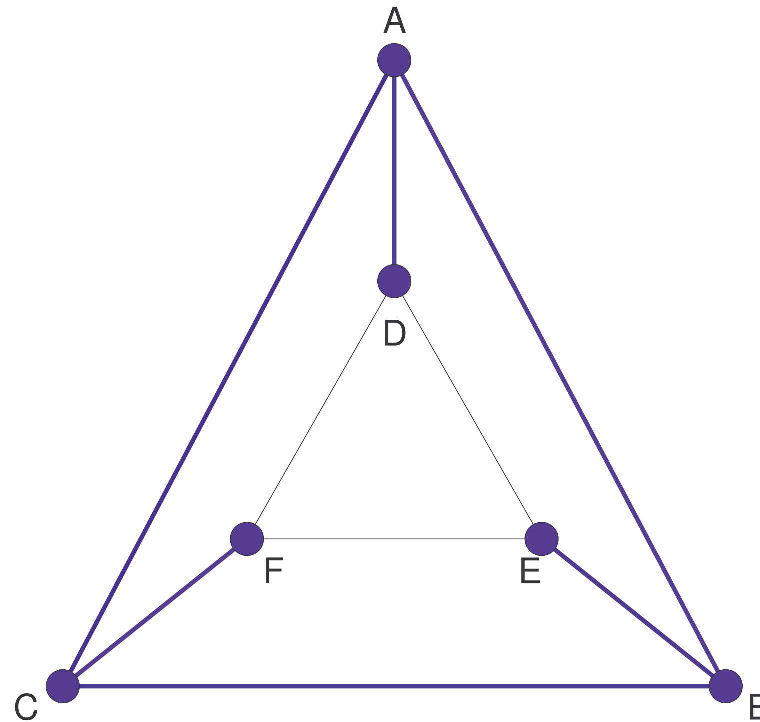
$$ABACDABCBEBACF$$

Example – a BFS traversal Path



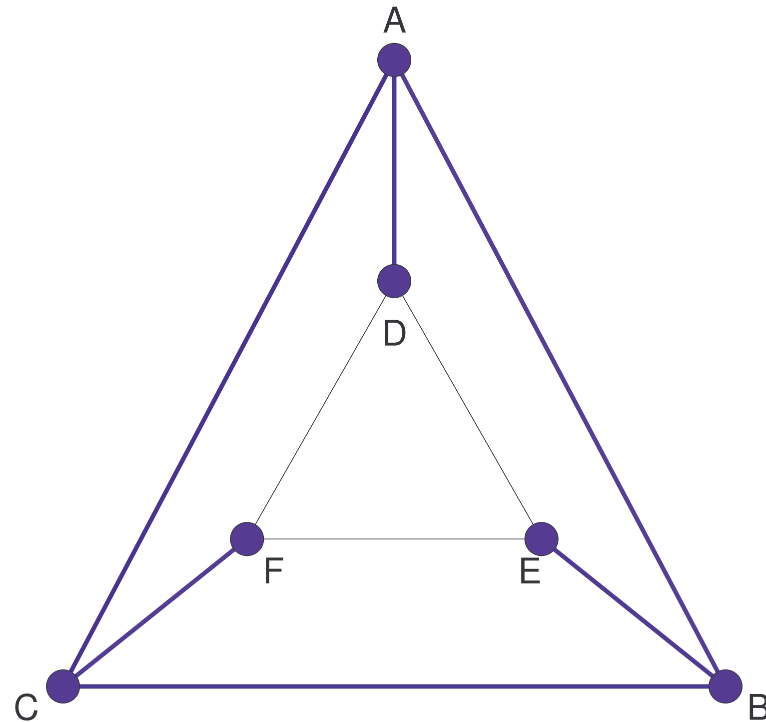
$$ABACDABCBEBACFC$$

Example – a BFS traversal Path



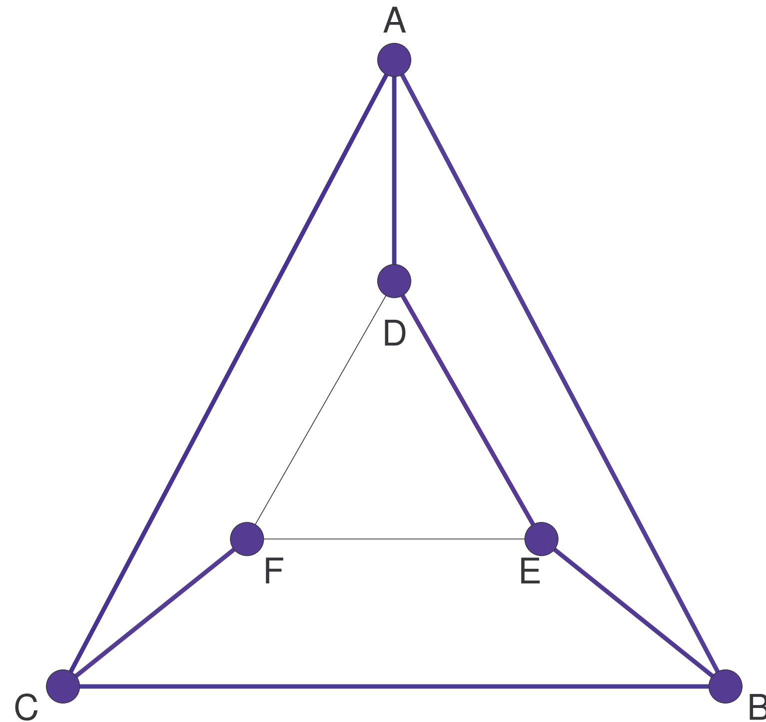
$$ABACDABCBEBACFCA$$

Example – a BFS traversal Path



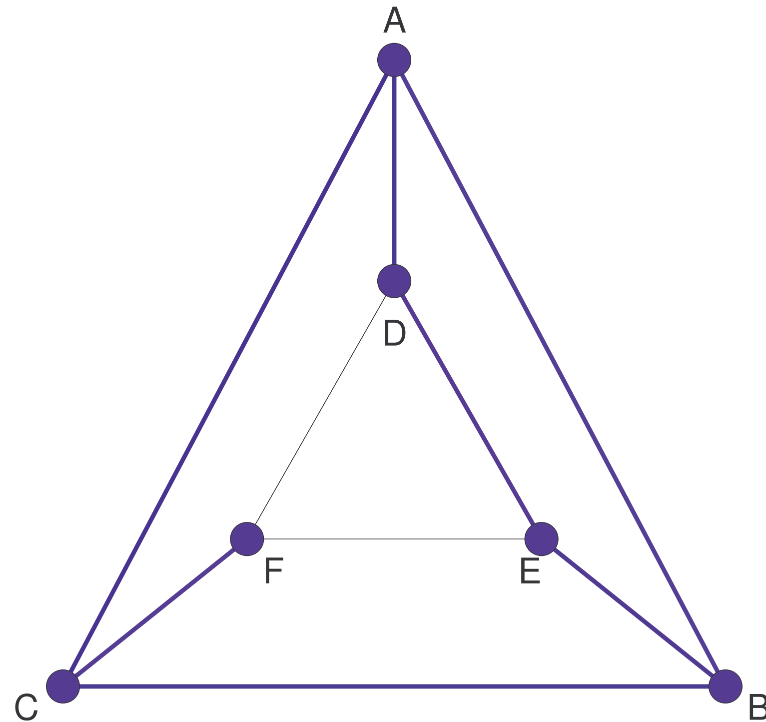
$$ABACDABCBEBACFCAD$$

Example – a BFS traversal Path



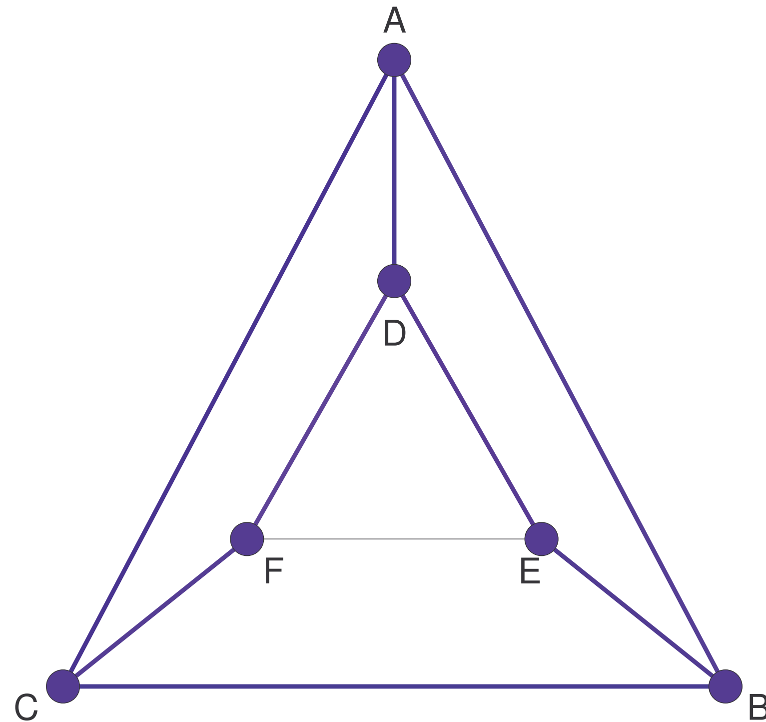
$$ABACDABCBEBACFCADE$$

Example – a BFS traversal Path



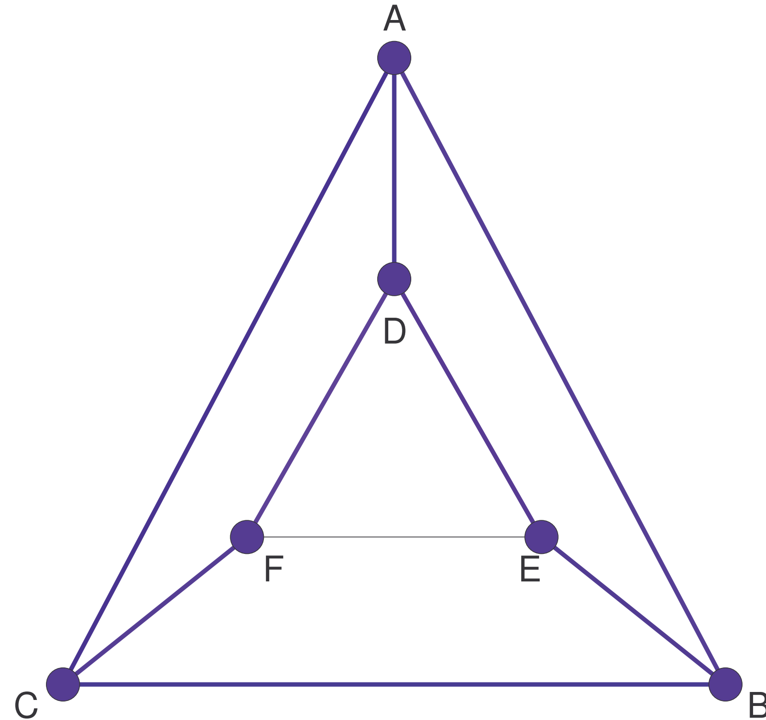
$$ABACDABCBEBACFCADED$$

Example – a BFS traversal Path



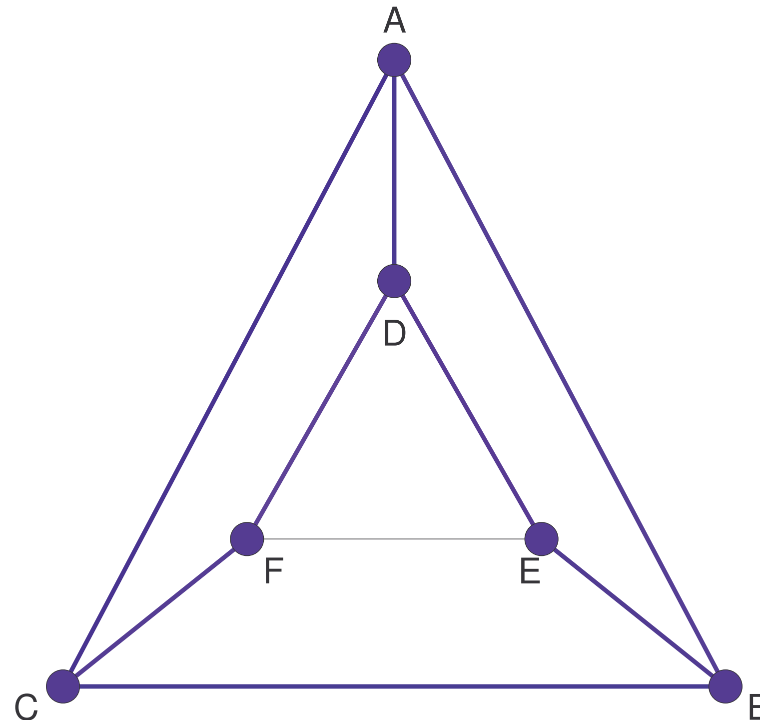
$$ABACDABCBEBACFCADEDF$$

Example – a BFS traversal Path



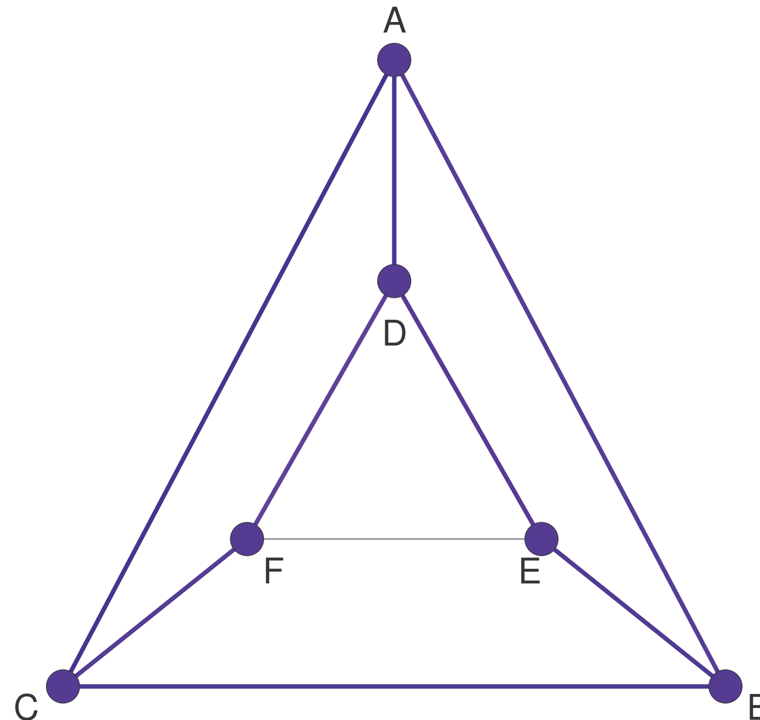
$$ABACDABCBEBACFCADEDFD$$

Example – a BFS traversal Path



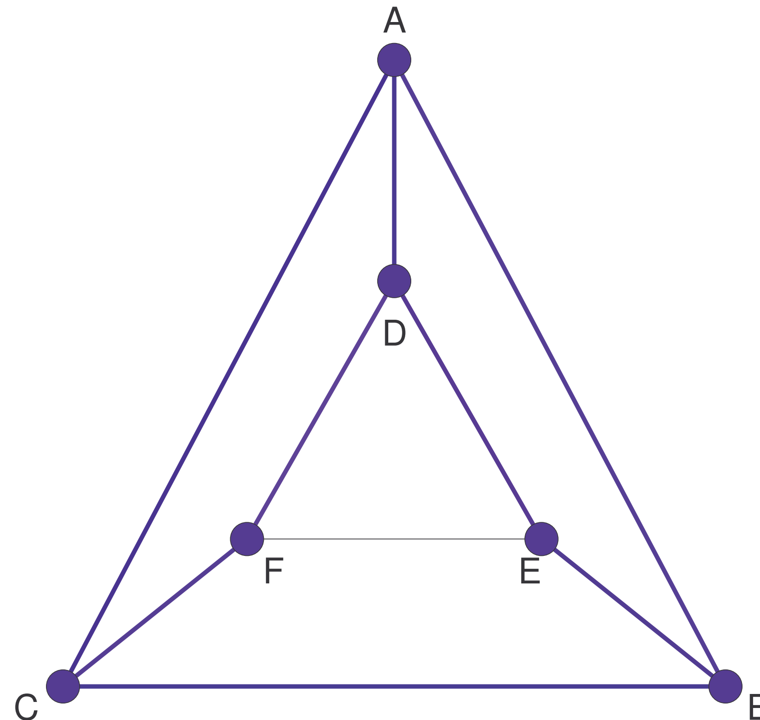
$$ABACDABCBEBACFCADEDFDA$$

Example – a BFS traversal Path



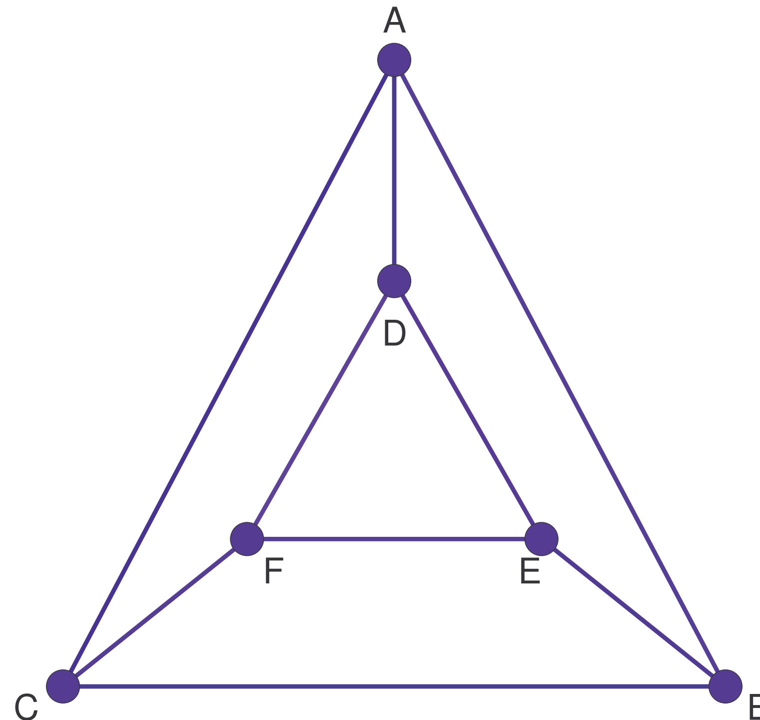
$$ABACDABCBEBACFCADEDFDAB$$

Example – a BFS traversal Path



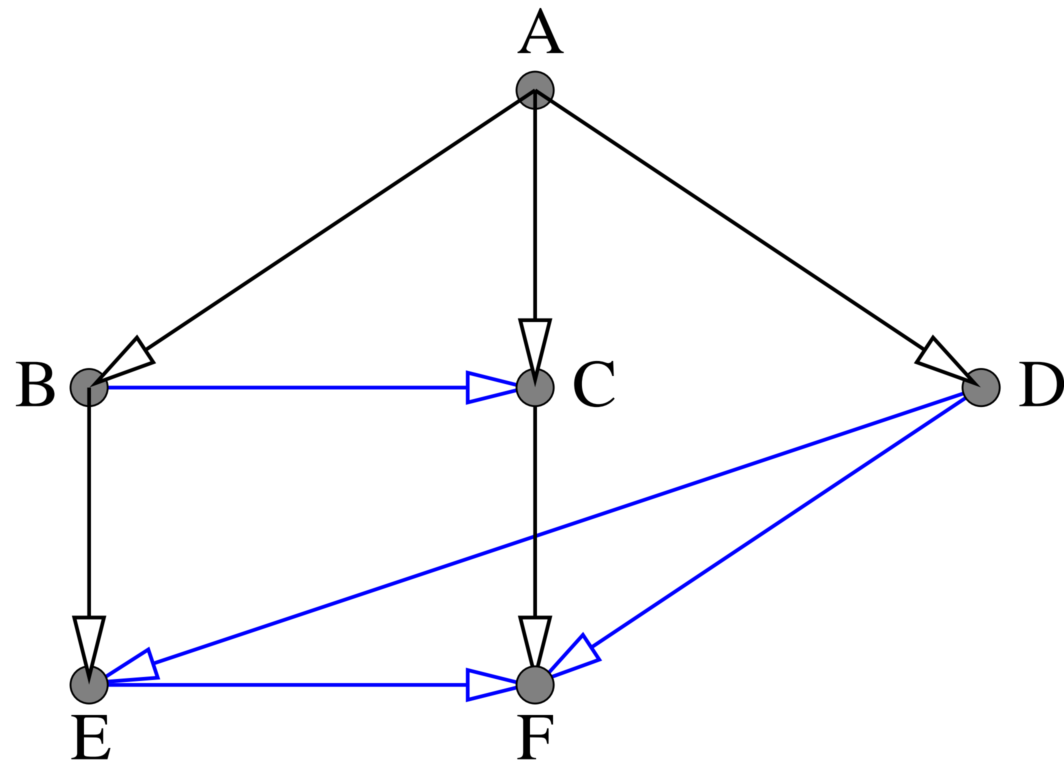
$$ABACDABCBEBACFCADEDFDABE$$

Example – a BFS traversal Path



$$ABACDABCBEBACFCADEDFDABEF$$

Example – the BFS Traversal Tree



$$ABACADABCBEBACFCADEDFDABEF$$

Variables for the BFS Procedure

Definitions:

- The distance between vertex u and vertex v is the length in edges of the shortest path from u to v.
- The level of a vertex in the tree is its distance to the root of the tree.

⋆ A level function $Level(v)$:

- $Level(v) = 0$ if v is a root of a tree in the forest.
- $Level(v) = \ell$ if v is at level ℓ in its tree.

Variables for the BFS Procedure

- ⋆ A FIFO queue Q:
  - $x = First(Q)$: Delete x the first element in Q.
  - $Last(Q) = x$: Add x as the last element in Q.
  - $Create(Q)$: Create an empty queue Q.
  - $Empty(Q)$: Check if the queue Q is empty.
- ⋆ A parenthood function $\Pi(v)$ in the traversal forest:
  - $\Pi(v) = nil$ if v is a root of a tree in the forest.
  - $\Pi(v)$ is v's parent in the traversal tree that contains v.

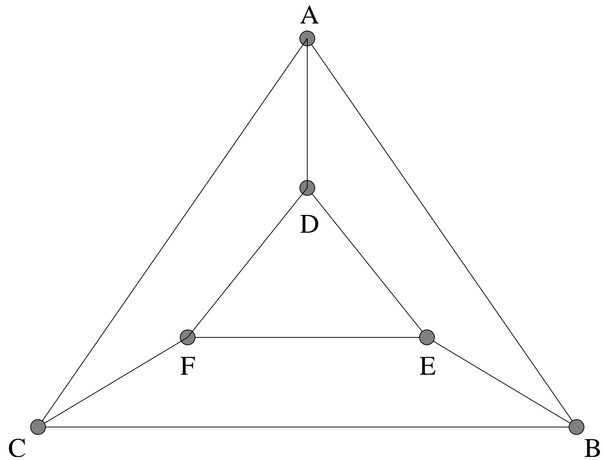
The BFS Procedure – Initial Call

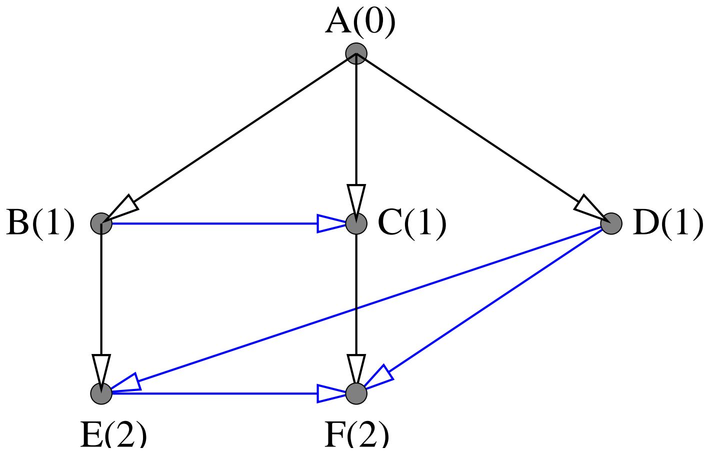
BFS(G)
for each vertex $v \in V$ do
$Level(v) = \infty$
$\Pi(v) = Nil$
for each vertex $r \in V$ do
if $Level(r) = \infty$ then
BFS-Visit(r)

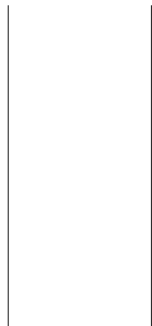
The BFS Procedure – for Each Connected Component

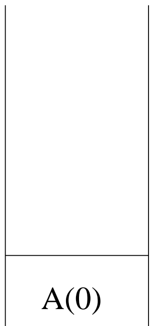
BFS-Visit(r)
  $Level(r) = 0$
  $Create(Q)$
  $Last(Q) = r$
  while (not $Empty(Q)$) do
    $v = First(Q)$
    for each neighbor u of v do
    if $Level(u) = \infty$ then
      $\Pi(u) = v$
      $Level(u) = Level(v) + 1$
      $Last(Q) = u$

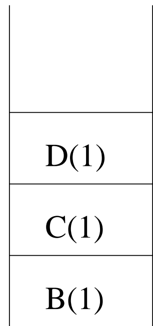
BFS – Example

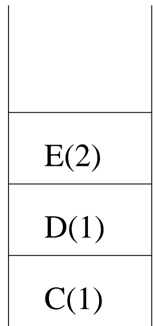


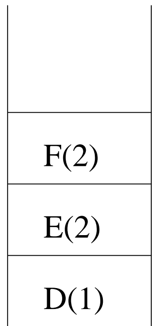


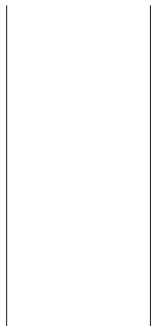












BFS – Correctness

Lemma: BFS visits all the vertices.

Proof: Each vertex changes its level ℓ as follows

$$\ell = \infty \rightarrow 0 \leq \ell < \infty$$

Lemma: BFS traverses all the edges.

Proof: For each vertex, BFS examines all of its incident edges in undirected graphs and all of its outgoing edges in directed graphs.

BFS – Traversal Path

Undirected graphs: The traversal path is not explicit. Traversing back the edges is done implicitly due to the queue handling.

Directed graphs: The traversal path sometimes uses the wrong direction due to the queue handling.

BFS – Time Complexity

Adjacency lists: $\Theta(n+m)$.

- $\star$ $\Theta(m)$: Each edge is examined once in directed graphs and twice in undirected graphs.
- $\star$ $\Theta(n)$: Each vertex gets a level $\ell < \infty$ exactly once.

Adjacency matrix: $\Theta(n^2)$.

- $\star$ $\Theta(n)$: For each vertex, for examining all of its incident edges in undirected graphs and all of its outgoing edges in directed graphs. of each vertex.

The BFS Traversal Tree

Lemma: There are no **forward** edges neither in directed graphs nor in undirected graphs.

Proof: A **forward** edge would be a **tree** edge.

Lemma: There are no **back** edges in undirected graphs.

Proof: A **back** edge would be a **tree** edge while examined in the other direction.

The BFS Traversal Tree

Lemma: In undirected graphs, $Level(u) = Level(v)$ or $Level(u) = Level(v) - 1$ for a **cross** edge (u, v)

Proof:

- $\star$ u is added to the queue before v.
- $\star$ Otherwise BFS would examine (v, u) before (u, v).
- $\star$ Therefore, $Level(u) \leq Level(v)$.
- $\star$ $Level(u) < Level(v) - 1 \Rightarrow (u, v)$ would be a **tree** edge.

BFS – Application

Problem: Let G be a connected undirected graph and let u and v be 2 vertices. Find one of the shortest paths from u to v and its length.

Algorithm:

- ⋆ Run the BFS algorithm starting with the vertex v.
- ⋆ Terminate when the vertex u is visited the first time.
- ⋆ The length of the shortest path from u to v is the level of vertex u in the BFS traversal tree.
- ⋆ The path $(u, \Pi(u), \Pi(\Pi(u)), \ldots, v)$ is one of the shortest paths from u to v.

Time complexity $O(n + m)$: the same as BFS.

Diameter of a Graph

Assumption: Let G be a connected undirected graph.

Notation: For 2 vertices u and v, denote by $dist(u, v)$ the length of the shortest path from u to v in G.

Definition: The diameter of G, denoted by $D(G)$, is the longest shortest path in G:

$$D(G) = \max_{u,v \in G} \{dist(u, v)\} \ .$$

Problem: Find the diameter of G.

Algorithm for any Graph G

Algorithm:

- ⋆ For all vertices v, run BFS starting with the vertex v.
- ⋆ Let u be one of the farthest vertices from v and let $r(v) = dist(v, u)$.
- ⋆ $D(G) \max_v \{r(v)\}$.

Correctness: By definition of BFS.

Complexity: $O(nm)$ for running BFS n times.

Diameter of a Tree

Problem: Find the diameter of a tree T.

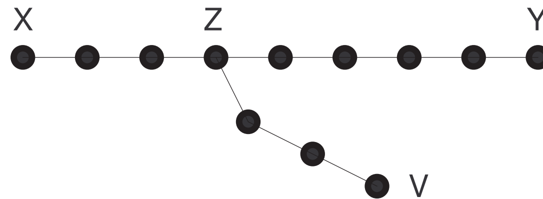
Algorithm:

- ⋆ Run BFS starting with an arbitrary vertex v.
- ⋆ Let u be one of the farthest vertices from v.
- ⋆ Run BFS starting with the vertex u.
- ⋆ Let w be one of the farthest vertices from u.
- ⋆ $D(G) = dist(u, w)$.

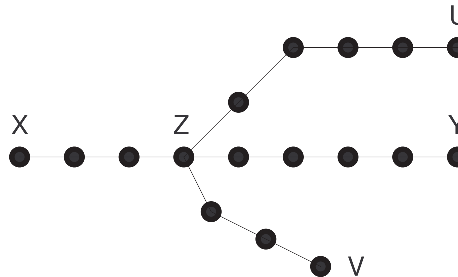
Complexity: $O(m)$ for running BFS twice.

Correctness

- Let x and y be 2 vertices such that $D(G) = dist(x, y)$.
- If v is either x or y, then $dist(v, w) = D(G)$ and therefore $dist(u, w) = D(G)$.
- Assume that $dist(v, y) \geq dist(v, x)$.
- Let z be the vertex that connects v to the path x to y.
  - z can be v or y.

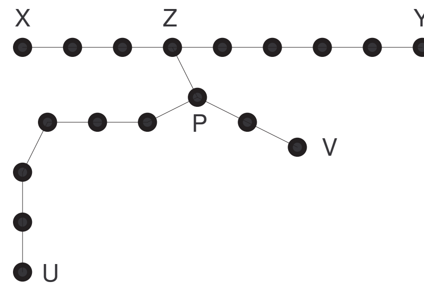


Correctness: Case I



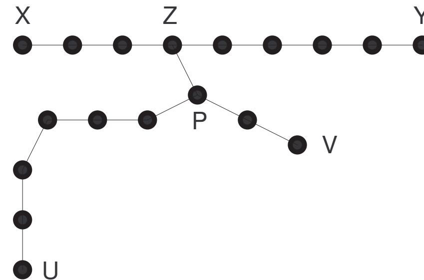
- ⋆ Assume that z is on the path from v to u.
- ⋆ By definition, $dist(v, u) \geq dist(v, y)$.
- ⋆ Therefore, $dist(z, u) \geq dist(z, y)$.
- ⋆ Therefore, $D(G) = dist(x, u)$.
- ⋆ By definition, $dist(u, w) \geq dist(u, x)$.
- ⋆ Therefore, $D(G) = dist(u, w)$.

Correctness: Case II



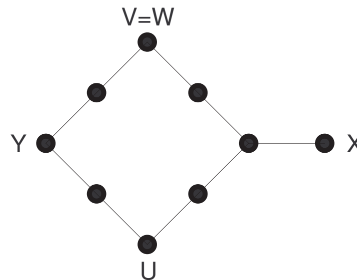
- ⋆ Assume that z is not on the path from v to u.
- ⋆ Let p be the vertex on the path from v to u and the path from v to z.
  - – p can be v.
- ⋆ By definition, $dist(v, u) \geq dist(v, y)$.
- ⋆ Therefore, $dist(p, u) \geq dist(p, y)$.

Correctness: Case II



- $\star$ $dist(z,u) > dist(p,u)$ and $dist(z,y) < dist(p,y)$ by definition.
- $\star$ Therefore, $dist(z,u) > dist(z,y)$.
- $\star$ Therefore, $dist(x,u) > dist(x,y)$.
- $\star$ A contradiction since $D(G) = dist(x,y)$.

The Algorithm Fails for Non-Trees



- $\star$ u is the only farthest vertex from v.
- $\star$ $v = w$ is the only farthest vertex from u.
- $\star$ $dist(x, y) > dist(u, w)$.

Strongly Connected directed Graphs

Definition: A directed graph $G = (V, E)$ is strongly connected if for any pair of vertices $u, v \in V$ there exists a directed path from u to v.

Problem: Given a directed graph $G = (V, E)$, determine if G is strongly connected.

Algorithm I

$\star$ For all vertices $v \in V$, run DFS or BFS starting with the vertex v. If there exists a vertex u that is not reachable from v, then return **NO** else return **YES**.

Correctness: By definition of the traversal procedures DFS and BFS.

Complexity: $O(nm)$ for running DFS or BFS n times.

Algorithm II

- ⋆ For some vertex w, run DFS or BFS on G starting with w.
- ⋆ If there exists a vertex u that is not reachable from w, then return **NO**.
- ⋆ Reverse the direction of all the edges in G to get a new directed graph G'.
- ⋆ Run DFS or BFS on G' starting with w.
- ⋆ If there exists a vertex v that is not reachable from w, then return **NO**.
- ⋆ Else, return **YES**.

Algorithm II – Correctness

The algorithm returns NO:

- ⋆ The algorithm produces an evidence of 2 vertices with no directed path between them.
- ⋆ Either there exists a vertex u such that there is no directed path from w to u in G.
- ⋆ Or there exists a vertex v such that there is no directed path from w to v in G' which is equivalent to having no directed path from v to w in G.

Algorithm II – Correctness

The algorithm returns YES:

- ⋆ Let $v, u \in V$ be any pair of vertices in G.
- ⋆ By the traversal result on G, it follows that there is a directed path from w to u in G.
- ⋆ By the traversal result on G', it follows that there is a directed path from w to v in G' which is equivalent to a directed path from v to w in G.
- ⋆ Combining the path from v to w with the path from w to u generates the path from v to u.

Algorithm II – Complexity

- ⋆ $O(n+m)$ for running DFS or BSF twice.
- ⋆ $O(n+m)$ for reversing the direction of all edges to produce the graph G'.
  - – Scan all the outgoing list to generate new outgoing lists.
- ⋆ $O(n+m)$ overall complexity.