

EECS3101 Algorithms (Summer'26)

Digest of Tutorial 6, June 15

Tracing DFS on a Directed Graph: Colours, Time Stamps, Edge Types

Jackie Wang

HOW TO USE THIS DIGEST

This digest **is** meant to help you identify the main ideas, important examples, common pitfalls, and practice targets from the lecture. It is **not** a replacement for attending class, watching the lecture recording, reviewing the slides and iPad notes, or doing your own practice and reflection. The digest intentionally summarizes only the roadmap and highlights; **you are expected to fill in the details** by revisiting the lecture materials, tracing examples, writing code, solving problems, and checking your own understanding. A good way to use this digest is as a checklist: after reviewing the lecture, you should be able to explain and apply the items listed under “*What you should be able to do*”.

This digest is an AI-assisted study roadmap reviewed by Jackie. It was produced from the lecture transcript, a carefully designed tool workflow, and Jackie’s actual hand-written iPad notes. Although the digest is reviewed before being posted, it **might** still contain typos, inconsistencies, omissions, or errors, just like any manually prepared course material. Therefore, use it as a study guide, **not** as an unquestionable authority. If something seems unclear or inconsistent, you are responsible for asking for clarification and verifying your understanding. Possible errors in the digest should **not** be treated as permission to submit incorrect work, or as a substitute for understanding the course materials.

TABLE OF CONTENTS

Main Ideas	2
Roadmap	2
Key Terms, Notation, and Structures	5
Key Examples and Demonstrations	6
Common Pitfalls and Misconceptions	7
What You Should Be Able to Do	8
Suggested Practice	9
Where to Review	10
Closing Reminder	10

Main Ideas

- Tutorial 6 was a practice-focused directed-graph DFS trace, continuing the directed DFS notation from Lecture 12.
- The main skill was to trace DFS from a specified start vertex while simultaneously tracking the stack, vertex colors, discovery times, finish times, the DFS tree, and edge classifications.
- The example used a directed graph with vertices A, B, C, D, E, F , $|V| = 6$, $|E| = 11$, start vertex A , and alphabetical order for choosing outgoing edges.
- Before tracing, Jackie emphasized graph visualization: draw the directed graph from V and E , check every vertex and edge, and avoid a messy layout that makes arrows difficult to follow.
- The trace illustrated the expected counts: a single DFS tree on six reachable vertices has $|V| - 1 = 5$ discovery edges, the remaining six edges are non-tree edges, and the final timestamp is $2 \cdot |V| = 12$.
- The tutorial reinforced that only push and pop events increment the global timer t ; classifying a back, forward, or cross edge does not change the stack or the timer.
- Jackie connected this skill to assessment preparation: similar edge-classification and tracing tasks may appear on tests or exams, and Programming Test 1 is on June 22 after reading week.

Roadmap

Reviewing the directed DFS notation from Lecture 12

Jackie began by reminding students that this tutorial continues the directed DFS material from Thursday. The relevant background is the notation page from the directed DFS lecture: discovery time d , finish time f , vertex colors white/grey/black, DFS trees, and edge categories. A specific notation overloading warning was repeated: the DFS d for discovery time is not the same as the d used earlier in the context of self-balancing BSTs.

Course logistics before the trace

Jackie reminded students that reading week starts on Tuesday, Assignment 1 is due on Wednesday, and Programming Test 1 is on Monday, June 22. Students were told to read the posted programming-test guide and follow Jackie's later reminders about room-entry procedure and test expectations. Jackie also mentioned a Wednesday 1:00 p.m. Zoom office hour and explained that Assignment 2 would involve DFS on an undirected graph, while this tutorial gives one more directed DFS practice example.

Reading a directed graph from V and E

The tutorial exercise did not begin with only a finished drawing. It specified the directed graph by a vertex set and an edge set, so the first task was to visualize the graph. Jackie emphasized that this is an assessment-relevant skill: before starting a trace, verify that all vertices are present, every directed edge has the correct direction, and no extra edge has been inserted. The specific graph had

$$V = \{A, B, C, D, E, F\}$$

and directed edges are

$$\{(A, B), (A, C), (A, D), (B, C), (C, A), (D, E), (D, F), (E, A), (E, F), (F, B), (F, C)\}$$

The pair (A, C) and (C, A) was discussed as allowed in a simple directed graph: they are opposite-directed (bi-directional) edges, not a self-loop and not parallel copies of the same directed edge.

Pre-trace checks: counts and expectations

Before performing the trace, Jackie used the graph size to predict what the completed trace should look like. Since $|V| = 6$, a DFS tree covering all six reachable vertices should have $6 - 1 = 5$ discovery edges. Since $|E| = 11$, the remaining $11 - 5 = 6$ edges must be non-tree edges, classified as back, forward, or cross edges. Since each vertex is pushed once and popped once, the timer should end at $2 \cdot |V| = 12$, with $t = 0$ reserved for the initial all-white state. These counts are not just facts to memorize; they are error checks during tracing.

Setting up the trace tables

Jackie set up several records at once: a push/pop table, a table of $d(v)$ and $f(v)$, a time-indexed color table for white/grey/black vertices, a stack, and a DFS tree diagram. The start vertex was A , so A becomes the root of the DFS tree. The initial row $t = 0$ has all vertices white, the stack empty, and no discovery or finish times defined. Jackie also posed review questions for each color: whether d is defined, whether f is defined, and whether the vertex is currently in the stack.

Beginning the trace: A , B , and C

The trace starts by pushing A , making A grey, and setting $d(A) = 1$. From A , the outgoing edges are processed in alphabetical order, so $A \rightarrow B$ is the first discovery edge. Then $B \rightarrow C$ is the next discovery edge. At this point the DFS tree contains $A \rightarrow B \rightarrow C$, the stack top is C , and $d(A) = 1$, $d(B) = 2$, $d(C) = 3$. Jackie stressed that the current decision point is always the top of the stack, not an older vertex lower in the stack.

First non-tree edge: $C \rightarrow A$ as a back edge

From C , the only outgoing edge to consider is $C \rightarrow A$. Since A is an active ancestor still in the stack, this edge is classified as a back edge. This classification happens while $t = 3$, and because it is not a discovery edge, the timer and stack do not change. Jackie highlighted this as a likely test-style question: students may be asked not only what type of edge it is, but also at what time t it was identified.

Backtracking from C and B

After $C \rightarrow A$ is classified, C has no remaining outgoing edge to process, so C is popped, colored black, and assigned $f(C) = 4$. Then B also has no remaining outgoing edge, so B is popped, colored black, and assigned $f(B) = 5$. This part of the trace illustrated the difference between edge-identification labels and timestamps: edge labels record the chronological order of edge consideration, while d and f record push/pop times.

Returning to A : forward edge then new branch

With A again at the top of the stack, Jackie considered the remaining outgoing edges from A in order. The edge $A \rightarrow C$ is a forward edge because C is already a finished descendant of A in the DFS tree; this is identified while $t = 5$, with no push or pop. Then $A \rightarrow D$ discovers a new vertex, so D is pushed, colored grey, and assigned $d(D) = 6$. Jackie also pointed out that D should be attached as a child of A in the DFS tree, because A is the vertex from which D was discovered.

Exploring the D, E, F branch

From D , alphabetical order chooses $D \rightarrow E$ before $D \rightarrow F$, so E is discovered with $d(E) = 7$. From E , alphabetical order considers $E \rightarrow A$ before $E \rightarrow F$, so $E \rightarrow A$ is a second back edge: it points to the active ancestor A and is identified while $t = 7$. Then $E \rightarrow F$ discovers F , assigning $d(F) = 8$. At this point all vertices have been discovered, but the trace is not finished because several outgoing edges and finish times still need to be handled.

Cross edges from F , then finishing the branch

From F , the outgoing edges $F \rightarrow B$ and $F \rightarrow C$ are processed alphabetically and classified as cross edges. In the DFS tree, F lies on the branch $A - D - E - F$, while B and C lie on a different branch under A . Neither target is an ancestor or descendant of F . Both cross edges are identified while $t = 8$, and neither changes the stack. After that, F is popped with $f(F) = 9$, and E is popped with $f(E) = 10$.

Final forward edge and completing the trace

After returning to D , the remaining edge $D \rightarrow F$ is a forward edge because F is a finished descendant of D . Jackie noted this as analogous to $A \rightarrow C$. Then D is popped with $f(D) = 11$,

and finally A is popped with $f(A) = 12$. The completed trace matches the earlier predictions: six pushes, six pops, five discovery edges, six non-tree edges, and final timestamp 12.

Key Terms, Notation, and Structures

- **DFS run:** One execution of depth-first search starting from a designated start vertex.
- **Start vertex:** The vertex where the DFS run begins; in this tutorial the start vertex is A , which becomes the root of the DFS tree.
- **DFS tree:** The tree formed by the discovery edges in one DFS run; in this example it spans all six vertices reached from A .
- **Discovery edge / tree edge:** An edge that leads to a white vertex and expands the DFS tree.
- **Back edge:** A non-tree edge pointing to an active ancestor on the stack, such as $C \rightarrow A$ or $E \rightarrow A$ in this trace.
- **Forward edge:** A non-tree edge pointing to a finished descendant in the DFS tree, such as $A \rightarrow C$ or $D \rightarrow F$.
- **Cross edge:** A non-tree edge between vertices with no ancestor/descendant relation in the DFS tree, such as $F \rightarrow B$ and $F \rightarrow C$ in this trace.
- **White vertex:** A vertex not yet discovered; its discovery and finish times are not yet defined, and it is not in the stack.
- **Grey vertex:** A discovered but unfinished vertex; its discovery time is defined, its finish time is not yet defined, and it is in the stack.
- **Black vertex:** A finished vertex; both discovery and finish times are defined, and it has been popped from the stack.
- t : The global DFS timer; in this tutorial $t = 0$ is the initial state and the first discovery event has time 1.
- $d(v)$ or $d(v)$: The discovery time of vertex v , assigned when v is pushed and turns grey.
- $f(v)$ or $f(v)$: The finish time of vertex v , assigned when v is popped and turns black.
- **Stack discipline:** At each step, decide what to do from the top of the stack; do not jump back to an older vertex unless backtracking has actually popped vertices above it.
- **Alphabetical adjacency order:** The visiting rule used in the tutorial; if several outgoing edges are available, consider their destination vertices alphabetically unless Jackie specifies a different order.
- **Simple directed graph with bidirectional edges:** Opposite directed edges such as (A, C) and (C, A) are allowed because they are not self-loops and not duplicate parallel copies of the same directed edge.

Key Examples and Demonstrations

Drawing the graph from vertex and edge sets

Setup: The tutorial exercise provided $V = \{A, B, C, D, E, F\}$ and a set of eleven directed edges rather than asking students to trace from a pre-digested answer.

Main point: Before any DFS trace, the graph itself must be correct. Jackie emphasized checking every vertex, every directed edge, and every arrow direction. A wrong diagram makes the entire trace unreliable.

What to practise: Recreate the graph from the listed V and E without looking at the final iPad diagram, then check it edge by edge against the notes before starting DFS.

Precomputing trace checks from $|V|$ and $|E|$

Setup: The graph has $|V| = 6$ and $|E| = 11$, and the DFS starts at A and reaches all vertices.

Main point: A completed single DFS tree on six vertices should contain 5 discovery edges. Since there are eleven total edges, six edges remain to be classified as non-tree edges. Since every vertex is pushed and popped once, the final timer should be 12.

What to practise: Before retracing the example, write down the expected values 5, 6, and 12, then use them to check whether your trace missed an edge, a push, a pop, or a timestamp.

The full directed DFS trace from A

Setup: Start at A , use alphabetical order for outgoing edges, and maintain the stack, colors, discovery/finish times, DFS tree, and edge labels.

Main point: The trace produces discovery times

$$d(A) = 1, d(B) = 2, d(C) = 3, d(D) = 6, d(E) = 7, d(F) = 8.$$

The finish times are

$$f(A) = 12, f(B) = 5, f(C) = 4, f(D) = 11, f(E) = 10, f(F) = 9.$$

The discovery edges are $A \rightarrow B$, $B \rightarrow C$, $A \rightarrow D$, $D \rightarrow E$, and $E \rightarrow F$. The non-tree edges are $C \rightarrow A$ and $E \rightarrow A$ as back edges, $A \rightarrow C$ and $D \rightarrow F$ as forward edges, and $F \rightarrow B$ and $F \rightarrow C$ as cross edges.

What to practise: Do not merely copy the final table. Reproduce the trace step by step and say, at each edge, whether it pushes a new vertex or only classifies a non-tree edge.

Classifying edges at a specific time

Setup: Jackie highlighted that a test question may ask when an edge is identified, not only what type it is.

Main point: In the trace, $C \rightarrow A$ is identified as a back edge when $t = 3$; $A \rightarrow C$ is identified as a forward edge when $t = 5$; $E \rightarrow A$ is identified as a back edge when $t = 7$;

$F \rightarrow B$ and $F \rightarrow C$ are identified as cross edges when $t = 8$; and $D \rightarrow F$ is identified as a forward edge after E and F have finished.

What to practise: For each non-tree edge, write the current stack, current t , target color, and tree relationship. This forces you to justify the label rather than guess from the final drawing.

Review questions on vertex colors

Setup: The final notes page asks, for each color, whether d is defined, whether f is defined, and whether the vertex is in the stack.

Main point: The color table is not decoration. White, grey, and black encode the vertex's exact DFS status: undiscovered, active, or finished.

What to practise: Fill in the answers for white, grey, and black vertices, then verify them against the trace at several different time rows.

Common Pitfalls and Misconceptions

Starting the trace before checking the graph drawing

Pitfall: Reading V and E quickly, drawing an incomplete or wrong directed graph, and then tracing from that drawing.

Why it causes problems: A missing arrow, reversed arrow, or extra edge changes the stack evolution, edge classifications, and timestamps.

How to avoid it: Before tracing, check the diagram against V and E exactly: all six vertices, all eleven directed edges, correct directions, no extras.

Confusing the initial state $t = 0$ with a discovery time

Pitfall: Assigning $d(A) = 0$ because the table begins at $t = 0$.

Why it causes problems: The tutorial uses $t = 0$ for the all-white initial state. The first actual event is pushing A , so $d(A) = 1$.

How to avoid it: Treat $t = 0$ as setup only. Start discovery and finish times at 1, and check that the final time is $2 \cdot |V|$.

Changing focus after a non-tree edge

Pitfall: After identifying $C \rightarrow A$ as a back edge, jumping to A 's outgoing edges because the edge points to A .

Why it causes problems: Non-tree edges do not push, pop, or change the current stack top. Jumping focus breaks DFS order and produces wrong finish times.

How to avoid it: Always ask, "What is currently on top of the stack?" Continue processing that vertex until all of its outgoing edges are handled, then pop it.

Popping a vertex just because an edge points to it

Pitfall: Seeing $E \rightarrow A$ and trying to pop A , or assuming A is finished because it was discovered earlier.

Why it causes problems: A vertex can be active for a long time. A stays on the stack until all branches below it have been completed.

How to avoid it: Pop only the stack-top vertex, and only after all of its outgoing edges have been processed.

Treating edge labels as timestamps

Pitfall: Mixing up the chronological edge-identification numbers with d , f , or t .

Why it causes problems: Edge labels count the order in which edges are considered. The timer increments only when vertices are pushed or popped.

How to avoid it: Keep separate columns or annotations: one for edge identification order, one for the global time t , and one for $d(v)/f(v)$.

Misclassifying forward and cross edges

Pitfall: Calling every edge to a black vertex a forward edge, or treating all non-tree edges as back edges.

Why it causes problems: Directed DFS has more categories than undirected DFS. A black descendant gives a forward edge, while a black vertex in another branch gives a cross edge.

How to avoid it: Use the DFS tree: descendant already finished means forward; different branch with no ancestor/descendant relation means cross.

Forgetting the specified adjacency order

Pitfall: Choosing $D \rightarrow F$ before $D \rightarrow E$, or choosing $E \rightarrow F$ before $E \rightarrow A$, despite the alphabetical-order assumption.

Why it causes problems: Different visitation orders can produce different traces, so tests must specify an order and students must follow it exactly.

How to avoid it: List each vertex's outgoing neighbors in the required order before tracing, then mark them off as they are considered.

What You Should Be Able to Do

After this tutorial, you should be able to:

- draw a directed graph correctly from a given set of vertices and directed edges;
- explain why opposite-directed (bi-directional) edges such as (A, C) and (C, A) do not by themselves violate the simple-graph assumption;
- determine the start vertex and root of a DFS tree for a specified DFS run;

- precompute expected trace checks such as $|V| - 1$ discovery edges and final timestamp $2 \cdot |V|$;
- maintain a stack-based directed DFS trace using the top-of-stack rule;
- update vertex colors correctly from white to grey to black;
- assign $d(v)$ when a vertex is pushed and $f(v)$ when a vertex is popped;
- distinguish edge-identification order from timestamp values;
- construct the DFS tree from discovery edges only;
- classify directed edges as discovery, back, forward, or cross using color, stack status, and ancestor/descendant relationships;
- reproduce the main Tutorial 6 trace from A and justify why $C \rightarrow A$, $A \rightarrow C$, $E \rightarrow A$, $F \rightarrow B$, $F \rightarrow C$, and $D \rightarrow F$ receive their classifications;
- explain why a non-connected graph may require more than one DFS run to visit all vertices and edges;
- use the posted Programming Test 1 guide and Jackie's office-hour support to prepare for the June 22 test.

Suggested Practice

- Review Lecture 12's directed DFS notation before redoing this tutorial trace; especially review colors, d , f , and the four directed edge types.
- Redraw the Tutorial 6 graph from the given V and E sets without looking at the final diagram, then check all eleven edges.
- Recreate the full trace table with rows $t = 0, 1, \dots, 12$, including white/grey/black sets, stack contents, push/pop order, and d/f values.
- For each edge in the graph, write its edge-identification order and type; then separately write the value of t when each non-tree edge is identified.
- Answer the page-5 review questions for each color: whether d is defined, whether f is defined, and whether the vertex is in the stack.
- Redo the trace with a 10- to 15-minute timer after you have understood it once; the first goal is correctness, then speed and tidiness.
- Make a small variant by changing the start vertex, or by changing the adjacency order, such as reverse alphabetical order, and observe which parts of the trace change.
- Read the Programming Test 1 guide, prepare questions during reading week, and use the Wednesday Zoom office hour if any assignment or test-preparation issue remains unclear.

Where to Review

- Review the Tutorial 6 iPad/PDF notes page 1 for the topic: DFS run on a directed graph.
- Review page 2 for the vertex and edge sets, the directed graph drawing, the start vertex A , the counts $|V| = 6$, $|E| = 11$, five discovery edges, six non-tree edges, and final timestamp 12.
- Review page 3 for the reminder that more than one DFS run may be needed when a graph is not connected; Jackie will revisit the directed/undirected details after reading week.
- Review page 4 for the completed trace: stack push/pop order, d/f table, white/grey/black table, DFS tree, and the classification of back, forward, and cross edges.
- Review page 5 for the vertex-color review questions; use them as a compact self-test before attempting another DFS trace.
- Rewatch the Tutorial 6 recording around the parts where Jackie explains why $C \rightarrow A$ is back, why $A \rightarrow C$ is forward, why $F \rightarrow B$ and $F \rightarrow C$ are cross, and why non-tree edges do not change the stack.
- Revisit Lecture 12 slides 31–32 and the Lecture 12 digest for the directed DFS terminology that this tutorial assumes.
- Check the posted Programming Test 1 guide and Jackie's course announcements for test logistics, allowed procedures, and reminders.

Closing Reminder

This tutorial digest is a roadmap for a directed DFS trace, not a substitute for doing the trace yourself. The skill only becomes reliable when you draw the graph, maintain the stack, update colors and timestamps, classify each edge, and check your result against the expected counts. Revisit Jackie's notes and recording, redo the Tutorial 6 table actively, and use reading week to turn this example into a repeatable tracing method before Programming Test 1.