Lecture 15: Applications of DFS: SCC

CS 341: Algorithms

Thursday, March 7\textsuperscript{th} 2019
Outline For Today

1. Strongly Connected Components in Directed Graphs
Recall: Connectivity in Undirected Graphs

Graph diagram showing connectivity with nodes labeled A, B, C, D, E, F, G, H, I, J, K, L.
Recall: Connectivity in Undirected Graphs
Connectivity in Directed Graphs
“Strong Connectivity” in Directed Graphs

SCC₁

L → I → J → K

SCC₂

A → D → H → B

SCC₃

E → F → G

SCC₄

C → B
Formal Definition of SCCs

- Given a directed graph $G(V, E)$
- $u$ and $v$ are strongly connected iff:
  $$u \sim v \text{ AND } v \sim u$$
- An SCC of $G$ is a set $C \subseteq V$ s.t
  1. $\forall (u, v) \in C$, $u$ and $v$ are strongly connected
  2. $C$ is non-empty and maximal, i.e. $\forall v' \text{ in } V-C$, $v'$ is not strongly connected to any vertex in $C$
Two-tiered Structure

- Given a directed graph $G(V, E)$
- Consider $G^{SCC}$: the meta-graph of its SCCs
- We’ll call this “SCC graph of $G$”
- **Claim:** $G^{SCC}$ is a DAG!

Each directed graph has a second higher-level structure which is a DAG.
Example of Two-tiered Structure
Example of Two-tiered Structure

SCC$_1$

SCC$_2$

SCC$_3$

SCC$_4$
Example of Two-tiered Structure

Claim: $G^{SCC}$ has to be a DAG!
Proof Sketch of $G^{SCC}$ being a DAG:

This is 1 SCC

Therefore, can’t have cycles in $G^{SCC}$
So We Can Topologically Order $G^{SCC}$

Example Topological Order of SCCs:

$SCC_1$  $SCC_2$  $SCC_4$  $SCC_3$

Will be important for Kosararaju’s Algorithm!
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs

Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Recall Algorithm for Undirected Graphs
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?
Can DFS/BFS Also Identify SCCs?

Identified 3 SCCs, not 1!
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?
What If We Got Very Lucky?

Now We Have Identified the SCCs Correctly!
Need to start BFS/DFS from a vertex in reverse topological order of $G^{SCC}$
Key Idea in Kosaraju’s Algorithm

- If we can start a BFS/DFS from vertices in reverse topological order of $G^{SCC}$
- In other words, if we can always pick a start vertex from a sink SCC
- We can use BFS/DFS to identify the components correctly!

Goal: Do some preprocessing to guarantee that we always start a traversal from vertex from a sink SCC!

How?
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
Let's Try DFS & Finishing Times (1)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)

Graph with nodes labeled A to I and edges connecting them. The nodes are colored and numbered from 1 to 11.
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
DFS & Finishing Times (2)
Looks like the latest finishing time vertex is always in $\text{SCC}_1$! WHY?
Definition: Finishing Time of an SCC

Let “finishing time of an SCC$_i$” in DFS traversal be the maximum finishing time of the vertices in SCC$_i$

\[ f[SCC_i] = \max \text{ over } v \in SCC_i \ f[v] \]
Finishing Times of Simulation (1)

- $f[SCC_1] = 12$
- $f[SCC_2] = 9$
- $f[SCC_3] = 4$
- $f[SCC_4] = 6$
Finishing Times of Simulation (2)

 SCC₁

 f[SCC₁] = 12

 SCC₂

 f[SCC₂] = 8

 SCC₃

 f[SCC₃] = 3

 SCC₄

 f[SCC₄] = 7
Key Lemma about finishing times of SCCs

\[ \text{In } G^{\text{SCC}} \text{ if } \text{SCC}_i \rightarrow \text{SCC}_j, \text{ then} \]
\[ \text{***} f[\text{SCC}_i] > f[\text{SCC}_j] \text{***} \]

Will prove this later
Extended Key Lemma

\[ \text{In } G^{\text{SCC}} \text{ if } \text{SCC}_i \sim \text{SCC}_j, \text{ then } \]
\[ f[\text{SCC}_i] > f[\text{SCC}_j] \]

Can prove by induction on the length of the paths using Key Lemma.
Final Tricks

- Recall we want to identify sinks NOT sources!
- So far it looks like we can identify sources.
- What if we took the reverse of $G \Rightarrow G_{\text{rev}}$
- And ran DFS on $G_{\text{rev}}$ and studied the finishing times
Original Graph G
$G_{\text{rev}}$
$G_{	ext{rev}}$ has the exact same SCCs as $G$!

Why?
Properties of $G_{\text{rev}}$

1. $G_{\text{rev}}$ and $G$ have the same SCCs
2. The $(G_{\text{rev}})^{\text{SCC}}$ is the reverse of $G^{\text{SCC}}$
Running DFS on $G_{rev}$

By Key Lemma we should get:


DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev

The diagram above illustrates a directed graph with nodes labeled from 1 to 10, and edges connecting them. The finishing times are indicated by the numbers inside the nodes.

1. F
2. G
3. C
4. K
5. J
6. I
7. H
8. A
9. D
10. E

The nodes and edges represent the order in which nodes are visited during a Depth-First Search (DFS) algorithm, with finishing times reflecting the order in which nodes are completed.
DFS & Finishing Times on Grev

Graph representation:

- Edges: 
  - A to D
  - A to H
  - B to E
  - C to F
  - D to E
  - D to G
  - E to F
  - E to G
  - H to I
  - H to J
  - I to J
  - I to L
  - J to K
  - K to L

Node colors:
- Red: A, D, H, I, J, K, L
- Blue: B, C, E, F, G

DFS traversal:
- Start at A
- Visit D, H, I, J, K, L
- Finish at L

Finished times:
- A: 8
- B: 9
- C: 10
- D: 6
- E: 2
- F: 1
- G: 3
- H: 7
- I: 5
- J: 4
- K: 4
- L: 5

DFS order:
A, D, H, I, J, K, L, B, C, E, F, G

Finishing times:
A: 8, B: 9, C: 10, D: 6, E: 2, F: 1, G: 3, H: 7, I: 5, J: 4, K: 4, L: 5
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev

Graph representation of DFS and finishing times on a graph. The graph shows nodes labeled A, B, C, D, E, F, G, H, I, J, K, L, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, and 11 with directed edges connecting them.
DFS & Finishing Times on Grev
DFS & Finishing Times on Grev

SCCs by decreasing finishing times

SCC₁

SCC₂

SCC₃

SCC₄
Finally: Kosaraju’s Algorithm

procedure kosarajusSCC(DAG G):
  1. run DFS on G\text{rev} and keep finishing times f
  2. run DFS/BFS in G:
     pick start vertices by decreasing f times
     propagate the ID of each new start vertex
     during traversal

Again, the labels will implicitly identify the SCCs!
Kosaraju’s Algorithm Simulation

◆ First Step: DFS on $G_{rev}$
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G

![Diagram of Kosaraju’s Algorithm Simulation]
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

- Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G

Diagram of the Kosaraju’s algorithm simulation with vertices labeled from 1 to 12 and edges connecting the vertices in a certain order.
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G

Diagram of a graph with nodes labeled A, B, C, D, E, F, G, H, I, J, K, and L. The nodes are colored to indicate the order of visiting during DFS.
Kosaraju’s Algorithm Simulation

- Second Step: DFS on G

Diagram of the Kosaraju’s Algorithm simulation with nodes L, K, J, I, A, D, H, E, F, C, B, G labeled and directed edges between them, illustrating the depth-first search process.
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G

Diagram of the Kosaraju’s Algorithm Simulation, showing the vertices and edges of the graph.
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

 önemli Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G

![Graph diagram showing the Kosaraju’s Algorithm simulation with vertices labeled 1 to 12 and edges connecting them. The second step of the algorithm is applied on the graph G.]
Kosaraju’s Algorithm Simulation

- Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

- Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Simulation

◆ Second Step: DFS on G
Kosaraju’s Algorithm Analysis

- **Run-time:** $O(n + m)!$
- **Correctness Intuition:**

  Recall Key Lemma: $SCC_i \rightarrow SCC_j$, then
  \[***f[SCC_i] > f[SCC_j]***\]

  Corollary: In $G_{rev}$, the highest $f[v]$ will be in a source SCC => In $G$ $v$ will be in a sink SCC
  ⇒ A BFS/DFS from $v$ will only identify $v$’s SCC
  ⇒ Once $v$’s SCC is “peeled-off” next highest $f[v]$ also has to be a sink by induction
  ⇒ A BFS/DFS from $v$ will only identify $v$’s SCC
  ⇒ Inductively finding all SCCs
Proof of Key Lemma:

In $G^{SCC}$ if $SCC_i \rightarrow SCC_j$, then $f[SCC_i] > f[SCC_j]$

Let $u$ be the first vertex visited in $SCC_i$ or $SCC_j$

Again, break into 2 cases by whether $u$ is in $SCC_i$ or $SCC_j$

Case 1: $u$ is in $SCC_i$. Then:

by extended key lemma 1 (EKL1) of Topological Sort

$f[u]$ will be greater than any vertex in $SCC_j$

Case 2: $u$ is in $SCC_j$. Then by (EKL1):

$f[u] > f[y]$ for any vertex in $SCC_j$

$f[u] < f[x]$ for any vertex in $SCC_i$

$=> f[SCC_i] > f[SCC_j]$