CSCI3160 Design and Analysis of Algorithms (2025 Fall)

White Path Theorem and Strongly Connected Components

Instructor: Xiao Liang[1]

Department of Computer Science and Engineering
The Chinese University of Hong Kong

[1]These slides are primarily based on materials prepared by Prof. Yufei Tao (please refer to Prof. Tao's version from 2024 Fall for the original content). Some modifications have been made to better align with this year's teaching progress, incorporating student feedback, in-class interactions, and my own teaching style and research perspective.

DFS Algorithm and White Path Theorem

Recalling DFS (1/2)

Algorithm description:

- Let $G = (V, E)$ be a directed simple[2] graph.
- In the beginning, color all vertices in the graph **white**.
- Create an empty tree T. /* this will be called a "DFS tree" */
- Create a stack S, and then:
  - Pick an arbitrary vertex v
  - Push v into S, and color it **gray** /* gray means "in the stack" */
  - Make v the root of T

(to be continued ...)

[2] Here, "simple" means no self-loops — i.e., no edge from a vertex to itself.

Recalling DFS (2/2)

Repeat the following until S is empty.

1. Let v be the vertex that currently tops the stack S /* do not remove v from S yet */
2. Does v still have a white out-neighbor?

   2.1 If **YES**: let it be u.
   - Push u into S and color u **gray** /* gray means "in the stack" */
   - Make u a child of v in the DFS-tree T.

   2.2 If **NO**: pop v from S and color v **black** /* black means "this node is done" */

If there are still white vertices, repeat the above by restarting from an arbitrary white vertex v', creating a new DFS-tree rooted at v'.

(end of description)

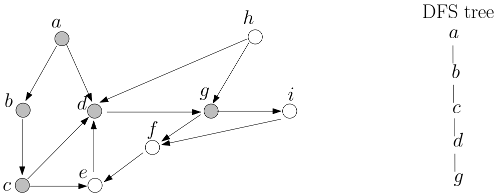
White Path Theorem

Theorem (White Path Theorem)

Let u be a vertex in G. Consider the moment when u enters the stack. Then, a vertex v will become a proper descendant of u in the DFS-forest **if and only** *if at the current moment we can go from u to v by traveling* **on white vertices only** *(i.e., there's a white path from u to v).*

Example: middle of execution

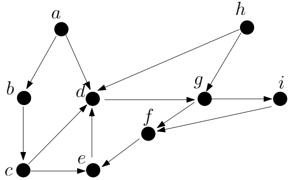
Consider the moment in our previous example when g just entered the stack. $S = (a, b, c, d, g)$.



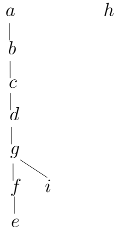
We can see that g can reach f, e, and i by hopping on only white vertices. Therefore, f, e, and i are proper descendants of g in the DFS-forest; and g has no other descendants.

Example: end of execution

The end.



DFS forest



$S = ()$.

Strongly Connected Components

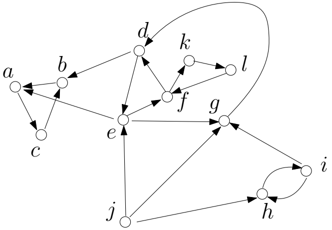
Strongly Connected Component

Let $G = (V, E)$ be a directed graph.

A **strongly connected component** (SCC) of G is a subset S of V that satisfies the following two properties:

1. for any two vertices $u, v \in S$, graph G has a path from u to v and a path from v to u;
2. S is maximal in the sense that we cannot put any more vertex into S without breaking the above property.

Example



- $\{a, b, c\}$ is an SCC.
- $\{a, b, c, d\}$ is not an SCC.
- $\{d, e, f, k, l\}$ is not an SCC (because we can still add vertex g).
- $\{e, d, f, k, l, g\}$ is an SCC.

Property of SCCs

SCCs are Disjoint

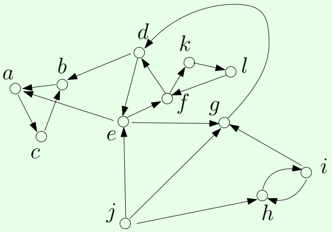
Lemma 1: Suppose that S_1 and S_2 are both SCCs of G. Then, $S_1 \cap S_2 = \emptyset$.

The proof is easy and left to you.

The SCC Problem

Given a directed graph $G = (V, E)$, the goal of the **strongly connected components problem** is to divide V into disjoint subsets, each being an SCC.

Example:



We should output: $\{a, b, c\}$, $\{d, e, f, g, k, l\}$, $\{h, i\}$, and $\{j\}$.

DFS-based Algorithm for SCC

We will introduce a DFS-based algorithm to solve the SCC problem.

At a high level, this algorithm operates in 3 stages, invoking the DFS algorithm twice: once on the original graph G, and once on the so-called **reverse graph** G^{rev}.

An example of reverse graph:

(a) Original graph G (b) Reverse Graph G^{rev}

Algorithm Description (1/5)

Algorithm

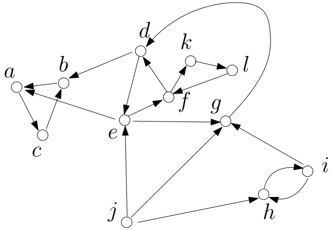
Step 1: Run DFS on G, and list the vertices by the order they turn black (i.e., popped from the stack).

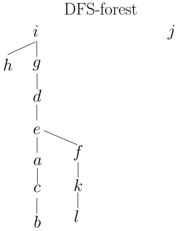
- If vertex $u \in V$ is the i-th turning black, we **label** u with i.

(to be continued ...)

Algorithm Description (2/5)

Example





Start DFS from i and re-start from j.
The following is a possible turn-black order: $h, b, c, a, l, k, f, e, d, g, i, j$. E.g.:

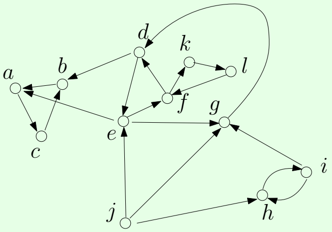
- The label of c is 3.
- The label of g is 10.
- The label of i is 11.
- The label of j is 12.

Algorithm Description (3/5)

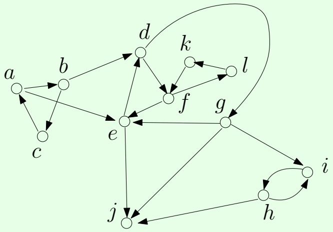
Algorithm

Step 2: Obtain the reverse graph G^{rev} by reversing the directions of all the edges in G.

Example:



Input graph



Reverse graph

Algorithm Description (4/5)

Algorithm

Step 3: Perform DFS on G^{rev} subject to the following rules:

- **Rule 1:** Start at the vertex with the largest label.
- **Rule 2:** When a restart is needed, do so from the white vertex with the largest label.

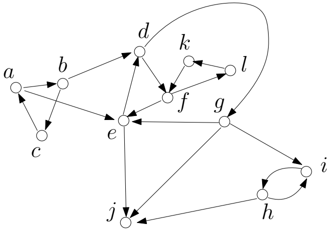
Output the set of vertices in each DFS-tree as an SCC.

(end of description)

Algorithm Description (5/5)

Example

Vertices in ascending order of label: *h*, *b*, *c*, *a*, *l*, *k*, *f*, *e*, *d*, *g*, *i*, *j*.
Reverse graph G^{rev}:



Start DFS from *j*, which finishes immediately and discovers only *j*.

- First SCC: $\{j\}$

Restart from *i*, which finishes after discovering *i* and *h*

- Second SCC: $\{i, h\}$

Restart from *g*, which finishes after discovering *g*, *e*, *d*, *f*, *l*, and *k*

Summary of the Algorithm

Step 1: Run DFS on G, and list the vertices by the order they turn black (i.e., popped from the stack).

Step 2: Obtain the reverse graph G^{rev} by reversing the directions of all the edges in G.

Step 3: Perform DFS on G^{rev} subject to the following rules:

- **Rule 1:** Start at the vertex with the largest label.
- **Rule 2:** When a restart is needed, do so from the white vertex with the largest label.

Time Complexity

Theorem: Our SCC algorithm finishes in $O(|V| + |E|)$ time.

The proof is left as a regular exercise.

Next, we will prove that the algorithm correctly returns all the SCCs.

Proof of Correctness

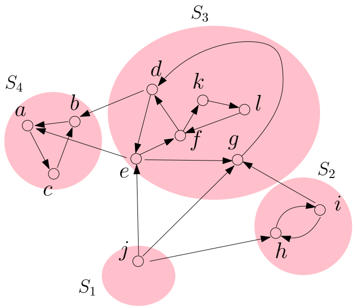
SCC Graph

Suppose that the input graph G has SCCs $S_1, S_2, ..., S_t$ for some $t \geq 1$.

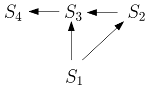
The **SCC graph** G^{scc} is defined as follows:

- Each vertex in G^{scc} is a distinct SCC in G.
- For every two distinct vertices (a.k.a. SCCs) S_i and S_j ($1 \leq i, j \leq t$), G^{scc} has an edge from S_i to S_j if some vertex of S_i has an edge in G to a vertex of S_j.

Example



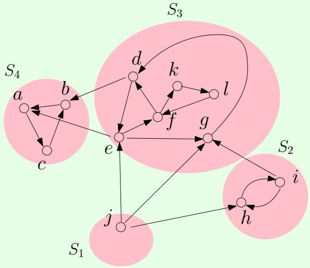
SCC Graph

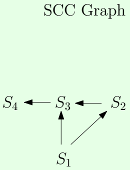


SCC Graph

For each SCC S_i ($i \in [1, t]$), define

$$\textbf{label}(S_i) = \max_{v \in S_i} \text{label of } v$$





Vertices in ascending order of label: $h, b, c, a, l, k, f, e, d, g, i, j$.
$label(S_1) = 12$, $label(S_2) = 11$, $label(S_3) = 10$, $label(S_4) = 4$

Lemma of SCC Ordering

Lemma 2: If SCC S_i (for some $i \in [1, t]$) has an edge to SCC S_j (for some $j \in [1, t]$) in G^{scc}, then $label(S_i) > label(S_j)$.

The proof is straightforward. Try to work it out on your own first, then check your solution against the proof provided on the next slide.

Proof of Lemma 2

Proof: Let u be the first vertex in $S_i \cup S_j$ that turns gray in DFS (i.e., u is the first vertex in $S_i \cup S_j$ discovered by DFS).

- If $u \in S_i$, u has a white path to every vertex in $S_i \cup S_j$. By the white path theorem:
  - u turns black after all the vertices in S_j; /* since S_i has an edge to S_j */
  - u is the last vertex in S_i turning black.

  The above facts imply $label(S_i) > label(S_j)$.
- If $u \in S_j$, first note that:
  - u has a path to every vertex in S_j, but no path to any vertex in S_i. /* since S_i has an edge to S_j */

  By the white path theorem, u turns black after all the vertices in S_j and before every vertex in S_i. This again implies $label(S_i) > label(S_j)$.

$\square$

It follows from **Lemma 2** that without loss of generality, we can always choose the name for SCCs so that $S_1, S_2, ..., S_t$ satisfying the following condition:

$$label(S_1) > label(S_2) > ... > label(S_t).$$

Using the ordering, we have the following corollary:

Corollary 3: Fix any $i \in [1, t]$. Consider any vertex $u \in S_i$. In G^{rev} (i.e., the reverse graph), if (v, u) is an incoming edge of u and yet $v \notin S_i$, then v belongs to some S_j with $j > i$.

Proof: As (v, u) is in G^{rev}, G has an edge from u to v. Hence, S_i has an edge to S_j in G^{scc}.

By Lemma 2, $label(S_i) > label(S_j)$, which means $j > i$. □

Lemma 4: Consider the DFS on G^{rev} (in Step 3 of our algorithm). For each $i \in [1, t]$, S_i is exactly the set of vertices in the i-th DFS-tree produced.

Before we proceed to the proof of lemma 4, Convince yourself that lemma 4 implies the correct of our algorithm.

Proving Lemma 4 (1/2)

Proof: We will prove the claim by induction on i.

Case $i = 1$:

Let u be the vertex in S_1 having the largest label; u is the root of the first DFS-tree. Consider the beginning moment of the first DFS on G^{rev}.

- As S_1 is an SCC, u has a white path to every other vertex in S_1.
- By Corollary 3, u has no path (let alone a white path) to any vertex outside S_1. /* Think how to prove this. Hint: think about the "crossing edge." */

By the white path theorem, all and only the vertices in S_1 are descendants of u in the first DFS tree. The claim thus holds for $i = 1$.

Proving Lemma 4 (2/2)

Case $i = k$:

Assuming that the claim holds for $i = k - 1$ (where $k \geq 2$), next we prove its correctness for $i = k$.

Let u be the vertex in S_k having the largest label; u is the root of the k-th DFS-tree. Consider the beginning moment of the k-th DFS on G^{rev}.

- All the vertices in $S_1, S_2, ..., S_{k-1}$ are black.
- As S_k is an SCC, u has a white path to every other vertex in S_k.
- By Corollary 3, u has no white path to any vertex in $S_{k+1}, S_{k+2}, ..., S_t$. /* Though u could have paths to vertices in $S_1, S_2, \ldots, S_k$, those path won't be white due to the first bullet */

By the white path theorem, all and only the vertices in S_k are descendants of u in the k-th DFS tree. The claim thus holds for $i = k$. □