Introduction to Algorithms

Mong-Jen Kao (高孟駿)

Tuesday 10:10 – 12:00

Thursday 15:30 – 16:20

Graph Algorithms

– Graph problem pervades computer science, and Algorithms for graphs are fundamental to the field.

Applications of the DFS Algorithm

Let’s examine graph problems
that are solved by DFS-like algorithms.

Example 1.

Topological Sort

Produce a consistent linear ordering of the vertices for a directed acyclic graph (DAG).

Topological Sort

- Let $G = (V, E)$ be a directed acyclic graph (DAG).
  - i.e., G contains no cycle.
- The topological sort problem is to produce a linear ordering $\pi : V \mapsto \{1,2,\ldots,n\}$ of the vertices, where $n = |V|$, such that
  - $\pi_u \neq \pi_v$ for any $u, v \in V$ with $u \neq v$, and
  - for any directed edge $(u, v) \in E$, we have $\pi_u < \pi_v$.
- In other words, produce an ordering of the vertices such that no edge points backwards in the ordering.

Topological Sort

- The topological sort problem can be solved by the DFS algorithm.

- Topological-Sort(G) - $G = (V, E)$ is directed acyclic.

A. Let Q be an empty list.

B. Call DFS(G).
As each vertex is finished in the DFS-Visit call,
insert the vertex in the front of Q.

C. Return Q.
// Q stores the vertices in the descending order of their finish times.

Analysis of the Algorithm

- It is clear that this algorithm runs in $O(|V| + |E|)$ time.
- To prove the correctness,
  let us verify the 4 types of edges in the DFS-forest.
  - By the parenthesis theorem, tree edges *point to vertices with earlier finish times*.
  - By the definition, forward edges and cross edges point to black vertices which are already finished.

These vertices come after in the linear ordering.

Analysis of the Algorithm

- It is clear that this algorithm runs in $O(|V| + |E|)$ time.
- To prove the correctness,
  let us verify the 4 types of edges in the DFS-forest.
  - There is no back edge in the resulting DFS-forest.
    - Any back edge forms at least one cycle in the graph.
- Hence, none of these edges point backward in the ordering produced by the algorithm.

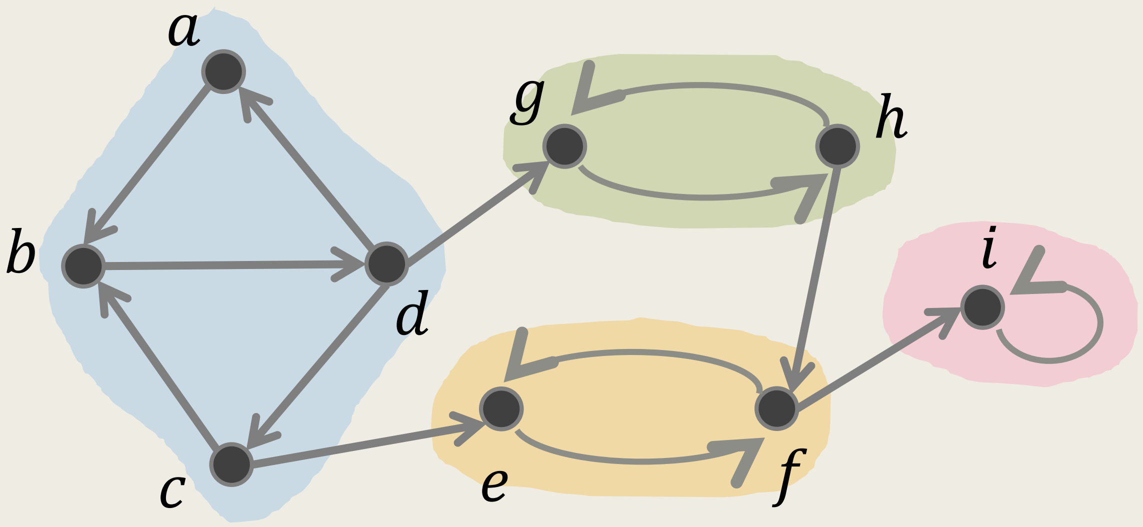
Example 2.

Strongly Connected Components

Partition the vertices into maximal components such that
every vertex pair within one component is *reachable from both directions*.

Strongly Connected Components

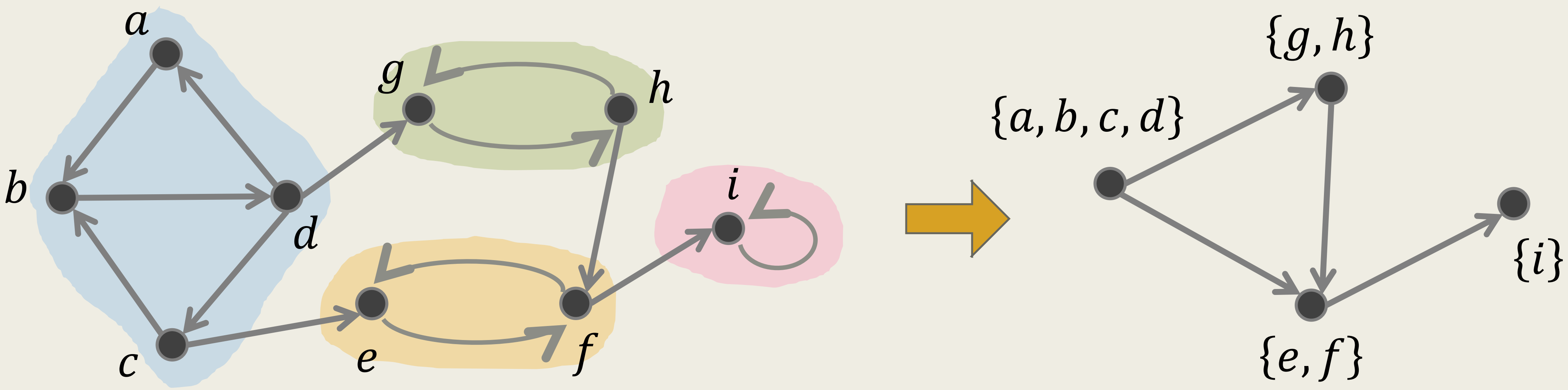
- Let $G = (V, E)$ be a directed graph (digraph).
  - A *<u>strongly connected component</u>* (SCC) is a ***maximal vertex subset*** $C \subseteq V$ such that for any $u, v \in C$, vertices u and v are *<u>reachable from each other</u>*.



Note that, $\{a, b, d\}$ is not an SCC since it is not maximal in size.

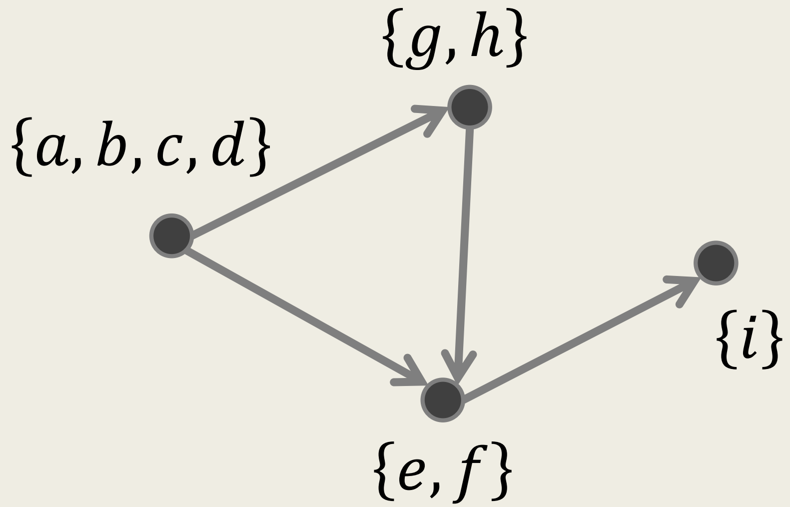
Strongly Connected Components

- Let $G = (V, E)$ be a directed graph (digraph).
  - Let $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ be a partition of V into SCCs.
  - Consider the set E_C of edges in E that connects components in $\mathcal{C}$.
  - Then, $G_C = (\mathcal{C}, E_C)$ is ***acyclic***.



Strongly Connected Components

- Define $E^T \coloneqq \{ (v,u) : (u,v) \in E\}$ be the edges in E with their directions reversed.
  - Define $G^T = \left(V, E^T\right)$ to be the transpose of G.
- Then, $C \subseteq V$ is an SCC for G if and only if C is an SCC for G^T.
- Furthermore, if $C, C' \subseteq V$ are SCCs such that C' is reachable from C in G_C, then C' is not reachable from C in G_C^T.



Strongly Connected Components

- With the information computed by the DFS algorithm, we can compute the set of SCCs in $O(|V| + |E|)$ time.

- Strongly-Connected-Components(G) - $G = (V, E)$ is directed.

A. Call DFS(G), and use a queue to maintain the vertices in decreasing order of their finish times.

B. Call DFS$\left(G^T\right)$, but in the main loop of the algorithm, consider the vertices in decreasing order of their finish times.

C. Report each tree in the DFS forest created by DFS$\left(G^T\right)$ as an SCC.

Example 3.

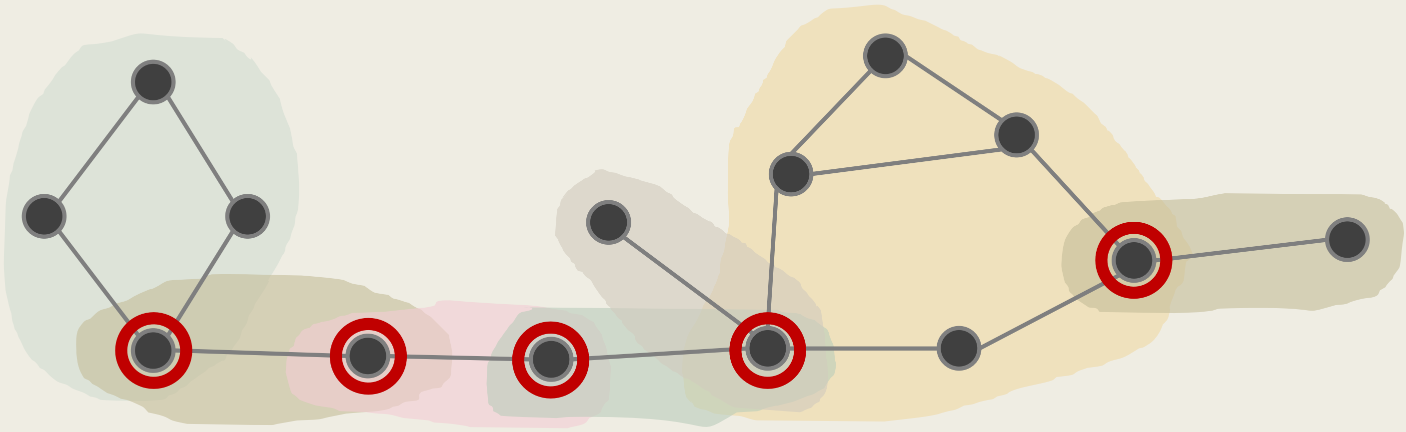
Bi-connected Components

and Articulation Points

Computing the cut vertices and
2-vertex connected components for an undirected graph.

Articulation Points / Cut Vertices

- Let $G = (V, E)$ be a connected undirected graph.
  - A vertex $v \in V$ is a ***cut vertex*** / ***articulation point*** for G if $G - \{v\}$ is disconnected.
  - A ***biconnected component*** is a *maximal vertex subset* $B \subseteq V$ that induces a connected subgraph ***with no cut vertex***.



Articulation Points / Cut Vertices

- We can use the DFS algorithm to compute the set of bi-connected components and also the set of articulation points in $O(|V| + |E|)$ time.

- Consider the DFS tree G_π computed by the DFS algorithm.
  - We have the following properties.

Lemma 1.

Let r be the root of G_π. Then r is an articulation point ***if and only if*** it has at least two children nodes in G_π.

Articulation Points / Cut Vertices

- Consider the DFS tree G_π computed by the DFS algorithm.
  - We have the following properties.

Lemma 2.

Let $v \in G_\pi$ be a non-root vertex.

Then v is an articulation point ***if and only if*** it has a child s such that there is ***no back edge*** from s or any descendant of s to a proper ancestor of v.

- We can use the DFS algorithm to compute the set of bi-connected components and also the set of articulation points in $O(|V| + |E|)$ time.

- Get-Articulation-Points(v, ℓ) - $v \in V$ the current vertex with depth ℓ.

A. // Initializations

Set $\text{visited}[v] \leftarrow \text{true}$,

$\text{depth}[v] \leftarrow \ell$,

$\text{low}[v] \leftarrow \ell$, // Highest depth reachable from any descendant of v <u>*via back edges*</u>.

$\text{childCount} \leftarrow 0$, and

$\text{isCutVertex} \leftarrow \text{false}$.

- Get-Articulation-Points(v, ℓ) - $v \in V$ the current vertex with depth ℓ.

A. // Initializations

B. For each $u \in N(v)$, do the following.

   - If $\text{visited}[u] = \text{false}$ then

     1. Set $\pi[u] \leftarrow v$ and $\text{childCount} \leftarrow \text{childCount} + 1$.
     2. Get-Articulation-Points$(u, \ell + 1)$.
     3. If $\text{low}[u] \geq \text{depth}[v]$, // by Lemma 2
        then set $\text{isCutVertex} \leftarrow \text{true}$.
     4. Set $\text{low}[v] \leftarrow \min(\text{low}[v], \text{low}[u])$.

     else if $u \neq \pi[v]$, then // (v, u) is a back edge

     1. Set $\text{low}[v] \leftarrow \min(\text{low}[v], \text{depth}[u])$.

- Get-Articulation-Points(v, ℓ) - $v \in V$ the current vertex with depth ℓ.

A. // Initializations

B. For each $u \in N(v)$, do the following.

   – …

C. // by Lemma 1 and Lemma 2

   If ($\pi_v \neq \mathrm{NIL}$ and $\mathrm{isCutVertex} = \mathrm{true}$) or

   ($\pi_v = \mathrm{NIL}$ and $\mathrm{childCount} > 1$) then

   – Output v as an articulation point.

With an extra stack, this algorithm can be modified
to output all biconnected components in $O(|V| + |E|)$ time as well.