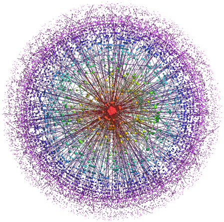

THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

Algorithms and Data Structures

Lecture 14 Connectivity and Components

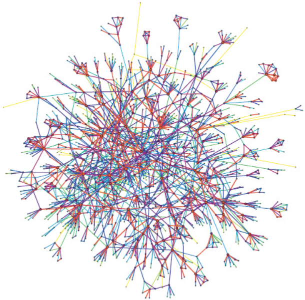
Jiamou Liu
The University of Auckland



Decomposing Graphs

Why decompose graphs?

[Divide-and-Conquer] Often, when we solve a problem on graph, it is much more efficient to decompose the graph into components, solve the problem on individual components, then combine the solutions.

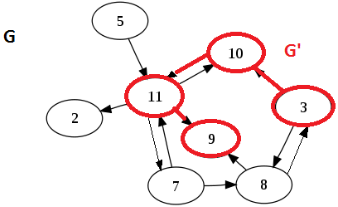


Subgraphs

Definition [Subgraphs]

Let $E \subseteq V^2$, and $V' \subseteq V$.

- we use $E \restriction V'$ to denote the set $\{(u,v) \in E \mid u,v \in V'\}$.
- A subgraph of a digraph $G = (V,E)$ is a digraph $G' = (V',E')$ where $V' \subseteq V$ and $E' \subseteq E \restriction V'$.
- If $E' = E \restriction V'$, then G' is an induced subgraph of G.

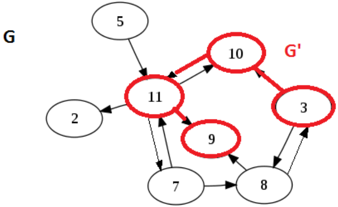


Subgraphs

Definition [Subgraphs]

Let $E \subseteq V^2$, and $V' \subseteq V$.

- we use $E \restriction V'$ to denote the set $\{(u,v) \in E \mid u,v \in V'\}$.
- A subgraph of a digraph $G = (V,E)$ is a digraph $G' = (V',E')$ where $V' \subseteq V$ and $E' \subseteq E \restriction V'$.
- If $E' = E \restriction V'$, then G' is an induced subgraph of G.



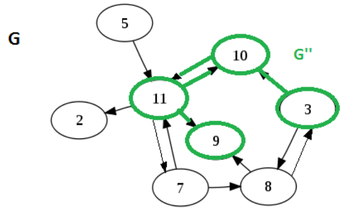
Let $V' = \{3,9,10,11\}$. $E \restriction V' = \{(10,11),(11,10),(3,10),(11,9)\}$

A subgraph is $(\{3,9,10,11\},\{(3,10),(10,11),(11,9)\})$

Definition [Subgraphs]

Let $E \subseteq V^2$, and $V' \subseteq V$.

- we use $E \upharpoonright V'$ to denote the set $\{(u, v) \in E \mid u, v \in V'\}$.
- A subgraph of a digraph $G = (V, E)$ is a digraph $G' = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E \upharpoonright V'$.
- If $E' = E \upharpoonright V'$, then G' is an induced subgraph of G.



Let $V' = \{3, 9, 10, 11\}$. $E \upharpoonright V' = \{(10, 11), (11, 10), (3, 10), (11, 9)\}$

An induced subgraph is $(\{3, 9, 10, 11\}, \{(3, 10), (10, 11), (11, 9), (11, 10)\})$

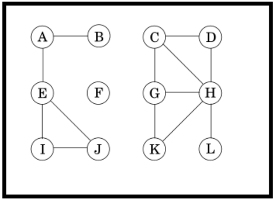
Decomposing Undirected Graphs

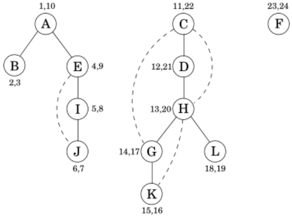
Connectivity in Undirected Graphs

- Recall a node is reachable from another if there is a path linking these two nodes
- Here reachability is an equivalence relation:
  - (reflexivity) Any node u is reachable from itself.
  - (symmetry) If v is reachable from u then u is reachable from v.
  - (transitivity) If v is reachable from u, u is reachable from w, then v is reachable from w.
- We may decompose the graph into equivalence classes: Two nodes are in the same class if they are reachable from each other
- Each equivalence class is a connected component

Definition [Undirected Connectivity]

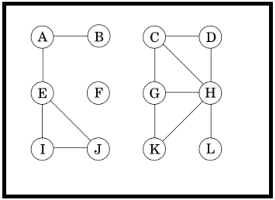
A connected components is the induced subgraph of a maximal set of nodes that are pairwise reachable.
An undirected graph is connected if it contains only one connected component.

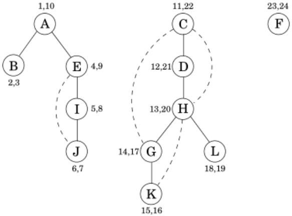




Definition [Undirected Connectivity]

A connected components is the induced subgraph of a maximal set of nodes that are pairwise reachable.
An undirected graph is connected if it contains only one connected component.





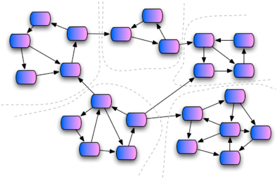
DFS and CC

- We may use DFS to decide if two nodes are in the same CC.
- Running time: $O(m + n)$.

Decomposing Directed Graph

Definition [Directed Connectivity]

In a digraph G, we say that two nodes u, v are in the same strongly connected component (SCC) if there is a path from u to v and a path from v to u. A digraph is strongly connected if it contains only one SCC.



Extreme special cases:

- If G is acyclic, then every node is itself a SCC.
  Therefore there are n SCCs in G.

Extreme special cases:

- If G is acyclic, then every node is itself a SCC.
  Therefore there are n SCCs in G.
- If G is a cycle, then G is a SCC.
  Therefore there is only 1 SCCs in G.

Extreme special cases:

- If G is acyclic, then every node is itself a SCC.
  Therefore there are n SCCs in G.
- If G is a cycle, then G is a SCC.
  Therefore there is only 1 SCCs in G.
- If G is undirected, then u, v are in the same SCC whenever u can reach v.
  Therefore checking SCC is same as reachability.

Extreme special cases:

- If G is acyclic, then every node is itself a SCC.
  Therefore there are n SCCs in G.
- If G is a cycle, then G is a SCC.
  Therefore there is only 1 SCCs in G.
- If G is undirected, then u, v are in the same SCC whenever u can reach v.
  Therefore checking SCC is same as reachability.

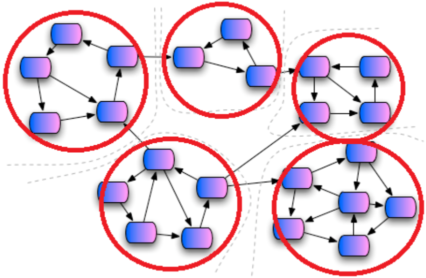
SCC Problem.

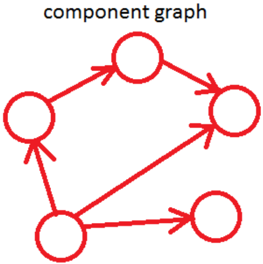
- INPUT: A digraph G,
- OUTPUT: All the strongly connected components of G.

Meta-Graph

Definition [Meta-Graph]

Given a graph G. If we collapse all nodes in the same SCCs together, only keeping the edges between different components, then we get the meta-graph, G^{SCC}.





Source and Sink

Meta-Graph

Let G be a digraph, and G^{SCC} be its meta-graph after collapsing every SCC into one node.

- The meta-graph G^{SCC} must be acyclic.

  Why?

Source and Sink

Meta-Graph

Let G be a digraph, and G^{SCC} be its meta-graph after collapsing every SCC into one node.

- The meta-graph G^{SCC} must be acyclic.

  Why?

- We can linearise G^{SCC}.

Source and Sink

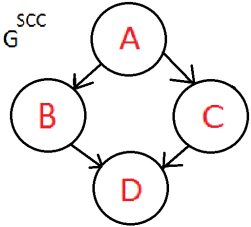
Meta-Graph

Let G be a digraph, and G^{SCC} be its meta-graph after collapsing every SCC into one node.

- The meta-graph G^{SCC} must be acyclic.

  Why?
- We can linearise G^{SCC}.
- Source: A node in G^{SCC} with no incoming edge.
- Sink: A node in G^{SCC} with no outgoing edge.

Consider the following meta-graph:

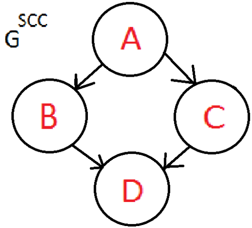


A is a source, *D* is a sink.

Observation

If we run DFS on a node in a sink, then we will find all nodes in this sink.

Consider the following meta-graph:



A Plan for Finding SCC

Given G. Repeat the following:

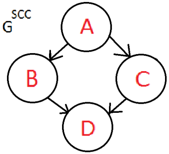
1. Find a node u in a sink
2. Run dfs_explore(G, u)
3. Declare all visited nodes an SCC. Take those nodes out.

Question. How do we find a node in a sink?

Observe:

- We are given the graph G, but no information about G^{SCC}.
- We can find a node in a source:

  Run DFS. Take the node that is finished last.



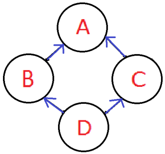
- But running DFS on a source does not work.

Question. Does finding a source node help in finding a sink node?

Question. Does finding a source node help in finding a sink node?

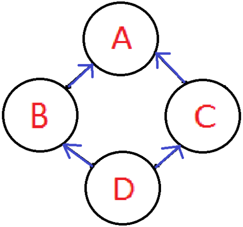
Fact

Let G be a digraph. Let G^T be the transpose of G: the digraph obtained from G by reversing the direction of every edge.



- G and G^T have the same SCCs.
- A source in G becomes a sink in G^T.

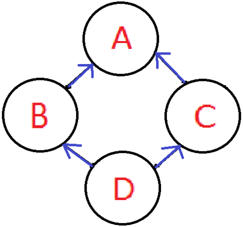
DFS and Strongly Connected Components



Example. When the edges are reversed, A, B, C, D are still SCCs.

- Let x be the last finished node in DFS.
  Running dfs_explore(G^T, x) will compute A.
- Let y be the last node that is finished in the remaining graph.
- Continue for B, D (decreasing order of *post*)

DFS and Strongly Connected Components



Example. When the edges are reversed, A, B, C, D are still SCCs.

- Let x be the last finished node in DFS.

  Running dfs_explore(G^T, x) will compute A.
- Let y be the last node that is finished in the remaining graph.

  Running dfs_explore(G^T, y) will compute C.
- Continue for B, D (decreasing order of *post*)

The following algorithm takes as input any digraph G, outputs all the SCCs of G.

Algorithm SCC(G)

INPUT: a digraph G
OUTPUT: SCCs of G
stack $\leftarrow$ empty stack
Run $dfs(G)$, at the same time do:
    When a node is finished, push it onto a *stack*
$G^T \leftarrow G$ with all edges reversed
for each u in *stack* (in popped order)
    Run dfs_explore(G^T, u)
    The nodes visited by **explore** is the SCC of u.

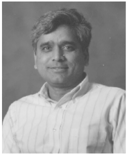
Running time: $O(m + n)$.

Discussion

- Essentially, the algorithm runs DFS twice: first time on G, then on G^T.
  In the second time, when no where to go, select the next node in decreasing order of finishing time of the first DFS.

Discussion

- Essentially, the algorithm runs DFS twice: first time on G, then on G^T.
  In the second time, when no where to go, select the next node in decreasing order of finishing time of the first DFS.
- The algorithm is called the Kosaraju-Sharir algorithm



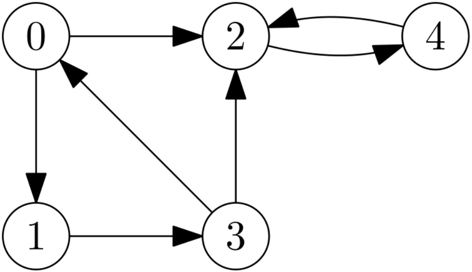
"At some point, the learning stops and the pain begins."

------- S. Rao Kosaraju

Summary

- Directed acyclic graph (DAG)
- Acyclicity problem: DFS-based algorithm (no back edge)
- Linearisation problem:
  - Zero in-degree algorithm: $O(n(m+n))$
  - DFS-based algorithm (decreasing finishing order): $O(m+n)$

Question 1. For the following digraph, perform the algorithm taught above and find all SCCs. Show working.




THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND