Strongly Connected Components and Minimum Spanning Trees
CSCI 700 - Algorithms I

Andrew Rosenberg
Last Time

- Depth-First Search
- Breadth-First Search
Today

- Strongly Connected Components
- Minimum Spanning Trees: Prim’s Algorithm
Ordering Of Nodes using DFS

We can track the **discovery time** $d[v]$ and **finish time** $f[v]$ of each node.

- Sorting by discovery time gives a **preorder traversal**
- Sorting by finish time gives a **postorder traversal**
- Is a node $u$ is a DFS ancestor of a node $v$?
  - $d[u] < d[v] < f[v] < f[u]$
- Is a node $u$ to the left is a node $v$?
  - $d[u] < f[u] < d[v] < f[v]$
Ordering Of Nodes using DFS

Diagram:

- Nodes: a, b, c, d, e, f, g, h, i
- Connections:
  - a to b
  - b to c, e
  - c to d, e
  - e to g, h
  - f to i

Graph structure:

```
  a -- b
  |    |
  c <--- e <--- f
  |    |
  d -- g <--- h -- i
```
A Graph $G$ is **strongly connected** if for all $u, v \in V$, $u$ and $v$ are mutually reachable.

We can decompose any Graph into a set of **Strongly Connected Components** – SCCs.

Every node in a cycle appears in the same SCC.
How can we identify SCCs?
Identifying SCCs

Test reachability from every node to every other node.

Runtime?
Identifying SCCs

Test reachability from every node to every other node.

Runtime?
\[ O(n(n + e)) = O(n^2 + ne) \]
Identifying SCCs

Test reachability from every node to every other node.

Runtime?
\[ O(n(n + e)) = O(n^2 + ne) \]

How can we do better?

- Run a DFS – keep track of \( f[v] \).
- Reverse the Graph.
- Run a DFS on the reversed Graph – traversing in order of **decreasing** \( f[v] \).
- The DFS Tree contains only the strongly connected components with internal edges included.

Runtime?
Identifying SCCs

Test reachability from every node to every other node.

Runtime?
\[ O(n(n + e)) = O(n^2 + ne) \]

How can we do better?

- Run a DFS – keep track of \( f[v] \).
- Reverse the Graph.
- Run a DFS on the reversed Graph – traversing in order of decreasing \( f[v] \).
- The DFS Tree contains only the strongly connected components with internal edges included.

Runtime?
\[ O(3(n + e)) = O(n + e) \]
Identifying SCCs

a (1,19) -> b (2,18) -> c (3,8) -> d (4,5) -> e (6,7) -> f (9,16) -> g (10,13) -> h (11,12) -> i (14,15)
Identifying SCCs

Reverse the edges
Identifying SCCs

a (1,19) → b (2,18) → c (3,8) → d (4,5) → e (6,7)

f (9,16) → g (10,13) → h (11,12) → i (14,15)
Identifying SCCs
Identifying SCCs

Diagram showing the relationships between nodes labeled a, b, c, d, e, f, g, h, and i, each with associated numbers in parentheses.
Identifying SCCs

a (1,19) → b (2,18) → c (3,8) → d (4,5) → e (6,7) → f (9,16) → i (14,15) → g (10,13) → h (11,12)
Identifying SCCs

Diagram showing the structure of SCCs with nodes labeled as follows:
- a (1,19)
- b (2,18)
- c (3,8)
- d (4,5)
- e (6,7)
- f (9,16)
- g (10,13)
- h (11,12)
- i (14,15)
Identifying SCCs
Identifying SCCs
Identifying SCCs

a (1,19)
b (2,18)
c (3,8)
d (4,5)
e (6,7)
f (9,16)
g (10,13)
h (11,12)
i (14,15)
Identifying SCCs

Diagram showing nodes labeled with coordinates: a(1,19), b(2,18), c(3,8), d(4,5), e(6,7), f(9,16), g(10,13), h(11,12), i(14,15). Arrows indicate the direction of edges between the nodes.
Identifying SCCs

The diagram shows a directed graph with the following nodes and edges:

- **a (1,19)**
- **b (2,18)**
- **c (3,8)**
- **d (4,5)**
- **e (6,7)**
- **f (9,16)**
- **g (10,13)**
- **h (11,12)**
- **i (14,15)**

The edges are directed as follows:

- **a** → **b**
- **b** → **c**
- **c** → **d**
- **d** → **e**
- **e** → **g**
- **g** → **h**
- **h** → **i**
- **i** → **b**
- **f** → **g**

This diagram represents a strongly connected component graph.
Identifying SCCs
Prove that the SCC procedure identifies all strongly connected components. Must show:

- If $u$ and $v$ are mutually reachable then they are in the same DFS tree generated by the second DFS.
- If $u$ and $v$ are in the same DFS tree, then they are mutually reachable.
1. If $u$ and $v$ are mutually reachable in $G$, then they are mutually reachable in $G_R$.

If they are mutually reachable in $G_R$ then they will appear in the same SCC.

Case 1) The DFS finds $u$ first. Since $u$ is reachable from $v$ in $G$, then $v$ is reachable from $u$ in $G_R$. If $v$ is reachable from $u$ in $G_R$, the DFS subtree from $u$ will include $v$.

Case 2) The reverse. The DFS finds $v$ first. Since $v$ is reachable from $u$ in $G$, then $u$ is reachable from $v$ in $G_R$. If $u$ is reachable from $v$ in $G_R$, the DFS subtree from $v$ will include $u$. 
2. If $u$ and $v$ are in the same DFS tree in $G_R$, then they are mutually reachable.

- Let $x$ be the root of the DFS tree containing $u$ and $v$.
- Want to show: $x$ and $u$ are mutually reachable, and $x$ and $v$ are mutually reachable. (Only one is necessary)
- $f[v] < f[x]$, otherwise we would have started the DFS with $v$.
- Since $x$ can reach $v$ in the reverse graph $G_R$, $v$ can reach $x$ in the original graph $G$.

- Partition the first DFS into four regions when the active node is $x$.
  - Black nodes – completed.
  - Grey nodes – ancestors of $x$.
  - White nodes reachable from $x$.
  - White nodes not reachable from $x$. 
SCC Proof

- $f[\text{greynodes}] > f[x]$
- $f[\text{nodesnotreachablefrom}x] > f[x]$.

- Recall $f[v] < f[x]$, thus, $v$ is either black or in reachable from $x$.
- $v$ cannot be black, because $v$ can reach $x$ in $G$.
- Thus, $v$ must be below $x$, therefore reachable from $x$ in $G$.
- Since $x$ and $v$ are mutually reachable in $G$, they are mutually reachable in $G_R$.

- The same logic holds for $u$.
- Thus, $u$ and $v$ are mutually reachable in $G_R$ if they are in the same DFS Tree.
Definition of a Spanning Tree

A **spanning tree** $T$ of a graph $G$ is a subgraph that contains

- Every vertex $v \in V$
- Edges $e \in E$ such that $T$ is a tree – acyclic.
Example of Spanning Trees - DFS
Example of Spanning Trees - BFS
Example of Spanning Trees - Other
Not a Spanning Tree
Not a Spanning Tree
If the edges $e \in E$, have weights, a **minimum spanning tree** is a spanning tree with minimal cost.
If the edges $e \in E$, have weights, a **minimum spanning tree** is a spanning tree with minimal cost.
If the edges \( e \in E \), have weights, a **minimum spanning tree** is a spanning tree with minimal cost.
Minimum Spanning Tree Algorithm

Proposition: Start at any node. Include the lowest cost “fringe” edge.
Claim: This greedy choice works.

Prim’s Algorithm
Example of Prim’s Algorithm
Example of Prim’s Algorithm
Example of Prim’s Algorithm
Example of Prim’s Algorithm
Example of Prim’s Algorithm
Example of Prim’s Algorithm
Example of Prim’s Algorithm

[Diagram of a network graph with labeled nodes and edges, showing the process of Prim’s algorithm for finding a minimum spanning tree.]
Example of Prim’s Algorithm
Example of Prim’s Algorithm
MST-Prim(G,w,r)

for $u \in V[G]$ do
    $key[u] = \infty$; $\pi[u] \leftarrow \emptyset$
end for

$key[r] \leftarrow 0$

$Q \leftarrow V[G]$

while $Q \neq \emptyset$ do
    $u \leftarrow \text{ExtractMin}(Q)$
    for $v \in \text{Adjacent}(u)$ do
        if $v \in Q$ and $w(u,v) < key[v]$ then
            $key[v] = w(u,v)$; $\pi[u] \leftarrow u$
        end if
    end for
end while
Greedy Strategy for a Generic MST growth.

Manage a set of edges $A$, such that prior to each iteration of the algorithm, $A$ is the subset of a minimum spanning tree.

- At each step, identify an edge $(u, v)$ that can be added to $A$ without violating the invariant.
- We call $(u, v)$ a **safe edge**.

How do we identify **safe edges**?
Identifying safe edges

Safe edges

Let $G$ be a connected, undirected graph with weight functions $w$ defined on $E$. Let $A$ be a subset of $E$ that is included in a MST. Let $(S, V − S)$ be any cut of $G$ that respects $A$. Let $(u, v)$ be a light edge crossing $(S, V − S)$. Then $(u, v)$ is safe for $A$.

- A cut $(S, V − S)$ of an undirected graph $G$, is a partition of $V$.
- An edge crosses a cut if one of the end points are in $S$, and the other is in $V$.
- A cut respects a set of edges $A$ if no edges in $A$ cross the cut.
- A edge that crosses the cut is a light edge, if its weight is minimal crossing the cut.
Let $T$ be a MST that includes $A$, but doesn’t include light edge $(u, v)$. (If it does, we’re done.)

We will construct another MST $T'$ that includes $A \cup (u, v)$ using the cut-and-paste technique. Showing that $(u, v)$ is a safe edge for $A$.

Assume the edge $(u, v)$ forms a cycle with the edges on the unique path $p$ from $u$ to $v$ in $T$.

Thus, there is some edge $(x, y)$ on the path $p$ that also crosses the cut.

$(x, y)$ is not in $A$ because the cut respects $A$.

Since, $(x, y)$ is on the unique path removing $(x, y)$ separates $T$ into two components.

Adding $(u, v)$ connects these two components, forming a new spanning tree $T'$. 
Now we need to show that $T'$ is Minimal.

Since $(u, v)$ is a light edge crossing the cut $(S, V - S)$, and $(x, y)$ also crosses the cut, $w(u, v) \leq w(x, y)$.

Thus:

$w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)$

Since $T$ was a minimum spanning tree, so is $T'$.

Since $T'$ is a minimum spanning tree containing $(u, v)$, $(u, v)$ is safe for $A$. 
In Prim’s algorithm, we identify the cut \((S, V - S)\), by starting at one node, and building up \(A\) by including light edges, and expanding \(A\).

Next time:

- Another way to identify safe edges and build up \(A\) – Kruskal’s Algorithm
Next time

- Another Minimum Spanning Tree Algorithm: Kruskal's
- Shortest Paths: Bellman-Ford.