Graphs and trees Advanced graph algorithms

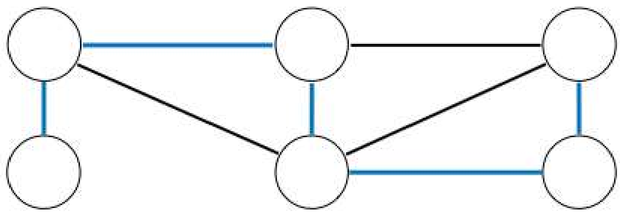
Advanced graph algorithms

By Afshine Amidi and Shervine Amidi

Spanning trees

Definition

A spanning tree of an undirected graph $G = (V, E)$ is defined as a subgraph that has the minimum number of edges $E' \subseteq E$ required for all vertices V to be connected.

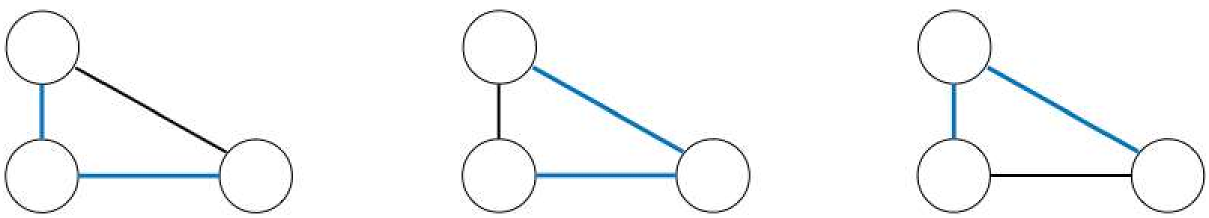


REMARK

In general, a connected graph can have more than one spanning tree.

Cayley's formula

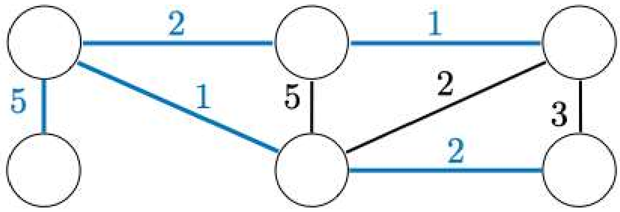
A complete undirected graph with N vertices has N^{N-2} different spanning trees.



For instance, when $N = 3$, there are $3^{3-2} = 3$ different spanning trees:

Minimum spanning tree

A minimum spanning tree (MST) of a weighted connected undirected graph is a spanning tree that minimizes the total edge weight.



For example, the MST shown above has a total edge weight of 11.

REMARK

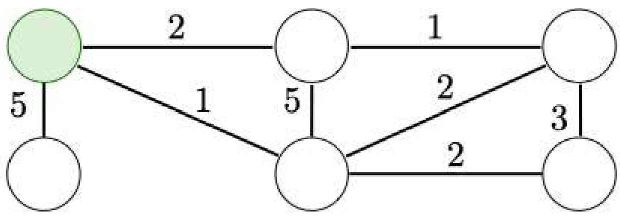
A graph can have more than one MST.

Prim's algorithm

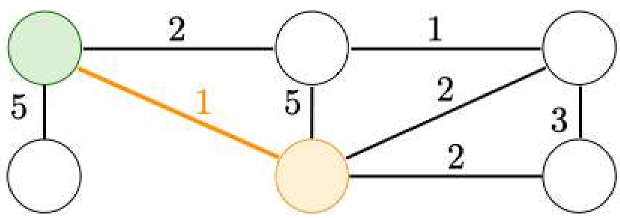
Prim's algorithm aims at finding an MST of a weighted connected undirected graph.

A solution is found in $\mathcal{O}(|E|\log(|V|))$ time and $\mathcal{O}(|V|+|E|)$ space with an implementation that uses a min-heap:

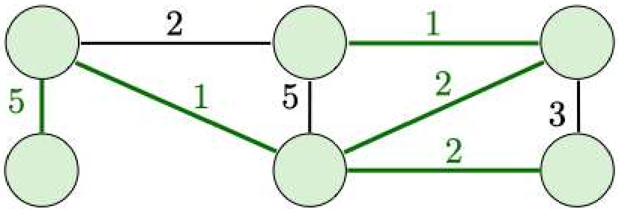
- *Initialization:* Pick an arbitrary node in the graph and visit it.



- *Compute step:* Repeat the following procedure until all nodes are visited:
  - Pick an unvisited node that connects one of the visited notes with the smallest edge weight.
  - Visit it.



- *Final step:* The resulting MST is composed of the edges that were selected by the algorithm.

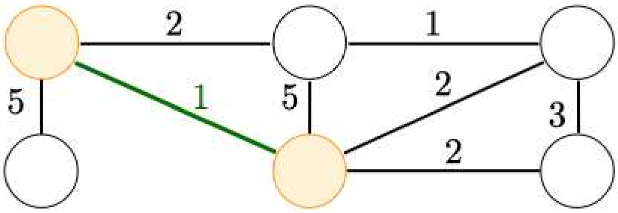


Kruskal's algorithm

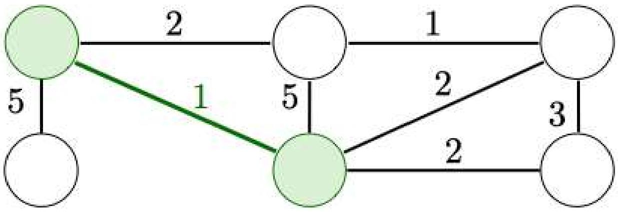
The goal of Kruskal's algorithm is to find an MST of a weighted connected undirected graph.

A solution is found in $\mathcal{O}(|E|\log(|E|))$ time and $\mathcal{O}(|V|+|E|)$ space:

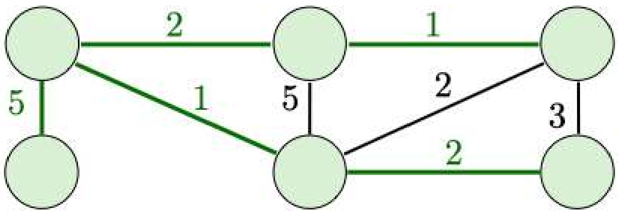
- *Compute step:* Repeat the following procedure until all nodes are visited:
  - Pick an edge that has the smallest weight such that it does not connect two already-visited nodes.



  - Visit the nodes at the extremities of the edge.



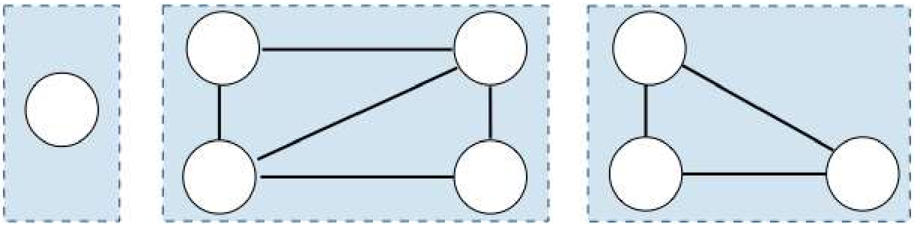
- *Final step:* The resulting MST is composed of the edges that were selected by the algorithm.



Components

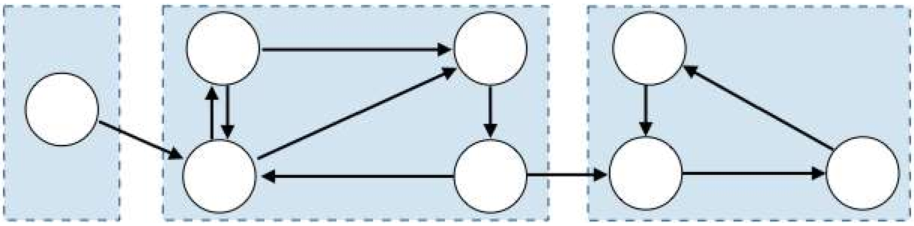
Connected components

In a given undirected graph, a connected component is a maximal connected subgraph.



Strongly connected components

In a given directed graph, a strongly connected component is a maximal subgraph for which a path exists between each pair of vertices.



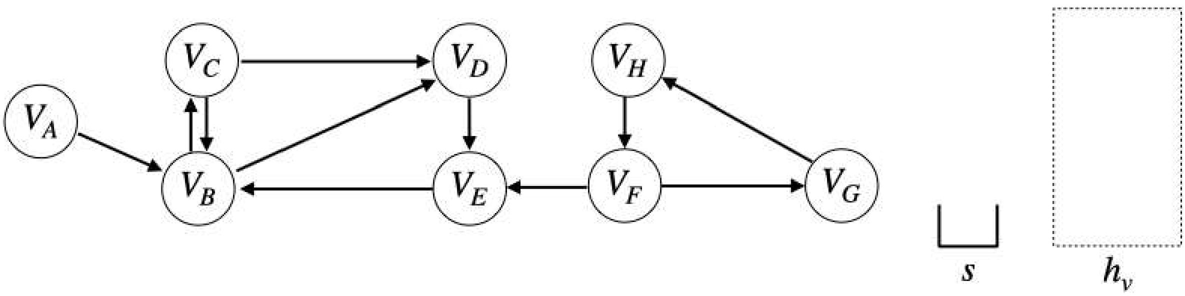
Kosaraju's algorithm

Kosaraju's algorithm aims at finding the strongly connected components of a directed graph.

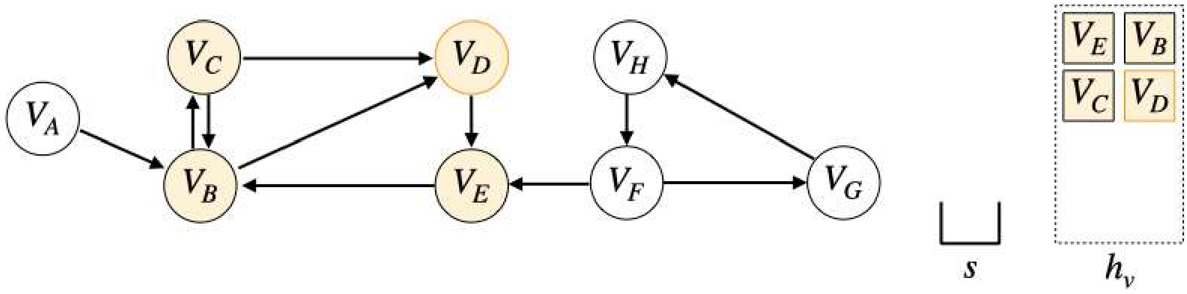
The solution is found in $\mathcal{O}(|V| + |E|)$ time and $\mathcal{O}(|V|)$ space:

First-pass DFS On the original graph, perform a DFS.

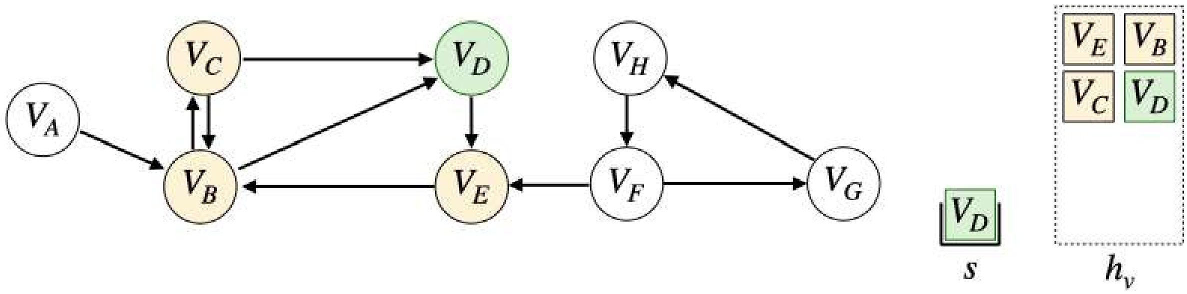
- *Initialization:* The following quantities are used:
    - An initially empty hash set h_v that keeps track of the visited nodes.
    - An initially empty stack s that keeps track of the order at which the nodes have had their neighbors visited.



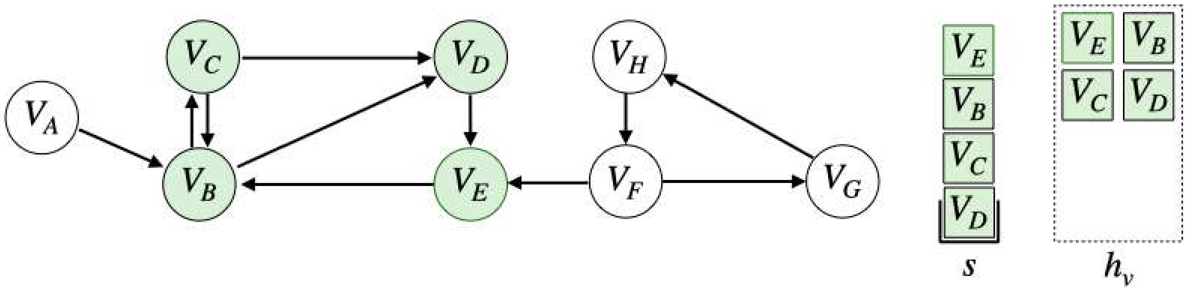
- *Compute step:* Perform the following actions:
    - Pick an arbitrary node and visit it by adding it to h_v. Recursively visit all its neighboring nodes.



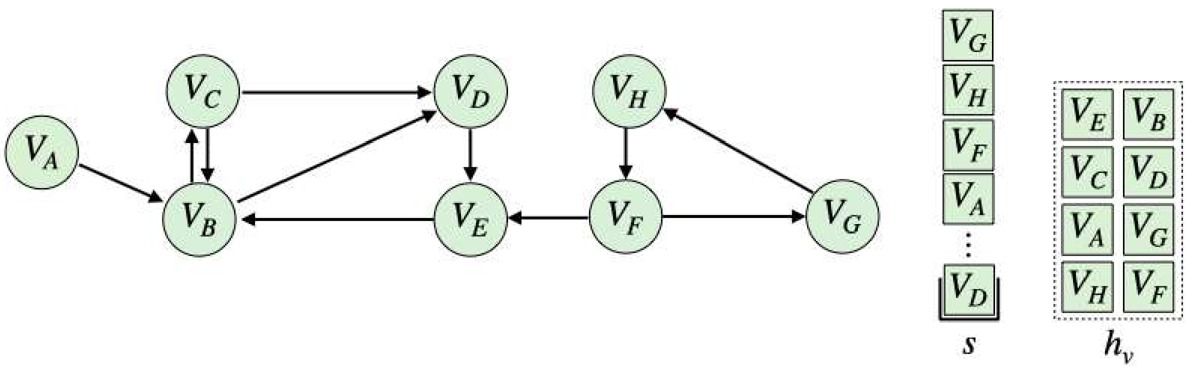
- Once a visited node has all its neighbors visited, push the visited node into the stack s.



The recursive procedure adds the remaining nodes to the stack s.



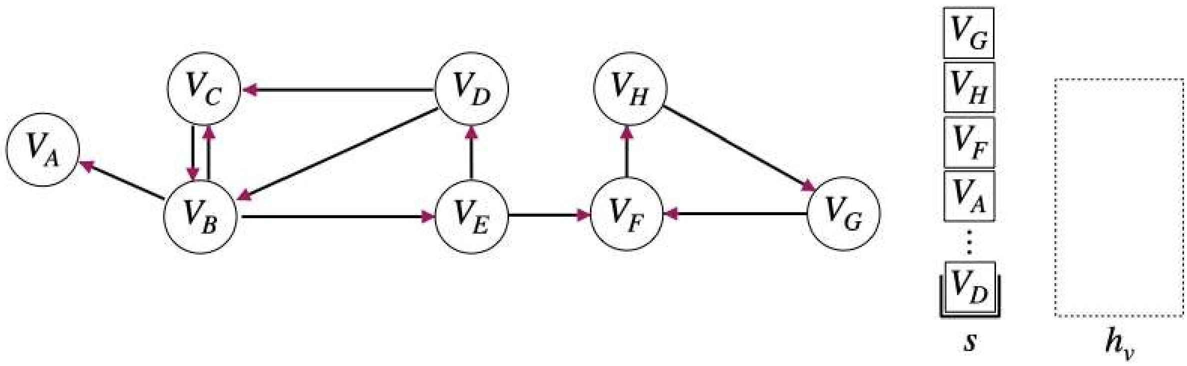
The same process is successively initiated on the remaining unvisited nodes of the graph until all nodes are visited.



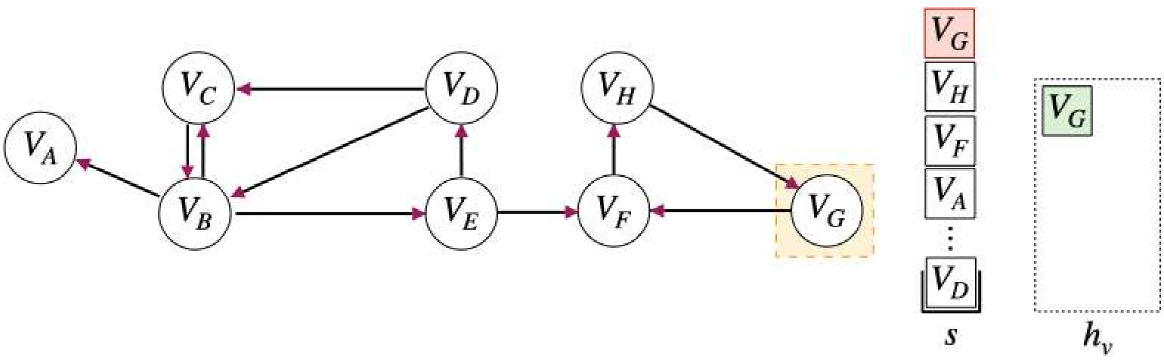
Second-pass DFS On the reverted graph, perform a DFS to determine the strongly connected components.

- *Initialization:* The following quantities are used:
  - The stack s obtained from the first-pass DFS.

- An initially empty hash set h_v that keeps track of the visited nodes.

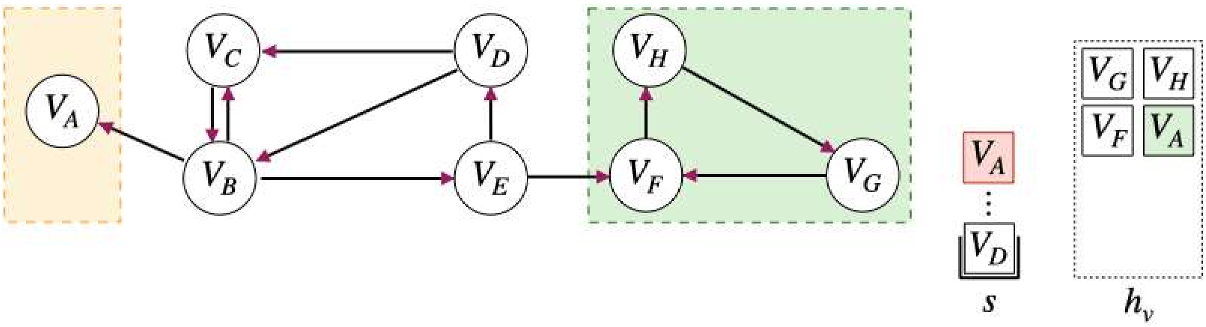


- *Compute step:* Pop a node from the stack.
  - *Node is not visited:* This node is the representative of a new strongly connected component. Add it to h_v. Perform a DFS starting from that node. For the nodes that are being explored:
    - *DFS node is not visited:* Add it to h_v and add the node to the hash set identifying the strongly connected component.

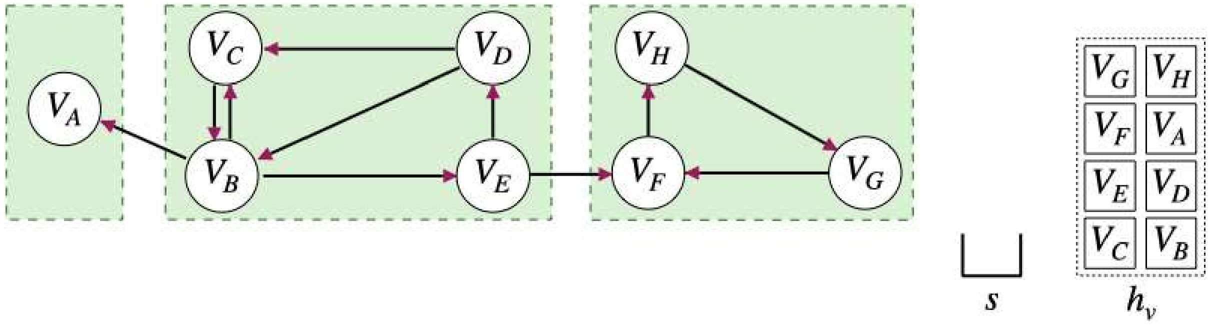


    - *DFS node is visited:* Skip it.
  - *Node is visited:* Skip it.

  Repeat this process again...



- *Final step:* ...until the stack is empty.



WANT MORE CONTENT LIKE THIS?

Subscribe here to be notified of new Super Study Guide releases!