

Trees (continued)

Definitions:

A **spanning tree** of a graph G is a tree that is a subgraph of G and includes every vertex of G .

A **minimum spanning tree** of a weighted graph G is a tree that is a spanning tree of least weight.

Spanning trees are **different** if their sets of edges are different.

Matrix-Tree Theorem (Kirchhoff):

For any connected graph $G = (V, E)$ of size n , the number of spanning trees is

$$\text{Cofactor}_{ij} (\text{diag}(\text{degrees}(V)) - \text{Adjacency}(G)) \quad \forall i, j \in \{1..n\}.$$

Theorem: The number M_n of labeled trees with n vertices is n^{n-2} .

Proof for $n \geq 3$: in effect, we can compute M_n as the number of spanning trees of the complete graph K_n using Kirchhoff theorem:

$$\begin{aligned} M_n &= \text{cofactor}_{11}(nI_n - 1) = \det(nI_{n-1} - 1) \\ &= \det \begin{bmatrix} n-1 & -1 & \cdots & -1 \\ -1 & n-1 & \cdots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \cdots & n-1 \end{bmatrix} = n^{n-2}. \end{aligned}$$

In the above $(n-1) \times (n-1)$ matrix, add rows 2 through $n-1$ to the first row, then subtract the first column from each other column. Then the computation of the determinant becomes straightforward.

Greedy minimum spanning tree algorithms

Kruskal:

Given a connected weighted graph $G = (V, E, W)$, initialize E' to an empty set. Then

For $k=1..n-1$,

Add to E' an edge $e \in E \setminus E'$ such that
the resultant E' has no circuits, and
with this restriction, e has minimal weight.

The result is the minimum spanning tree $T = (V, E')$.

Prim:

Given a connected weighted graph $G = (V, E, W)$, select any $v \in V$ and initialize T to a tree of one vertex: $T = (V', E')$, $V' = \{v\}$, $E' = \emptyset$. Then

For $k=1..n-1$,

Add to T an edge $e \in E \setminus E'$ such that
 e is incident with a vertex in V' , and
with this restriction, e has minimal weight.

This algorithm has an implementation with the complexity $O(n^2)$.

A lower bound for the weight of a minimum Hamiltonian cycle:

Given a connected weighted graph $G = (V, E, W)$ with a minimum Hamiltonian cycle H and a minimum spanning tree T , removal of any edge from H results in a spanning tree of G , therefore

$W(H) \geq W(T) + W(e)$, where e is the shortest edge in G .

This lower bound can be further improved (G&P, page 389).

Acyclic digraphs

Definition:

A labeling $\{v_i, i=1..n\}$ of an acyclic digraph $G = (V, E)$ of size n is called **canonical (canonical ordering)** iff $v_i v_j \in E \rightarrow i < j$.

Lemma: any set S of vertices in an acyclic digraph contains a vertex with zero indegree in S .

Proof by contradiction (cycles).

Theorem: a digraph G is acyclic iff it has a canonical labeling.

Proof. (\leftarrow) A canonical labeling excludes the possibility for any walk to come to its origin, since every step increases the label index. Therefore, there are no cycles in G .

(\rightarrow) Let $S_0 = G$. Then, by the Lemma, S_0 has a vertex of zero indegree. Call it v_0 . The rest of G (call it S_1), according to Lemma, has a vertex v_1 of zero indegree. The rest of the rest of G (call it S_2), according to Lemma, has a vertex v_2 of zero indegree, and so on. Thus constructed sequence $\{v_0, v_1, v_2, \dots\}$ is a canonical labeling, because, by construction, there are no arcs that return from S_i , $\forall i$.

Definition: a digraph is a **rooted tree** with **root** v iff the unoriented graph is a tree, and v is the only vertex with indegree 0.

Proposition: in a rooted directed tree with root v , there is a unique path from v to every other vertex.

Proof (existence). To construct the path from a given vertex to the root, follow reversed arcs: by the above definition, they always exist for a non-root node. Running into a previously visited node is impossible: there are no cycles. The process terminates at the root.

Corollary: in a rooted directed tree, every vertex other than root has indegree 1.

Uninformed Search Algorithms

Breadth-First Search of a Tree

In this strategy, levels of the tree are searched sequentially.

```

Queue := {root}
While (Queue)
    node := Pop (Queue)
    Search (node)
    Queue := Append (Queue, Expand (node))
  
```

Here *Queue* is the list of nodes that need to be processed, the function *Search* performs the search (whatever its goal is) of the given node, the function *Pop* extracts the first node from *Queue* (it returns the first element and shortens *Queue*); the function *Expand* returns the list of children of the node, the function *Append* (x, y) appends elements of y at the end of the list x . Formally speaking, *Queue* becomes “false” when it’s empty. Complexity is $O(b^m)$, where b (the **branching factor**) is the average number of children, and m is the height of the tree.

Depth-First Search of a Tree

In this strategy, leaves are searched sequentially, together with nodes encountered first time on the way to a leaf.

```

Stack := {root}
While (Stack)
    node := Pop (Stack)
    Search (node)
    Stack := Push (Expand (node), Stack)
  
```

Here the function *Pop* extracts the top node from the stack, and the function *Push* adds nodes on top of the stack. Complexity is $O(b^m)$.

Depth-First Search of a Graph

One possible implementation is given below. A new element with respect to the previous case is that visited nodes need to be labeled in order to avoid processing them twice. This is done with the function *Label*. The function *Unlabeled* selects only unlabeled vertices from a given list. The function *Reduce* removes duplicates from the list, leaving the first instances. The function *Expand* here returns all adjacent vertices of a given vertex.

```

Stack := {any vertex}
While (Stack)
  vertex := Pop (Stack)
  Search (vertex)
  Label (vertex)
  Stack := Reduce (Push ( Unlabeled (Expand (vertex) ), Stack ))

```

In addition to searching the graph, this algorithm is also potentially capable of returning a spanning tree and the associated canonical labeling (Figure 1). Because of the necessity to process every edge of the graph of the size n , the complexity of the algorithm is $O(n^2)$.

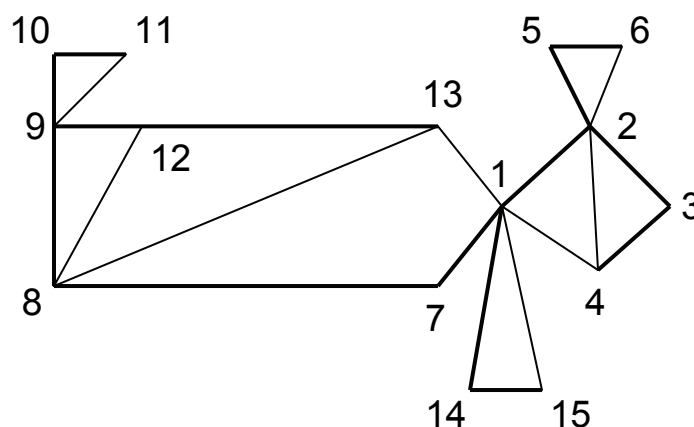


Figure 1. Example of a graph with a spanning tree (fat lines) and its canonical labeling that result from application of the depth-first search algorithm (from G&P, page 398).

The One-Way Street Problem

Definitions:

To orient a graph is to assign orientation to every edge of the graph

A graph has a **strongly connected orientation** if it is possible to orient it so that it will be strongly connected (can travel between any two points respecting orientations).

An edge of a connected graph is called a **bridge** (or **cut edge**) if its deletion renders the graph disconnected.

Theorem:

A graph has a strongly connected orientation iff it is connected and has no bridges.

Proof: (\rightarrow) If there is a bridge, assigning an orientation to it makes one of the two ends of it unreachable from the other end.

(\leftarrow) Otherwise, one can use the following algorithm.

Algorithm for assigning a strongly connected orientation:

Let T be a labeled spanning tree produced by a depth-first search in a connected graph G that has no bridges. For each edge ij of G , assign the orientation $i \rightarrow j$, if $ij \in T$. Otherwise, assign $j \rightarrow i$.

Proof: By induction.

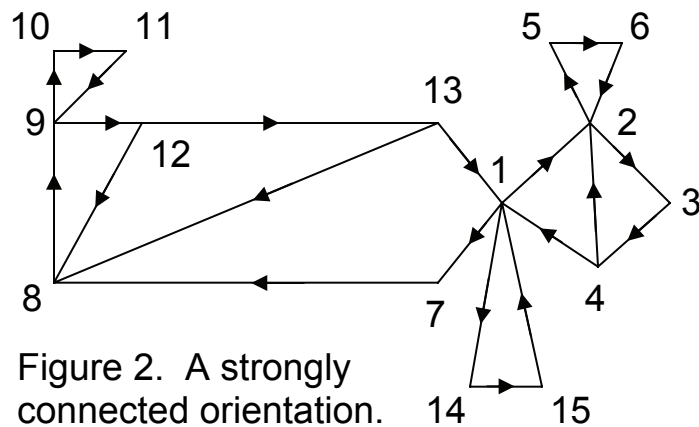


Figure 2. A strongly connected orientation.