Tree

Recognition

Recognition Tests

A *forest* is an acyclic, undirected graph, and a *tree* is a connected forest. Depending on the subfield, there are various conventions for generalizing these definitions to directed graphs.

In one convention, directed variants of forest and tree are defined in an identical manner, except that the direction of the edges is ignored. In effect, each directed edge is treated as a single undirected edge. Then, additional restrictions are imposed to define *branchings* and *arborescences*.

In another convention, directed variants of forest and tree correspond to the previous convention's branchings and arborescences, respectively. Then two new terms, *polyforest* and *polytree*, are defined to correspond to the other convention's forest and tree.

Summarizing:

```
+-----------------------------+
| Convention A | Convention B |
+=============================+
| forest       | polyforest   |
| tree         | polytree     |
| branching    | forest       |
| arborescence | tree         |
+-----------------------------+
```

Each convention has its reasons. The first convention emphasizes definitional similarity in that directed forests and trees are only concerned with acyclicity and do not have an in-degree constraint, just as their undirected counterparts do not. The second convention emphasizes functional similarity in the sense that the directed analog of a spanning tree is a spanning arborescence. That is, take any spanning tree and choose one node as the root. Then every edge is assigned a direction such there is a directed path from the root to every other node. The result is a spanning arborescence.

NetworkX follows convention "A". Explicitly, these are:

undirected forest

An undirected graph with no undirected cycles.

undirected tree

A connected, undirected forest.

directed forest

A directed graph with no undirected cycles. Equivalently, the underlying graph structure (which ignores edge orientations) is an undirected forest. In convention B, this is known as a polyforest.

directed tree

A weakly connected, directed forest. Equivalently, the underlying graph structure (which ignores edge orientations) is an undirected tree. In convention B, this is known as a polytree.

branching

A directed forest with each node having, at most, one parent. So the maximum in-degree is equal to 1. In convention B, this is known as a forest.

arborescence

A directed tree with each node having, at most, one parent. So the maximum in-degree is equal to 1. In convention B, this is known as a tree.

For trees and arborescences, the adjective “spanning” may be added to designate that the graph, when considered as a forest/branching, consists of a single tree/arborescence that includes all nodes in the graph. It is true, by definition, that every tree/arborescence is spanning with respect to the nodes that define the tree/arborescence and so, it might seem redundant to introduce the notion of “spanning”. However, the nodes may represent a subset of nodes from a larger graph, and it is in this context that the term “spanning” becomes a useful notion.

`is_tree`(G)	Returns True if `G` is a tree.
`is_forest`(G)	Returns True if `G` is a forest.
`is_arborescence`(G)	Returns True if `G` is an arborescence.
`is_branching`(G)	Returns True if `G` is a branching.

Branchings and Spanning Arborescences

Algorithms for finding optimum branchings and spanning arborescences.

This implementation is based on:

> J. Edmonds, Optimum branchings, J. Res. Natl. Bur. Standards 71B (1967), 233–240. URL: http://archive.org/details/jresv71Bn4p233

`branching_weight`(G[, attr, default])	Returns the total weight of a branching.
`greedy_branching`(G[, attr, default, kind, seed])	Returns a branching obtained through a greedy algorithm.
`maximum_branching`(G[, attr, default, ...])	Returns a maximum branching from G.
`minimum_branching`(G[, attr, default, ...])	Returns a minimum branching from G.
`maximum_spanning_arborescence`(G[, attr, ...])	Returns a maximum spanning arborescence from G.
`minimum_spanning_arborescence`(G[, attr, ...])	Returns a minimum spanning arborescence from G.
`ArborescenceIterator`(G[, weight, minimum, ...])	Iterate over all spanning arborescences of a graph in either increasing or decreasing cost.
`Edmonds`(G[, seed])	Edmonds algorithm [R5a58a7577195-1] for finding optimal branchings and spanning arborescences.

Encoding and decoding

Functions for encoding and decoding trees.

Since a tree is a highly restricted form of graph, it can be represented concisely in several ways. This module includes functions for encoding and decoding trees in the form of nested tuples and Prüfer sequences. The former requires a rooted tree, whereas the latter can be applied to unrooted trees. Furthermore, there is a bijection from Prüfer sequences to labeled trees.

`from_nested_tuple`(sequence[, ...])	Returns the rooted tree corresponding to the given nested tuple.
`to_nested_tuple`(T, root[, canonical_form])	Returns a nested tuple representation of the given tree.
`from_prufer_sequence`(sequence)	Returns the tree corresponding to the given Prüfer sequence.
`to_prufer_sequence`(T)	Returns the Prüfer sequence of the given tree.

Operations

Operations on trees.

`join`(rooted_trees[, label_attribute])	Returns a new rooted tree with a root node joined with the roots of each of the given rooted trees.

Spanning Trees

Algorithms for calculating min/max spanning trees/forests.

`minimum_spanning_tree`(G[, weight, ...])	Returns a minimum spanning tree or forest on an undirected graph `G`.
`maximum_spanning_tree`(G[, weight, ...])	Returns a maximum spanning tree or forest on an undirected graph `G`.
`minimum_spanning_edges`(G[, algorithm, ...])	Generate edges in a minimum spanning forest of an undirected weighted graph.
`maximum_spanning_edges`(G[, algorithm, ...])	Generate edges in a maximum spanning forest of an undirected weighted graph.
`SpanningTreeIterator`(G[, weight, minimum, ...])	Iterate over all spanning trees of a graph in either increasing or decreasing cost.

Decomposition

Function for computing a junction tree of a graph.

`junction_tree`(G)	Returns a junction tree of a given graph.

Exceptions

Functions for encoding and decoding trees.

Since a tree is a highly restricted form of graph, it can be represented concisely in several ways. This module includes functions for encoding and decoding trees in the form of nested tuples and Prüfer sequences. The former requires a rooted tree, whereas the latter can be applied to unrooted trees. Furthermore, there is a bijection from Prüfer sequences to labeled trees.

`NotATree`	Raised when a function expects a tree (that is, a connected undirected graph with no cycles) but gets a non-tree graph as input instead.