- Author: Petre Iordanescu
- Publisher: RENware Systems
- Date: 2021 July
- Tags: software alghoritms and data structures
- Language: EN, partial RO

About brief series. These materials are derived from author teaining sessions as course notes for students. It is intended as presentation of basic topics regarding a concept. These materials are intended to present for a concept whst is good for, its main topics and what to deep research to find out and learn more about it.*

Graphs in software. Overview

Generalities. A brief of theory

A graph is a network of nodes. Depending on some factors (enumerated below), graps can be classified in useful categories, the most important thing (for this article purpose) being their usage and applicability in software systems.

Basic properties

Here are enumerated the most basic components and properties of a graph. These, for sure are not all, are enumerated those used in all applications and are fundamental to understand graphs.

Node

A node is an element that contains information about "something / anything" that is used in the graph network. Any node contains its core information represented (or can be useful represented) in graph. See also note 1.

Edge

An edge links two nodes. The edge must have a meaning, ie what is good for that link between nodes or what represents that link.

For example in a social network, between any two nodes can be more links (edges) with different meanings. Assuminga that a node is a person in that network, there could be these kind if edges:

- X is friend with Y
- X follow Y
- X likes a post of Y
- and so on...

NOTE: In most cases the edge is also named *vertex*.

Edge direction

Represents the direction of an edge in a directed graph. Usual is a pair of modes, the *from* and *to* nodes.

Pay careful that from and to attributes used here does not always are in visual accordance with a drawing using arrows. The name of yhe edge should be the relevant factor.

Fan. Fan in. Fan out

These are properties of a node. Fan in and out are relevant in an clearly / explicit directed graph.

Fan in represent the number of edges tha *enter* in a node and fan out is the number of edges that leave frim a node.

Fan is the total number of edges, ie the fan in plus fan out.

Root(s)

Strictly, a root is a node with fan in equal to zero (0).

Please pay attention in non directed graph, where root has no formal meaning but for convenience it is largely used. If there is no explicit statement regarding root definition is better to ask.

Leaf(s)

Strictly, a leaf is a node with fan out equal to zero (0). The same remark as for root is applicable here.

Classifications

- by *directionality* there are **directed graphs** or **not directed graphs** (these are usual named simple graphs and not specifying that are not directed, ie undetsating that default are not directed). RO terms are "graf orientat" and "graf neorietat".
- by *inside cycles* can have **cyclic graphs** or **acyclic graphs**, tish referring to property of a node to 'return to itself' by using a route (doesn't matter which or how long is it). RO terms are "graf ciclic", respectively "graf aciclic".
- by *number of parents of any node* can have graphs with all nodes having no more than one parent, these kind of graphs being named **trees**.

Extended properties

- if a directed graph has at least one *cycle* . A graph without cycles is called *hamiltonian*
- if a tree has more than one root (in this case is named *forest*)
- if a graph has more than one root. In these case or are more separated graphs ir roots are in fact leaves in a nin directed graph
- there are nodes with fan zero. These are completly isolated nodes (sometimes named singularities)
- a gtaph without any leaf is a graph with at least one cycle

---

Applications

Here are enumerated the most known important application of graphs in computing engineering field.

Dependency graphs

This is a kind of usage in which a graph represents the dendency relation (*edge*) between any two components(*node*).

This is usually used in:

- operating systems to keep eviddnce of locks between resources. Therefore a cycle in this graph demote a *deadlock* and operating system can prevent and solve them.
- in spreadsheet applications to determine the order of evaluating cells, by keeping the telation *cell X use cell Y*

Shortest path

In this usage, the graph represents tbe roads (*edges*) between two locations (*nodes*). The problem solved by this alghoritm is to find the shortest route between a givem set of two nodes.

The real applications use more edges between two nodes, with meanings like (as useful examples) :

- average time spent to walk between nodes
- the distance between two nodes
- number of fuel stations between two modes
- and so on

This approach allow for being mote clear and specific about *short path*, ie, as time, as kilometers, and so on.

Spanning tree alghoritm

Spanning tree alghoritm (STP Spanning Tree Protocol, see also note 3) was developed by Radia Perlman (see note 4) as solvind the problem of **networking loops** when use redundant routers as fail safe mechanisms.

In fact this is a network (as graph) in which we are interested to find a route (possible shortest) between two modes and to avoid as much as possible the usage of other routes between that two nodes. More specific, the algoritm detects cycles in graph and avoid them by isolating and forbidding some routes until cycles are eliminated.

Best cost route

This is a particular kind of *shortest path* in which edges represent the cost.

Diagnose / faults tree

These are in essence *dependency trees* in which nodes are different fault types and edges the cause of faults. The hierarchical representation (as tree) allows for a gradual detarmination of fault causes starting with a current potential risk (cause).

They are also represented and kwown sd "fish bone" diagrams.

Project network (PERT)

In yhis application, nodes are projecy activities and edges represent the technical dependency between two activities. Most project management problems are solved by this graph, like total project duration, critical path, and so on.

These project networks / graphs are also known as PERT diagrams (Program Evaluation and Review Technique).

Critical path

This is a particularly problem id project networks, and deals with finding the maximum length route (as duration) and activities that are traversed thru that route (or routes).

This route is known as "project critical path" because any change in activities from this route will affect the project finish date.

Sintax trees

These are very common structures in compilers and allow for an expression evaluation and validity checking without the usage of brackets regardless of expression complexity.

They are a generalisation of RPN, Reverse Polish Notation.

Notes and biographical references

- 1. For example in a computer network, the nodes are computers, printers, any other equipments.
- 2. I recommend reading all Dijkstra alghoritms, as being fundamental in software. Dijkstra alghoritms (https://en.m.wikipedia.org/wiki/Dijkstra%27s_algorithm)
- 3. [Spanning Tree Protocol] (https://en.m.wikipedia.org/wiki/Spanning_Tree_Protocol (https://en.m.wikipedia.org/wiki/Spanning_Tree_Protocol))
- 4. About Radia Perlman (https://en.m.wikipedia.org/wiki/Radia_Perlman)