Graphs and graph algorithms

An introduction for Python

Victor Morel
https://victor-morel.net/

Chalmers University of Technology

morelv@chalmers.se

11^{th} November 2022

Outline

You use graphs everyday

Graphs are everywhere

- When you use a GPS 
- When you run a request on a search engine 
- When you connect on social media 
- When anything you use is based on a neural network

Sorry I lied

I was not accurate:

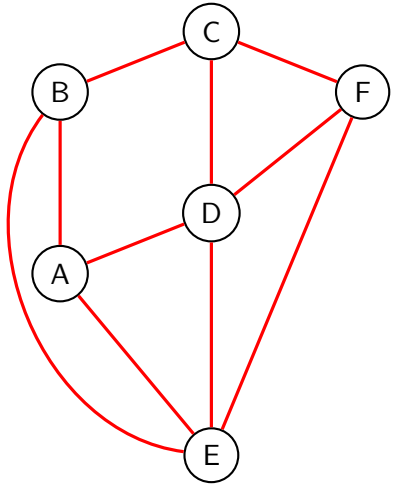
- What I described were networks
- But networks are represented with graphs

So what is a graph? 🤔

Basic definition

In its simplest form

- A graph G is composed of
  - A set of vertices V
  - Also called nodes
  - Connected through edges E
  - Also called links
- $G = (V, E)$



A graph can be represented like this

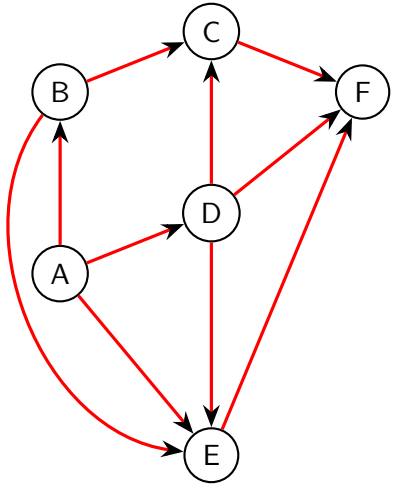
The end

For the less perspicacious: that's a joke, there's plenty of other stuff to see.

Directed graphs

Some graphs are directed

- Constraint on the edges
- Edges have a direction

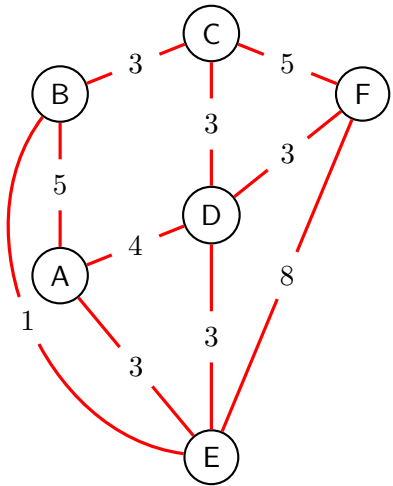


A directed graph can be represented like this

Weighted graphs

Some graphs are weighted

- Constraint on the edges here again
- Edges have a weight
- Also called a cost
- Typical of a certain kind of graph we will study later



A weighted graph can be represented like this

Vocabulary

Degree Number of vertices

Adjacent nodes Nodes connected through an edge

Arc In a directed graph, an edge (yup that's all)

Loop In a directed graph, an edge that connects a node to itself

Successor In a directed graph, node j is a successor of i if $\exists$ arc (i,j)

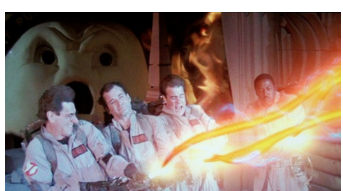
Predecessor In a directed graph, node j is a predecessor of i if $\exists$ arc (j,i)

Connected For a regular graph, it means all nodes are connected. For a directed graph, it means that for every nodes (i,j) $\exists$ a path between i and j (semiconnected to be accurate)

Planar graph A graph is planar if it can be embedded in a plane (edges don't cross)

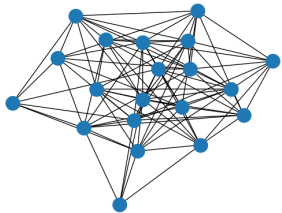
You want a planar graph?

Don't cross $\rightarrow$



Outline

Random networks are random



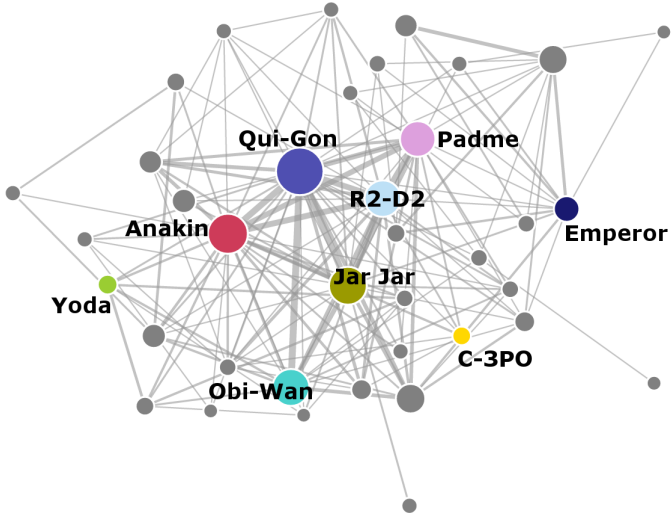
Beautiful isn't it?

Too random

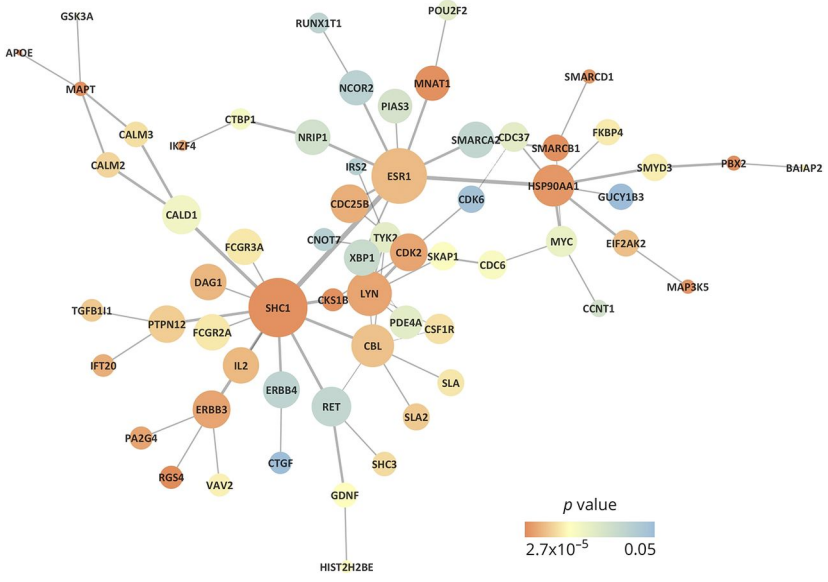
Random networks miss some characteristics:

- Scale-free (few super connected nodes, lots of little connected)
- Small world (can easily reach any node from any other)
- High clustering coefficient (high density of ties)

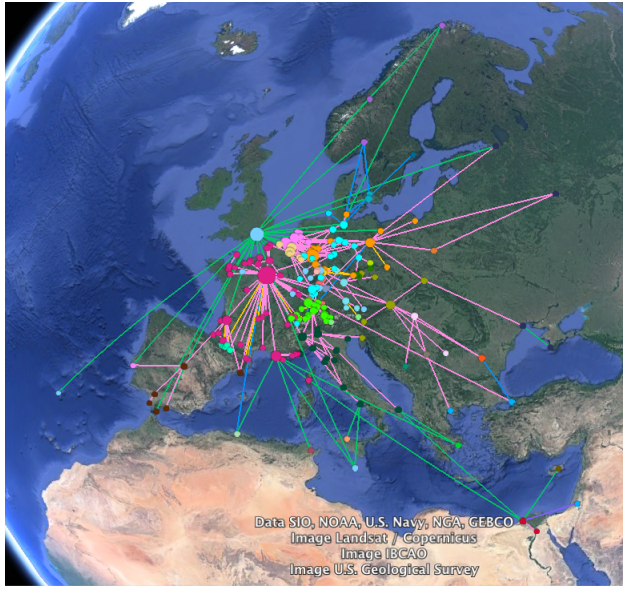
Social networks



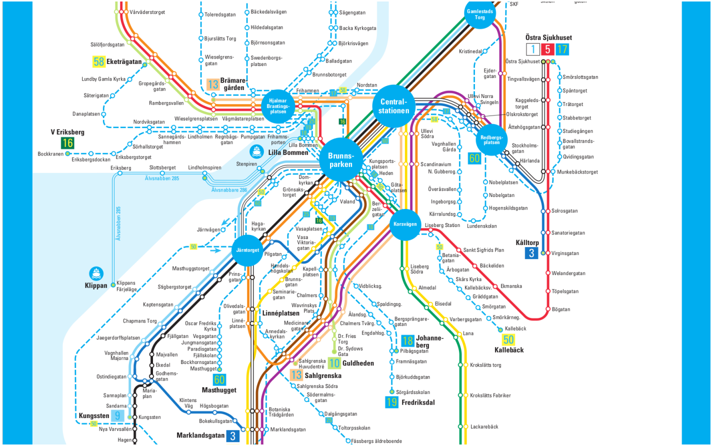
Biological networks



Transport networks



Use-case: tram network in Göteborg



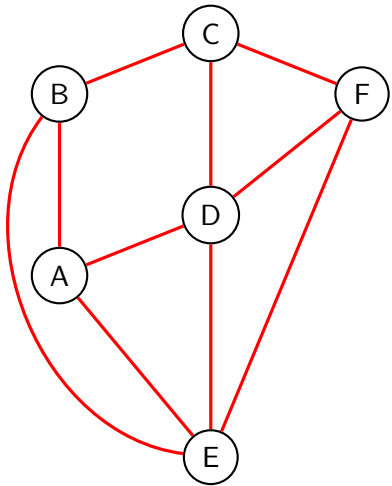
Network graphs are usually weighted and undirected

They can be directed (but not in our case)

Outline

Graphs for humans: diagrams

We humans like this:

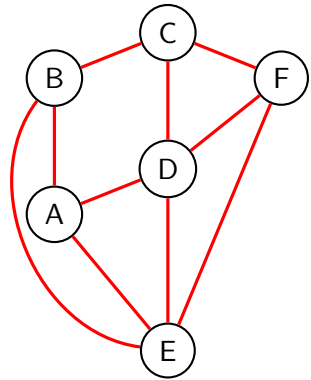


But not exactly machine-readable

Graphs for machines: adjacency matrix

Mathematicians like this:

$$A_{6\times 6} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$



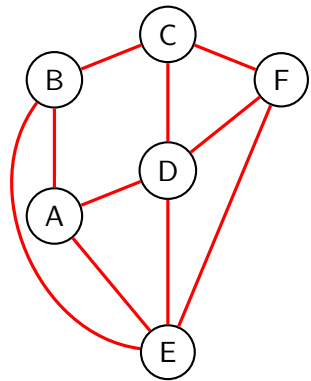
Previous graph

Good machine-representation! 👍

Graphs for machines: adjacency list

Computers also like this:

adjlist = {
    A: [B, D, E],
    B: [A, C, E],
    C: [B, D, F],
    D: [A, C, E, F],
    E: [A, B, D, F],
    F: [C, D, E]
}



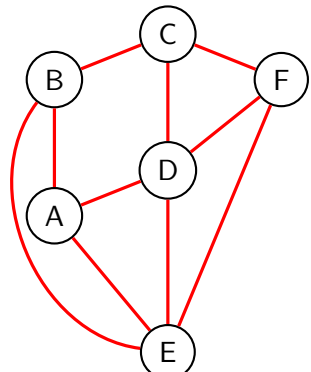
Previous graph

Graphs for snakes: two lists

Reptiles also like this representation:

vertices = [A, B, C, D, E, F]
edges =
[(A,B),(A,D),(A,E),(B,C),(B,E),(C,D),
(C,F),(D,E),(D,E),(E,F)]

↑ This is a typical notation



Still the same graph

Outline

Graph in python - manual implementation

graph_skeleton.py

```python
class Graph:
    def __init__(self, start=None, values = None,
                 directed=False):
        self._adjlist = {}
        if values is None:
            values = {}
        self._valuelist = values
        self._isdirected = directed
        # plus some code for building a graph from a 'start' object
        # such as a list of edges
        # here are some of the public methods to implement
    def vertices(self):
    def edges(self):
    def neighbours(self,v):
    def add_edge(self,a,b):
    def add_vertex(self,a):
    def is_directed(self):
    def get_vertex_value(self, v):
    def set_vertex_value(self, v, x):


class WeightedGraph(Graph):
    #missing something?

    def set_weight(self, a, b, w):
    def get_weight(self, a, b):
```

Graph in python - NetworkX

NetworkX

Network Analysis in Python

Outline

Diameter

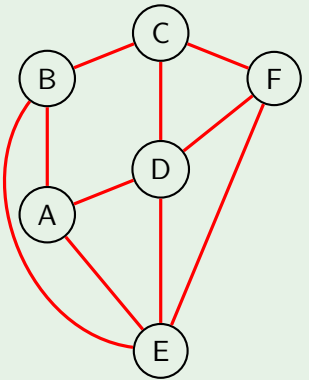
Definition

The diameter d of a graph is the greatest distance between any pair of vertices.

In NetworkX

nx.diameter(G)

Example



Diameter is 2

Degree centrality

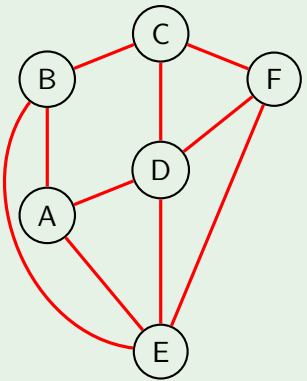
Definition

The degree centrality for a node v is the fraction of nodes it is connected to (minus itself).

In NetworkX

nx.degree(G)

Example



Degree centrality for each node is:
A: 0.6, B: 0.6, C: 0.6, D: 0.8, E: 0.8, F: 0.6

Clustering coefficient

Definition

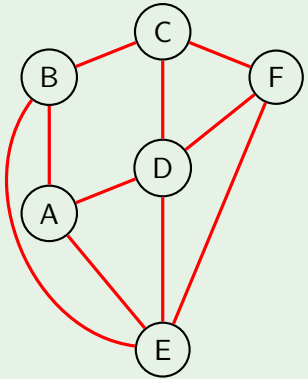
The clustering coefficient C_i for a vertex v_i is given by a proportion of the number of links between the vertices within its neighbourhood divided by the number of links that could possibly exist between them (in simple terms).

$C_i = \frac{2|\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{k_i(k_i - 1)}$ otherwise

In NetworkX

nx.clustering(G), nx.average_clustering(G)

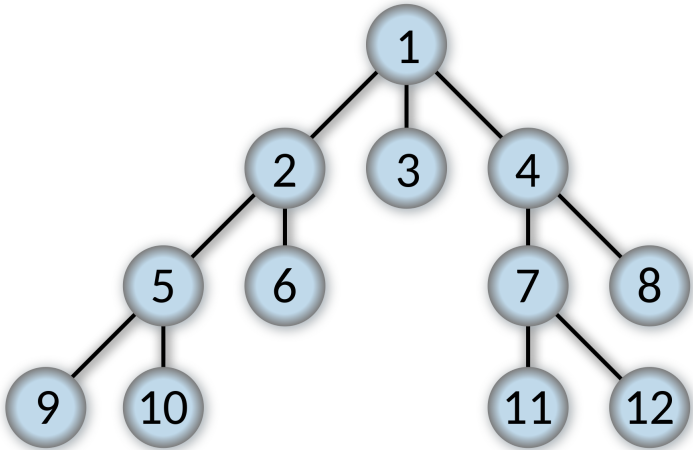
Example



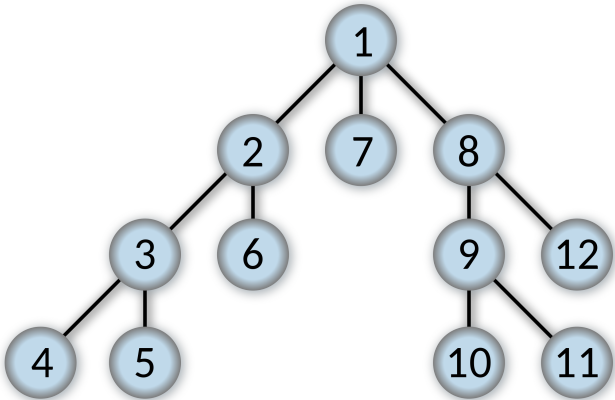
Clustering coefficient for each node is: A: 0.66, B: 0.33, C: 0.33, D: 0.5, E: 0.5, F: 0.66. Average is: 0.5

Outline

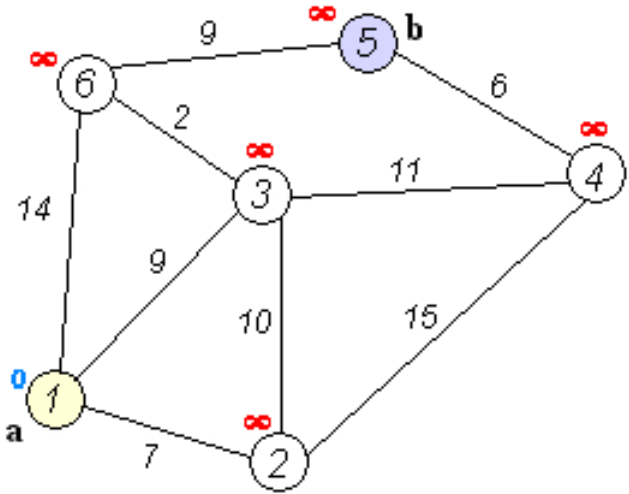
Breadth-first search



Depth-first search



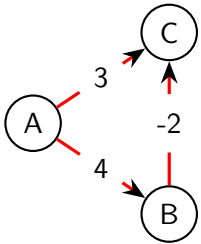
Dijkstra's algorithm



Going further

Shortest path with negative values?

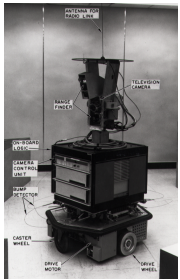
Go for Bellman-Ford



Not so clever, Dijkstra hey? (Start with A)

Want something bit more efficient?

Go for A* (uses heuristics)



Exterminaaate

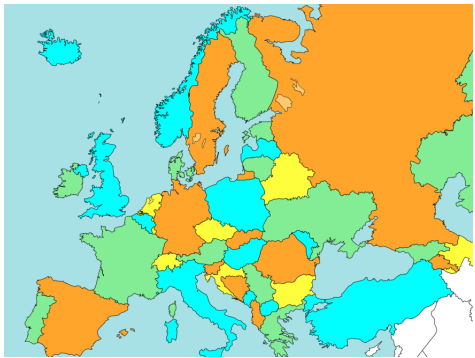
https://en.wikipedia.org/wiki/Shortest_path_problem

Outline

Graphs: other applications

Coloring

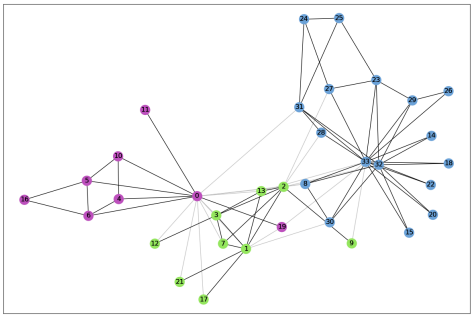
Adjacent vertices always have different colors



Only four colors needed

Social networks

- Clustering
- Community detection



A famous example: the Karate Club