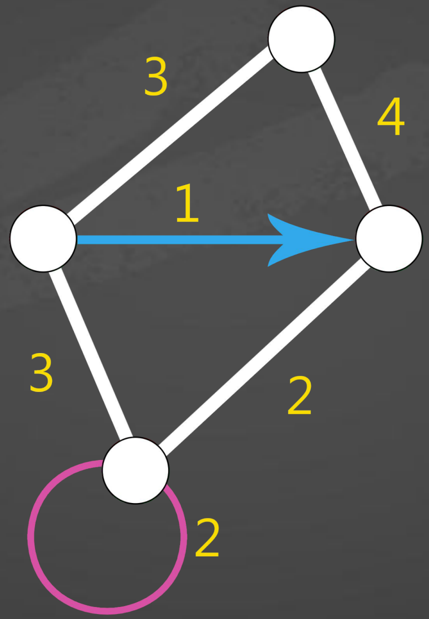
Multigraph

Group F
André Flores, up201907001
Diogo Faria, up201907014
Tiago Rodrigues, up201906087

Graph

Nodes & Edges

Edges connect nodes in the graph



Directed

Edges could have orientation, restricting the direction

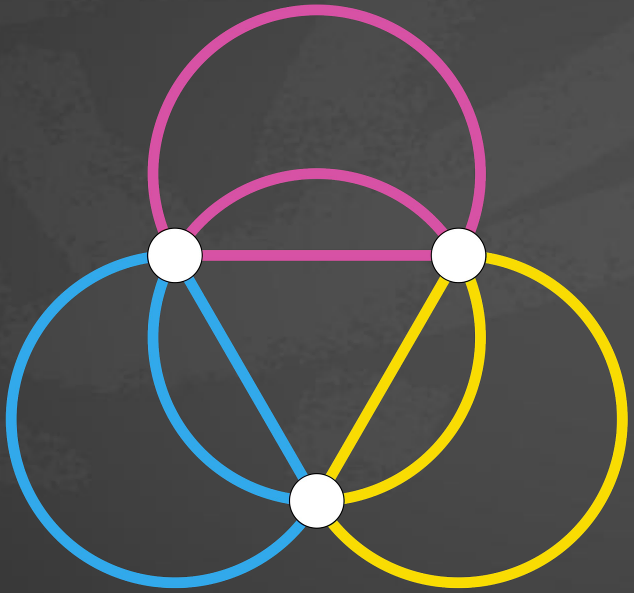
Self-Loops

Edge that joins a vertex to itself

Weighted

Edges could have a weight associated

Multigraph



It's a graph

It's nothing more than a graph with...

Parallel Edges

There can be multiple edges between the same two nodes

Or is it?

Ambiguity

The term multigraph is ambiguous enough that researchers have argued for its deprecation[1][2].

Parallel Edges

Literature disagrees on whether parallel edges are just permitted[3][4] or required[5][6][7].

Self-Loops

Researchers also disagree on whether self-loops are permitted[8][9] or forbidden[3][4][7]. Others stay silent on the issue[4][5][6].

Identity

Edges can either be labelled or unlabelled

Applications

- Multigraphs are used to formally specify multidimensional networks, that is, networks with multiple kinds of relations.
- Normal graphs aren't appropriate to describe many real-world relationships.

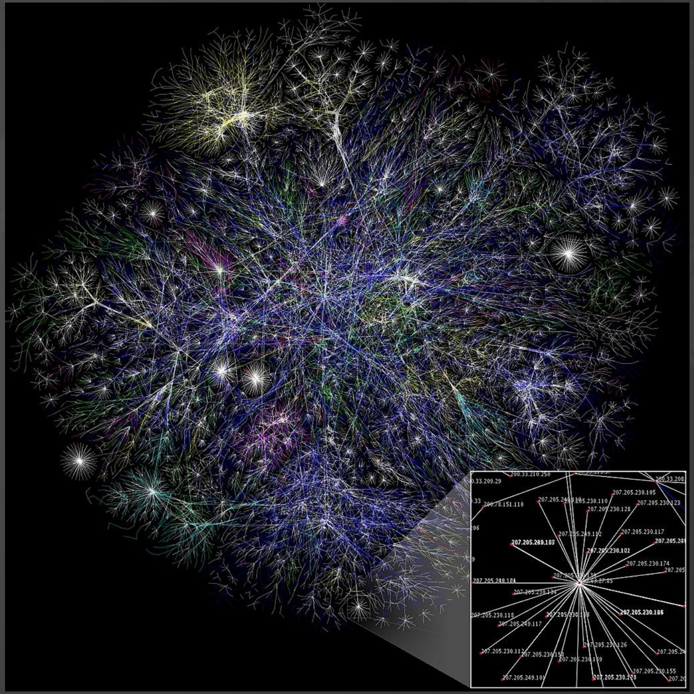
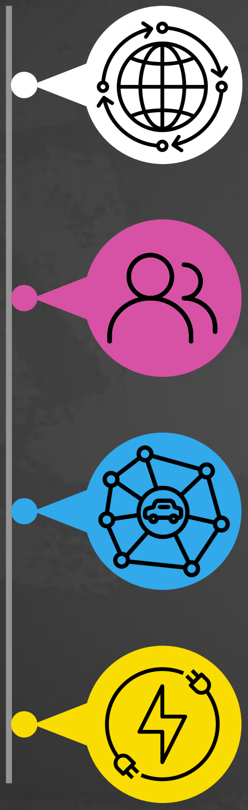


Fig.1 - Partial Internet Map from 2005

Applications



Internet Routing

Node represents a network
Edge represents a path

Social Networks

Node represents a person
Edge represents a relationship (friendship, family ties, ...)

Transportation Networks

Node represents a location
Edge represent a route

Electrical Networks

Node represents a component (resistor, capacitor, ...)
Edge represents a wire

Operations

1

Add node/edge

2

Remove node/edge

3

Find a node

4

Find the shortest path

1 Add node/edge

2 Remove node/edge

3 Find a node

- Usually, Nodes are stored in sets and edges in either sets (labelled edges) or multisets (unlabelled edges).
- Time complexity for these operations depends solely on the data structure in which the sets and multisets are implemented.
- Hash Tables are good choices for the average case: O(1) for adding and finding a node and O(m) for removing.

4 Find the shortest path

- Regular algorithms don’t usually work out-of-the-box on multigraphs, they usually need to be altered due to having the addition of multiple edges between the same nodes.
- One such example is Dijkstra’s algorithm, which when traversing the graph needs to be able to know the weight of the edges between two nodes.
- It also has a caveat regarding the edges’ weights: it only works with positive integers or real numbers.
- In order for the algorithm to work on multigraphs, it needs the ability to select the smallest edge between two nodes.

4 Find the shortest path

```
//MW is a Min-Weight set with all W(u, v), with W(u, v) being the edge with minimal weight between 'u' and 'v'
function Generalised-Dijkstra(Graph, MW, source):
   for each vertex v in Graph.Vertices:         //For each vertex in Graph
       d[v] ← ∞                                 //Start with all distances to source with infinite
       prev[v] ← Null                           //Set all the previous visited vertices as Null
       add v to Q                               //Add vertex to unvisited vertex list

   d[source] ← 0                                //Distance from source to source is 0

   while Q is not empty:                        //While there are unvisited vertices
       u ← vertex in Q with min d[u]            //Start with the closest vertex
       remove u from Q                          //Remove it from the unvisited list

       for each neighbour v of u still in Q:    //For each vertice connected to 'u' that hasn't been visited
           w ← MW(u, v).weight                  //Select the edge with smallest weight between the vertexes
           alt ← d[u] + w                       //Consider possible alternate route
           if alt < d[v]:                       //If alternate route better than the one already known
               d[v] ← alt                       //Update the distance from source
               prev[v] ← u                      //Update previous visited vertex

   return d[], prev[]                           //Return results
```

References

1. West, D. B. Introduction to Graph Theory, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2000.
2. Weisstein, Eric W. "Multigraph." From MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/Multigraph.html
3. Harary, F. Graph Theory. Reading, MA: Addison-Wesley, p. 10, 1994.
4. Gross, J. T. and Yellen, J. Graph Theory and Its Applications. Boca Raton, FL: CRC Press, 1999.
5. Skiena, S. Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Reading, MA: Addison-Wesley, 1990.
6. Pemmaraju, S. and Skiena, S. Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Cambridge, England: Cambridge University Press, 2003.
7. Zwillinger, D. (Ed.). CRC Standard Mathematical Tables and Formulae, 31st ed. Boca Raton, FL: CRC Press, 2003.
8. Hartsfield, N. and Ringel, G. Pearls in Graph Theory: A Comprehensive Introduction, 2nd ed. San Diego, CA: Academic Press, 1994.
9. Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. Introduction to Algorithms, 2nd ed. Cambridge, MA: MIT Press, 2001.
10. Biswas, S. S., Alam, B., & Doja, M. N. Generalization of Dijkstra's Algorithm for Extraction of Shortest Paths in Directed multigraphs. J. Comput. Sci., 9(3), p. 377-382, 2013.

Figures

1. By The Opte Project bkrqayd232@gmail.com - Originally from the English Wikipedia