

Graph Theory *lite*

What am I looking at?

Daiwei Chen

<2019-03-06 Wed>

Contents

1	Graph Theory <i>lite</i>	1
1.1	Paths	2
1.2	Graphs	2
1.3	Subgraphs	2
2	Keeping Track of Graphs	2
2.1	Adjacency List	2
2.2	Adjacency Matrix	3
3	Weighted Graphs	3
4	Minimal Spanning Tree	3
5	Prim's Algorithm	3

1 Graph Theory *lite*

When you would like to model problems with "complex" relationships.

Flights Catching flights and changing planes.

Internet Routing The internet is a giant graph. Figuring out the number of hops and which hops to take in order to get to your final destination.

Circuit Boards Making sure connections are efficient.

Basically, things that are connected to each other. A Graph Consists of a set of **vertices** and **nodes** that are *connected* by edges.

Tools: Python - Networkx, c/c++ - nauty. Assumptions:

- No Self Loops
- Edges are Unique

1.1 Paths

Path List of vertices l such that $(l[i], l[i + 1]) \in E$.

On each small step of a path, the 2 nodes must have an edge between them.

Simple Path Path without repeated vertices.

Cycle A (mostly) simple path but the 1st and last vertices are the same. Only the beginning and end are repeated.

Path Length The number of edges in a path.

1.2 Graphs

Acyclic Graph A graph without **ANY** cycles.

Connected Graph A graph in which for some a there is a path from every vertex to every vertex.

Tree Connected acyclic graph.

- $|V| - 1$ edges.

1.3 Subgraphs

Subgraph Only contains vertices and edges of the original graph. Edges must have 2 end-points.

Spanning Tree A subgraph of G that contains all nodes in G and is a tree.

2 Keeping Track of Graphs

These 2 are not the only two styles of graph representations. There are many, many more and more specialized representations for the correct use case.

2.1 Adjacency List

Just a list, for example:

A	->	B, C, D
B	->	A, C, E
C	->	A, B
D	->	A, E
E	->	B, D

2.2 Adjacency Matrix

Shows if one node is connected to another.

	A	B	C	D	E
A	0	1	1	1	0
B	1	0	1	0	1
C	1	1	0	0	0
D	1	0	0	0	1
E	0	1	0	1	0

3 Weighted Graphs

Just a graph, but the edges has been assigned a value. For example, on a switch, different physical connections could have a weight of how much bandwidth each one gets?

4 Minimal Spanning Tree

You want to find the spanning tree where the total weight is the smallest.

5 Prim's Algorithm

A minimal spanning tree algorithm.

```
def prims(V:set, E:set):
    tree_vertex_set = {V[random(len(V))]} # The vertex starts with ANY vert we want to start
    tree_edge_set = {}

    for i=1..len(V)-1:
        # This very much depends on the way you're storing your edges
        find the minimal edge $e_m$ = ($u_m$, $V_m$) such that $u_m \notin$ tree_vertex_set
        tree_edge_set.add($e_m$)
        tree_vertex_set.add($V_m$)

    return tree_edge_set
```

The way that this algorithm works, is that on each run of every vertex, you always find the minimum edge from the starting point, and move on to the next one. Then you do the same until you run out of vertices.