Proof of edge-critical strongly-connected digraph generator algorithm

version 1.4

Index

- Definition
  1. Graph
  2. Strongly-connected digraph
  3. edge-critical strongly-connected digraph
  - Summary
- Problem Statement
  - Objective
  - Existing tools
- Prime digraph generation
  - Example and Motivation
  - Algorithm I - Find all prime digraphs
  - Results

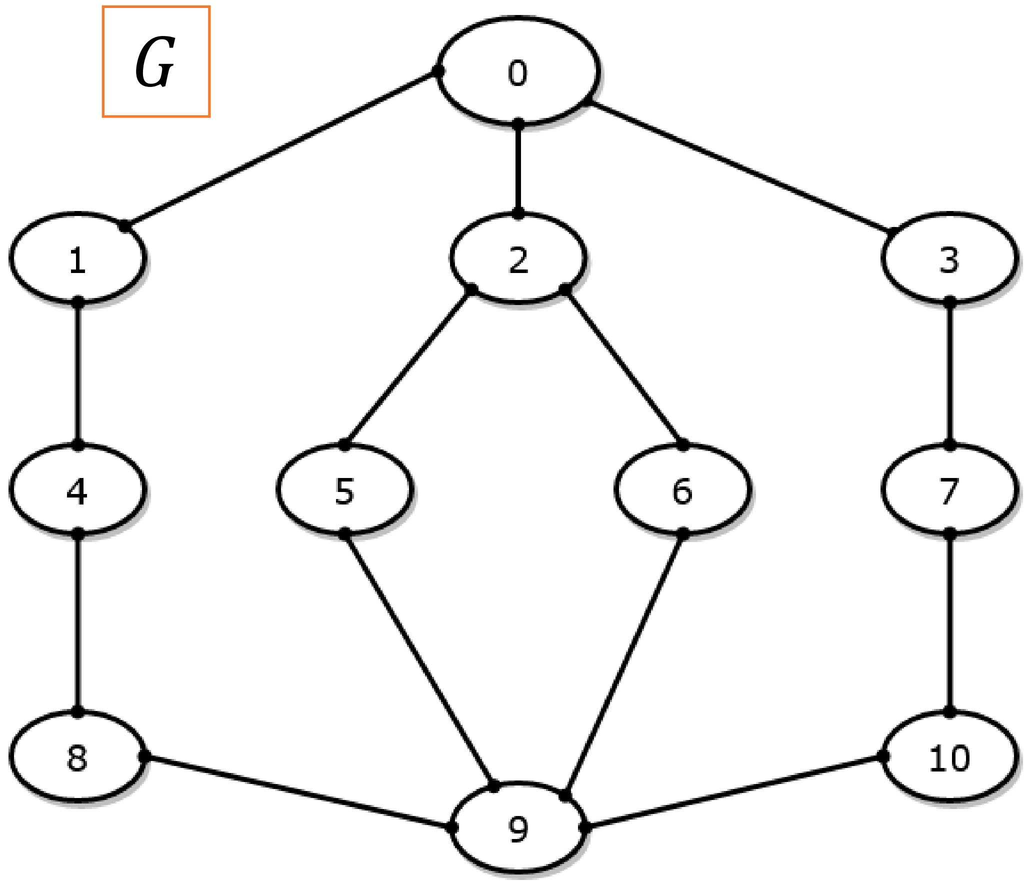
Index

- Proof of prime generator algorithm
    - Correctness of prime_generator
    - Completeness of prime_generator
    - Construction of BFS tree
    - Three cases
    - Case I – Two or more out-edges only
    - Case II – Two or more in-edges
    - Case III – One in-edge and at least one out-edge
- Conclusion
    - Conclusion
    - Further study
    - Reference

Graph[1]

A graph G is a triple consisting of a **vertex set** $\mathbf{V}(\boldsymbol{G})$, an **edge set** $\mathbf{E}(\boldsymbol{G})$, and a relation that associates with each edge two vertices (not necessarily distinct) called its endpoints.

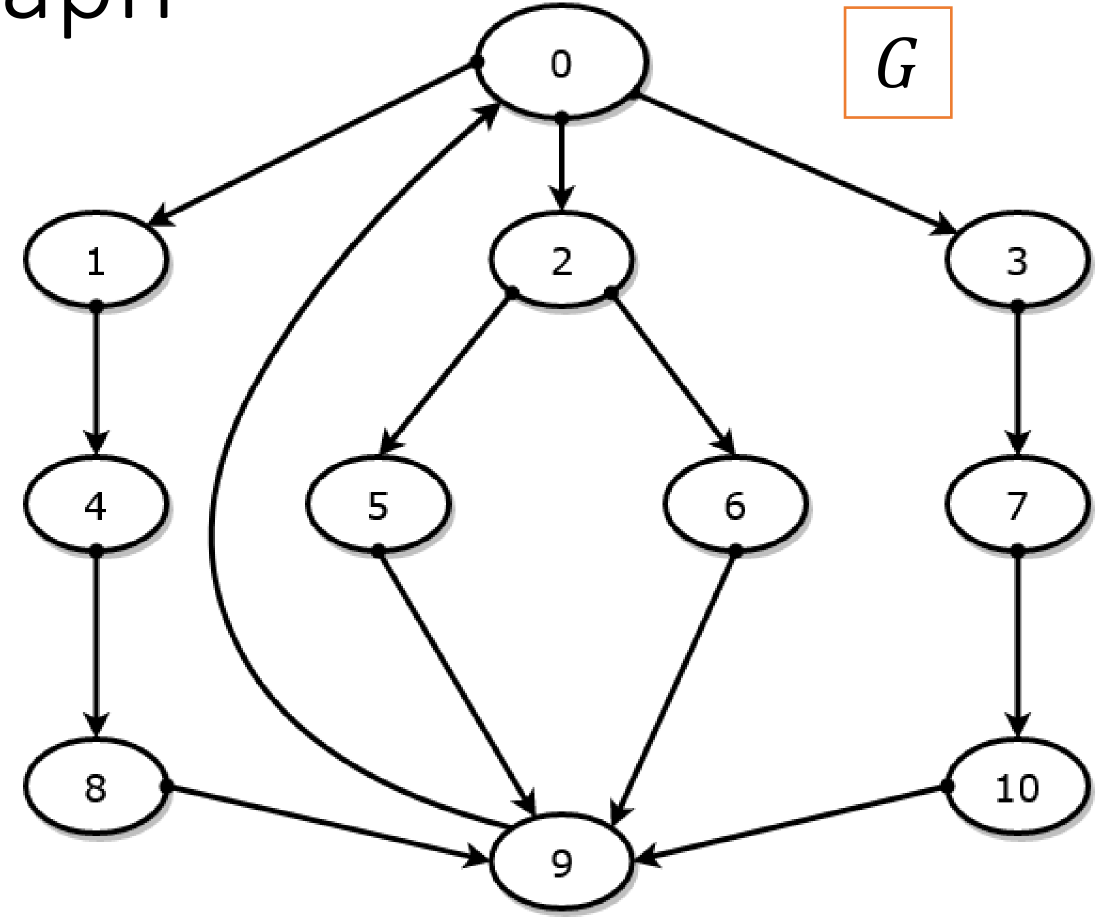
- The terms **vertex** and **node** may be used interchangeably.



$Here\ V(G) = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
$And\ E(G) = \{(0,1), (0,2), (0,3), (1,4), (2,5), (2,6), (3,7), (4,8), (5,9), (6,9), (7,10), (8,9), (9,10)\}$

strongly-connected digraph [3]

A digraph is strongly connected or strong if for each ordered pair u, v of vertices, there is a path from $u\ to\ v$.



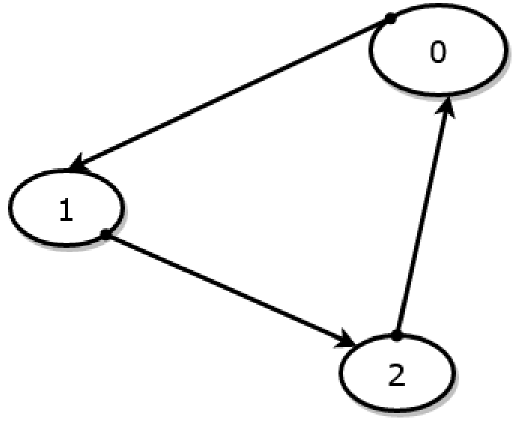
$Here\ V(G) = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
$And\ E(G) = \{(0,1), (0,2), (0,3), (1,4), (2,5), (2,6), (3,7), (4,8), (5,9), (6,9), (7,10), (8,9), (\mathbf{9}, \mathbf{0}), (10,9)\}$

edge-critical strongly-connected digraph

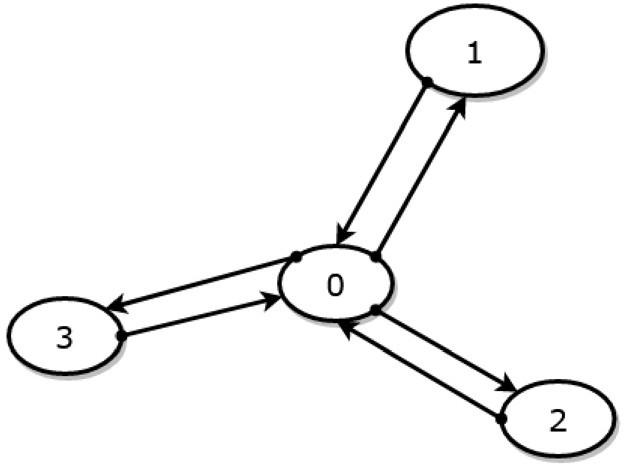
A strongly-connected digraph that ceases to be strongly-connected if **any of its edge is removed** is called edge-critical strongly-connected digraph.

Note: In the codebase [5], an edge-critical strongly-connected digraph is called **prime digraph.** We may use the terms interchangeably.

G



H



G and H are prime digraphs.

Note: All edge-critical graphs are 1-edge-connected graphs but not necessarily vice versa.

Definitions-Summary

Term	Definition
Graph	A graph G is a triple consisting of a **vertex set** $\mathbf{V}(\boldsymbol{G})$, an **edge set** $\mathbf{E}(\boldsymbol{G})$, and a relation that associates with each edge two vertices (not necessarily distinct) called its endpoints.
Directed Graph	A directed graph or digraph G is a triple consisting of a vertex set $\mathrm{V}(G)$, an edge set $\mathrm{E}(G)$, and a function assigning each edge an **ordered pair of vertices**.
strongly-connected Directed Graphs	A digraph is strongly connected or strong if for each ordered pair u, v of vertices, there is a path from $u\ to\ v$.
k-edge-connected digraph	A strongly-connected digraph is k-edge-connected directed graph if it remains strongly-connected whenever fewer than k edges are removed.
edge-critical strongly-connected digraph	A strongly-connected digraph that ceases to be strongly-connected if any of its edge is removed is called edge-critical strongly-connected digraph.
Isomorphic Graphs	Two graphs G and H are said to be isomorphic if there exists a function f from nodes of G to nodes of H, such that (u, v) is an edge of G if and only if $(f(u), f(v))$ is an edge of H.

Objective

- Find an algorithm to generate all prime (edge-critical strongly-connected) directed graphs of $n + 1$ nodes given a set of all prime digraphs of n nodes.
- Prove the algorithm's correctness and completeness.

Tools at hand

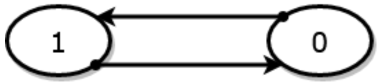
Following algorithms/tools that have been studied in detail is assumed to be present at hand for achieving the mentioned objectives.

1. strong-connection test[9,10]: An algorithm that tests whether the given graph is strongly-connected or not. It does so by finding the strongly connected components using Depth First Search algorithm. It is referred as **`is_strongly_connected`**`(digraph)`
2. Prime test [5]: An algorithm that tests whether given graph is edge-critical strongly-connected or not. It does so by removing an edge and checking for strong-connectedness. This is done for all edges. It is referred as **`is_prime`**`(digraph)`

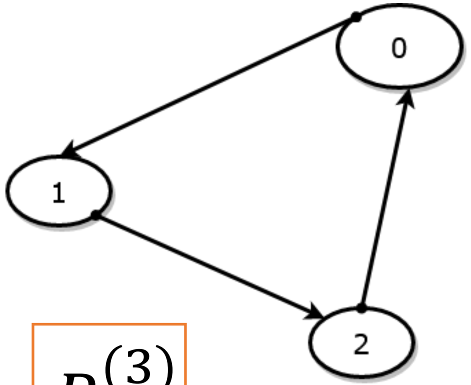
Note: **`is_strongly_connected`** is available in the networkx package while **`is_prime`** is available on link mentioned in reference.

Further Work: Find a better algorithm to conclude primality of directed graph.

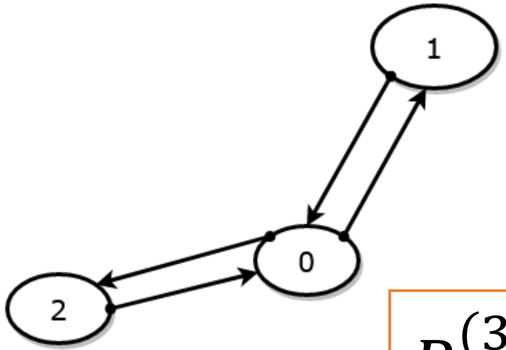
All prime digraphs of 2,3, and 4 nodes



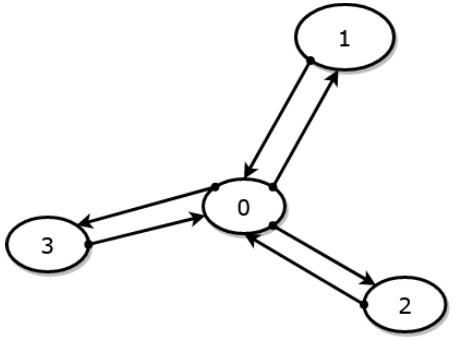
$P_1^{(2)}$



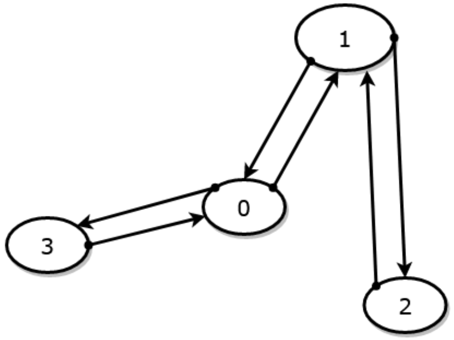
$P_1^{(3)}$



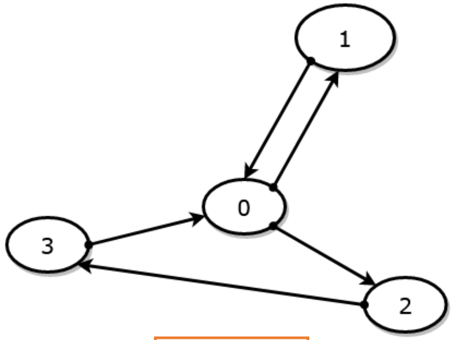
$P_2^{(3)}$



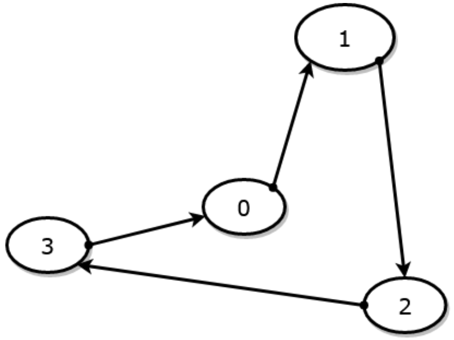
$P_1^{(4)}$



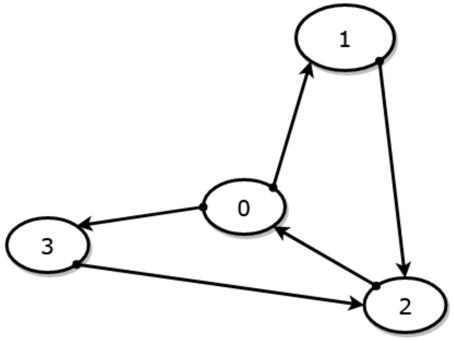
$P_2^{(4)}$



$P_3^{(4)}$



$P_4^{(4)}$



$P_5^{(4)}$

Motivation of Prime digraphs

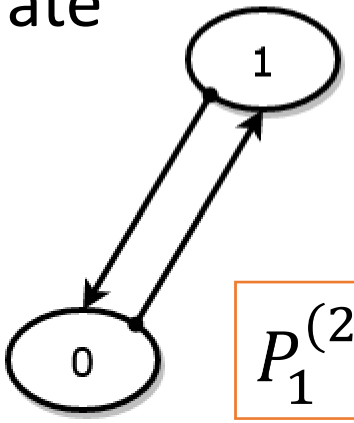
- The prime (edge-critical strongly-connected) digraphs are basic unit of construction for any strongly-connected graph.
- If even a single prime digraph is available for a given number of nodes, then one can add edges randomly with some probability p adhering to $G(n, p)$ Erdős–Rényi model[11]. This results into a digraph that is assured to be strongly-connected.
- In order to choose a random prime digraph for reduced bias in digraph selection, it is best to have list of all prime digraphs for a given node count n.

Algorithm I – Find all prime digraphs

- Given an exhaustive list of all prime digraphs CDG_n with n nodes, we employ Algorithm I to build CDG_{n+1}, a list of all prime digraphs with $n+1$ nodes.
- Code present at: https://github.com/oyeluckydps/directed_graph/blob/master/prime_DiGraph_Generator.py
- A digraph is chosen from CDG_n iteratively. It has nodes labelled from $0\ to\ n-1$.
- Start with an empty list CDG_{n+1}.
- Two new digraph are generated following these steps:
  - add a new node (n).
  - Add $(a, n)\ and\ (n, b)$ edges to the original graph to generate $first\_possible$.
  - Remove (a, b) from the $first\_possible$ to generate $second\_possible$.

Algorithm I – Find all prime digraphs

- There are n^2 possibilities for (a, b) and all of them are explored for new prime generation.
- Check if $first\_possible$ or $second\_possible$ is prime graph. If prime, then add it to list CDG_{n+1}.
- The correctness of Algorithm I to generate exhaustive list of all prime digraphs of $n + 1$ is proved in following slides.
- One may start with single prime digraph of 2 nodes and generate prime digraphs for other node values iteratively.



$\boxed{P_1^{(2)}}$

Algorithm I – Find all prime digraphs

```
def compute_next_primes(cdg_n):
        cdg_n1 = []; already_checked_digraphs = {}
        for DG_n in cdg_n:
                for in_edge, out_edge in new_edge_possibilities:
                        first_possible = DG_n.add_edges_from_list([in_edge, out_edge])
                        second_possible = first_possible.remove_edge(in_edge[0], out_edge[1])
                        if is_prime(first_possible):
                                cdg_n1.append(first_possible)
                        if is_prime(second_possible):
                                cdg_n1.append(second_possible)
        return all_primes
```

- **new_edge_possibilities** is list of all possibilities for in_edge and out_edge in following form:

$$\begin{aligned}&[\big((0,n),(n,0)\big),\big((1,n),(n,0)\big),\big((2,n),(n,0)\big),\dots\big((n-1,n),(n,0)\big),\\ &\ \big((0,n),(n,1)\big),\big((1,n),(n,1)\big),\big((2,n),(n,1)\big),\dots\big((n-1,n),(n,1)\big),\\ &\dots\\ &\big((0,n),(n,n-1)\big),\big((1,n),(n,n-1)\big),\dots\dots\ \big((n-1,n),(n,n-1)\big)]\end{aligned}$$

Result: Find all prime digraphs

- Algorithm I was used to generate all prime digraphs for 2 to 10 nodes. It can be found at: https://github.com/oyeluckydps/directed_graph/blob/master/using_isomorphic_hashCDG.csv
- Count of prime digraphs for given number of nodes is:

Number of nodes	Total Prime Digraphs
2	1
3	2
4	5
5	15
6	63
7	288
8	1526
9	8627
10	52021

Correctness of prime generator

- RTP: The prime generator algorithm outputs a set of graphs and each graph is a prime graph.
- A graph (either *first_possible* or *second_possible*) is only added to the CDG_n1 if it is prime i.e. if it passes the **is_prime** test thus each graph in the CDG_n1 is prime.

Completeness of prime generator

- To prove that all primes of n+1 nodes are generated when prime generator is employed, we instead prove a different result and show that this results lead to completeness of prime generator.
- We prove

$$A\ prime\ graph\ must\ contain\ a\ node\ with\ degree\ 2.$$

- Let us first prove that a prime graph of $n + 1\ nodes$, having a node with degree 2 implies that the prime graph will be present in the output of the Algorithm I.
- Without the loss of generalization assume that $node\ n$ of prime graph P has degree 2 and the edges are $e_1 = (\mathrm{a}, n)\ and\ e_2 = (n, b)$ on $node\ n$.
- Now, we require to prove that either
  - Graph G_1 with edges $\mathcal{E}(P)\backslash\{e_1, e_2\}$ and nodes $D(P)\backslash\{n\}$ is prime or
  - Graph G_2 with edges $(\mathcal{E}(P)\backslash\{e_1, e_2\}) \cup \{(a, b)\}$ and nodes $D(P)\backslash\{n\}$ is prime

Completeness of prime generator

- Now, we require to prove that either
  - Graph G_1 with edges $\mathcal{E}(P)\backslash\{e_1, e_2\}$ and nodes $D(P)\backslash\{n\}$ is prime or
  - Graph G_2 with edges $(\mathcal{E}(P)\backslash\{e_1, e_2\}) \cup \{(a, b)\}$ and nodes $D(P)\backslash\{n\}$ is prime
- First, we show that there exists a path from $any\ node\ x\ to\ another\ node\ y$ in the graph G_2. This is trivial. If we assume the contradictory that there doesn't exist a path from $node\ x\ to\ node\ y$ then the same would be true for original graph P too. This is a contradiction as P is prime graph.
- Next, any of the edges in G_2 except the edge (a, b) cannot be redundant.
- To prove this, let us assume the contradictory, that removing an edge of G_2 (other than (a, b)) results in strongly connected graph. Then the same would be true for original graph P too but that leads to a contradiction as P is a prime graph.
- Finally, (a, b) may be a redundant edge in which case the graph G_1 is prime otherwise, G_2 is prime.

Completeness of prime generator

- This proves that a prime graph of n+1 nodes with a node having two edges can always be found from a prime graph of n nodes by either
  - Adding a new node ‘n’ and adding an in-edge and out-edge from two nodes say a and b, OR
  - Removing an edge (a, b) followed by addition of node n and edges (a, n) and (n, b).

- Now, we shall prove that any prime graph has at least one node with two edges.

Construction of BFS tree

- Now, we shall prove that any prime graph has at least one node with two edges.
$$\min(\deg(n)) = 2 \; for \; n \in Nodes(P)$$
- Let us assume the contradictory and assume that each node has at least 3 edges.
- To prove this we construct a Breadth First Tree using the following mechanism.
  - Choose any node A in the prime graph P.
  - Using a as the root, add all nodes x as children of A, if (x, A) is an edge.
  - Repeat this process iteratively for the children of A, their children and so on.
  - Do not readd already added nodes including the node A in the tree construction.

Three cases

- Now, we shall concentrate on the three cases for the leafs of the BFS tree. It is known that the leaf has at least 3 edges. One of this is represented in the BFS tree, so there must be at least two other edges that are not represented in the BFS.
- Case 1:

$$The\ leaf\ \ has\ two\ or\ more\ out-edges\ only.$$

- Case 2:

$$The\ leaf\ \ has\ two\ or\ more\ in-edges.$$

- Case 3:

$$The\ leaf\ \ has\ one\ in-edge\ and\ at\ least\ one\ out-edge.$$

Two or more out-edges only

$$Case\ 1{:}\ The\ leaf\ has\ two\ or\ more\ out-edges\ only$$

- If the node has only out-edges (as the edge on the BFS tree is also an out-edge), then there does not exist a path from any other node to this node. So clearly, case 1 leads to contradiction.

Three cases

- Now, we shall concentrate on the three cases for the leafs of the BFS tree. It is known that the leaf has at least 3 edges. One of this is represented in the BFS tree, so there must be at least two other edges that are not represented in the BFS.
- ~~Case 1:~~

$$\cancel{The\ leaf\ \ has\ two\ or\ more\ out-edges\ only}.$$

- Case 2:

$$The\ leaf\ \ has\ two\ or\ more\ in-edges.$$

- Case 3:

$$The\ leaf\ \ has\ one\ in-edge\ and\ at\ least\ one\ out-edge.$$

Two or more in-edges

$$Case\ 2\text{: } The\ leaf\ has\ two\ or\ more\ in - edges$$

- We prove the base case, i.e. there exists a contradiction for two in-edges on the leaf.
- Suppose the leaf node is l and the two edges are (x, l) and (y, l) which are not a part of the BFS tree.
- Now there must exist a path from A to either x or y or both, otherwise there is no path from A to l.

Two or more in-edges

$$Case\ 2\text{: } The\ leaf\ has\ two\ or\ more\ in-edges.$$

- Without the loss of generality let us assume that there exists a path from A to x. This leads to either of the two cases:
  - All paths from A to x contain (y, l) edge
  - There exists a path from A to x independent of (y, l) edge.
- The first case implies that there exists a path from x to A to y to l. So, the (x, l) edge is redundant.
- In the second case, there exists a path y to A to x to l. So, the (y, l) edge is redundant.
- Thus it is not possible to have two (or more) in-edges on the leaf.
  - And this holds independent or number of out-edges on the leaf node.

Three cases

- Now, we shall concentrate on the three cases for the leafs of the BFS tree. It is known that the leaf has at least 3 edges. One of this is represented in the BFS tree, so there must be at least two other edges that are not represented in the BFS.
- ~~Case 1:~~

$$\cancel{The\ leaf\ has\ two\ or\ more\ out-edges\ only}.$$

- ~~Case 2:~~

$$\cancel{The\ leaf\ has\ two\ or\ more\ in-edges}.$$

- Case 3:

$$The\ leaf\ has\ one\ in-edge\ and\ at\ least\ one\ out-edge.$$

1 in-edge and at least 1 out-edge

$Case\ 3$: $The\ leaf\ has\ one\ in-edge\ and\ at\ least\ one\ out-edge.$

- We have proved that the leafs cannot have two or more in-edges or only out-edges. So, all leafs must have one in-edge and at least one out-edge that are not a part of BFS tree.
- Let us focus on a leaf l with edges (x, l) and (l, y) which are not the part of BFS tree.
- Then there cannot exist a path from A to l, independent of (x, l)
  - Otherwise it would imply that there exists a path from x to A to l, thus (x, l) is redundant.
- Also there cannot exist a path from A to y independent of (l, y)
  - Otherwise it would imply that there exists a path from l to A to y, thus (l, y) is redundant.

1 in-edge and at least 1 out-edge

$$Case\ 3\text{: } The\ leaf\ \ has\ one\ in - edge\ and\ at\ least\ one\ out - edge.$$

- Also there cannot exist a path from A to y independent of (l, y)
  - Otherwise it would imply that there exists a path from l to A to y, thus (l, y) is redundant.
- But this implies, that y must be a leaf.
  - If it has a child, let say y', then there is no path from A to y' in the prime graph P – which is not possible.
- Since y is a leaf so y must also have 1 in-edge and at least 1 out-edge and the in-edge is (l, y). Let the out edge be (y, z).
- Using the same logic, z must be leaf on the BFS tree with 1 in-edge and at least 1 out-edge.

1 in-edge and at least 1 out-edge

$Case\ 3$: $The\ leaf\ has\ one\ in - edge\ and\ at\ least\ one\ out - edge$.

- Using the same logic, z must be leaf on the BFS tree with 1 in-edge and at least 1 out-edge.
- This logic might repeat infinitely but the number of nodes on the leaf is finite.
- Thus it is not possible to have one in-edge and one or more out-edge either.
- **Hence, we have found contradiction in all the three cases when assumed that all nodes have three or more edges.**
- **Thus all prime graphs must have at least one node with 2 edges.**

Conclusion

- A algorithm to find the set of all prime (edge-critical strongly-connected) digraphs for any given number of nodes is found. The algorithm is iterative in nature.
- A theoretical proof for correctness and completeness of Alorithm I to generate exhaustive list of all prime digraphs of $n+1$ nodes is established.

Further study

- This is the first step in study of random strongly-connected directed graphs and further study may be conducted in following direction:
    1. Study of time/space complexity of the mentioned algorithm.
    2. Optimization of the mentioned algorithm.
    3. Optimization of the is_prime algorithm.
    4. Peer review of correctness and completeness of Algorithm I - to generate exhaustive list of all prime digraphs of $n + 1$ is provable.
    5. Deployment of strongly-connected digraphs in network studies.
    6. Resilience test of AI agents on games with underlying strongly-connected digraphs as playground.

Reference

1. Douglas B West (2001). 1.1.2 Introduction to Graph Theory. Pearson Education, Inc
2. Douglas B West (2001). 1.4.2 Introduction to Graph Theory. Pearson Education, Inc
3. Douglas B West (2001). 1.4.12 Introduction to Graph Theory. Pearson Education, Inc
4. Wikipedia contributors (2022a, June 25). K-edge-connected graph. https://en.wikipedia.org/wiki/K-edge-connected_graph
5. Verma, A. Kumar (2022, December 11). Directed Graph and analysis. GitHub. https://github.com/oyeluckydps/directed_graph
6. Wikipedia contributors (2022c, November 28). Graph isomorphism. Wikipedia. https://en.wikipedia.org/wiki/Graph_isomorphism
7. L. P. Cordella, P. Foggia, C. Sansone, M. Vento (2001). "An Improved Algorithm for Matching Large Graphs", 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen, pp. 149-159. https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.101.5342
8. Shervashidze, Nino, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt (2011). Weisfeiler Lehman Graph Kernels. Journal of Machine Learning Research. http://www.jmlr.org/papers/volume12/shervashidze11a/shervashidze11a.pdf

Reference

9. R. Tarjan (1972). Depth-first search and linear graph algorithms. SIAM Journal of Computing 1(2):146-160.
10. E. Nuutila and E. Soisalon-Soinen (1994). On finding the strongly connected components in a directed graph. Information Processing Letters 49(1): 9-14
11. Erdős, P.; Rényi, A. (1959). "On Random Graphs. I" (PDF). Publications Mathematicae. **6**: 290–297.