GSoC 19: GRAPH C++ Boost graph algorithms for pgRouting

1. Contact Details

2. Title

3. Brief Project Description

4. State of the Project Before GSoC

5. Benefits to Community

6. Deliverables

7. Timeline

8. Do you understand this is a serious commitment, equivalent to a full time paid summer internship or summer job?

9. Do you have any known time conflicts during the official coding period?

10. Studies
    What is your School and degree?
    Would your application contribute to your ongoing studies/degree? If so, how?

11. Programming and GIS
    Computing experience
    GIS experience as a user
    GIS programming and other software programming

12. GSoC Participation
    Have you participated to GSoC before?
    Have you submitted/will you submit another proposal for this year's GSoC to a different org?

13. pgRouting Application Requirements

14. Detailed Proposal
    topological sort
        Proposed Signature
        Usage
    transitive closure:
        Proposed Signature
        Usage
    pgr_lengauer_tarjan_dominator_tree

Proposed Signature
Usage
15. References

1. Contact Details

- Name: Hang Wu
- Country: China
- Email: nike0good@gmail.com
- Phone: +86-18650996599
- Location: Xi’an, China, +8:00 GMT
- Github: https://github.com/nike0good
- Personal blog: blog.csdn.net/nike0good
- Personal blog: nike0good.com

2. Title

Implement GRAPH C++ Boost graph algorithms for pgRouting.

3. Brief Project Description

My project will focus on implementing:

- topological sort

  Topological sort is a sorting algorithm. It is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering.[1]

- transitive closure

  The concept of transitive closure can be thought of as constructing a data structure that makes it possible to answer reachability questions. That is, can one get from node a to node d in one or more hops? A binary relation tells you only that node a is connected to node b, and that node b is connected to node c, etc.[2]

- lengauer tarjan dominator tree

  A dominator tree is a tree where each node's children are those nodes it immediately dominates. Because the immediate dominator is unique, it is a tree. The start node is the root of the tree.[3]

I propose to add the above 3 algorithms into pgRouting during the GSoC period.

4. State of the Project Before GSoC

pgRouting currently does not have these algorithms implemented.

Topological sort is a common sorting algorithm of graph. However, a single standard function does not exist.

Floyd's algorithm implemented in pgRouting can also answer reachability question. However, it has a higher run-time complexity. Transitive closure is required

Also lengauer tarjan dominator tree is not implemented before in pgRouting. So far, a single standard function does not exist.

5. Benefits to Community

These three algorithms help users in some cases.

For instance, if a person wants to go to various places, and there are some restrictions, like a place should go after another. Then topological sort works.

Secondly, sometimes people there are some blocks or traffic jam, and people want to know whether it is possible to go to one place from another place. They can use transitive closure.

What's more, lengauer tarjan dominator tree helps users know which vertices are 'important'. That is to say, if node u dominate node v, it means node u is important because one cannot travel from the root to node v without node u.

6. Deliverables

1. Implementation of topological sort for pgRouting.
   I need to construct function pgr_topological_sort() and it will return one of the possible ordering according to the DAG.
2. Implementation of transitive closure for pgRouting.
   I need to construct function pgr_transitive_closure() and it will return which node it can reach.
3. Implementation of lengauer tarjan dominator tree for pgRouting.
   I need to construct a function pgr_lengauer_tarjan_dominator_tree() and it will return which node it can reach.
   To construct dominator tree from a graph G, we have such steps:
    1. choose a node as root R
    2. DFS and relabeled the nodes.
    3. Calc the sdom.
    4. Calc some nodes' idom from sdom according to the formula.
    5. Again, calc the rest nodes' idom.

   PS: Union-Find Set is required during Step 4 and Step 5.

Each implemented function will be delivered with the relevant documentation and tests included.

7. Timeline

Community Bonding Period

➢ Set up the development environment.

- ➢ Interact with mentors, introduce myself to the community and actively get involved in the discussion.
- ➢ Get familiar with pgRouting's development style. Understand expected coding, documentation and testing standards set by pgRouting.
- ➢ Set up wiki page to keep track of weekly progress.
- ➢ Develop a better understanding of PostgreSQL, PostGIS, Pl/pgSQL and how they interact with pgRouting.
- ➢ Learn to create unit tests using pgTap.
- ➢ Implement simple dummy functions to better understand pgRouting.

First Coding Period

Time Period	Proposed Work
Week 1 (May 27th - June 2nd)	➢ Design pgr_topological_sort() function.
Week 2 (June 2nd - June 9th)	➢ Create a basic skeleton for documentation and tests.
Week 3 (June 10th - June 16th)	➢ Implement pgr_topological_sort() function along its helper files. ➢ Basic testing.
Week 4 (June 17th - June 23rd)	➢ Prepare a report for First Evaluation.

➢ First Evaluation Period:

- ○ Submit working pgr_topological_sort() function (albeit without documentation and pgTap tests).
- ○ Mentors evaluate me and vice-versa.

Second Coding Period

Time Period	Proposed Work

Week 5 (June 24th - June 30th)	➢ Work on feedback provided from the first evaluation. ➢ Prepare documentation for pgr_topological_sort() function.
Week 6 (July 1st - July 7th)	➢ Complete testing along with writing pgTap tests for pgr_topological_sort() function.
Week 7 (July 8th - July 14th)	➢ Design pgr_transitive_closure() function. ➢ Create a basic skeleton for documentation and tests.
Week 8 (July 15th - July 21st)	➢ Begin implementation of pgr_transitive_closure() function. ➢ Create a basic skeleton for documentation and tests. ➢ Design pgr_lengauer_tarjan_dominator_tree() function. ➢ Prepare a report for Second Evaluation.

➢ Second Evaluation Period:

- Submit documentation and all tests for pgr_topological_sort() function.
- Submit design of pgr_transitive_closure() function pgr_lengauer_tarjan_dominator_tree() function and along with a basic skeleton of its helper files, documentation and tests.
- Mentors evaluate me and vice-versa.

Third Coding Period

Time Period	Proposed Work
Week 9 (July 22nd - July 28th)	➢ Work on feedback provided from the second evaluation. ➢ Complete the implementation of pgr_transitive_closure() function.Each implemented function will be delivered with the relevant documentation and tests included.
Week 10 (July 29th - August 4th)	➢ Begin implementation of pgr_lengauer_tarjan_dominator_tree() function.
Week 11 (August 5th - August 11th)	➢ Complete testing along with writing pgTap tests for pgr_transitive_closure() function and pgr_lengauer_tarjan_dominator_tree() function.
Week 12 (August 12th - August 18th)	➢ Review, complete and finalize all documentation and tests. ➢ Create a detailed final report.

- ➢ **Final Evaluation Period:**
    - Submit complete project with all required functions, documentation and unit tests.
    - Submit final report and evaluation of mentors.

8. Do you understand this is a serious commitment, equivalent to a full time paid summer internship or summer job?

Yes, I understand that this is a serious commitment and the expectations from me are the same as any other full-time internship/job. I am ready to put in my best efforts, improve pgRouting and help the community.

9. Do you have any known time conflicts during the official coding period?

No.

10. Studies

What is your School and degree?

School:	Xi’an Jiaotong University
Degree:	Master of Technology in Computer Science and Technology

Would your application contribute to your ongoing studies/degree? If so, how?

Yes, this application will contribute to my ongoing Degree.

I am an algorithmic coder, and I always represent my university to host algorithm competitions. However, I lack the experience of practical projects. This project broadens my vision and makes a good influence on my resume.

11. Programming and GIS

Computing experience

- **Programming Languages:** C++, C, Python 3, MATLAB, Java, Pascal, HTML, CSS, SQL
- **Operating Systems:**Windows 10, MacOS
- **Tools:** Git
- **Relevant Courses Completed:** Data Structures, Java(Android).

GIS experience as a user

GIS programming and other software programming

- Worked on “Wuzi Chess”(Andriod Project) in Java&Android Class.
- Graduation Project, Research of Knowledge Graph for medicial.
- Public C++ related programming/projects:
    - Code submissions on various competitive programming platforms such as Codeforces, BZOJ, POJ, vjudge, Topcoder, etc. I have a small Github repository of a few C++ algorithm templates I use frequently in various algorithmic competitions.
    - Set problems for contest, such as ACM-ICPC, Asia Shenyang Regional Contest

12. GSoC Participation

Have you participated in GSoC before?

No, I have not participated in GSoC before. This year will be my first time applying.

Have you submitted/will you submit another proposal for this year's GSoC to a different org?

No.

13. pgRouting Application Requirements

The requirements for applying to pgRouting(under OSGeo) are mentioned here - https://github.com/pgRouting/pgrouting/wiki/GSoC-Ideas%3A-2019#pgrouting-application-requirements

The links to the respective issues:

- issue: Build locally pgRouting -> https://github.com/nike0good/GSoC-pgRouting/issues/1

- issue: Get familiar with C++-> https://github.com/nike0good/GSoC-pgRouting/issues/2
- issue: Get familiar with pgRouting on Github -> https://github.com/nike0good/GSoC-pgRouting/issues/3

14. Detailed Proposal

topological sort

A topological sort or topological ordering of a directed graph is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering. For instance, the vertices of the graph may represent tasks to be performed, and the edges may represent constraints that one task must be performed before another; in this application, a topological ordering is just a valid sequence for the tasks. A topological ordering is possible if and only if the graph has no directed cycles, that is, if it is a directed acyclic graph (DAG). Any DAG has at least one topological ordering, and algorithms are known for constructing a topological ordering of any DAG in linear time.[1]

complexity: O(|V|+|E|).

code in C++:

https://paste.ubuntu.com/p/6hwWGDJ7XD/

https://blog.csdn.net/nike0good/article/category/1273967

For instance,the DAG is

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

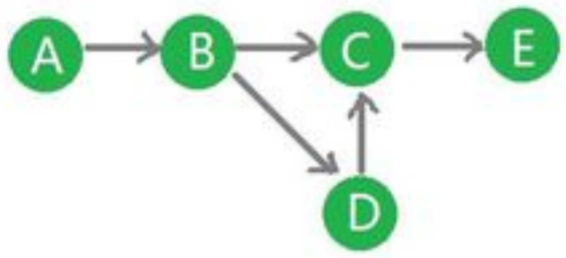

Fig: Example of DAG[4]

We label id $A.id=1 ,B.id = 2 ,\cdots, E.id=5$

And the topological sorted array of this graph is {A,B,D,C,E} (1,2,4,3,5 for id)

Proposed Signature

The possible variants are:

- pgr_topological_sort()

```
pgr_topological_sort(edges_sql )
RETURNS SET OF (seq, sorted_v)
```

Parameters[6]:

edges_sql:	an SQL query, which should return a set of rows with the following columns:

Edges_sql[6]: It should be an sql query which returns the following:

Column	Type	Default	Description
id	`ANY-INTEGER`		Identifier of the edge.
source	`ANY-INTEGER`		Identifier of the first end point vertex of the edge.
target	`ANY-INTEGER`		Identifier of the second end point vertex of the edge.
cost	`ANY-NUMERICAL`		Weight of the edge (source, target) ● When negative: edge (source, target) does not exist, therefore it's not part of the graph.
reverse_cost	`ANY-NUMERICAL`	-1	Weight of the edge (target, source), ● When negative: edge (target, source) does not exist, therefore it's not part of the graph.

Where:

ANY-INTEGER:	SMALLINT, INTEGER, BIGINT

ANY-NUMERICAL:	SMALLINT, INTEGER, BIGINT, REAL, FLOAT

Description of Return Values:

Column	Type	Description
seq	`INT`	Sequential value starting from 1.
sorted_v	`BIGINT`	Identifier the ordering of vertex.

Usage

The examples[7] in this section use the following Network for queries marked as directed and only cost column is used

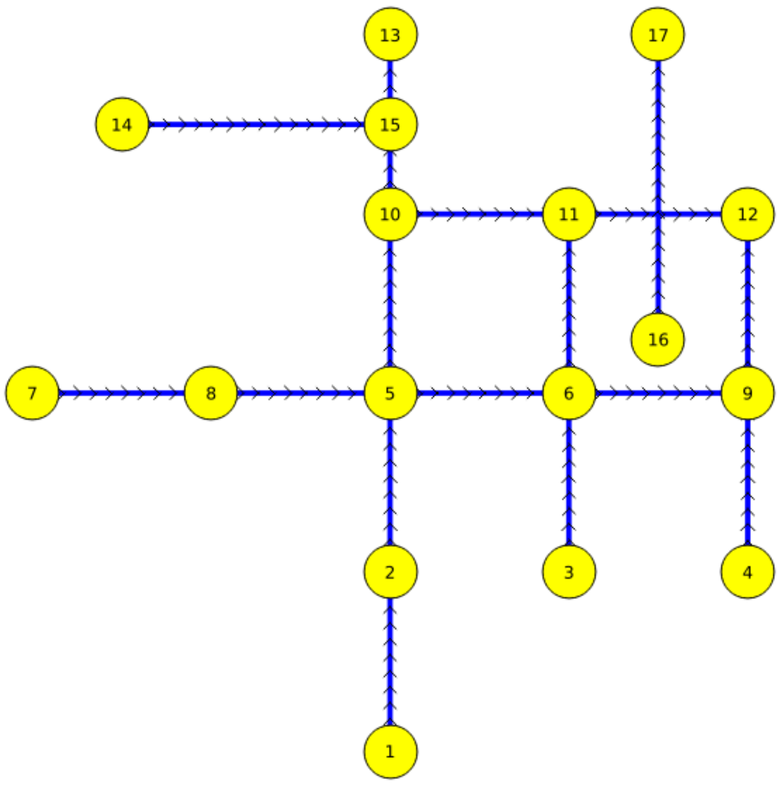

Fig: Example[7]

```
SELECT * FROM pgr_topological_sort(
    'SELECT id, source, target FROM edges_sql'
);
1,2,3,4,7,8,5,6,9,10,11,12,15,14,13,16,17
 seq | sorted_v
-----+---------
   1  |        1
   2  |        2
   3  |        3
   4  |        4
   5  |        7
   6  |        8
   7  |        5
   8  |        6
   9  |        9
   10 |        10
   11 |        11
   12 |        12
   13 |        14
   14 |        15
   15 |        13
   16 |        16
   17 |        17


(17 rows)
```

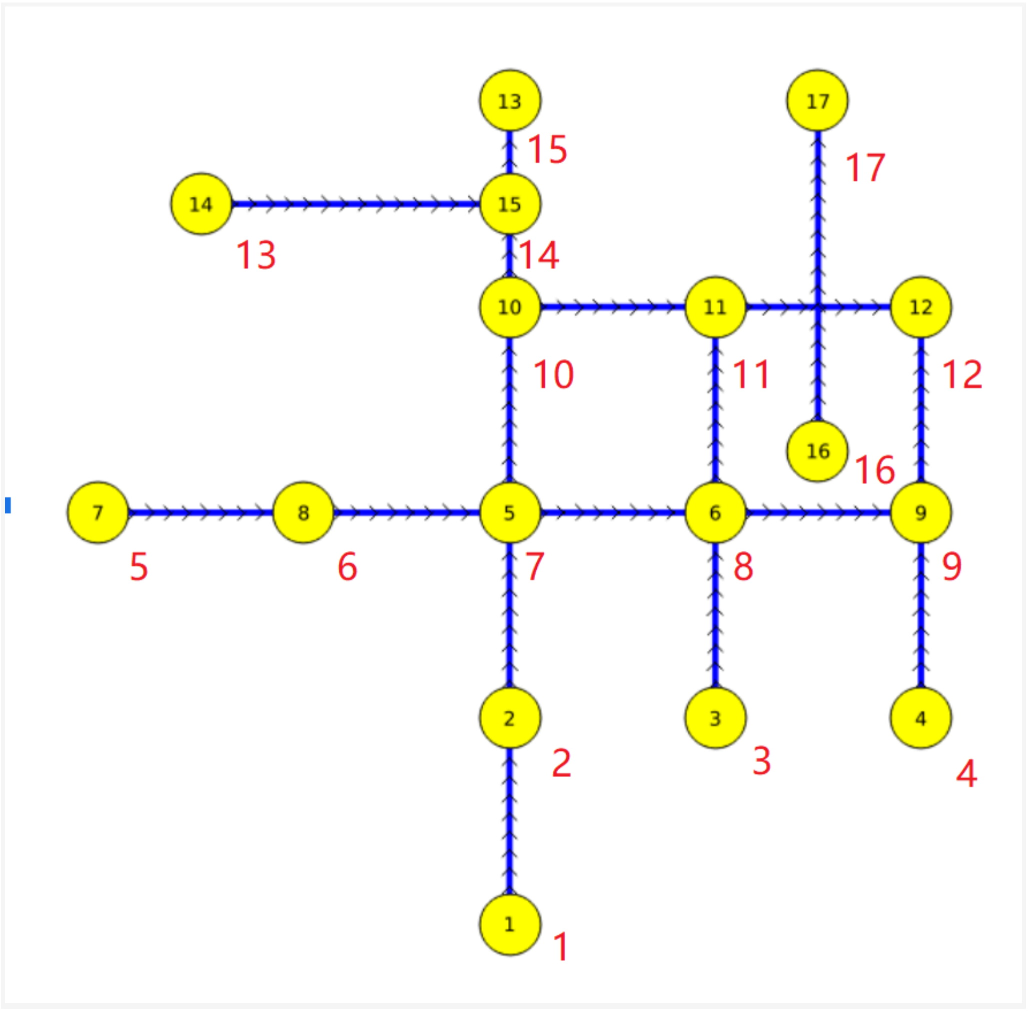

Fig: the ordering of the vertexs of the example DAG[7]

The figure tells us the result of sorted_v, topological ordering of a directed graph.

transitive closure:

The concept of transitive closure can be thought of as constructing a data structure that makes it possible to answer reachability questions. That is, can one get from node a to node d in one or more hops? A binary relation tells you only that node a is connected to node b, and that node b is connected to node c, etc. After the transitive closure is constructed, as depicted in the following figure, in an O(1) operation one may determine that node d is reachable from node a. The data structure is typically stored as a matrix, so if matrix[1][4] = 1, then it is the case that node 1 can reach node 4 through one or more hops.[2]

From my perspective, transitive closure is the basic idea of Floyd algorithm. We can know whether it is possible to find a path from node $u$ to node $v$ or not. Also we can use bitset(data structure) with bit operations to solve many problems.

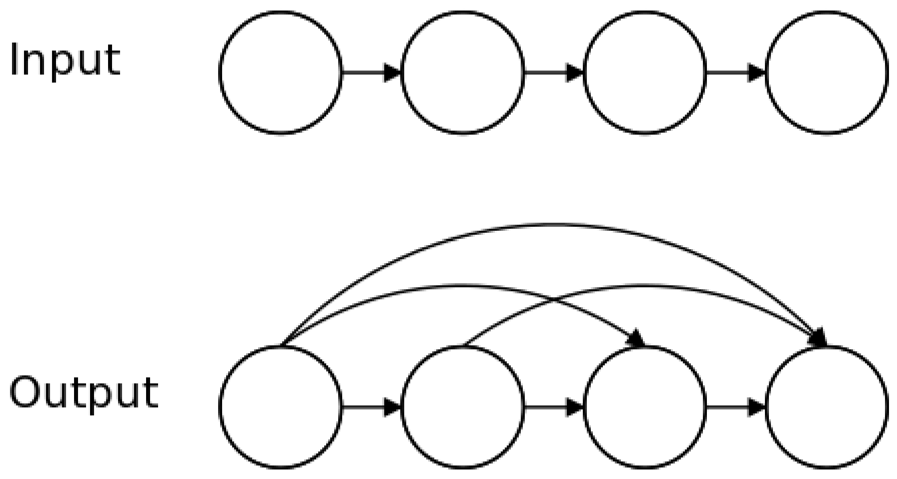

Fig: The picture was found in Wikipedia[2]

Proposed Signature

The possible variants are:

- pgr_transitive_closure()

```
pgr_transitive_closure(edges_sql )
RETURNS SET OF (seq, target_array )
```

Parameters[6]:

edges_sql:	an SQL query, which should return a set of rows with the following columns:

Edges_sql[6]: It should be an sql query which returns the following:

Column	Type	Default	Description
id	ANY-INTEGER		Identifier of the edge.
source	ANY-INTEGER		Identifier of the first end point vertex of the edge.

target	`ANY-INTEGER`		Identifier of the second end point vertex of the edge.
cost	`ANY-NUMERICAL`		Weight of the edge (source, target) ● When negative: edge (source, target) does not exist, therefore it’s not part of the graph.
reverse_cost	`ANY-NUMERICAL`	-1	Weight of the edge (target, source), ● When negative: edge (target, source) does not exist, therefore it’s not part of the graph.

Where:

ANY-INTEGER:	SMALLINT, INTEGER, BIGINT
ANY-NUMERICAL:	SMALLINT, INTEGER, BIGINT, REAL, FLOAT

Description of Return Values:

Column	Type	Description
seq	`INT`	Sequential value starting from 1.
vid	`BIGINT`	Identifier of the starting vertex.
target_array	`BIGINT[]`	Identifier of the union of the edges.

Usage

The examples[7] in this section use the following Network for queries marked as directed and only cost column is used

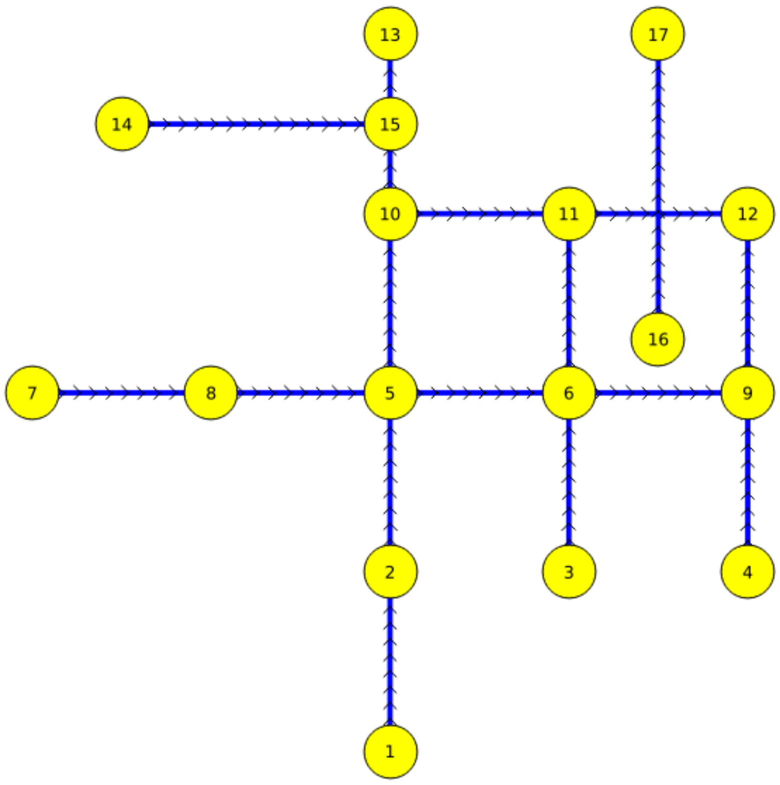

Fig: Example[7]

```
SELECT * FROM pgr_transitive_closure(
    'SELECT id, source, target FROM edges_sql'
);
 seq | vid| target_array
-----+----+-----
   1   | 1 |   {2,5,6,9,10,11,12,13,15}
   2   | 2 |   {5,6,9,10,11,12,13,15}
   3   | 3 |   {6,9,11,12}
   4   | 4 |    {9,12}
   5   | 5 |    {6,9,10,11,12,13,15}
   6   | 6 |    {9,11,12}
   7   | 7 |    {5,6,8,9,10,11,12,13,15}
   8   | 8 |   {5,6,9,10,11,12,13,15}
   9   | 9 |    {12}
   10  |10 |    {11,12,13,15}
   11  |11 |     {12}
   12  |12 |       {}
   13  |13 |     {}
   14  |14 |     {13,15}
   15  |15 |     {13}
   16  |16 |     {17}
   17  |17 |     {}

(17 rows)
```

For instance, what can 1 travel to:

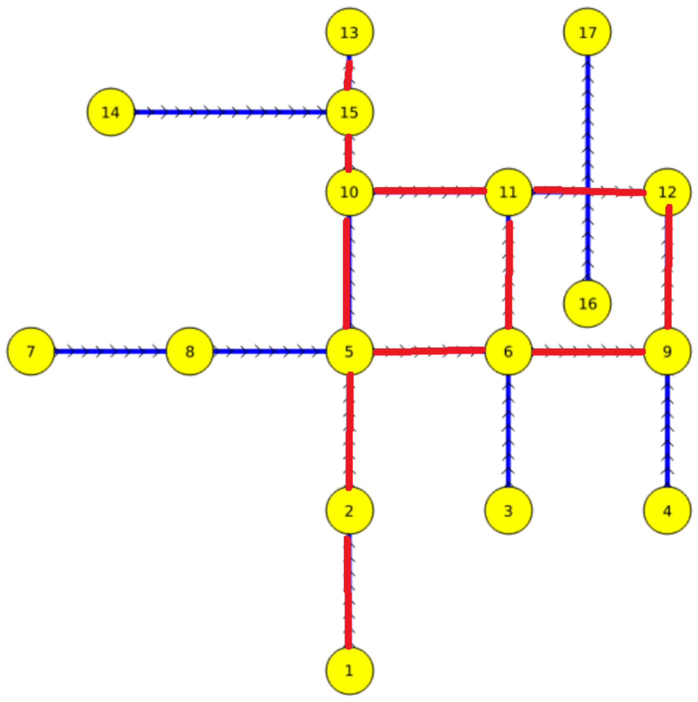

Fig: Example of reachability of vextex 1[7]

it is possible to find a path from node $1$ to node $2,5,6,9,10,11,12,13,15$.

pgr_lengauer_tarjan_dominator_tree

In computer science, in control flow graphs, a node d dominates a node n if every path from the entry node to n must go through d. Notationally, this is written as d dom n (or sometimes d {\displaystyle \gg } \gg n). By definition, every node dominates itself.

There are a number of related concepts:

A node d strictly dominates a node n if d dominates n and d does not equal n.

The immediate dominator or idom of a node n is the unique node that strictly dominates n but does not strictly dominate any other node that strictly dominates n. Every node, except the entry node, has an immediate dominator.

The dominance frontier of a node d is the set of all nodes n such that d dominates an immediate predecessor of n, but d does not strictly dominate n. It is the set of nodes where d's dominance stops.

A dominator tree is a tree where each node's children are those nodes it immediately dominates. Because the immediate dominator is unique, it is a tree. The start node is the root of the tree.[3]

We can use dominator tree to answer which we can not access from one specific node. Also the data structure was implemented in the boost.

To construct dominator tree from a graph G, we have such steps:

6. choose a node as root R
7. DFS and relabeled the nodes.
8. Calc the sdom.
9. Calc some nodes' idom from sdom according to the formula.
10. Again, calc the rest nodes' idom.

PS: Union-Find Set is required during Step 4 and Step 5.[5]

Proposed Signature

The possible variants are:

- pgr_lengauer_tarjan_dominator_tree()

```
pgr_lengauer_tarjan_dominator_tree(edges_sql,root )
RETURNS SET OF (id, sdom, idom)
```

Parameters:

Parameter	Type	Default	Description
edges_sql	`TEXT`		**Described above.**
root	`BIGINT`	**0**	**Identifier of the root vertex of the graph.**

Edges_sql[6]: It should be an sql query which returns the following:

Column	Type	Default	Description
id	**`ANY-INTEGER`**		**Identifier of the edge.**

source	`ANY-INTEGER`		Identifier of the first end point vertex of the edge.
target	`ANY-INTEGER`		Identifier of the second end point vertex of the edge.
cost	`ANY-NUMERICAL`		Weight of the edge (source, target) • When negative: edge (source, target) does not exist, therefore it's not part of the graph.
reverse_cost	`ANY-NUMERICAL`	-1	Weight of the edge (target, source), • When negative: edge (target, source) does not exist, therefore it's not part of the graph.

Where:

ANY-INTEGER:	SMALLINT, INTEGER, BIGINT
ANY-NUMERICAL:	SMALLINT, INTEGER, BIGINT, REAL, FLOAT

Description of Return Values:

Parameter	Type	Default	Description
id	`BIGINT`		Sequential value starting from 1.
vid	`BIGINT`		Identifier of the vertex.

sdom	`BIGINT`	**0**	**Identifier of the sdom vertex of the id vertex.** **If path from root vertex to id vertex does not exist, return the default value.**
idom	`BIGINT`	**0**	**Identifier of the idom vertex of the id vertex.** **If path from root vertex to id vertex does not exist, return the default value.**

Usage

The examples[7] in this section use the following Network for queries marked as directed and only cost column is used

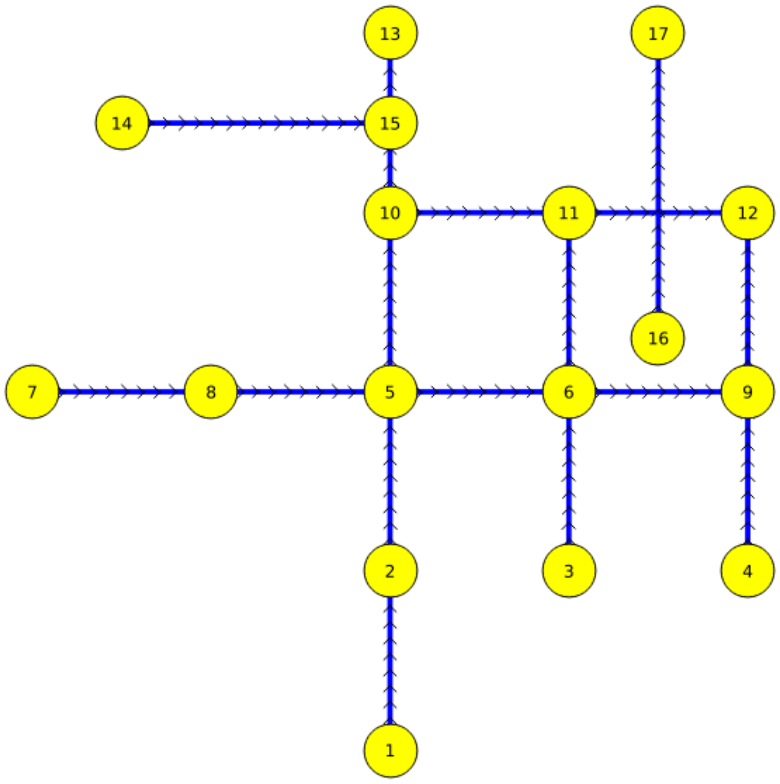

Fig: Example[7]

```
SELECT * FROM pgr_lengauer_tarjan_dominator_tree(
    'SELECT id, source, target FROM edges_sql',1
);
|id   | vid  | sdom | idom|
+-----+------+------+-----+
   1  |  1   |  0   |  0  |
   2  |  2   |  1   |  1  |
   3  |  3   |  0   |  0  |
   4  |  4   |  0   |  0  |
   5  |  5   |  2   |  2  |
   6  |  6   |  5   |  5  |
   7  |  7   |  0   |  0  |
   8  |  8   |  0   |  0  |
   9  |  9   |  6   |  6  |
   10 |  10  |  5   |  5  |
   11 |  11  |  6   |  5  |
   12 |  12  |  9   |  5  |
   13 |  13  |  10  | 15  |
   14 |  14  |  0   |  0  |
   15 |  15  |  10  | 10  |
   16 |  16  |  0   |  0  |
   17 |  17  |  0   |  0  |


(17 rows)
```

The sdom of 11 is 6 because there is a path go from 6 to 11 without going through nodes labeled less than 6.

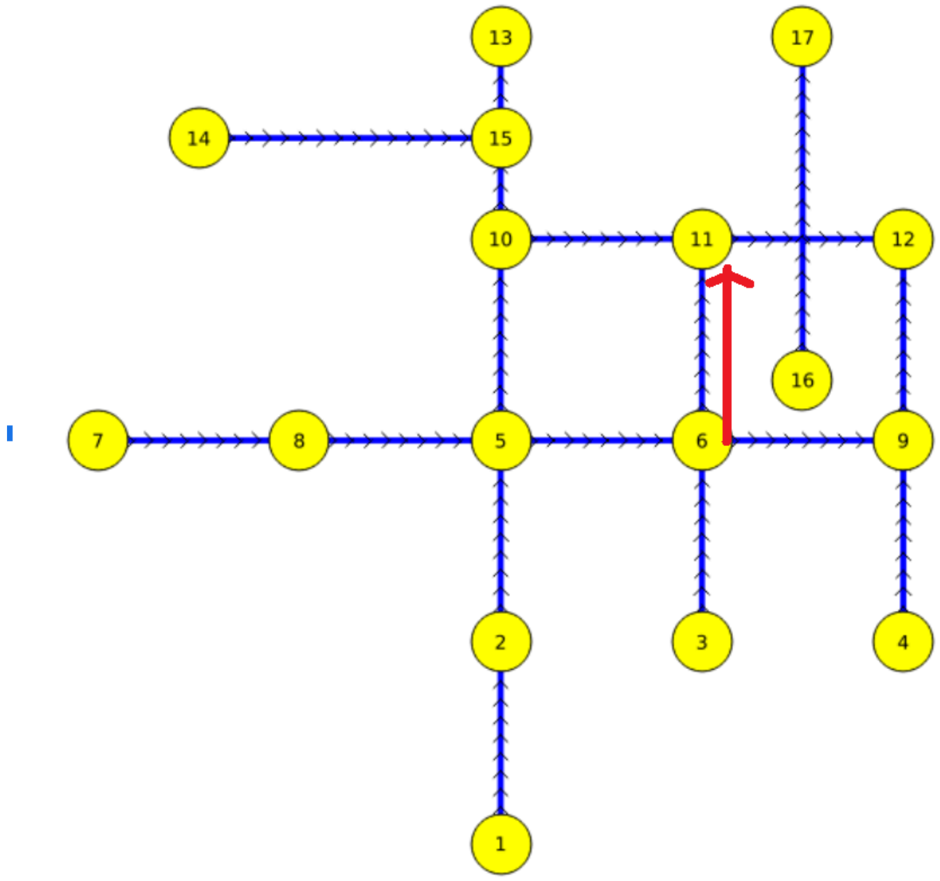

Fig: Example of sdom of vextex 11[7]

The idom of 11 is 6 because there is a path go from 5 to 11 and it is impossible to go from 1 to 11 without vertex 5, as well as vertex 6 is the unique node that strictly dominates 11 but does not strictly dominate any other node that strictly dominates 11.

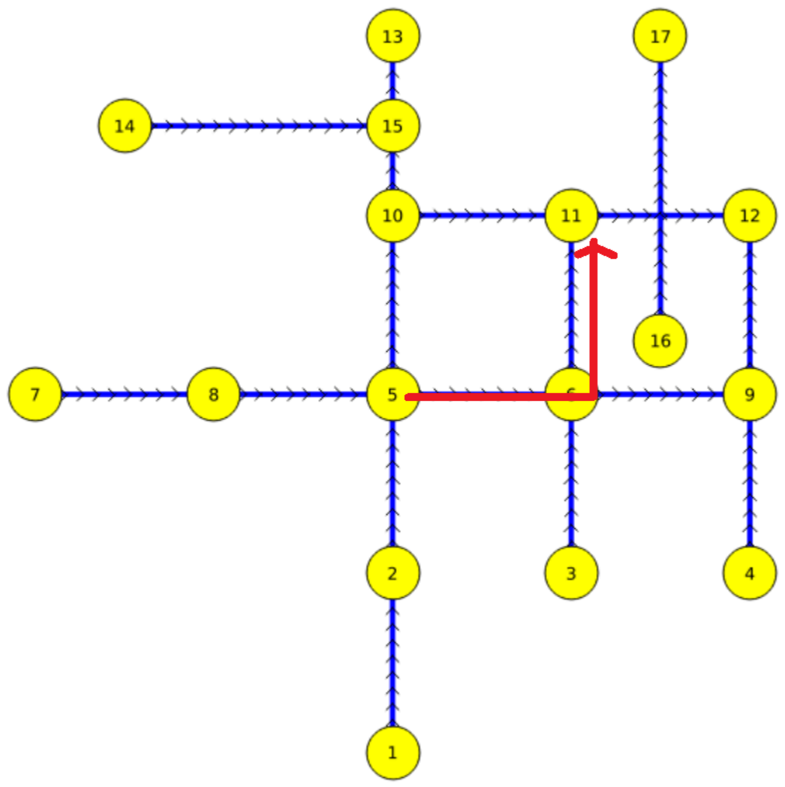

Fig: Example of idom of vextex 11[7]

Vextex 1,2 also dominate vertex 11, but they are not vertex 11’s idom.

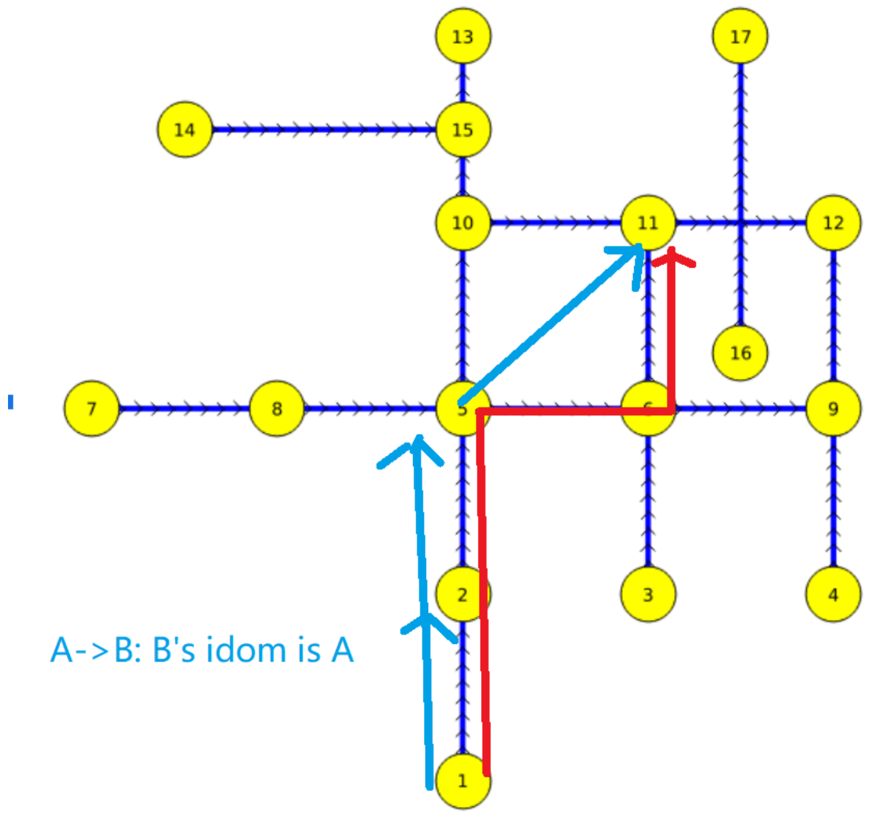

Fig: Example of idom and sdom relation, relations of idom construct the dominator tree[7]

15. References

[1]. Topological sorting From Wikipedia, the free encyclopedia
[2]. transitive closure From Wikipedia, the free encyclopedia
[3]. Dominator_(graph_theory)#Algorithms From Wikipedia, the free encyclopedia.
[4]. DAG From th7.com
[5]. lengtarj Persudo code & implement figures on cl.cam.ac.uk.
[6]. pgr_bdDijkstra Documentation
[7]. pgRouting Sample Data.