Introduction to GraphBLAS

A linear algebraic approach for concise, portable, and high-performance graph algorithms

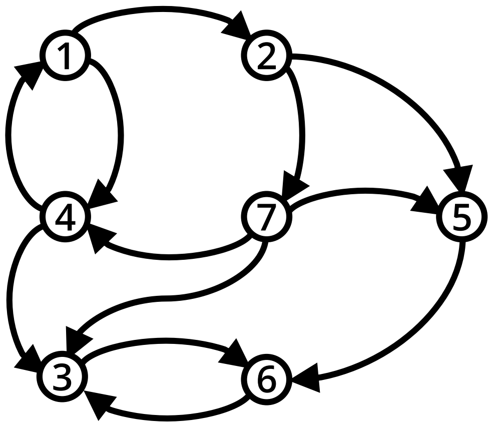
Gábor Szárnyas

2020/12/12

Motivation

GRAPHBLAS

- Graph algorithms are challenging to program
  - irregular access patterns → poor locality
  - caching and parallelization are difficult
- Optimizations often limit portability
- The **GraphBLAS** introduces an abstraction layer using the language of linear algebra
  - graph ≡ sparse matrix
  - traversal step ≡ vector-matrix multiplication



≡

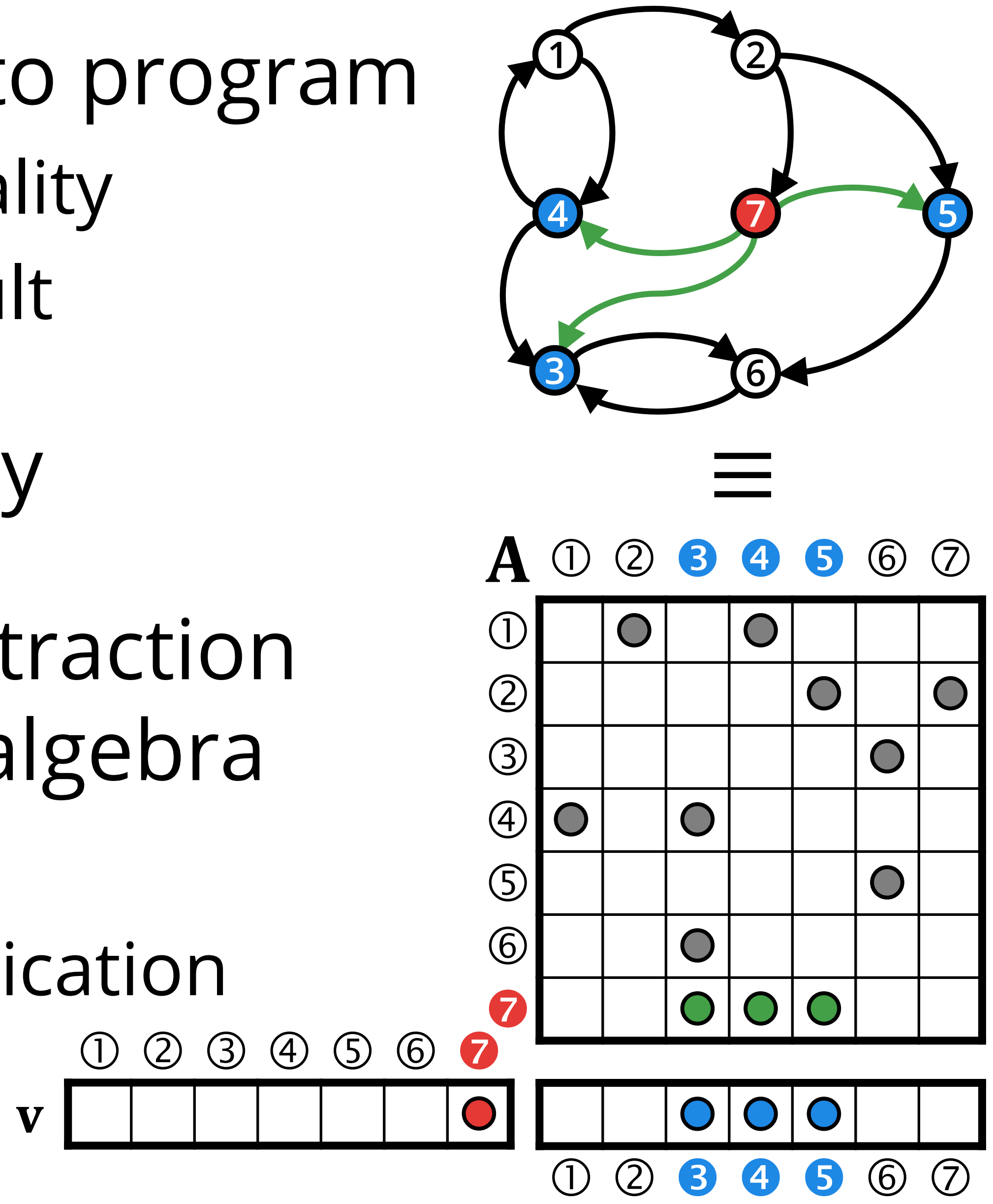
A	①	②	③	④	⑤	⑥	⑦
①		●		●			
②					●		●
③						●	
④	●		●				
⑤						●	
⑥			●				
⑦			●	●	●		

GRAPHBLAS

- Graph algorithms are challenging to program
  - irregular access patterns → poor locality
  - caching and parallelization are difficult

- Optimizations often limit portability

- The **GraphBLAS** introduces an abstraction layer using the language of linear algebra
  - graph ≡ sparse matrix
  - traversal step ≡ vector-matrix multiplication



A	1	2	3	4	5	6	7
1		●		●			
2					●		●
3						●	
4	●		●				
5						●	
6			●				
7			●	●	●		

	1	2	3	4	5	6	7
v							●
			●	●	●		

What makes graph computations difficult?

GRAPH PROCESSING CHALLENGES

connectedness

the “curse of connectedness”

computer architectures

contemporary computer architectures are good at processing linear and hierarchical data structures, such as *lists*, *stacks*, or *trees*

caching and parallelization

a massive amount of random data access is required, CPU has frequent cache misses, and implementing parallelism is difficult

B. Shao, Y. Li, H. Wang, H. Xia (Microsoft Research),
Trinity Graph Engine and its Applications,
IEEE Data Engineering Bulleting 2017

GRAPH PROCESSING CHALLENGES

Graph algorithm have a *high communication-to-computation ratio.*

Speedup with a CPU that has better arithmetic performance:

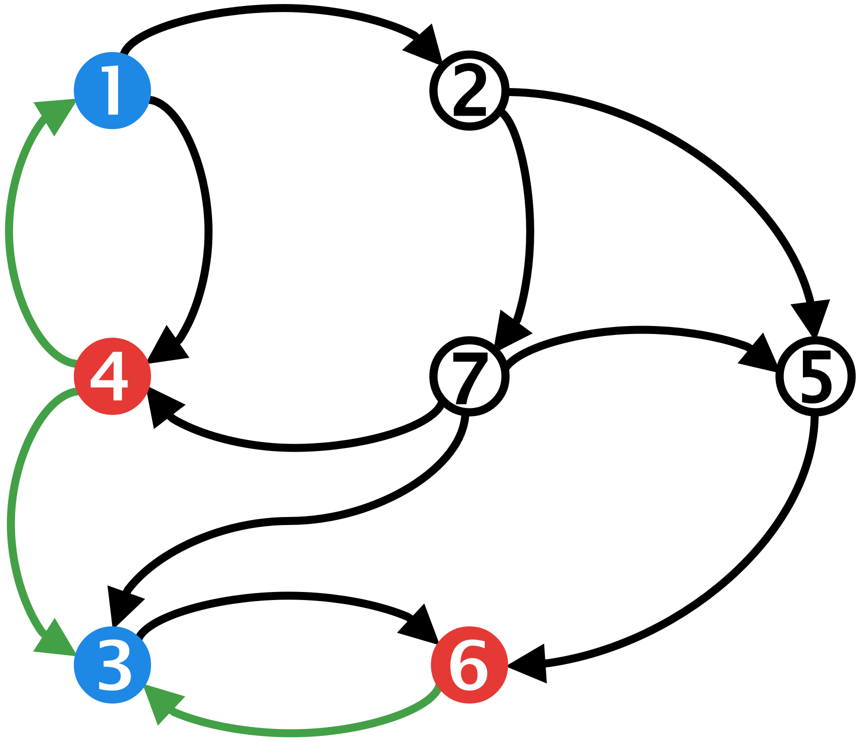
machine learning	→	**a lot**
relational queries	→	**some**
graph processing	→	**very little**

Standard latency hiding techniques break down, e.g. pre-fetching and branch prediction provide little benefit.

Use a data representation and computation model which are expressive, machine-friendly, and portable.

LINEAR ALGEBRA-BASED GRAPH PROCESSING

- Graphs are encoded as **sparse adjacency matrices.**
- Use **vector/matrix operations** to express graph algorithms.



A	❶	②	❸	④	⑤	⑥	⑦
①		1		1			
②					1		1
③						1	
❹	1		1				
⑤						1	
❻			1				
⑦			1	1	1		

	①	②	③	❹	⑤	❻	⑦
f				1		1	
fA	1		2				

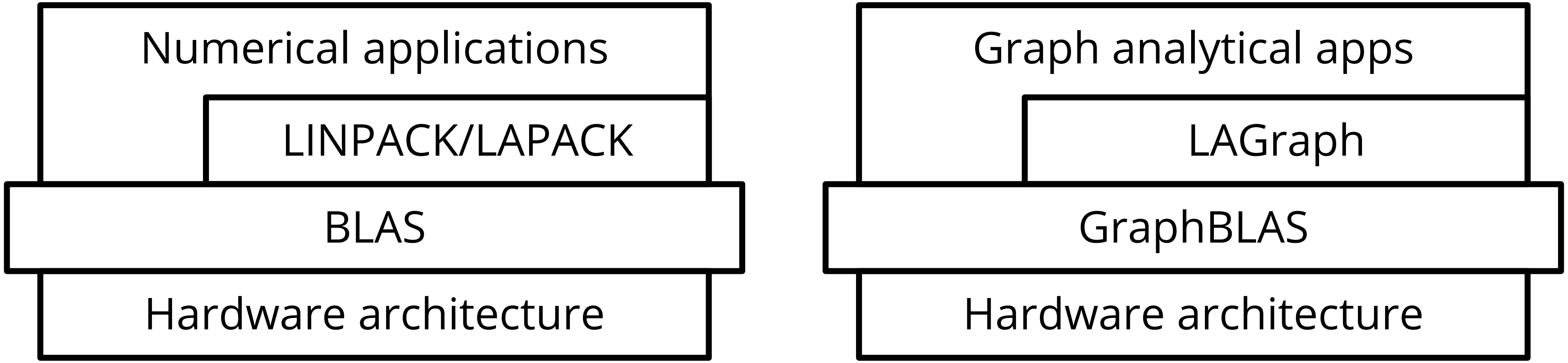
frontier

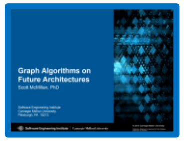
The GraphBLAS standard

THE GRAPHBLAS STANDARD

Goal: separate the concerns of the hardware, library, and application designers.

1979: BLAS	Basic Linear Algebra Subprograms
2001: Sparse BLAS	an extension to BLAS (little uptake)
2013: GraphBLAS	an effort to define standard building blocks for graph algorithms in the language of linear algebra




Graph Algorithms on Future Architectures

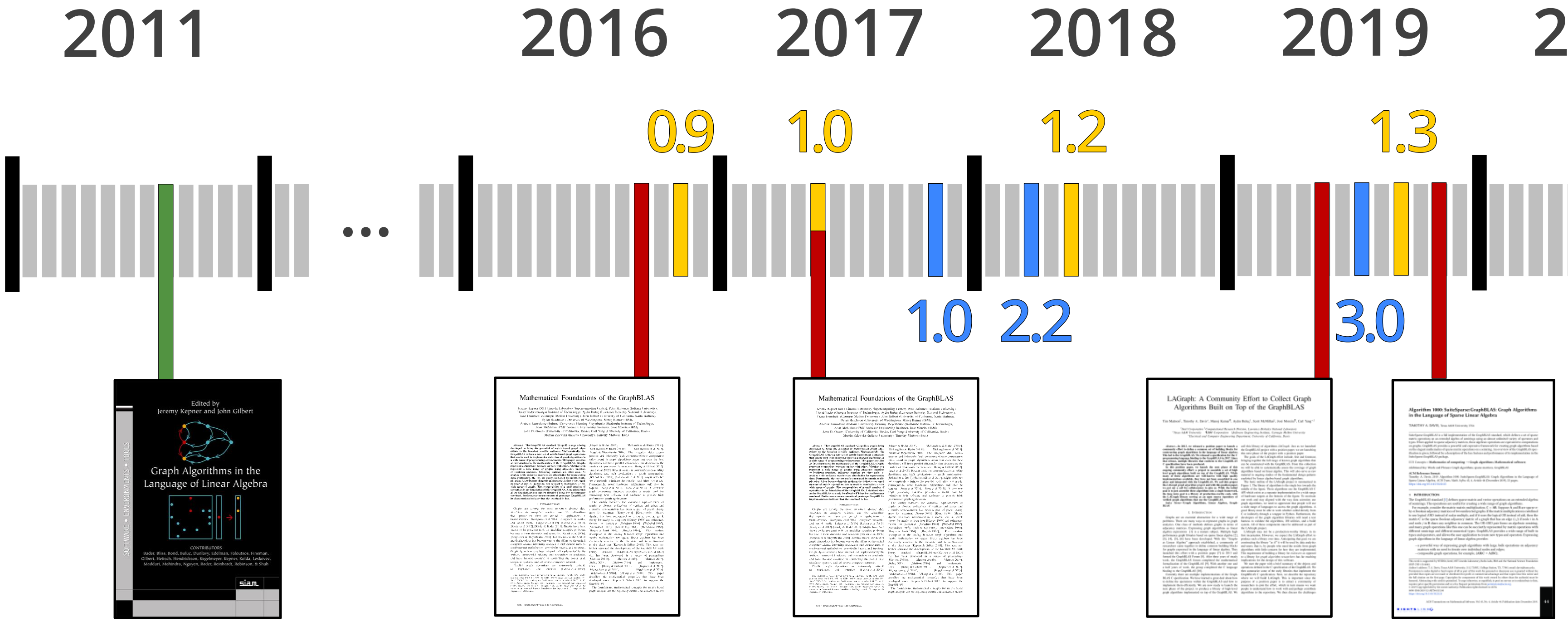
Scott McMillan, PhD

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

S. McMillan @ SEI Research Review (Carnegie Mellon University, 2015):
Graph algorithms on future architectures

GRAPHBLAS TIMELINE

Book — Papers — GraphBLAS standards — SuiteSparse:GraphBLAS releases


2011
2016
2017
2018
2019
2020
2021
...
0.9
1.0
1.2
1.3
1.0
2.2
3.0
Edited by
Jeremy Kepner and John Gilbert
Graph Algorithms in the Language of Linear Algebra
Mathematical Foundations of the GraphBLAS
Mathematical Foundations of the GraphBLAS
LAGraph: A Community Effort to Collect Graph Algorithms Built on Top of the GraphBLAS

Graph Algorithms in the Language of Linear Algebra

Mathematical foundations, HPEC

C API, GABB@ IPDPS

LAGraph, GrAPL @IPDPS

SuiteSparse: GraphBLAS, TOMS

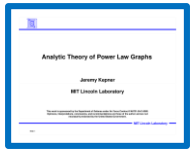
2020:

- SuiteSparse:GraphBLAS v4.0.1draft
- C++ API proposal
- Distributed API proposal

GRAPH ALGORITHMS IN LINEAR ALGEBRA

Notation: $n = |V|, m = |E|$. The complexity cells contain asymptotic bounds.
Takeaway: The majority of common graph algorithms can be expressed efficiently in LA.

problem	algorithm	canonical complexity Θ	LA-based complexity Θ
breadth-first search		m	m
single-source shortest paths	Dijkstra	$m + n \log n$	n^2
	Bellman-Ford	mn	mn
all-pairs shortest paths	Floyd-Warshall	n^3	n^3
minimum spanning tree	Prim	$m + n \log n$	n^2
	Borůvka	$m \log n$	$m \log n$
maximum flow	Edmonds-Karp	$m^2 n$	$m^2 n$
maximal independent set	greedy	$m + n \log n$	$mn + n^2$
	Luby	$m + n \log n$	$m \log n$



Analytic Theory of Power Law Graphs

Jeremy Kepner

MIT Lincoln Laboratory

Based on the table in J. Kepner:
Analytic Theory of Power Law Graphs,
SIAM Workshop for HPC on Large Graphs, 2008

Theoretically Efficient Parallel Graph Algorithms Can Be Fast and Scalable

See also L. Dhulipala, G.E. Blelloch, J. Shun:
Theoretically Efficient Parallel Graph Algorithms Can Be Fast and Scalable, SPAA 2018

KEY FEATURES OF GRAPHBLAS

- **Portable:** supports x86; GPUs (WIP), Arm (planned)
- **Efficient:** within one order-of-magnitude compared to hand-tuned code
- **Concise:** most textbook algorithms can be expressed with a few operations
- **Composable:** the output of an algorithm can be used as an input of a subsequent algorithm
- **Flexible:** can express algorithms on typed graphs and property graphs

Theoretical foundations of the GraphBLAS

DENSE MATRIX MULTIPLICATION

Definition:

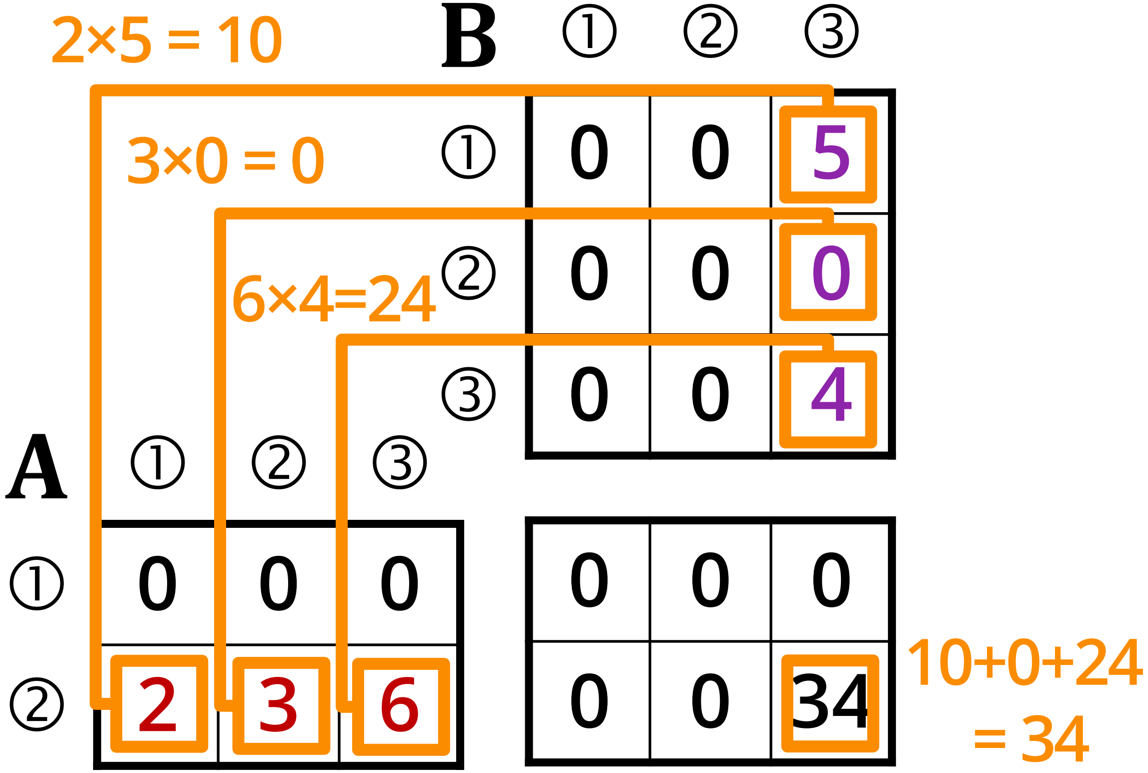
$$\mathbf{C} = \mathbf{AB}$$

$$\mathbf{C}(i,j) = \sum_k \mathbf{A}(i,k) \times \mathbf{B}(k,j)$$

Example:

$$\begin{aligned}\mathbf{C}(2{,}3) = \; & \mathbf{A}(2{,}1) \times \mathbf{B}(1{,}3) + \\ & \mathbf{A}(2{,}2) \times \mathbf{B}(2{,}3) + \\ & \mathbf{A}(2{,}3) \times \mathbf{B}(3{,}3)\end{aligned}$$

$$= 2 \times 5 + 3 \times 0 + 6 \times 4 = 34$$



2×5 = 10

3×0 = 0

6×4=24

$$\mathbf{B} \quad \begin{array}{c|ccc} & \textcircled{1} & \textcircled{2} & \textcircled{3} \\ \hline \textcircled{1} & 0 & 0 & \boxed{5} \\ \textcircled{2} & 0 & 0 & \boxed{0} \\ \textcircled{3} & 0 & 0 & \boxed{4} \end{array}$$

$$\mathbf{A} \quad \begin{array}{c|ccc} & \textcircled{1} & \textcircled{2} & \textcircled{3} \\ \hline \textcircled{1} & 0 & 0 & 0 \\ \textcircled{2} & \boxed{2} & \boxed{3} & \boxed{6} \end{array}$$

$$\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & \boxed{34} \end{array}$$

10+0+24

= 34

$$\mathbf{C} = \mathbf{A} \oplus.\otimes \mathbf{B}$$

SPARSE MATRIX MULTIPLICATION

Definition:

$$\mathbf{C} = \mathbf{AB} = \mathbf{A} \oplus.\otimes \mathbf{B}$$

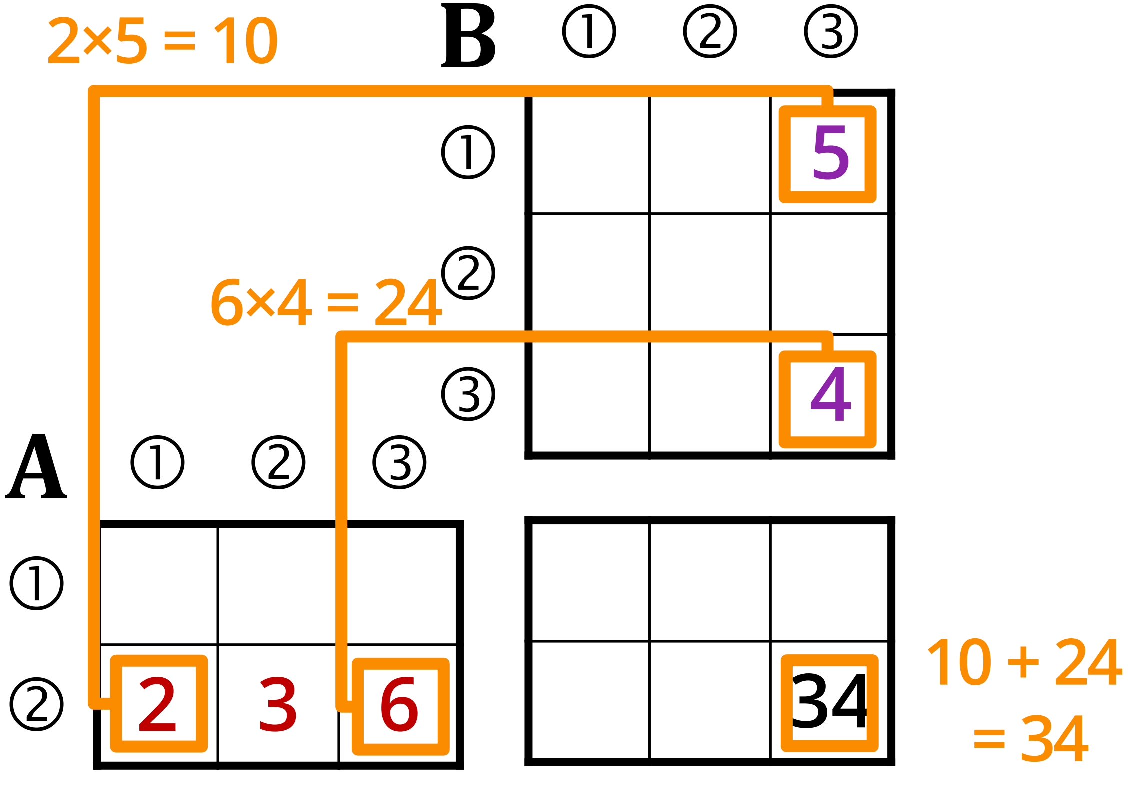
$$\mathbf{C}(i,j) = \bigoplus_{k \in \mathrm{ind}(\mathbf{A}(i,:)) \cap \mathrm{ind}(\mathbf{B}(:,j))} \mathbf{A}(i,k) \otimes \mathbf{B}(k,j)$$

Sparse matrix multiplication only evaluates the multiplication operator ⊗ for positions where there is a non-zero element in both $\mathbf{A}(i,k)$ and $\mathbf{B}(k,j)$.

Example:

$$\mathbf{C}(2,3) = \mathbf{A}(2,1) \times \mathbf{B}(1,3) + \mathbf{A}(2,3) \times \mathbf{B}(3,3)$$

$$= 2 \times 5 + 6 \times 4 = 34$$



$$\mathbf{C} = \mathbf{A} \oplus.\otimes \mathbf{B}$$

MATRIX MULTIPLICATION $\mathbf{C} = \mathbf{A} \oplus.\otimes \mathbf{B}$

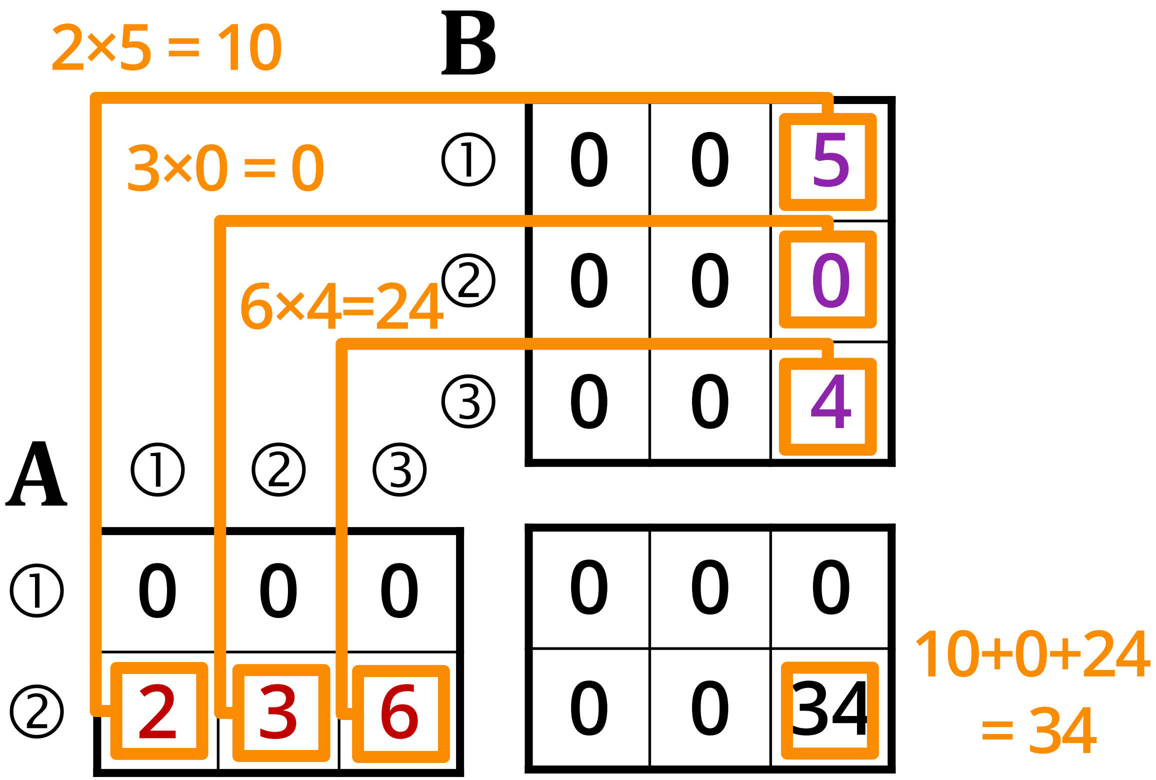
Multiplication on dense matrices

$$\mathbf{C}(i,j) = \bigoplus_{j} \mathbf{A}(i,k) \otimes \mathbf{B}(k,j)$$

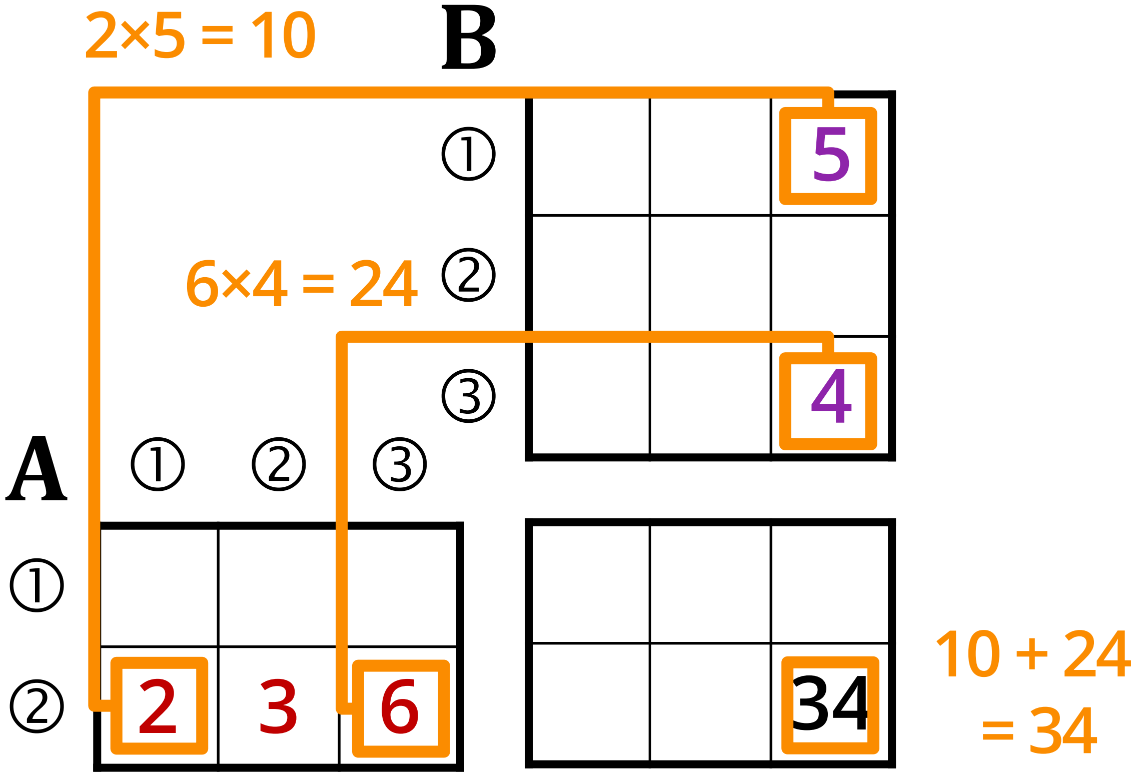
Multiplication on sparse matrices

$$\mathbf{C}(i,j) = \bigoplus_{k \in \mathrm{ind}\left(\mathbf{A}(i,:)\right) \cap \mathrm{ind}\left(\mathbf{B}(:,j)\right)} \mathbf{A}(i,k) \otimes \mathbf{B}(k,j)$$

Example: $\mathbf{C} = \mathbf{A} \oplus.\otimes \mathbf{B}$



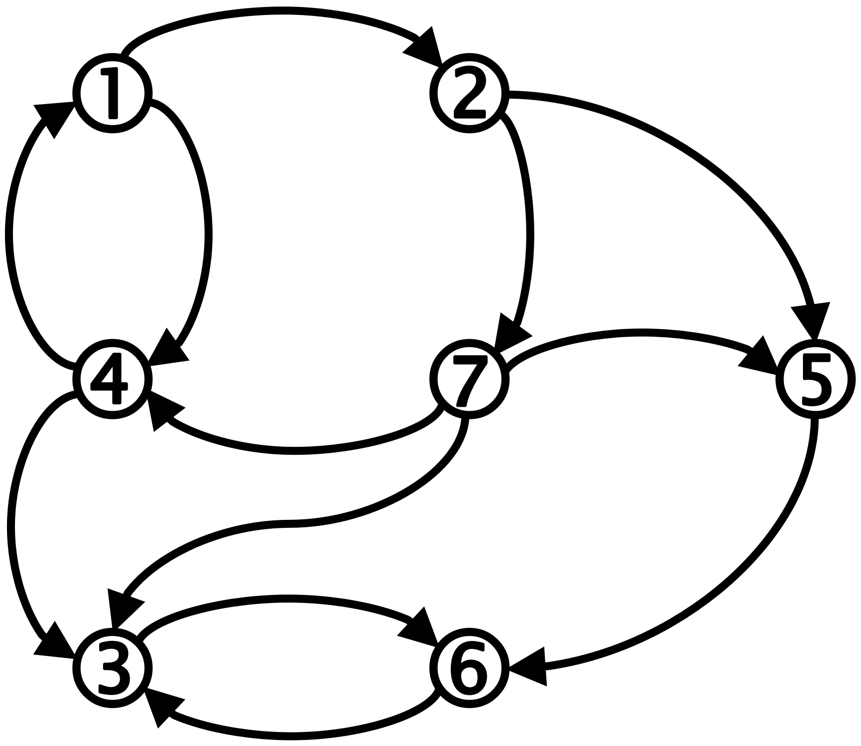
$$\mathbf{B} = \begin{array}{c|ccc} ① & 0 & 0 & 5 \\ ② & 0 & 0 & 0 \\ ③ & 0 & 0 & 4 \end{array} \qquad \mathbf{A} = \begin{array}{c|ccc} & ① & ② & ③ \\ \hline ① & 0 & 0 & 0 \\ ② & 2 & 3 & 6 \end{array} \qquad \begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 34 \end{array}$$



$$\mathbf{B} = \begin{array}{c|ccc} ① & & & 5 \\ ② & & & \\ ③ & & & 4 \end{array} \qquad \mathbf{A} = \begin{array}{c|ccc} & ① & ② & ③ \\ \hline ① & & & \\ ② & 2 & 3 & 6 \end{array} \qquad \begin{array}{ccc} & & \\ & & 34 \end{array}$$

ADJACENCY MATRIX

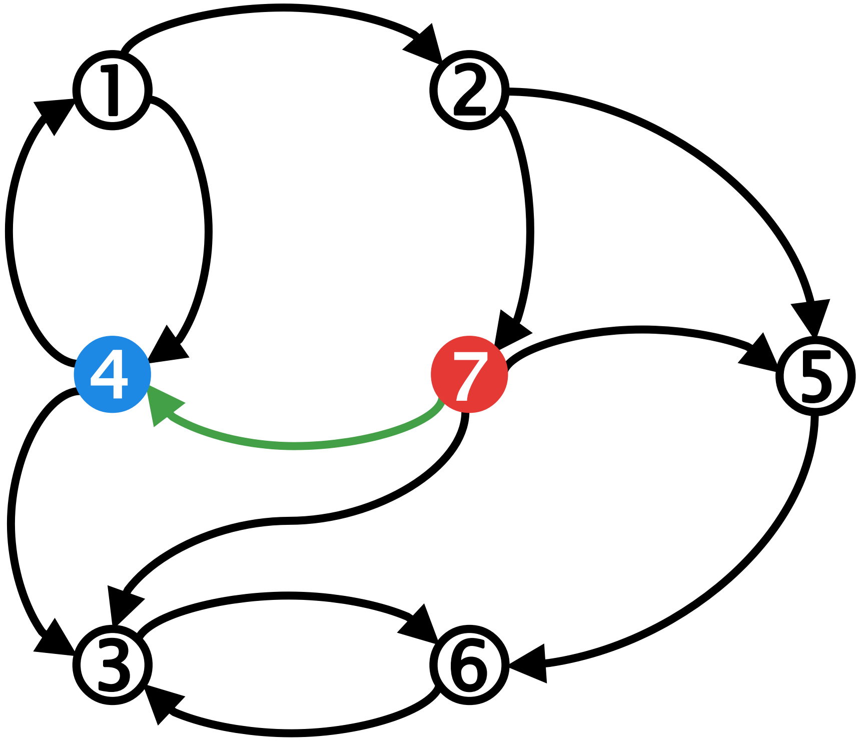
$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{if } (v_i, v_j) \notin E \end{cases}$$



A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②					1		1
③						1	
④	1		1				
⑤						1	
⑥			1				
⑦			1	1	1		

ADJACENCY MATRIX

$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{if } (v_i, v_j) \notin E \end{cases}$$

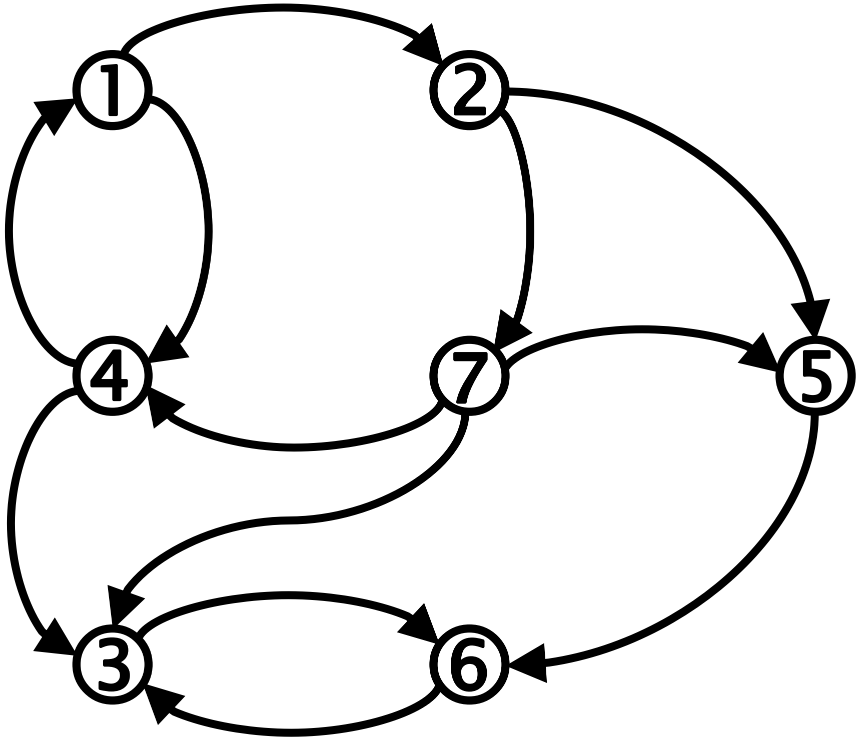


target

A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②					1		1
③						1	
④	1		1				
⑤						1	
⑥			1				
source ⑦			1	1	1		

ADJACENCY MATRIX

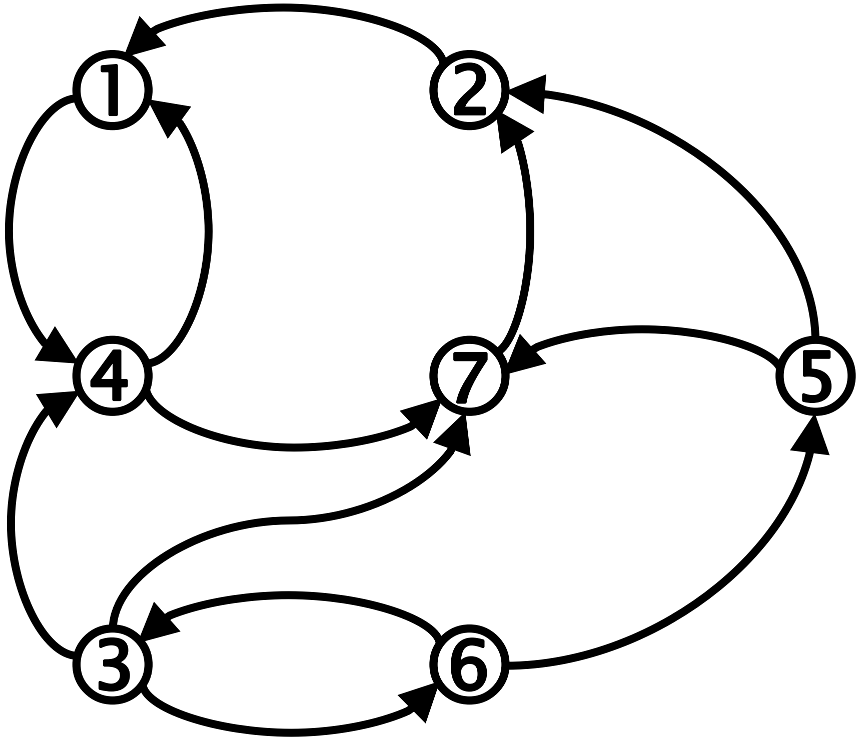
$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{if } (v_i, v_j) \notin E \end{cases}$$



A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②					1		1
③						1	
④	1		1				
⑤						1	
⑥			1				
⑦			1	1	1		

ADJACENCY MATRIX TRANSPOSED

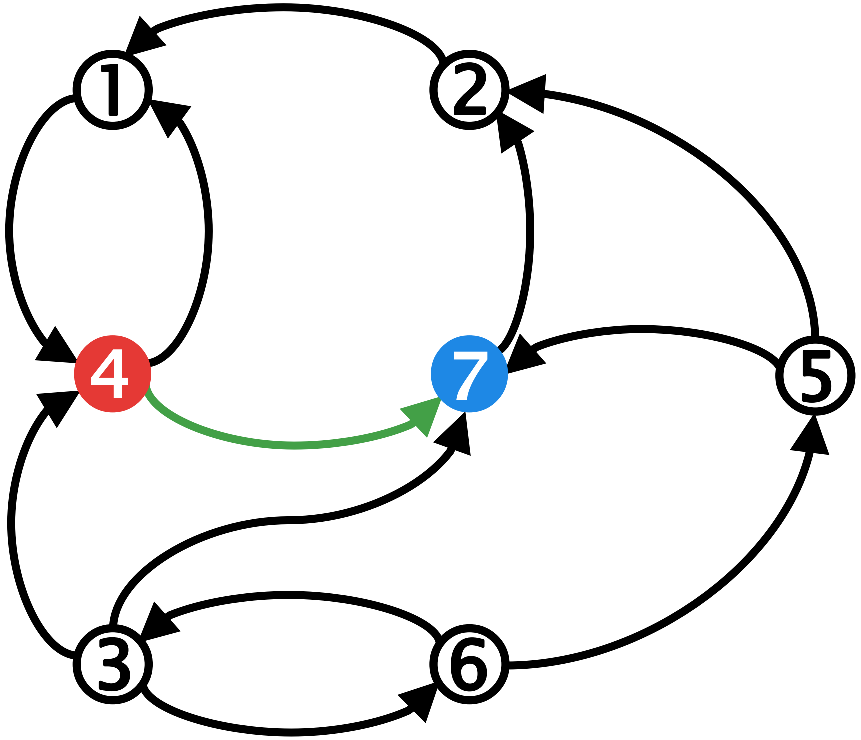
$$\mathbf{A}^{\top}_{ij} = \begin{cases} 1 & \text{if } (v_j, v_i) \in E \\ 0 & \text{if } (v_j, v_i) \notin E \end{cases}$$



$\mathbf{A}^{\top}$	①	②	③	④	⑤	⑥	⑦
①				1			
②	1						
③				1		1	1
④	1						1
⑤		1					1
⑥			1		1		
⑦		1					

ADJACENCY MATRIX TRANSPOSED

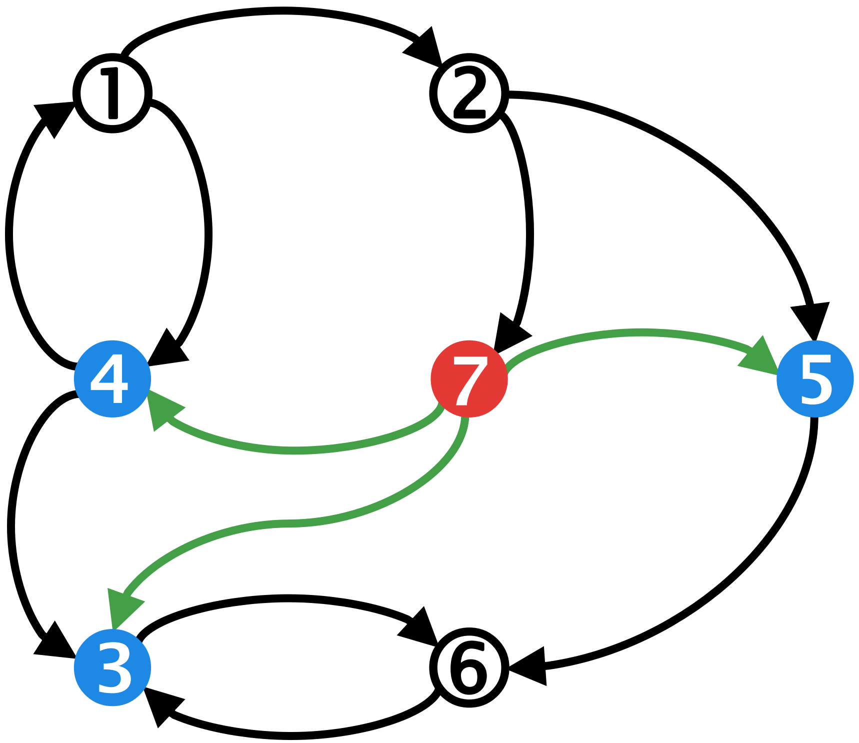
$$\mathbf{A}^{\top}_{ij} = \begin{cases} 1 & \text{if } (v_j, v_i) \in E \\ 0 & \text{if } (v_j, v_i) \notin E \end{cases}$$



target

$\mathbf{A}^{\top}$	①	②	③	④	⑤	⑥	❼
①				1			
②	1						
③				1		1	1
source ❹	1						1
⑤		1					1
⑥			1		1		
⑦		1					

GRAPH TRAVERSAL WITH MATRIX MULTIPLICATION



$\mathbf{f}\mathbf{A}^k$ means k hops in the graph

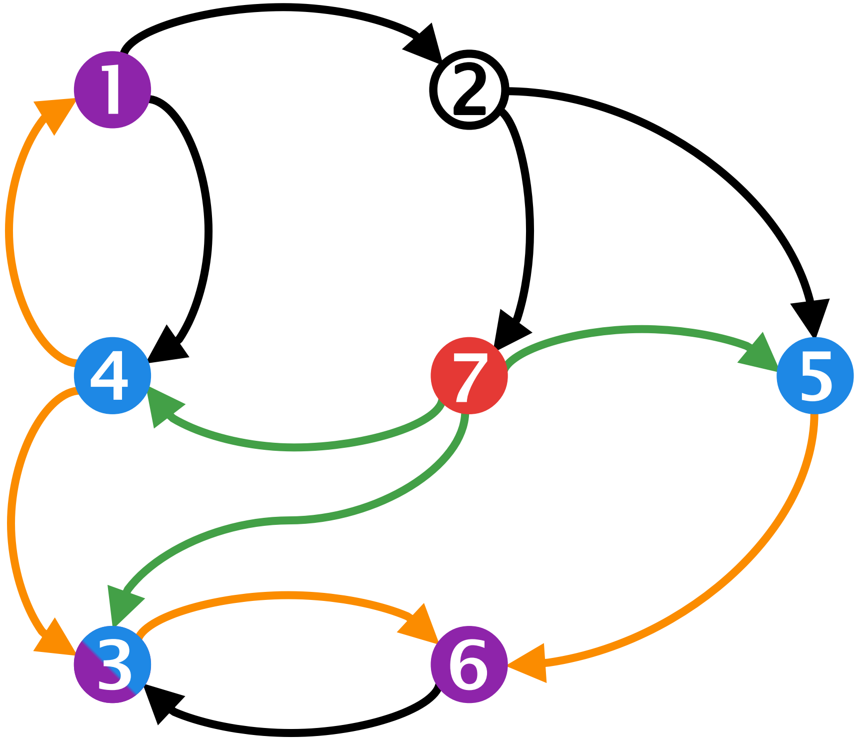
$\mathbf{A}$	①	②	❸	❹	❺	⑥	⑦
①		1		1			
②					1		1
③						1	
④	1		1				
⑤						1	
⑥			1				
❼			1	1	1		

	①	②	③	④	⑤	⑥	❼
$\mathbf{f}$							1

		1	1	1		
①	②	❸	❹	❺	⑥	⑦

one hop: $\mathbf{f}\mathbf{A}$

GRAPH TRAVERSAL WITH MATRIX MULTIPLICATION



$\mathbf{fA}^k$ means k hops in the graph

A	①	②	❸	❹	❺	⑥	⑦
①		1		1			
②					1		1
③						1	
④	1		1				
⑤						1	
⑥			1				
❼			1	1	1		

A	❶	②	❸	④	⑤	❻	⑦
①		1		1			
②					1		1
❸						1	
❹	1		1				
❺						1	
⑥			1				
⑦			1	1	1		

	①	②	③	④	⑤	⑥	❼
f							1

		1	1	1		
①	②	❸	❹	❺	⑥	⑦

one hop: $\mathbf{fA}$

1		1			2	
❶	②	❸	④	⑤	❻	⑦

two hops: $\mathbf{fA}^2$

GRAPHBLAS SEMIRINGS*

The $\langle D, \oplus, \otimes, 0 \rangle$ algebraic structure is a GraphBLAS semiring if

- $\langle D, \oplus, 0 \rangle$ is a commutative monoid using the addition operation $\oplus : D \times D \rightarrow D$, where $\forall a, b, c \in D$, if the following hold:
  - Commutative $a \oplus b = b \oplus a$
  - Associative $(a \oplus b) \oplus c = a \oplus (b \oplus c)$
  - Identity $a \oplus 0 = a$
- The multiplication operator is a closed binary operator $\otimes : D \times D \rightarrow D$.

The mathematical definition of a semiring requires that $\otimes$ is a monoid and distributes over $\oplus$. GraphBLAS omits these requirements.

SEMIRINGS

semiring	domain	$\oplus$	$\otimes$	0	graph semantics
any-pair	$\{\mathrm{T}, \mathrm{F}\}$	any	pair	F	traversal step
integer arithmetic	$\mathbb{N}$	$+$	$\times$	0	number of paths
min-plus	$\mathbb{R} \cup \{+\infty\}$	min	$+$	$+\infty$	shortest path

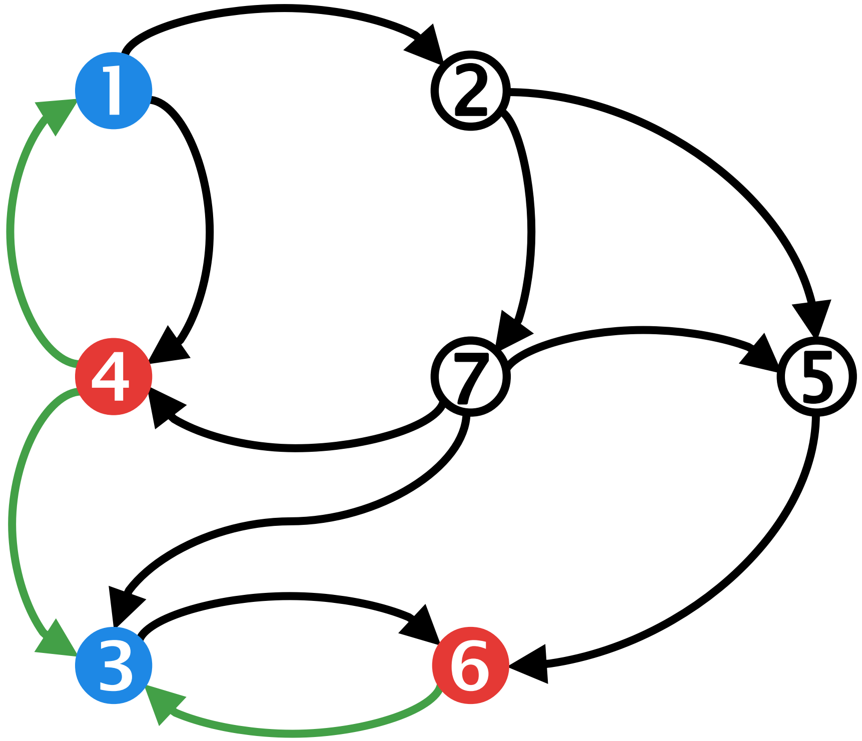
The default semiring is the conventional one:

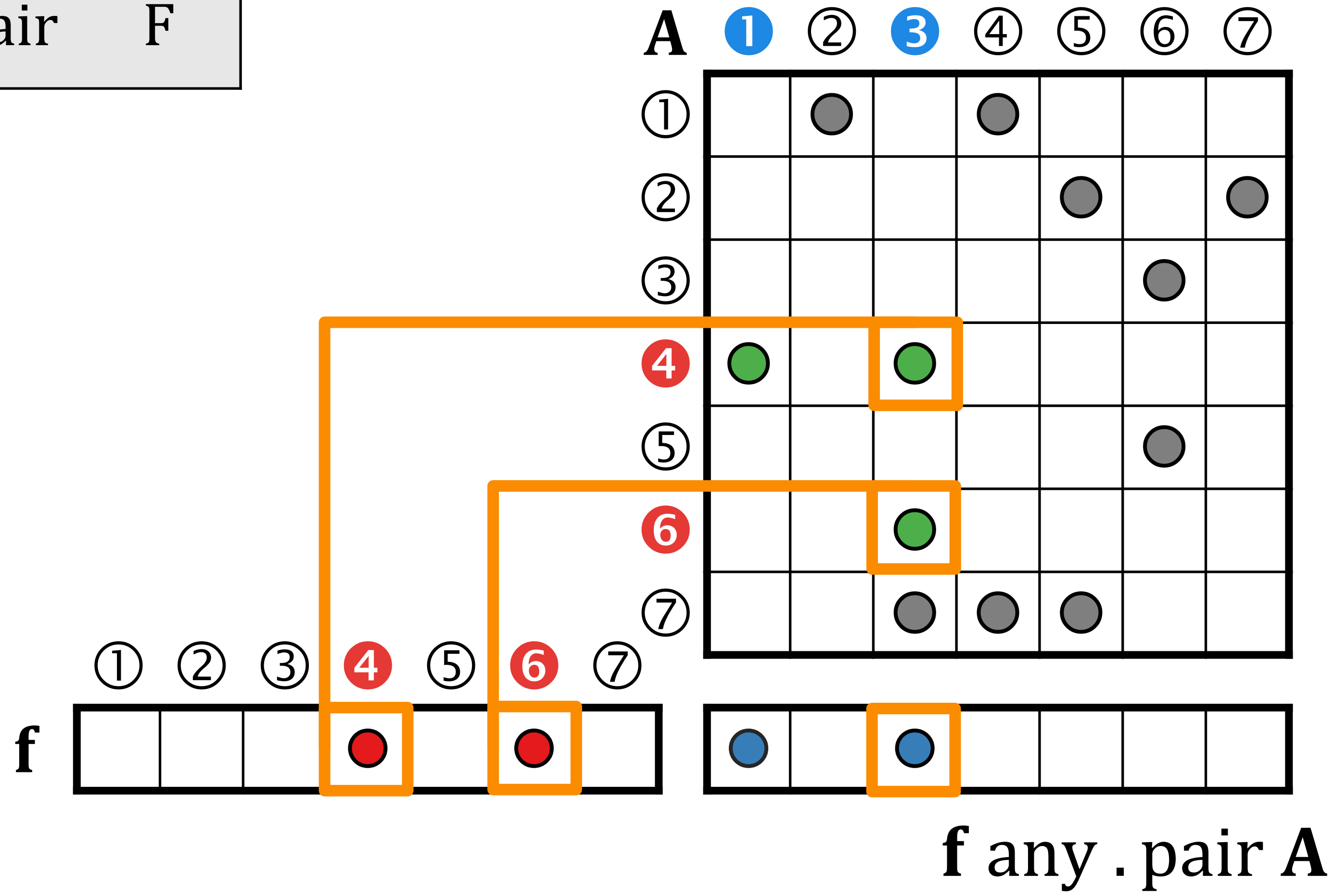
- $\otimes$ defaults to the arithmetic multiplication operator.
- $\oplus$ defaults to the arithmetic addition operator.

MATRIX-VECTOR MULTIPLICATION SEMANTICS

semiring	domain	$\oplus$	$\otimes$	0
any-pair	$\{T, F\}$	any	pair	F

Semantics: **traversal step**

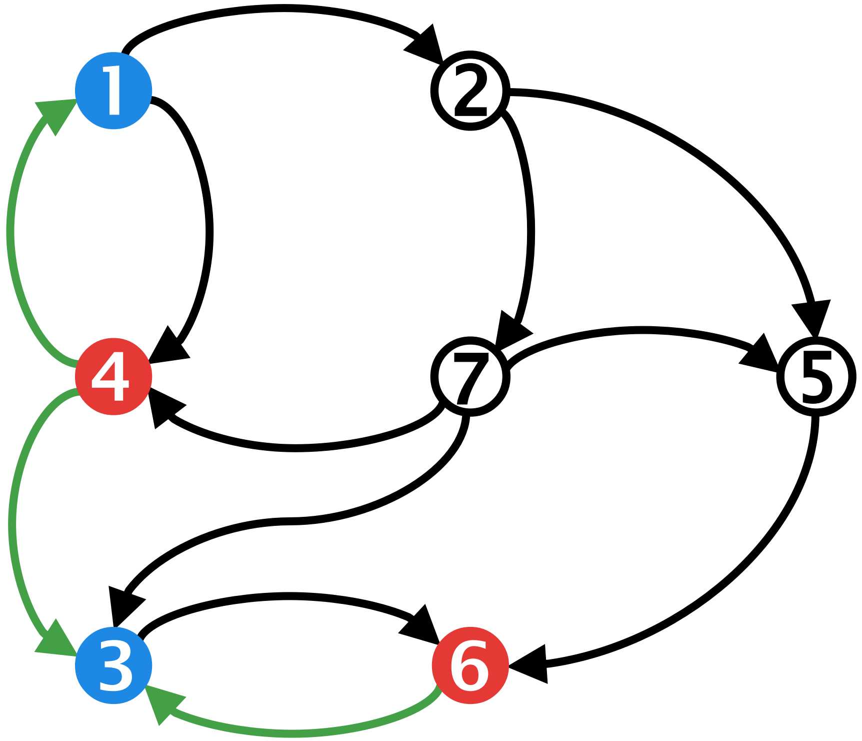




MATRIX-VECTOR MULTIPLICATION SEMANTICS

semiring	domain	⊕	⊗	0
integer arithmetic	ℕ	+	×	0

Semantics: **number of paths**



$$\begin{array}{c|ccccccc}
\mathbf{A} & ❶ & ② & ❸ & ④ & ⑤ & ⑥ & ⑦ \\
\hline
① & & 1 & & 1 & & & \\
② & & & & & 1 & & 1 \\
③ & & & & & & 1 & \\
❹ & 1 & & \boxed{1} & & & & \\
⑤ & & & & & & 1 & \\
❻ & & & \boxed{1} & & & & \\
⑦ & & & 1 & 1 & 1 & &
\end{array}$$

1·1=1

1·1=1

$$\begin{array}{c|ccccccc}
 & ① & ② & ③ & ❹ & ⑤ & ❻ & ⑦ \\
\hline
\mathbf{f} & & & & \boxed{1} & & \boxed{1} &
\end{array}$$

$$\begin{array}{|ccccccc|}
\hline
1 & & \boxed{2} & & & & \\
\hline
\end{array}$$

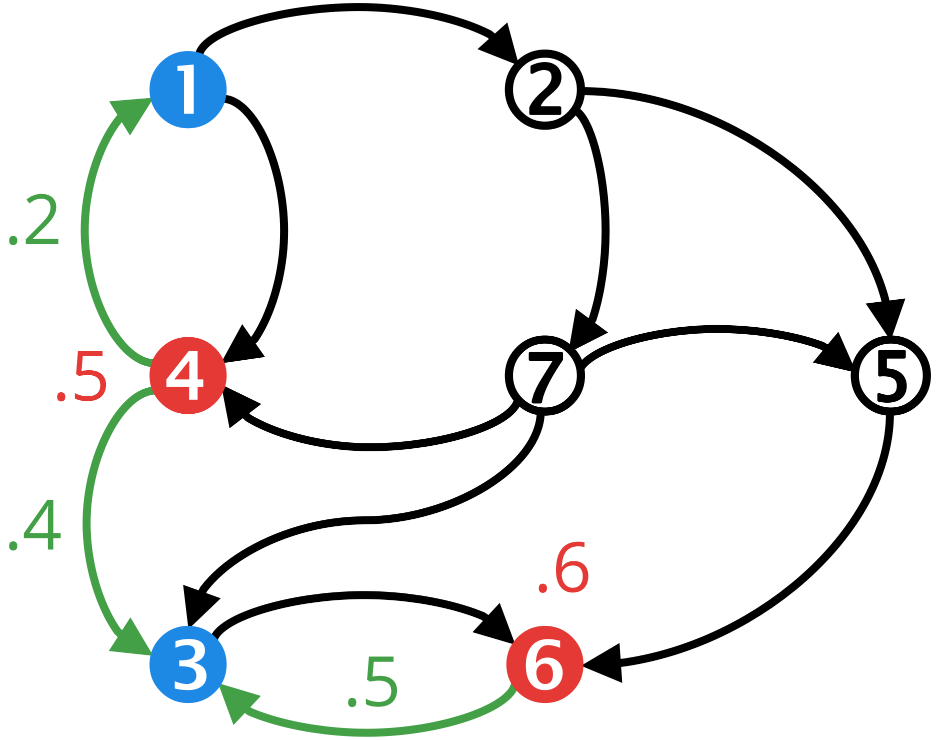
1+1=2

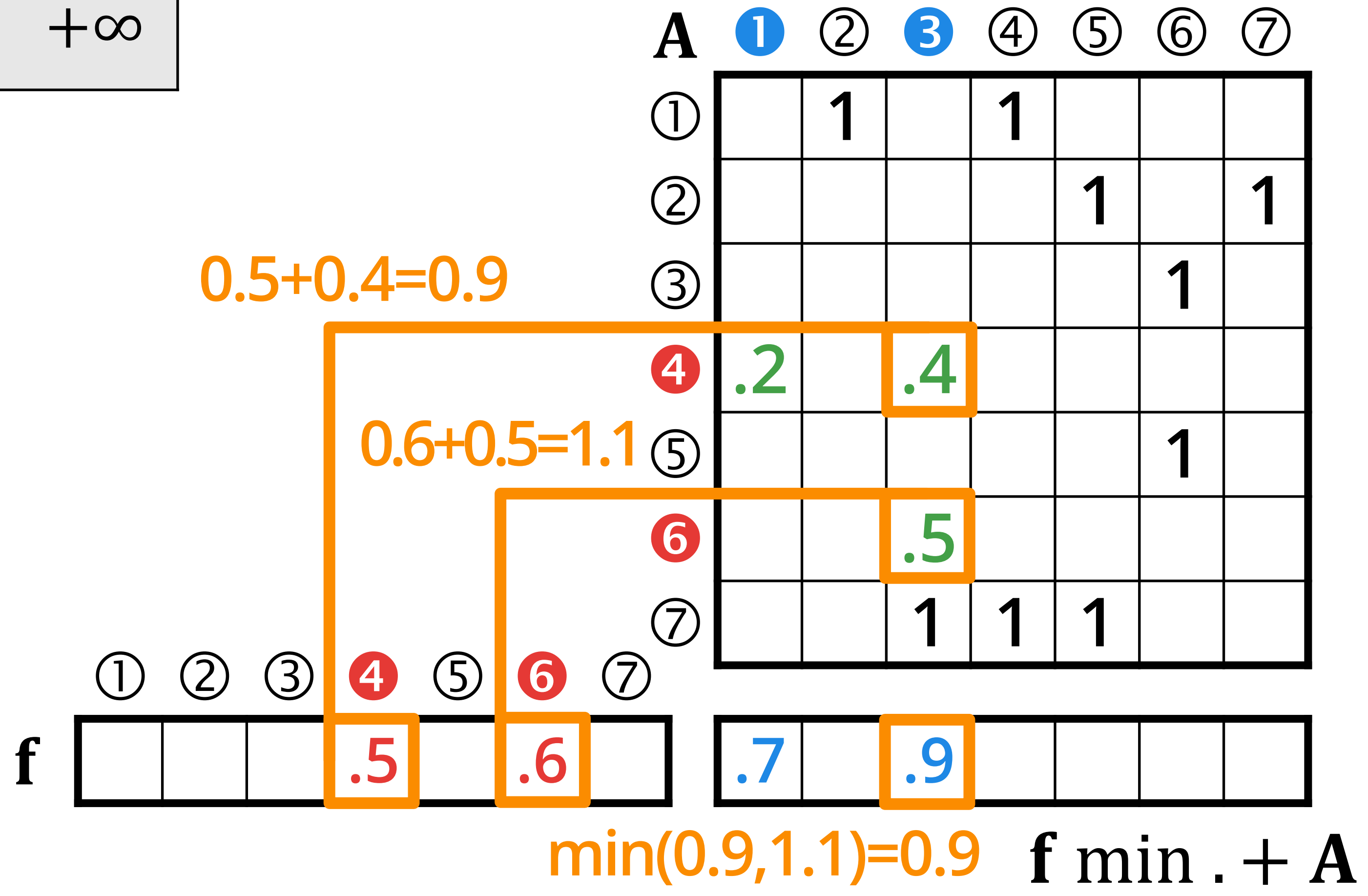
$\mathbf{f} \oplus.\otimes \mathbf{A}$

MATRIX-VECTOR MULTIPLICATION SEMANTICS

semiring	domain	⊕	⊗	0
min-plus	$\mathbb{R} \cup \{+\infty\}$	min	$+$	$+\infty$

Semantics: **shortest path**





$$\begin{array}{c|ccccccc}
\mathbf{A} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & & 1 & & 1 & & & \\
2 & & & & & 1 & & 1 \\
3 & & & & & & 1 & \\
4 & .2 & & .4 & & & & \\
5 & & & & & & 1 & \\
6 & & & .5 & & & & \\
7 & & & 1 & 1 & 1 & &
\end{array}$$

$$\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
\mathbf{f} & & & & .5 & & .6 &
\end{array}
\qquad
\begin{array}{ccccccc}
.7 & & .9 & & & &
\end{array}$$

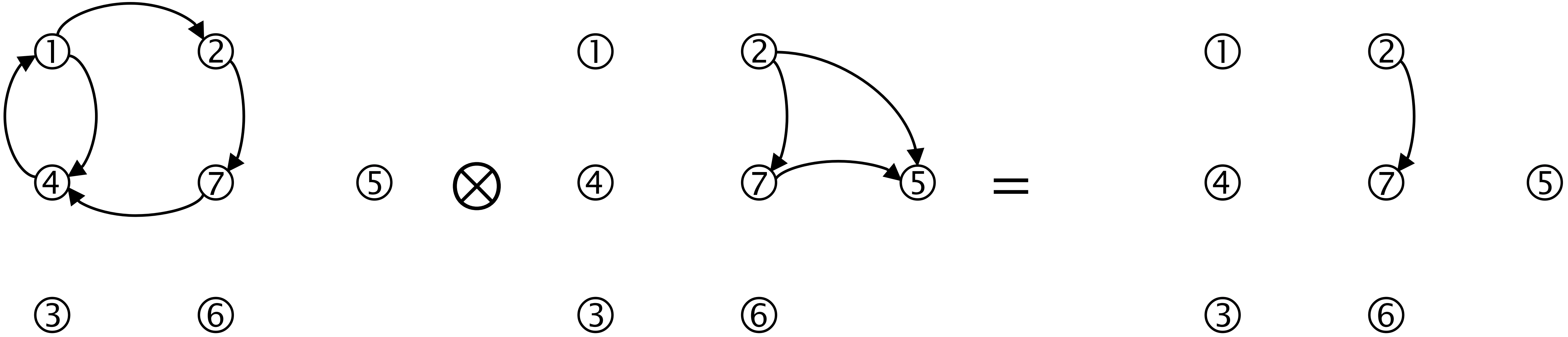
0.5+0.4=0.9

0.6+0.5=1.1

min(0.9,1.1)=0.9

$\mathbf{f}\ \mathrm{min}.{+}\ \mathbf{A}$

ELEMENT-WISE MULTIPLICATION: $\mathbf{A} \otimes \mathbf{B}$



A	①	②	③	④	⑤	⑥	⑦
①		●		●			
②							●
③							
④	●						
⑤							
⑥							
⑦				●			

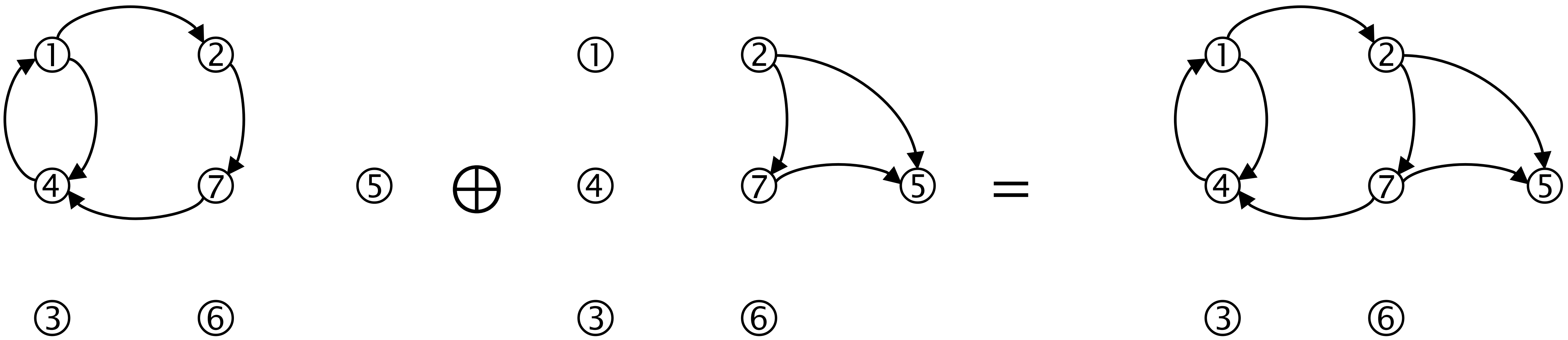
⊗

B	①	②	③	④	⑤	⑥	⑦
①							
②					●		●
③							
④							
⑤							
⑥							
⑦					●		

=

$\mathbf{A} \otimes \mathbf{B}$	①	②	③	④	⑤	⑥	⑦
①							
②							●
③							
④							
⑤							
⑥							
⑦							

ELEMENT-WISE ADDITION: **A** ⊕ **B**



A	①	②	③	④	⑤	⑥	⑦
①		●		●			
②							●
③							
④	●						
⑤							
⑥							
⑦				●			

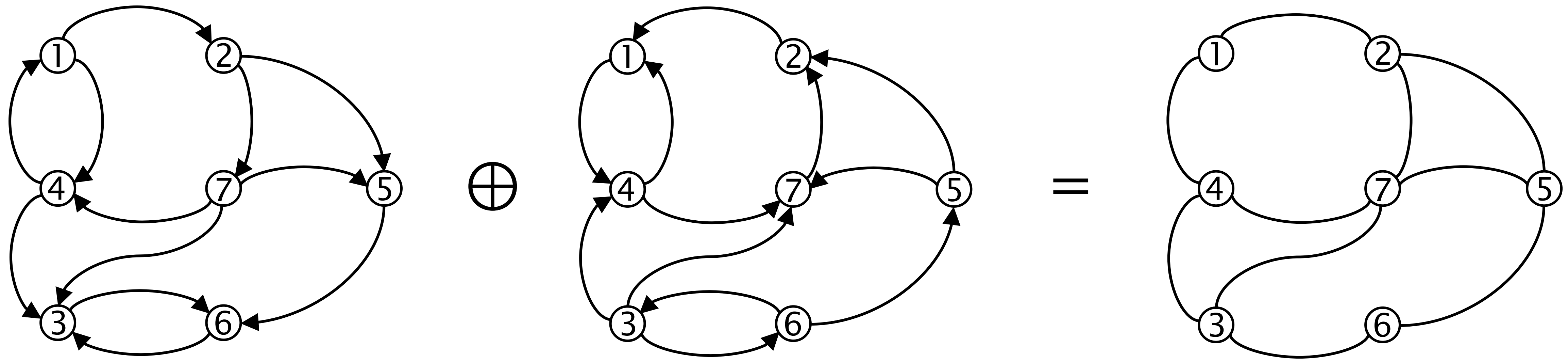
⊕

B	①	②	③	④	⑤	⑥	⑦
①							
②					●		●
③							
④							
⑤							
⑥							
⑦					●		

=

A ⊕ **B**	①	②	③	④	⑤	⑥	⑦
①		●		●			
②					●		●
③							
④	●						
⑤							
⑥							
⑦				●	●		

TURNING A GRAPH INTO UNDIRECTED: $\mathbf{A} \oplus \mathbf{A}^\mathsf{T}$



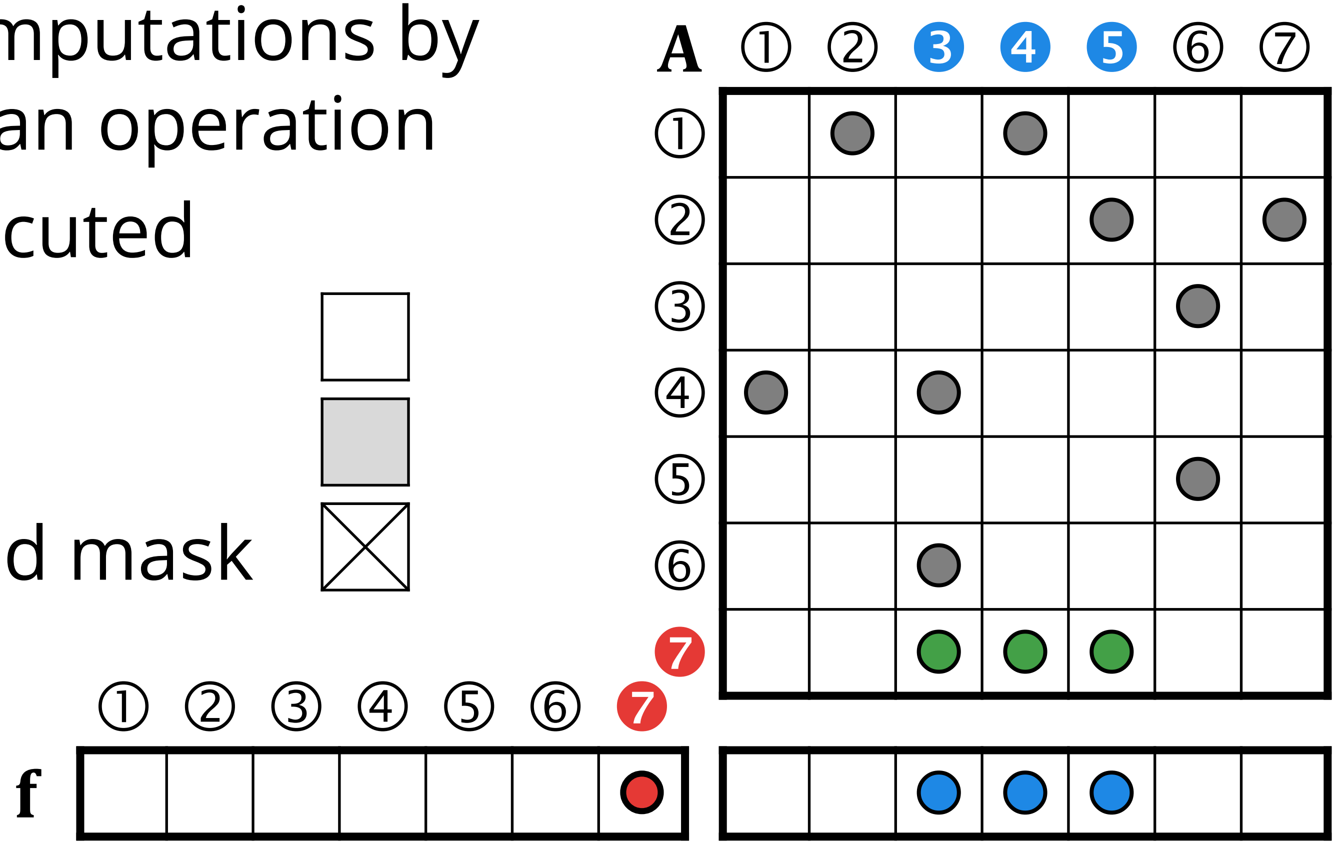
$$
\begin{array}{c|ccccccc}
\mathbf{A} & ① & ② & ③ & ④ & ⑤ & ⑥ & ⑦ \\
\hline
① & & \bullet & & \bullet & & & \\
② & & & & & \bullet & & \bullet \\
③ & & & & & & \bullet & \\
④ & \bullet & & \bullet & & & & \\
⑤ & & & & & & \bullet & \\
⑥ & & & \bullet & & & & \\
⑦ & & & \bullet & \bullet & \bullet & & \\
\end{array}
\;\oplus\;
\begin{array}{c|ccccccc}
\mathbf{A}^\mathsf{T} & ① & ② & ③ & ④ & ⑤ & ⑥ & ⑦ \\
\hline
① & & & & \bullet & & & \\
② & \bullet & & & & & & \\
③ & & & & \bullet & & \bullet & \bullet \\
④ & \bullet & & & & & & \bullet \\
⑤ & & \bullet & & & & & \bullet \\
⑥ & & & \bullet & & \bullet & & \\
⑦ & & \bullet & & & & & \\
\end{array}
\;=\;
\begin{array}{c|ccccccc}
\mathbf{A} \oplus \mathbf{A}^\mathsf{T} & ① & ② & ③ & ④ & ⑤ & ⑥ & ⑦ \\
\hline
① & & \bullet & & \bullet & & & \\
② & \bullet & & & & \bullet & & \bullet \\
③ & & & & \bullet & & \bullet & \bullet \\
④ & \bullet & & \bullet & & & & \bullet \\
⑤ & & \bullet & & & & \bullet & \bullet \\
⑥ & & & \bullet & & \bullet & & \\
⑦ & & \bullet & \bullet & \bullet & \bullet & & \\
\end{array}
$$

MASKING

Prevent redundant computations by reducing the scope of an operation

Operations can be executed

- **without a mask** □
- with a regular mask ▨
- with a complemented mask ☒



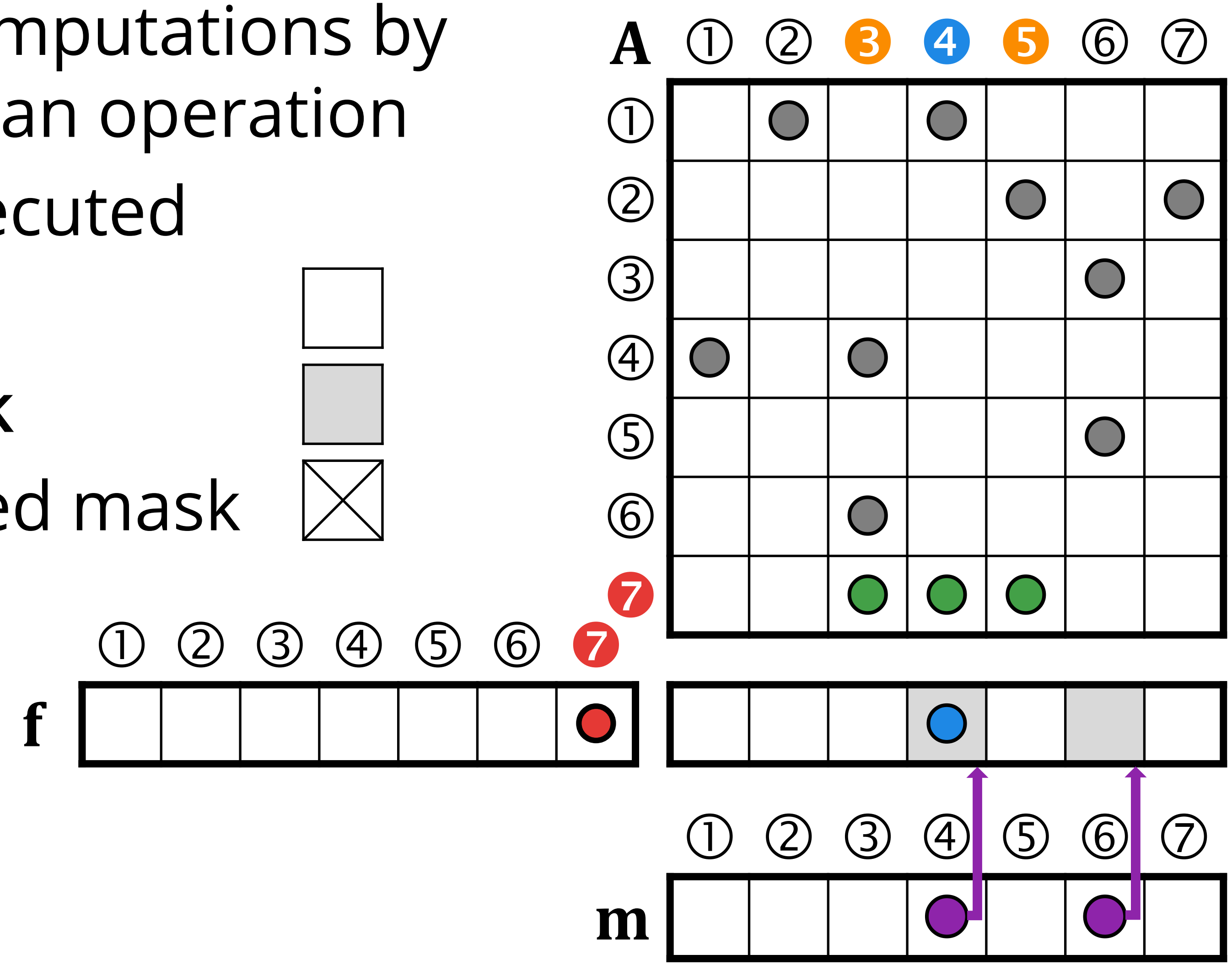
$\mathbf{w} =$
$\mathbf{f} \oplus.\otimes \mathbf{A}$

MASKING

Prevent redundant computations by reducing the scope of an operation

Operations can be executed

- without a mask □
- **with a regular mask** ▨
- with a complemented mask ☒



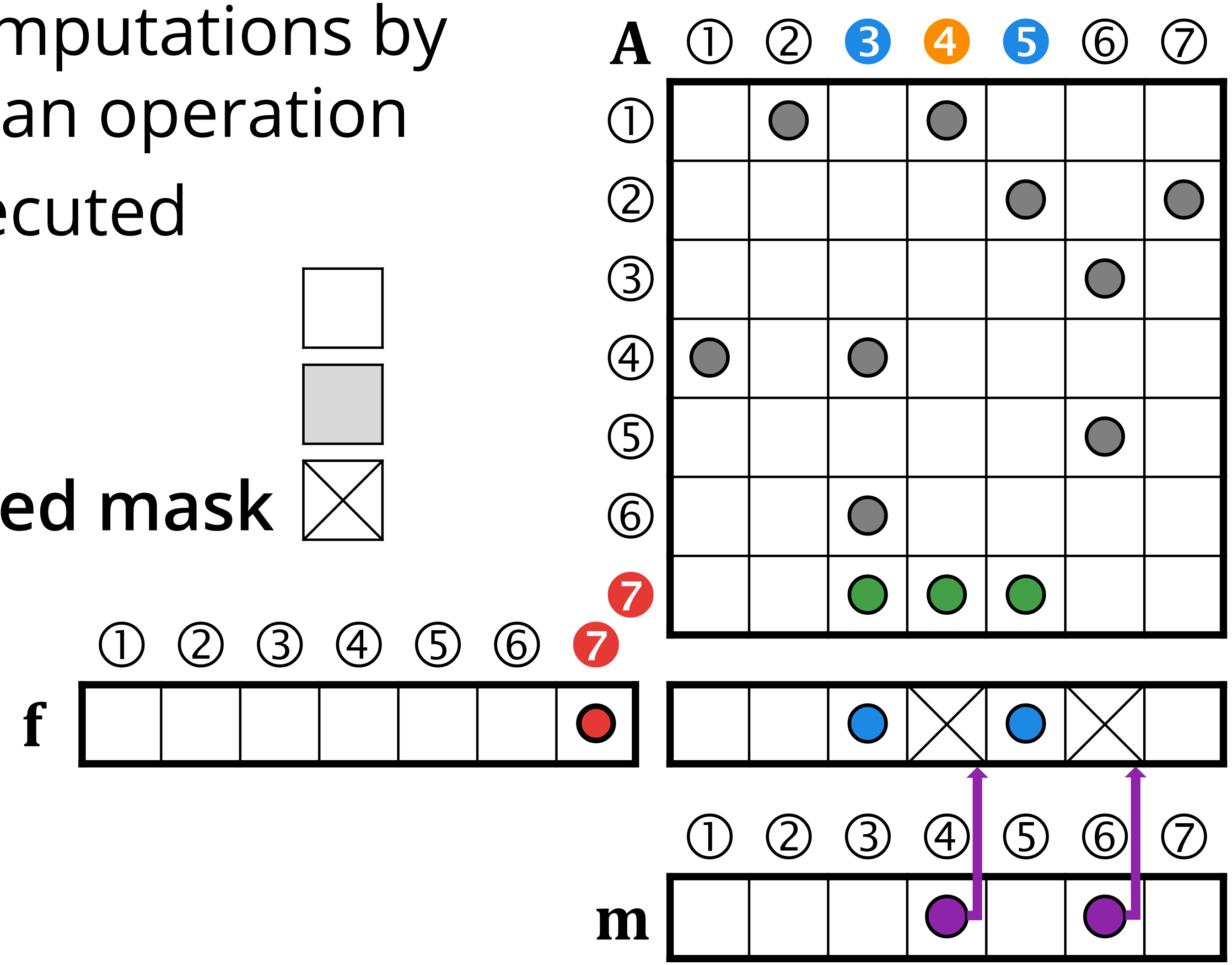
$\mathbf{w}\langle\mathbf{m}\rangle =$
$\mathbf{f} \oplus.\otimes \mathbf{A}$

MASKING

Prevent redundant computations by reducing the scope of an operation

Operations can be executed

- without a mask □
- with a regular mask ▨
- **with a complemented mask** ☒



$$\mathbf{w}\langle\neg\mathbf{m}\rangle = \mathbf{f} \oplus.\otimes \mathbf{A}$$

NOTATION*

- Symbols:
  - $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{M}$ – matrices
  - $\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{m}$ – vectors
  - s – scalar
  - i, j – indices
  - $\langle\mathbf{M}\rangle, \langle\mathbf{m}\rangle$ – masks
- Operators:
  - $\oplus$ – addition
  - $\otimes$ – multiplication
  - $\oslash$ – division
  - $^\top$ – transpose

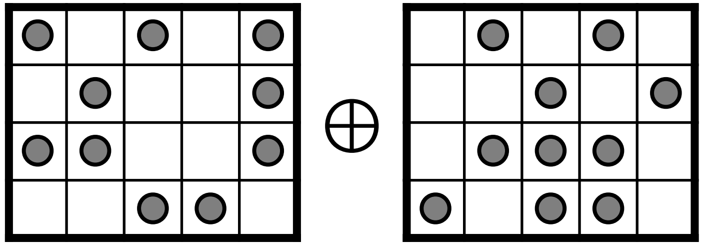
symbol	operation	notation
$\oplus.\otimes$	matrix-matrix multiplication	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{A} \oplus.\otimes \mathbf{B}$
	vector-matrix multiplication	$\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{v} \oplus.\otimes \mathbf{A}$
	matrix-vector multiplication	$\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{A} \oplus.\otimes \mathbf{v}$
$\otimes$	element-wise multiplication (set intersection of patterns)	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{A} \otimes \mathbf{B}$ $\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{u} \otimes \mathbf{v}$
$\oplus$	element-wise addition (set union of patterns)	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{A} \oplus \mathbf{B}$ $\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{u} \oplus \mathbf{v}$
f	apply unary operator	$\mathbf{C}\langle\mathbf{M}\rangle = f(\mathbf{A})$ $\mathbf{w}\langle\mathbf{m}\rangle = f(\mathbf{v})$
$[\oplus \cdots]$	reduce to vector	$\mathbf{w}\langle\mathbf{m}\rangle = \left[\bigoplus_j \mathbf{A}(:, j)\right]$
	reduce to scalar	$s = \left[\bigoplus_{ij} \mathbf{A}(i, j)\right]$
$\mathbf{A}^\top$	transpose matrix	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{A}^\top$

Vectors can act as both column and row vectors.

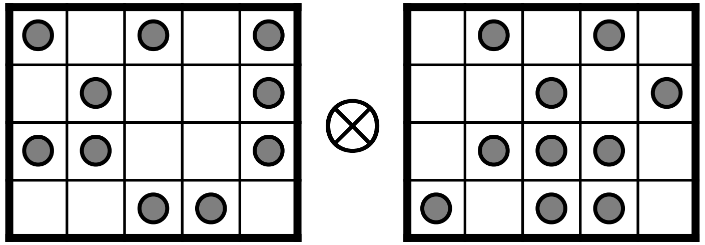
(Notation omitted for accumulator, selection, extraction, assignment…)

LINEAR ALGEBRAIC PRIMITIVES FOR GRAPHS #1

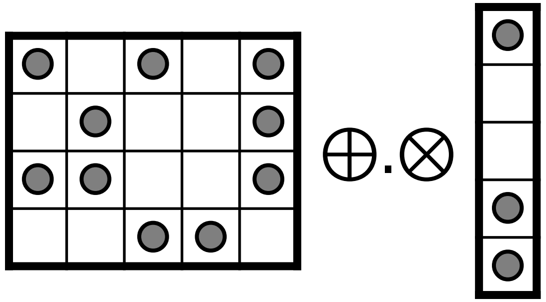
Element-wise addition:
union of non-zero elements



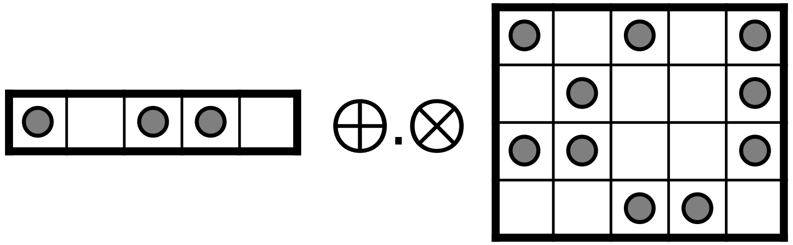
Element-wise multiplication:
intersection of non-zero elements



Sparse matrix times sparse vector:
process incoming edges

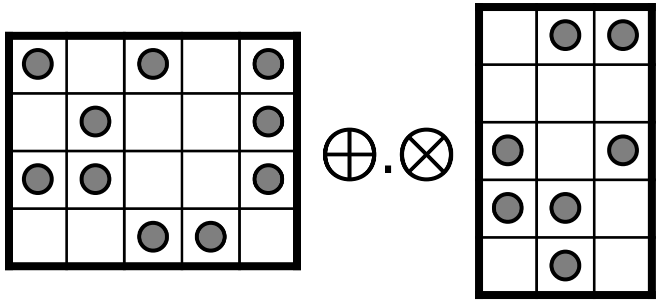


Sparse vector times sparse matrix:
process outgoing edges



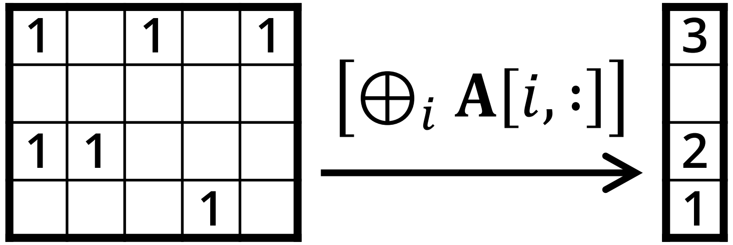
LINEAR ALGEBRAIC PRIMITIVES FOR GRAPHS #2

Sparse matrix times sparse matrix:
process connecting outgoing edges



$\oplus . \otimes$

Reduction:
aggregate values in each row

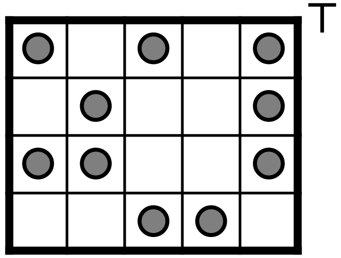


$\left[\bigoplus_i \mathbf{A}[i,:]\right]$

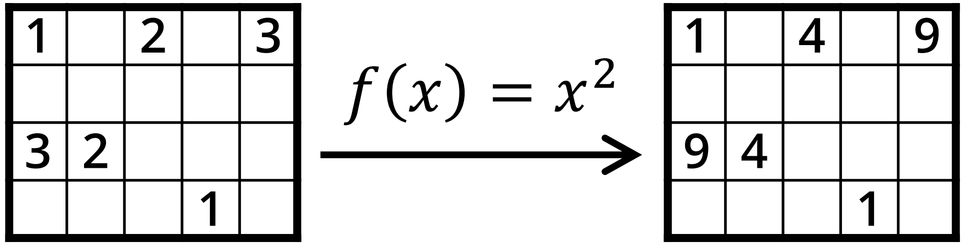
1		1		1
1	1			
			1	

3
2
1

Matrix transpose:
reverse edges



Apply:
apply unary operator on all values



$f(x) = x^2$

1		2		3
3	2			
			1	

1		4		9
9	4			
			1	

Graph algorithms in GraphBLAS

Breadth-first search

BFS: BREADTH-FIRST SEARCH

- **Algorithm:**
  - Start from a given vertex
  - “Explore all neighbour vertices at the present level prior to moving on to the vertices at the next level” [Wikipedia]
- **Variants:**
  - **Levels** compute traversal level for each vertex
  - **Parents** compute parent for each vertex
  - **MSBFS** start traversal from multiple source vertices

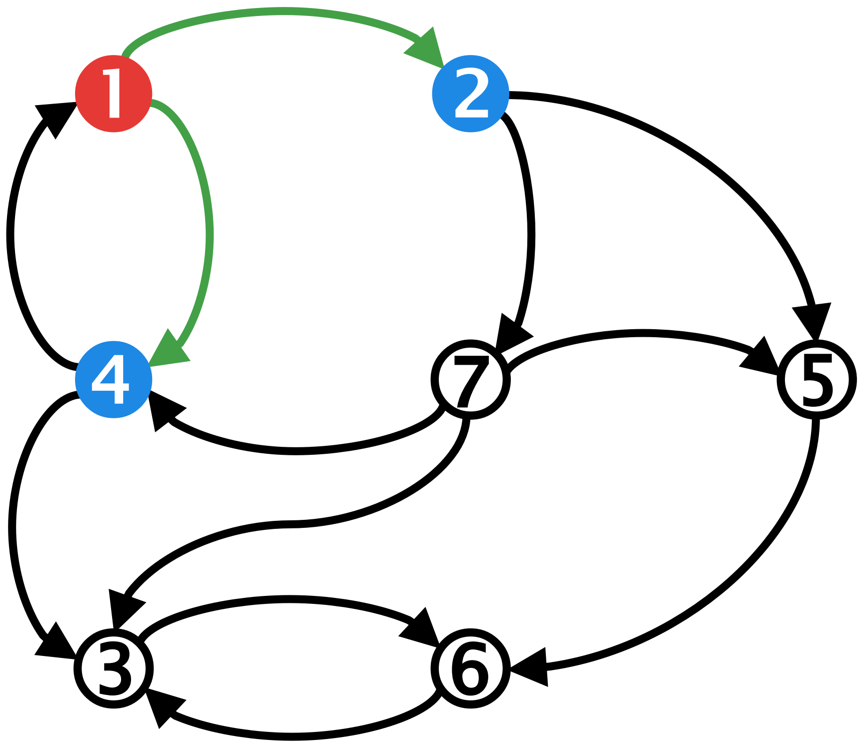
Graph algorithms in GraphBLAS

Breadth-first search / Levels

BFS – LEVELS

semiring	domain	⊕	⊗	0
any-pair	{T, F}	any	pair	F

level = 1



A	①	❷	③	❹	⑤	⑥	⑦
❶		●		●			
②					●		●
③						●	
④	●		●				
⑤						●	
⑥			●				
⑦			●	●	●		

	❶	②	③	④	⑤	⑥	⑦
f	●						

⇓ *level*

	❶	②	③	④	⑤	⑥	⑦
s	1						

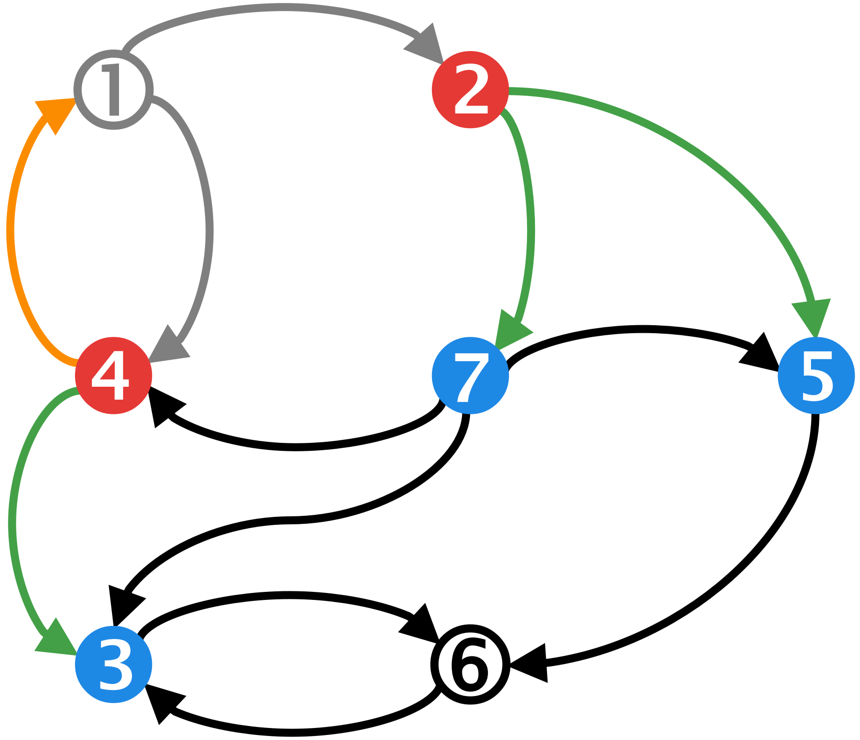
☒	●		●			

$\mathbf{f}\langle\neg\mathbf{s}\rangle = \mathbf{f}$ any.pair $\mathbf{A}$

BFS – LEVELS

semiring	domain	⊕	⊗	0
any-pair	{T, F}	any	pair	F

level = 2



A	①	②	③	④	⑤	⑥	⑦
①		●		●			
②					●		●
③						●	
④	●		●				
⑤						●	
⑥			●				
⑦			●	●	●		

	①	②	③	④	⑤	⑥	⑦
f		●		●			

☒	☒	●	☒	●		●

$\mathbf{f}\langle\neg\mathbf{s}\rangle = \mathbf{f}\ \text{any.pair}\ \mathbf{A}$

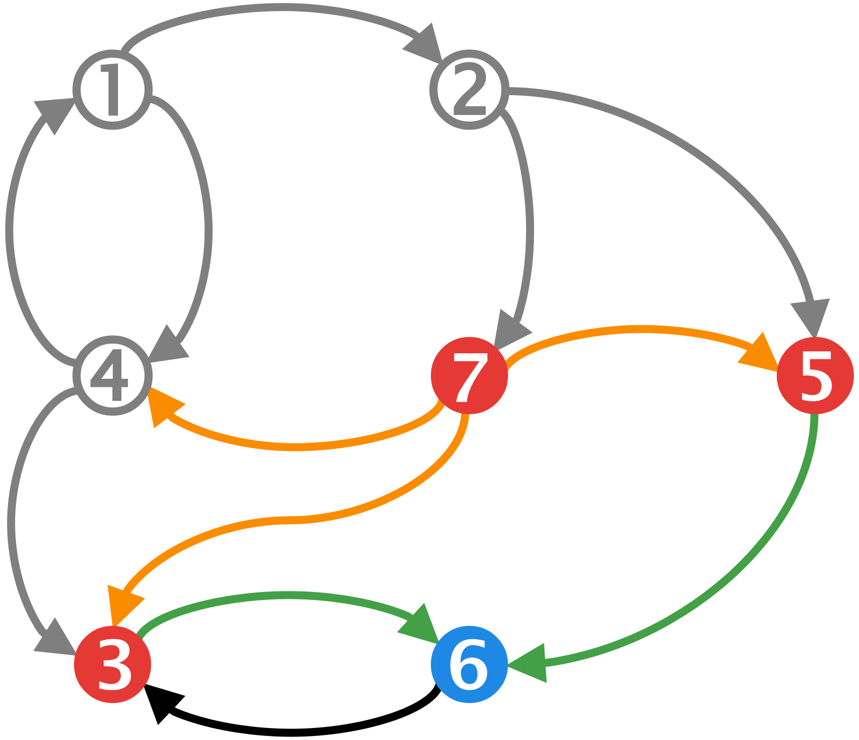
⇓ *level*

	①	②	③	④	⑤	⑥	⑦
s	1	2		2			

BFS – LEVELS

semiring	domain	⊕	⊗	0
any-pair	{T, F}	any	pair	F

level = 3



A	①	②	③	④	⑤	6	⑦
①		●		●			
②					●		●
3						●	
④	●		●				
5						●	
⑥			●				
7			●	●	●		

	①	②	3	④	5	⑥	7
f			●		●		●

⇓ *level*

	①	②	3	④	5	⑥	7
s	1	2	3	2	3		3

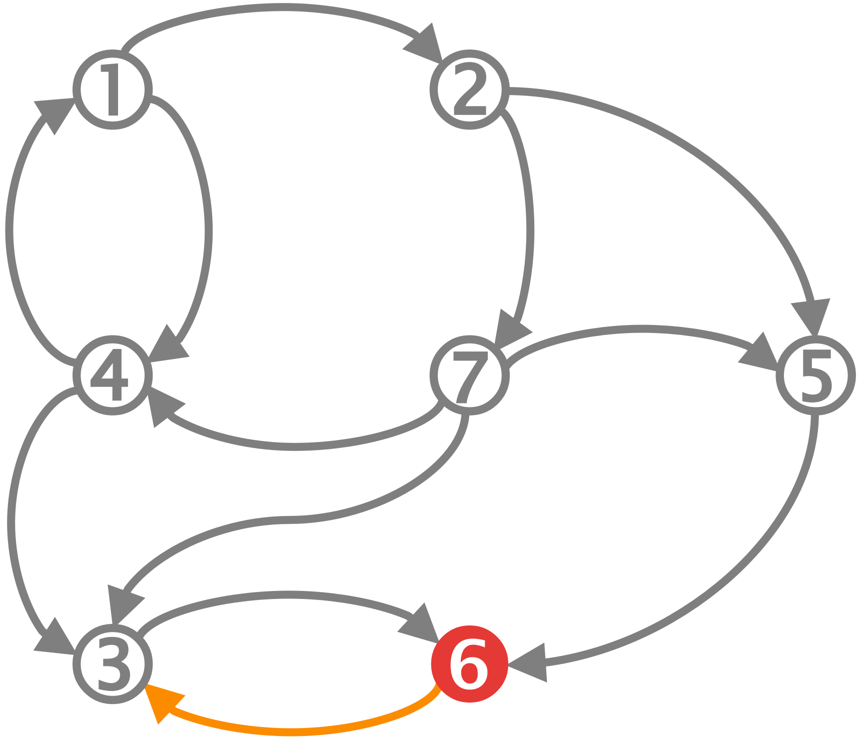
☒	☒	☒	☒	☒	●	☒

$\mathbf{f}\langle\neg\mathbf{s}\rangle = \mathbf{f}\ \text{any.pair}\ \mathbf{A}$

BFS – LEVELS

semiring	domain	⊕	⊗	0
any-pair	$\{T, F\}$	any	pair	F

$\mathit{level} = 4$



$$
\begin{array}{c|ccccccc}
\mathbf{A} & ① & ② & ③ & ④ & ⑤ & ⑥ & ⑦ \\
\hline
① & & ● & & ● & & & \\
② & & & & & ● & & ● \\
③ & & & & & & ● & \\
④ & ● & & ● & & & & \\
⑤ & & & & & & ● & \\
⑥ & & & ● & & & & \\
⑦ & & & ● & ● & ● & & \\
\end{array}
$$

$$
\begin{array}{c|ccccccc}
 & ① & ② & ③ & ④ & ⑤ & ⑥ & ⑦ \\
\hline
\mathbf{f} & & & & & & ● & \\
\end{array}
$$

☒ ☒ ☒ ☒ ☒ ☒ ☒

$\Downarrow$ *level*

$$
\begin{array}{c|ccccccc}
 & ① & ② & ③ & ④ & ⑤ & ⑥ & ⑦ \\
\hline
\mathbf{s} & 1 & 2 & 3 & 2 & 3 & 4 & 3 \\
\end{array}
$$

$\mathbf{f}\langle\neg\mathbf{s}\rangle = \mathbf{f}\ \text{any.pair}\ \mathbf{A}$

f is empty
→ terminate

BFS – LEVELS: ALGORITHM

- **Input:** adjacency matrix $\mathbf{A}$, source vertex s, #vertices n
- **Output:** vector of visited vertices $\mathbf{v}$ (integer)
- **Workspace:** frontier vector $\mathbf{f}$ (Boolean)

```
$\mathbf{f}(s) = \mathrm{T}$
for $\mathit{level} = 1$ to $n - 1$              *terminate earlier if $\mathbf{f}$ is empty
    $\mathbf{s}\langle\mathbf{f}\rangle = \mathit{level}$                    assign the level value to the vertices in the frontier
    $\mathrm{clear}(\mathbf{f})$                        clear the frontier $\mathbf{f}$
    $\mathbf{f}\langle\neg\mathbf{s}\rangle = \mathbf{f}\ \mathrm{any.pair}\ \mathbf{A}$    advance the frontier
```

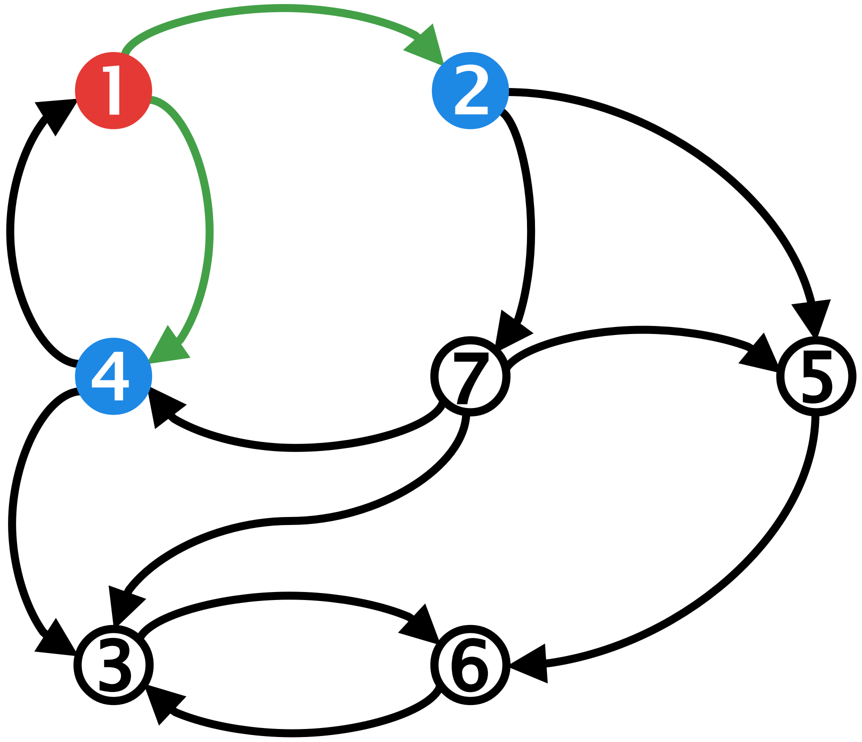
Graph algorithms in GraphBLAS

Breadth-first search / Parents

BFS – PARENTS

semiring	domain	⊕	⊗	0
min-first	$\mathbb{N}$	min	first	0

$$\text{first}(x, y) = x$$



A	①	②	③	④	⑤	⑥	⑦
①		●		●			
②					●		●
③						●	
④	●		●				
⑤						●	
⑥			●				
⑦			●	●	●		

	①	②	③	④	⑤	⑥	⑦
f	1						

☒	1		1			

$$\mathbf{f}\langle\neg\mathbf{p}\rangle = \mathbf{f}\ \text{min}\,.\,\text{first}\ \mathbf{A}$$

⇙ $\mathbf{p}\langle\mathbf{f}\rangle = \mathbf{f}$

⇓ $\mathbf{f}\langle\mathbf{f}\rangle = \mathbf{id}$

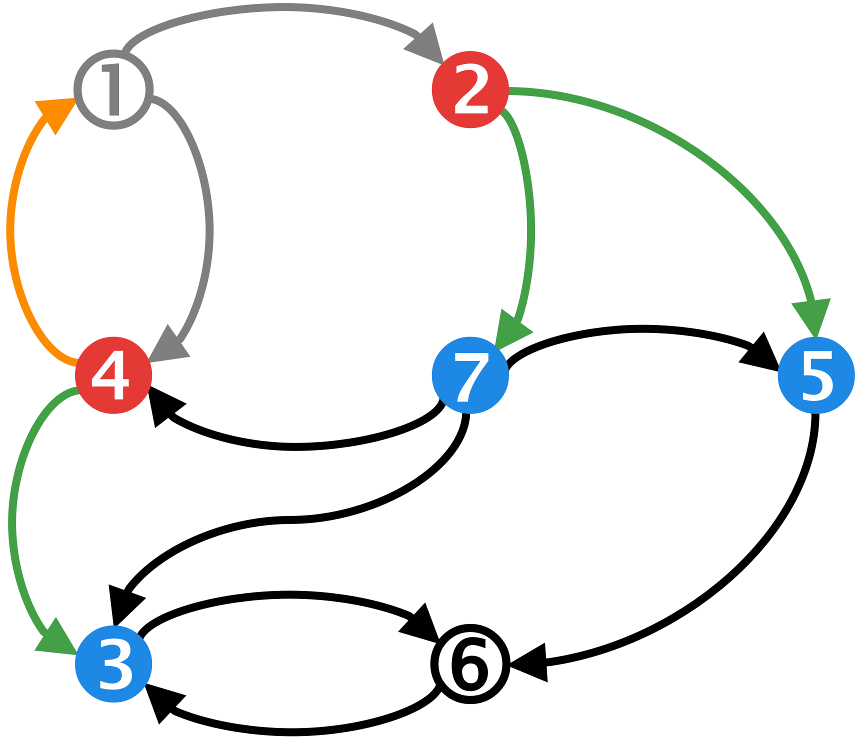
	①	②	③	④	⑤	⑥	⑦
p	0	1		1			

①	②	③	④	⑤	⑥	⑦
	2		4			

BFS – PARENTS

semiring	domain	$\oplus$	$\otimes$	0
min-first	$\mathbb{N}$	min	first	0

$$\text{first}(x, y) = x$$



$$\begin{array}{c|ccccccc}
\mathbf{A} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & & ● & & ● & & & \\
2 & & & & & ● & & ● \\
3 & & & & & & ● & \\
4 & ● & & ● & & & & \\
5 & & & & & & ● & \\
6 & & & ● & & & & \\
7 & & & ● & ● & ● & & \\
\end{array}$$

$$\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
\mathbf{f} & & 2 & & 4 & & & \\
\end{array}
\qquad
\begin{array}{ccccccc}
\boxtimes & \boxtimes & 4 & \boxtimes & 2 & & 2 \\
\end{array}$$

$$\mathbf{f}\langle\neg\mathbf{p}\rangle = \mathbf{f}\ \text{min}\,.\,\text{first}\ \mathbf{A}$$

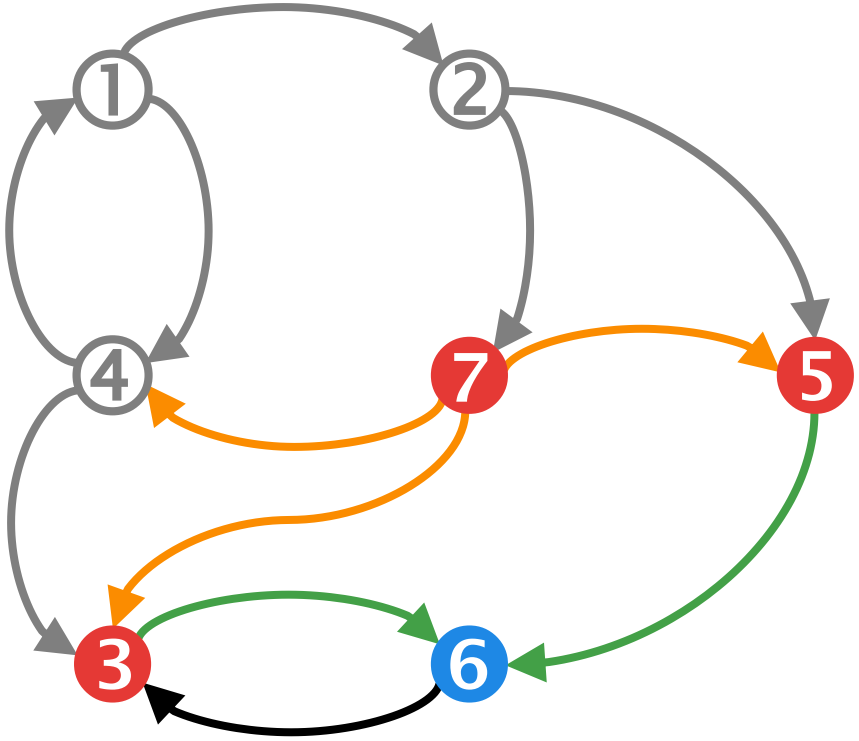
⇙ $\mathbf{p}\langle\mathbf{f}\rangle = \mathbf{f}$ ⇩ $\mathbf{f}\langle\mathbf{f}\rangle = \mathbf{id}$

$$\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
\mathbf{p} & 0 & 1 & 4 & 1 & 2 & & 2 \\
\end{array}
\qquad
\begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
 & & 3 & & 5 & & 7 \\
\end{array}$$

BFS – PARENTS

semiring	domain	⊕	⊗	0
min-first	ℕ	min	first	0

$$\mathrm{first}(x, y) = x$$



A	①	②	③	④	⑤	❻	⑦
①		●		●			
②					●		●
❸						●	
④	●		●				
❺						●	
⑥			●				
❼			●	●	●		

	①	②	❸	④	❺	⑥	❼
f			3		5		7

☒	☒	☒	☒	☒	3	☒

$$\mathbf{f}\langle\neg\mathbf{p}\rangle = \mathbf{f}\ \mathrm{min}\,.\,\mathrm{first}\ \mathbf{A}$$

⇙ $\mathbf{p}\langle\mathbf{f}\rangle = \mathbf{f}$

⇩ $\mathbf{f}\langle\mathbf{f}\rangle = \mathbf{id}$

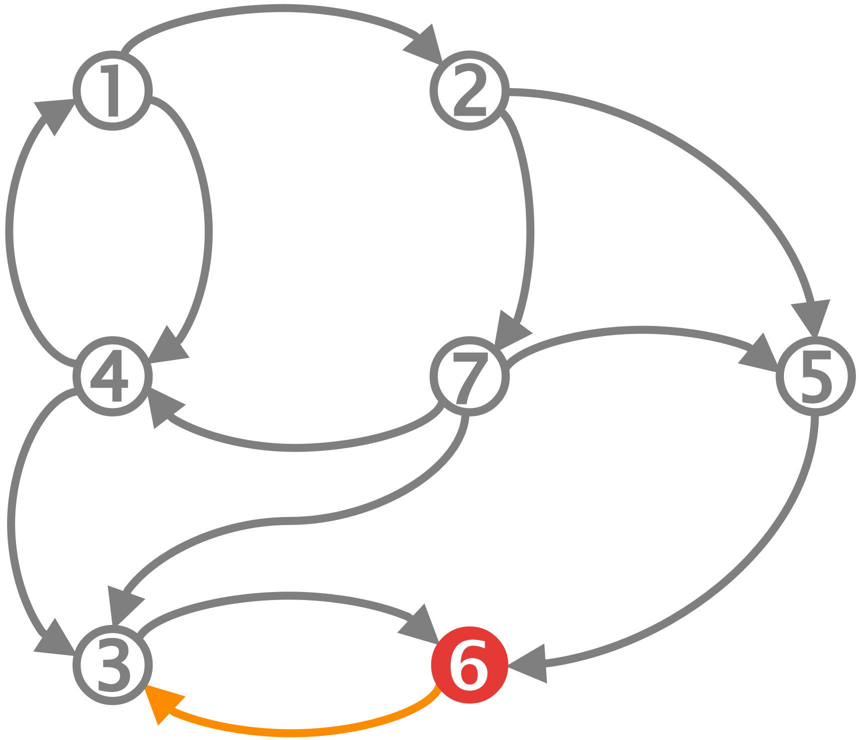
	①	②	③	④	⑤	❻	⑦
p	0	1	4	1	2	3	2

①	②	③	④	⑤	❻	⑦
					6	

BFS – PARENTS

semiring	domain	⊕	⊗	0
min-first	$\mathbb{N}$	min	first	0

$$\text{first}(x, y) = x$$



$$\mathbf{A}\quad \begin{array}{c|ccccccc} & ① & ② & ③ & ④ & ⑤ & ⑥ & ⑦ \\ \hline ① & & ● & & ● & & & \\ ② & & & & & ● & & ● \\ ③ & & & & & & ● & \\ ④ & ● & & ● & & & & \\ ⑤ & & & & & & ● & \\ ❻ & & & ● & & & & \\ ⑦ & & & ● & ● & ● & & \end{array}$$

	①	②	③	④	⑤	❻	⑦
f						6	

☒	☒	☒	☒	☒	☒	☒

$$\mathbf{f}\langle\neg\mathbf{p}\rangle = \mathbf{f}\ \text{min}\,.\,\text{first}\ \mathbf{A}$$

	①	②	③	④	⑤	⑥	⑦
p	0	1	4	1	2	3	2

f is empty
→ terminate

BFS – PARENTS: ALGORITHM

- **Input:** adjacency matrix $\mathbf{A}$, source vertex s, #vertices n
- **Output:** parent vertices vector $\mathbf{p}$ (integer)
- **Workspace:** vertex index vector $\mathbf{idx}$ (integer), frontier vector $\mathbf{f}$ (integer)

```
$\mathbf{idx} = [1\ 2\ \ldots\ n]$                                  we assume 1-based indexing here
$\mathbf{f}(s) = s$
$\mathbf{p}(s) = 0$
for $l = 1$ to $n - 1$                                  *terminate earlier if the frontier is empty
    $\mathbf{f}\langle\neg\mathbf{p}\rangle = \mathbf{f}\ \mathrm{min}.\mathrm{first}\ \mathbf{A}$    advance the frontier
    $\mathbf{p}\langle\mathbf{f}\rangle = \mathbf{f}$                                   assign parent ids to the frontier's vertices
    $\mathbf{f}\langle\mathbf{f}\rangle = \mathbf{idx}$                                 assign vertex ids $\mathbf{f}(i) = i$
```

BFS – PARENTS: OPTIMIZATIONS

- If getting deterministic results is not a requirement (i.e. any parent vertex can be returned), instead of $\mathrm{min.sel1st}$, one can use the $\mathrm{any.first}$ semiring.
- This optimization is allowed by the GAP Benchmark Suite.
- Direction-optimizing traversal (push/pull) can be exploited.
- The secondi (note the “i”) semiring can be used to express the BFS Parent algorithm. When using this semiring, one does not even have to look at the values in either $\mathbf{A}$ of $\mathbf{f}$.

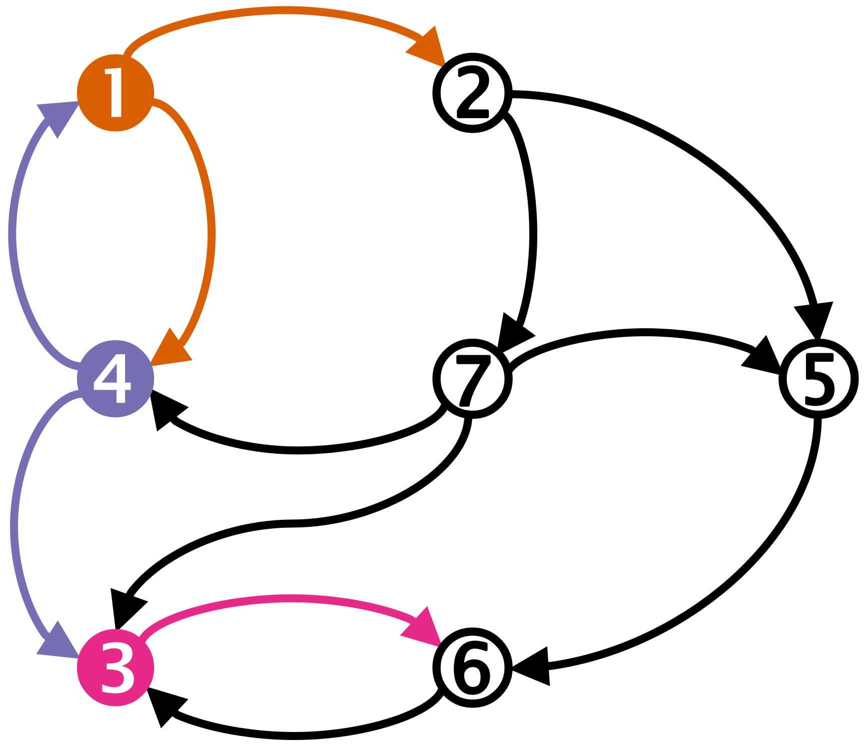
This algorithm is described in:
Evaluation of Graph Analytics Frameworks Using the GAP Benchmark Suite, IISWC 2020

Graph algorithms in GraphBLAS

Multi-source BFS

MULTI-SOURCE BFS – LEVELS

semiring	domain	⊕	⊗	0
any-pair	{T, F}	any	pair	F



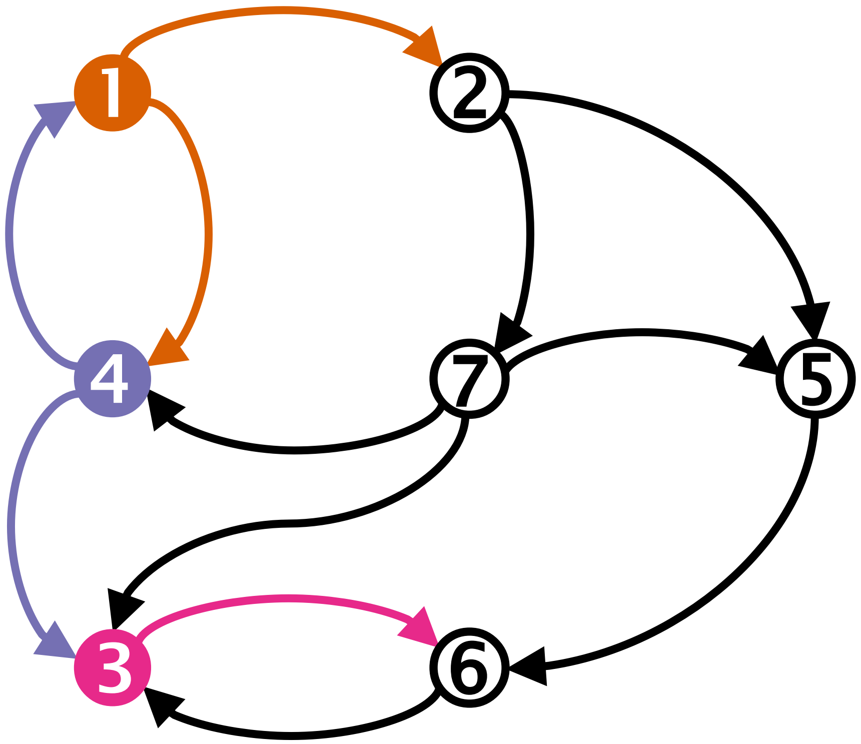
$$\begin{array}{c|ccccccc}
\mathbf{A} & ① & ② & ③ & ④ & ⑤ & ⑥ & ⑦ \\
\hline
1 & & \bullet & & \bullet & & & \\
② & & & & & \bullet & & \bullet \\
3 & & & & & & \bullet & \\
4 & \bullet & & \bullet & & & & \\
⑤ & & & & & & \bullet & \\
⑥ & & & \bullet & & & & \\
⑦ & & & \bullet & \bullet & \bullet & &
\end{array}$$

$$\begin{array}{c|ccccccc}
\mathbf{F} & 1 & ② & 3 & 4 & ⑤ & ⑥ & ⑦ \\
\hline
t1 & \bullet & & & & & & \\
t2 & & & \bullet & & & & \\
t3 & & & & \bullet & & &
\end{array}
\qquad
\begin{array}{ccccccc}
\times & \bullet & & \bullet & & & \\
 & & \times & & & \bullet & \\
\bullet & & \bullet & \times & & &
\end{array}$$

$$\mathbf{F}\langle\neg\mathbf{S}\rangle = \mathbf{F}\ \text{any.pair}\ \mathbf{A}$$

MULTI-SOURCE BFS – PARENTS

semiring	domain	⊕	⊗	0
min-first	ℕ ∪ {+∞}	min	first	+∞



A

	①	②	③	④	⑤	⑥	⑦
❶		●		●			
②					●		●
❸						●	
❹	●		●				
⑤						●	
⑥			●				
⑦			●	●	●		

F

	❶	②	❸	❹	⑤	⑥	⑦
t1	1						
t2			3				
t3				4			

☒	1		1			
		☒			3	
4		4	☒			

$$\mathbf{F}\langle\neg\mathbf{P}\rangle = \mathbf{F}\ \min.\text{first}\ \mathbf{A}$$

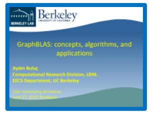
BFS – PERFORMANCE

- Naïve BFS impls can be slow on real graphs with skewed distributions – further optimizations are needed.
- Direction-optimizing BFS was published in 2012.
  - Switches between push ($\mathbf{vA}$) and pull ($\mathbf{A}^\top \mathbf{v}$) during execution:
    - Use the push direction when the frontier is small
    - Use the pull direction when the frontier becomes large
  - Adopted to GraphBLAS in 2018 (Yang et al.’s ICPP paper)

S. Beamer, K. Asanovic, D. Patterson: *Direction-Optimizing Breadth-First Search*, SC 2012

C. Yang, A. Buluç, J.D. Owens: *Implementing Push-Pull Efficiently in GraphBLAS*, ICPP 2018

C. Yang: *High-performance linear algebra-based graph framework on the GPU*, PhD thesis, UC Davis, 2019



GraphBLAS: concepts, algorithms, and applications

Aydın Buluç
Computational Research Division, LBNL
EECS Department, UC Berkeley

14th Scheduling Workshop
June 27, 2019, Bordeaux

A. Buluç: *GraphBLAS: Concepts, algorithms, and applications*, Scheduling Workshop, 2019

Graph algorithms in GraphBLAS

Single-source shortest paths

SSSP – SINGLE-SOURCE SHORTEST PATHS

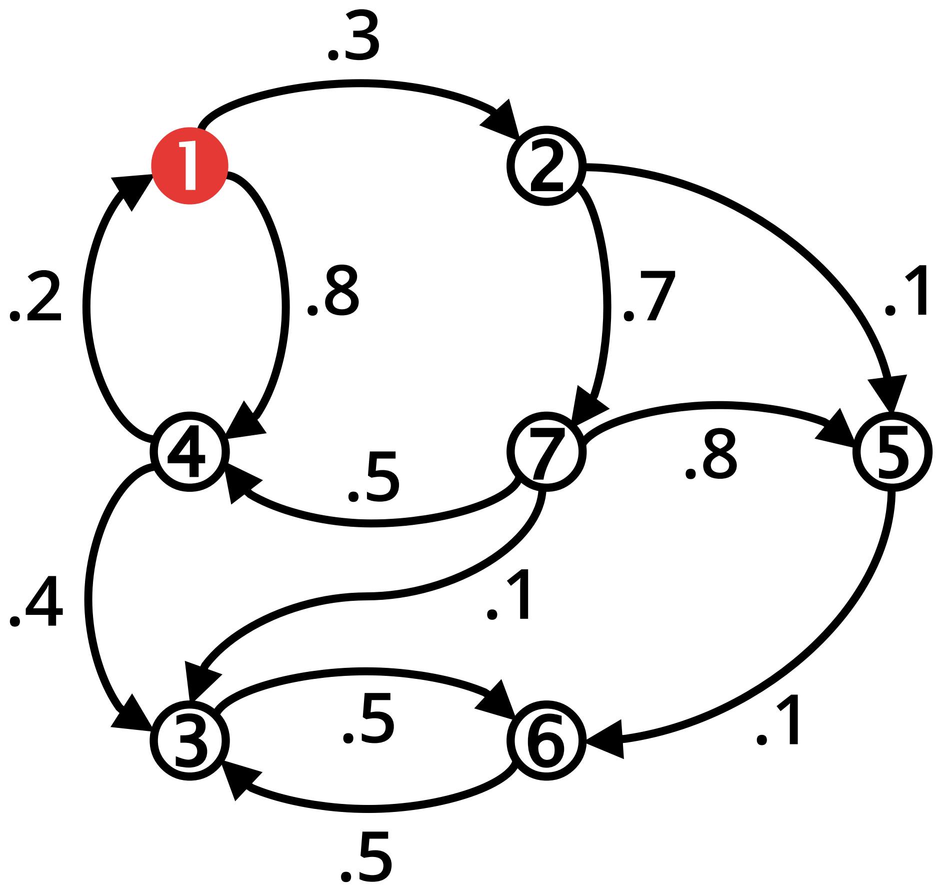
- **Problem:**
  - From a given start vertex s, find the shortest paths to every other (reachable) vertex in the graph

- **Bellman-Ford algorithm:**
  - Relaxes all edges in each step
  - Guaranteed to find the shortest path using at most $n - 1$ steps

- **Observation:**
  - The relaxation step can be captured using a VM multiplication
  - Unlike in BFS, there is no masking here, as revisiting edges that have been visited previously can be useful.

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	domain	⊕	⊗	0
min-plus	$\mathbb{R} \cup \{+\infty\}$	min	+	$+\infty$



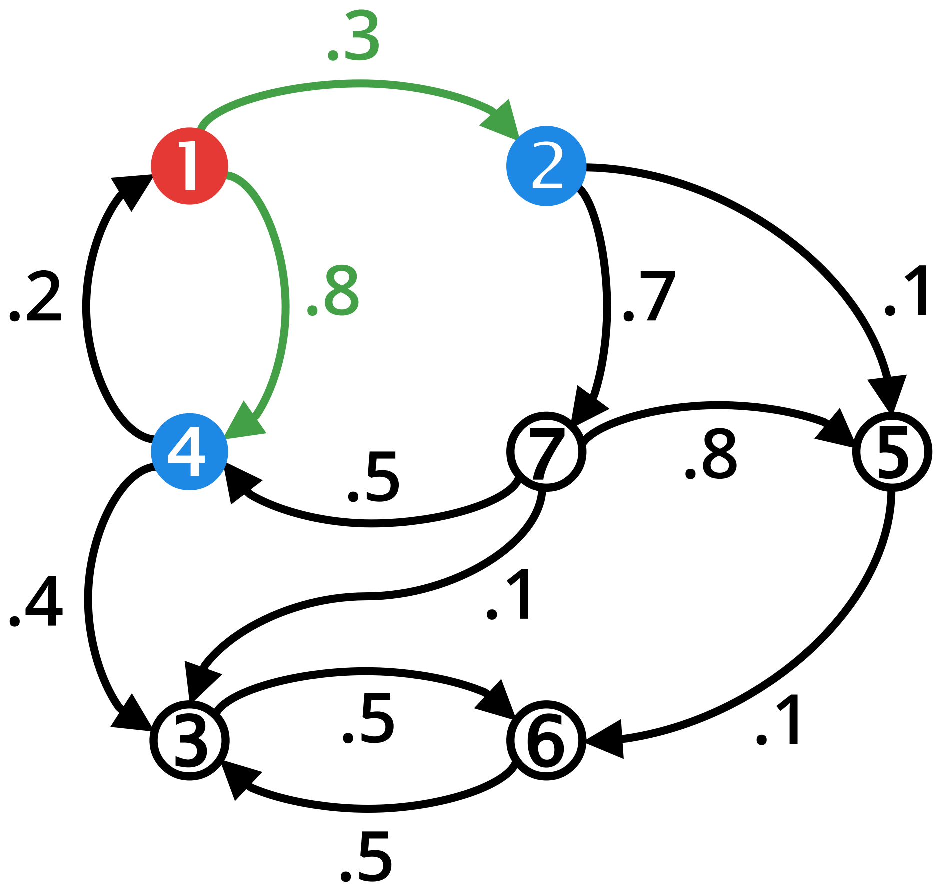
A	①	②	③	④	⑤	⑥	⑦
①	0	.3		.8			
②		0			.1		.7
③			0			.5	
④	.2		.4	0			
⑤					0	.1	
⑥			.5			0	
⑦			.1	.5	.9		0

	❶	②	③	④	⑤	⑥	⑦
d	0						

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	domain	⊕	⊗	0
min-plus	$\mathbb{R} \cup \{+\infty\}$	min	$+$	$+\infty$



A	①	②	③	④	⑤	⑥	⑦
❶	0	.3		.8			
②		0			.1		.7
③			0			.5	
④	.2		.4	0			
⑤					0	.1	
⑥			.5			0	
⑦			.1	.5	.9		0

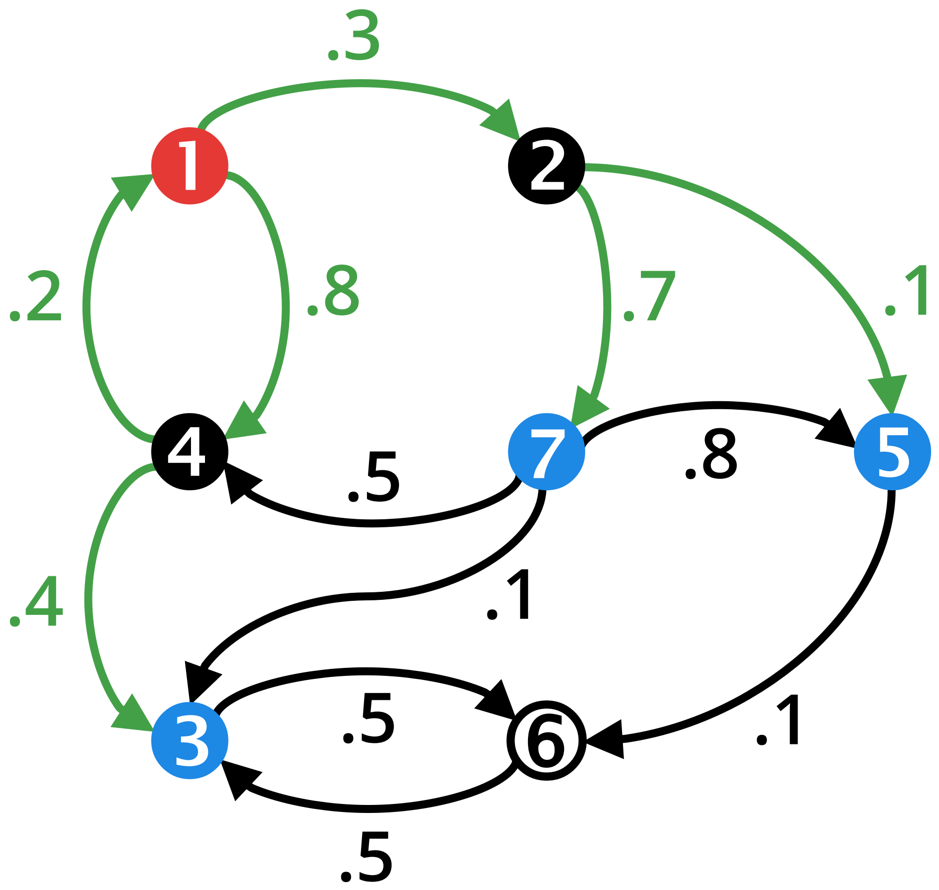
	❶	②	③	④	⑤	⑥	⑦
d	0						

0	.3		.8			

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	domain	⊕	⊗	0
min-plus	$\mathbb{R} \cup \{+\infty\}$	min	+	$+\infty$



A	①	②	③	④	⑤	⑥	⑦
❶	0	.3		.8			
❷		0			.1		.7
③			0			.5	
❹	.2		.4	0			
⑤					0	.1	
⑥			.5			0	
⑦			.1	.5	.9		0

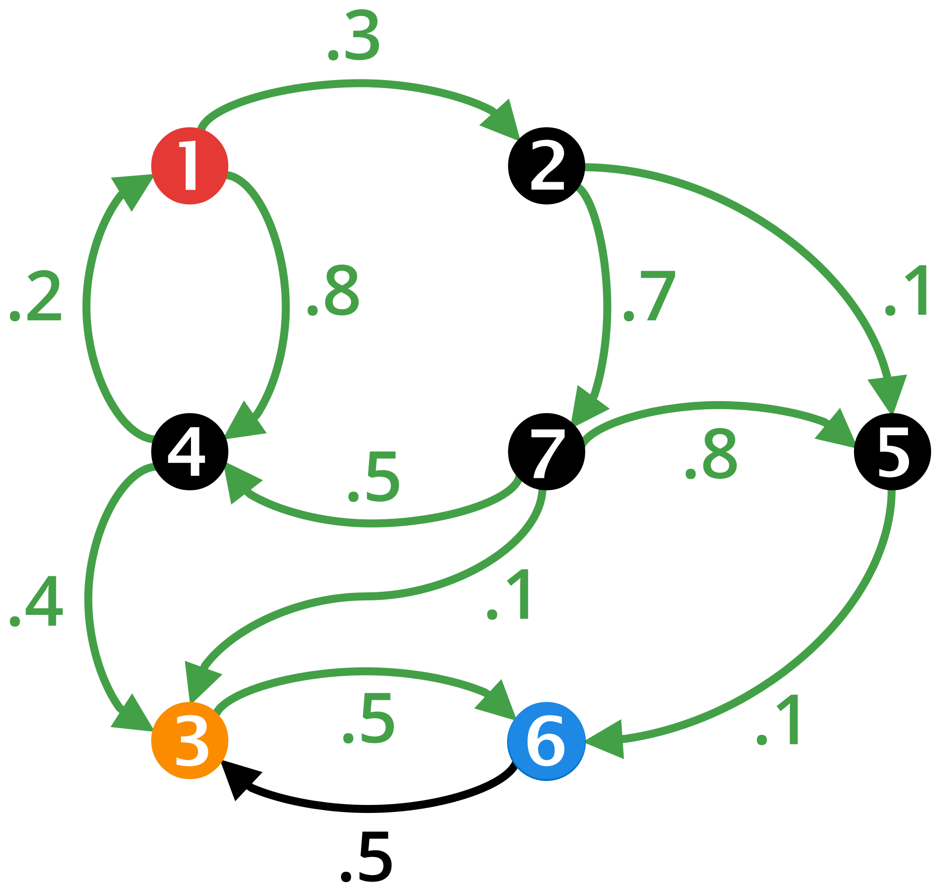
	❶	❷	③	❹	⑤	⑥	⑦
d	0	.3		.8			

0	.3	1.2	.8	.4		1

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	domain	⊕	⊗	0
min-plus	$\mathbb{R} \cup \{+\infty\}$	min	$+$	$+\infty$



A	①	②	③	④	⑤	⑥	⑦
❶	0	.3		.8			
❷		0			.1		.7
❸			0			.5	
❹	.2		.4	0			
❺					0	.1	
⑥			.5			0	
❼			.1	.5	.9		0

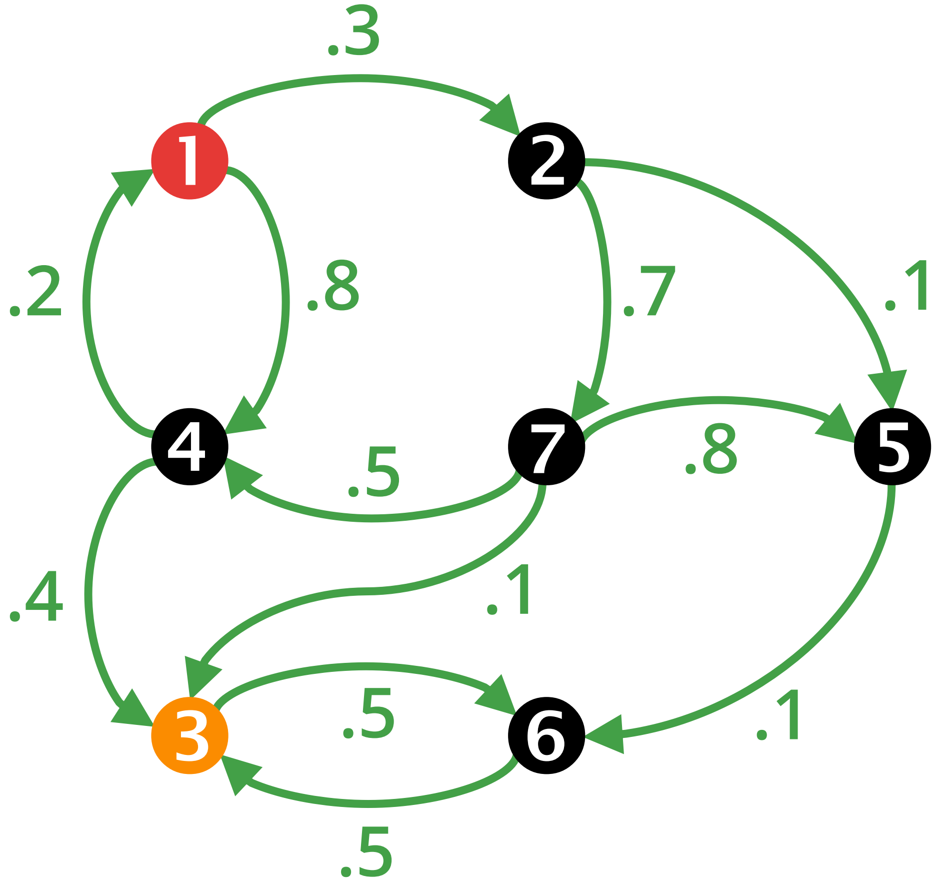
	❶	❷	❸	❹	❺	⑥	❼
d	0	.3	1.2	.8	.4		1

0	.3	1.1	.8	.4	.5	1

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	domain	⊕	⊗	0
min-plus	$\mathbb{R} \cup \{+\infty\}$	min	$+$	$+\infty$



$$
\begin{array}{c|ccccccc|}
\mathbf{A} & \text{①} & \text{②} & \text{③} & \text{④} & \text{⑤} & \text{⑥} & \text{⑦} \\
\hline
\text{❶} & 0 & .3 & & .8 & & & \\
\text{❷} & & 0 & & & .1 & & .7 \\
\text{❸} & & & 0 & & & .5 & \\
\text{❹} & .2 & & .4 & 0 & & & \\
\text{❺} & & & & & 0 & .1 & \\
\text{❻} & & & .5 & & & 0 & \\
\text{❼} & & & .1 & .5 & .9 & & 0 \\
\hline
\end{array}
$$

$$
\begin{array}{c|c|c|c|c|c|c|c|}
 & \text{❶} & \text{❷} & \text{❸} & \text{❹} & \text{❺} & \text{❻} & \text{❼} \\
\hline
\mathbf{d} & 0 & .3 & 1.1 & .8 & .4 & .5 & 1 \\
\hline
\end{array}
$$

$$
\begin{array}{|c|c|c|c|c|c|c|}
\hline
0 & .3 & 1 & .8 & .4 & .5 & 1 \\
\hline
\end{array}
$$

$\mathbf{d}$ min.+ $\mathbf{A}$

SSSP – ALGEBRAIC BELLMAN-FORD ALGO.

Input: adjacency matrix $\mathbf{A}$, source vertex s, #vertices n

$$\mathbf{A}_{ij} = \begin{cases} 0 & \text{if } i = j \\ w(e_{ij}) & \text{if } e_{ij} \in E \\ \infty & \text{if } e_{ij} \notin E \end{cases}$$

Output: distance vector $\mathbf{d}$ (real)

```
1.  $\mathbf{d} = [\infty\ \infty \dots \infty]$
2.  $\mathbf{d}(s) = 0$
3.  $\text{for } k = 1 \text{ to } n - 1$        *terminate earlier if we reach a fixed point
4.      $\mathbf{d} = \mathbf{d} \text{ min.+ } \mathbf{A}$
```

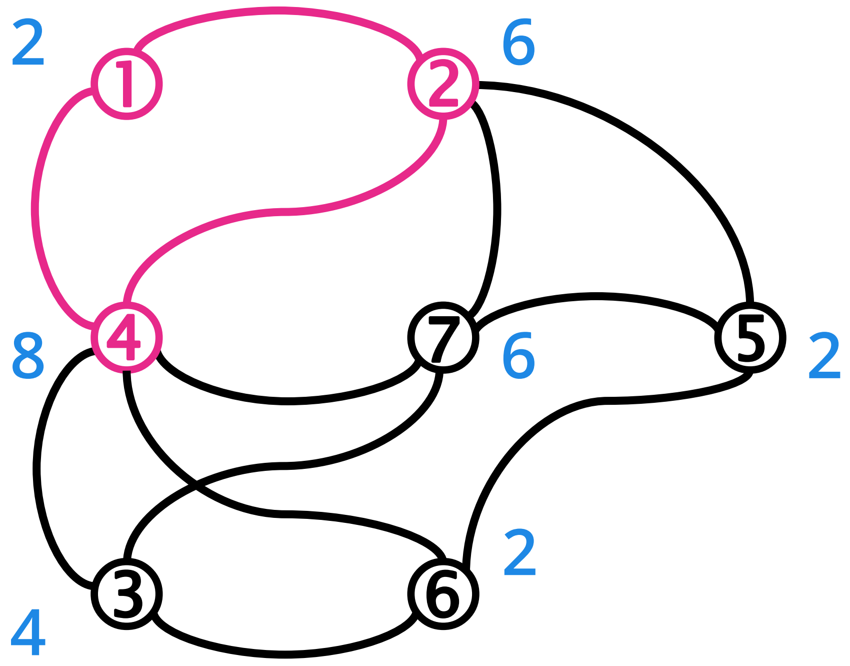
Optimizations for BFS (push/pull) also work here.

Graph algorithms in GraphBLAS

Triangle count / Definition

TRIANGLE COUNT

- IEEE GraphChallenge: an annual competition at the HPEC conference
- The task of the 2017 GraphChallenge was **triangle count:** given a graph G, count the number of triangles.
- **Triangle** = *"set of three mutually adjacent vertices in a graph"*
- Many solutions employed a linear algebraic computation model



Number of unique triangles:

$$\frac{30}{6}$$

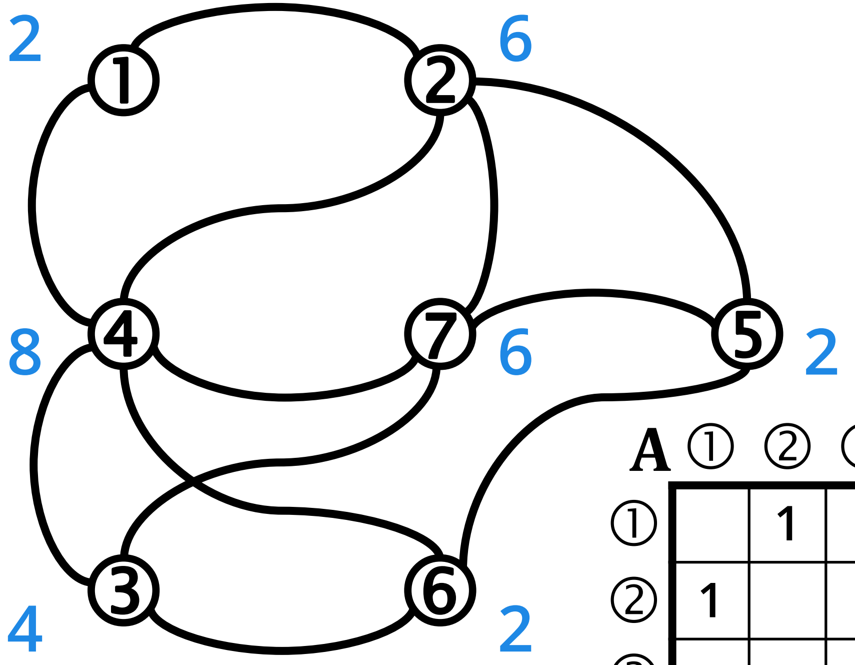
GraphChallenge.org: Raising the Bar on Graph Analytic Performance, HPEC 2018

Graph algorithms in GraphBLAS

Triangle count / Naïve algorithm

TC EXAMPLE: NAÏVE APPROACH

$$\mathbf{tri2} = \mathrm{diag}^{-1}(\mathbf{A} \oplus.\otimes\ \mathbf{A} \oplus.\otimes\ \mathbf{A})$$



A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

A ①	②	③	④	⑤	⑥	⑦
	1		1			
1			1	1		1
			1		1	1
1	1	1			1	1
	1				1	1
		1	1	1		
	1	1	1	1		

A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

2	1	1	1	1	1	2
1	4	2	2	1	2	2
1	2	3	2	2	1	1
1	2	2	5	3	1	2
1	1	2	3	3		1
1	2	1	1		3	3
2	2	1	2	1	3	4

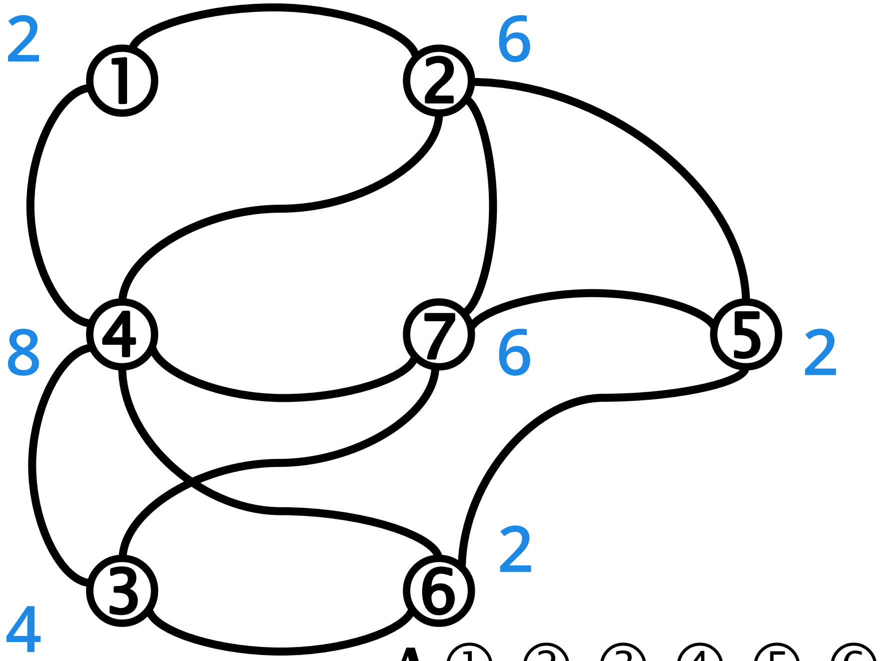
2	6	4	7	4	3	4
6	6	6	11	8	5	9
4	6	4	8	4	7	9
7	11	8	8	5	10	12
4	8	4	5	2	8	9
3	5	7	10	8	2	4
4	9	9	12	9	4	6

tri2
2
6
4
8
2
2
6

Graph algorithms in GraphBLAS

Triangle count / Masked algorithm

TC EXAMPLE: ELEMENT-WISE MULTIPLICATION



A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

$$\mathbf{TRI} = \mathbf{A} \oplus.\otimes \mathbf{A} \otimes \mathbf{A}$$

$$\mathbf{tri2} = \left[\bigoplus_j \mathbf{TRI}(:,j)\right]$$

A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

2	1	1	1	1	1	2
1	4	2	2	1	2	2
1	2	3	2	2	1	1
1	2	2	5	3	1	2
1	1	2	3	3		1
1	2	1	1		3	3
2	2	1	2	1	3	4

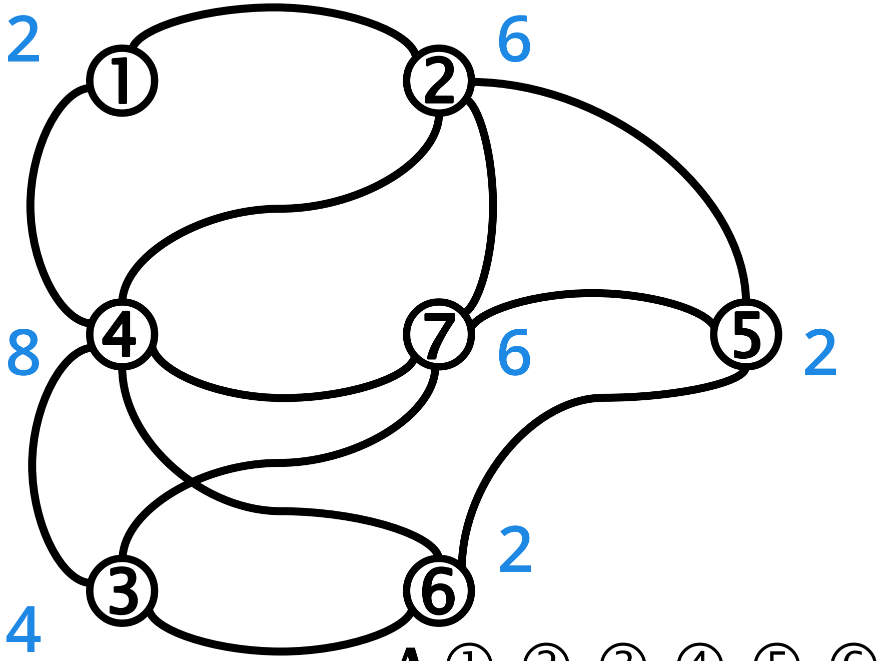
TRI

	1		1			
1			2	1		2
			2		1	1
1	2	2			1	2
	1					1
		1	1			
	2	1	2	1		

$$\xrightarrow{\left[\bigoplus_j \cdots\right]}$$

tri2
2
6
4
8
2
2
6

TC EXAMPLE: ELEMENT-WISE MULTIPLICATION



A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

Masking limits where the operation is computed. Here, we use **A** as a mask for $\mathbf{A} \oplus.\otimes \mathbf{A}$.

$$\mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} \oplus.\otimes \mathbf{A}$$

$$\mathbf{tri2} = \left[\bigoplus_j \mathbf{TRI}(:,j)\right]$$

A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

	1		1			
1			2	1		2
			2		1	1
1	2	2			1	2
	1					1
		1	1			
	2	1	2	1		

$$\xrightarrow{[+_j \cdots]}$$

tri2
2
6
4
8
2
2
6

Graph algorithms in GraphBLAS

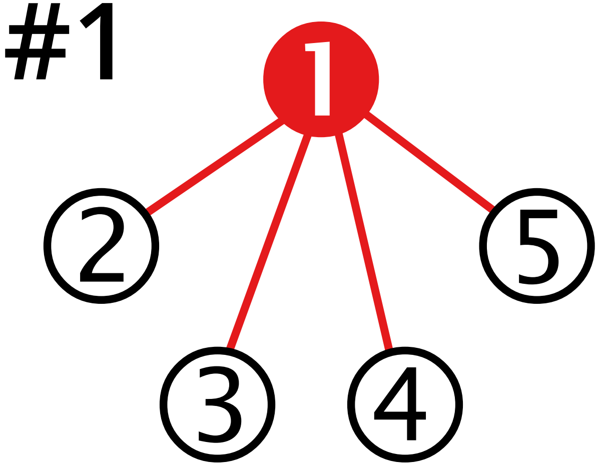
The importance of masking

THE IMPORTANCE OF MASKING

Q: Is masking absolutely necessary?

A: **Yes, it can reduce the complexity of some algorithms.**
We demonstrate this with two examples.

#1



$$\begin{array}{c|ccccc}
\mathbf{A} & \text{❶} & \text{②} & \text{③} & \text{④} & \text{⑤} \\
\hline
\text{❶} & & 1 & 1 & 1 & 1 \\
\text{②} & 1 & & & & \\
\text{③} & 1 & & & & \\
\text{④} & 1 & & & & \\
\text{⑤} & 1 & & & &
\end{array}
\qquad
\begin{array}{ccccc}
\mathbf{A}\ \text{❶} & \text{②} & \text{③} & \text{④} & \text{⑤} \\
\hline
 & 1 & 1 & 1 & 1 \\
1 & & & & \\
1 & & & & \\
1 & & & & \\
1 & & & &
\end{array}$$

A simple corner case is the star graph: there are $(n-1)^2$ wedges but none of them close into triangles.

We do quadratic work while it's clear that there are no triangles in the graph (it's a tree).

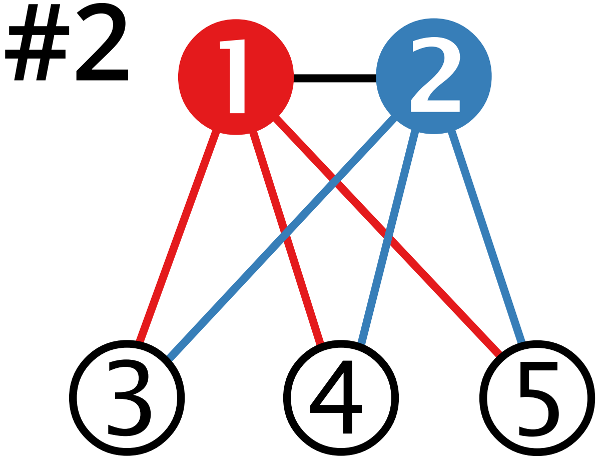
$$\begin{array}{c|ccccc}
\mathbf{A} & \text{❶} & \text{②} & \text{③} & \text{④} & \text{⑤} \\
\hline
\text{❶} & & 1 & 1 & 1 & 1 \\
\text{②} & 1 & & & & \\
\text{③} & 1 & & & & \\
\text{④} & 1 & & & & \\
\text{⑤} & 1 & & & &
\end{array}
\qquad
\begin{array}{|c|c|c|c|c|}
\hline
4 & & & & \\
\hline
 & 1 & 1 & 1 & 1 \\
\hline
 & 1 & 1 & 1 & 1 \\
\hline
 & 1 & 1 & 1 & 1 \\
\hline
 & 1 & 1 & 1 & 1 \\
\hline
\end{array}
\qquad
\begin{array}{|c|c|c|c|c|}
\hline
\ & \ & \ & \ & \ \\
\hline
\ & \ & \ & \ & \ \\
\hline
\ & \ & \ & \ & \ \\
\hline
\ & \ & \ & \ & \ \\
\hline
\ & \ & \ & \ & \ \\
\hline
\end{array}$$

$\mathbf{A}$

$\mathbf{A} \oplus.\otimes \mathbf{A}$

$\mathbf{A} \oplus.\otimes \mathbf{A} \oplus.\otimes \mathbf{A}$

#2



$$\begin{array}{c|ccccc} \mathbf{A} & ❶ & ❷ & ③ & ④ & ⑤ \\ \hline ❶ & & 1 & 1 & 1 & 1 \\ ❷ & 1 & & 1 & 1 & 1 \\ ③ & 1 & 1 & & & \\ ④ & 1 & 1 & & & \\ ⑤ & 1 & 1 & & & \end{array} \qquad \begin{array}{c|ccccc} \mathbf{A} & ❶ & ❷ & ③ & ④ & ⑤ \\ \hline & & 1 & 1 & 1 & 1 \\ & 1 & & 1 & 1 & 1 \\ & 1 & 1 & & & \\ & 1 & 1 & & & \\ & 1 & 1 & & & \end{array}$$

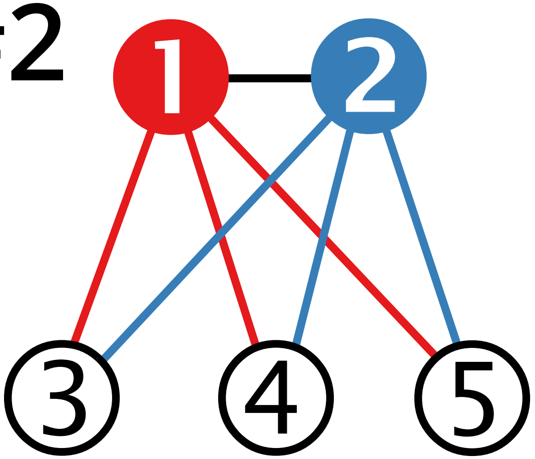
$$\begin{array}{c|ccccc} \mathbf{A} & ❶ & ❷ & ③ & ④ & ⑤ \\ \hline ❶ & & 1 & 1 & 1 & 1 \\ ❷ & 1 & & 1 & 1 & 1 \\ ③ & 1 & 1 & & & \\ ④ & 1 & 1 & & & \\ ⑤ & 1 & 1 & & & \end{array} \qquad \begin{array}{ccccc} 4 & 3 & 1 & 1 & 1 \\ 3 & 4 & 1 & 1 & 1 \\ 1 & 1 & 2 & 2 & 2 \\ 1 & 1 & 2 & 2 & 2 \\ 1 & 1 & 2 & 2 & 2 \end{array} \qquad \begin{array}{ccccc} & 3 & 1 & 1 & 1 \\ 3 & & 1 & 1 & 1 \\ 1 & 1 & & & \\ 1 & 1 & & & \\ 1 & 1 & & & \end{array}$$

$\mathbf{A}$ $\qquad$ $\mathbf{A} \oplus.\otimes \mathbf{A}$ $\qquad$ $\mathbf{A} \oplus.\otimes \mathbf{A} \oplus.\otimes \mathbf{A}$

A full bipartite graph $K_{2,3}$ with the vertices in the top partition connected.

A bipartite graph only has cycles of even length, so it's easy to see that all triangles will contain the two vertices in the top partition. Still, $\mathbf{A} \oplus.\otimes \mathbf{A}$ enumerates all wedges starting and ending in the bottom partition, thus performing a lot of unnecessary work.

#2



$$\begin{array}{c|ccccc} \mathbf{A} & \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} \\ \hline \textcircled{1} & & 1 & 1 & 1 & 1 \\ \textcircled{2} & 1 & & 1 & 1 & 1 \\ \textcircled{3} & 1 & 1 & & & \\ \textcircled{4} & 1 & 1 & & & \\ \textcircled{5} & 1 & 1 & & & \end{array}$$

Masking avoids the materialization of large interim data sets by ensuring that we only enumerate wedges whose endpoints are already connected.

$$\begin{array}{c|ccccc} \mathbf{A} & \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} \\ \hline \textcircled{1} & & 1 & 1 & 1 & 1 \\ \textcircled{2} & 1 & & 1 & 1 & 1 \\ \textcircled{3} & 1 & 1 & & & \\ \textcircled{4} & 1 & 1 & & & \\ \textcircled{5} & 1 & 1 & & & \end{array} \qquad \begin{array}{|ccccc|} & 3 & 1 & 1 & 1 \\ 3 & & 1 & 1 & 1 \\ 1 & 1 & & & \\ 1 & 1 & & & \\ 1 & 1 & & & \end{array} \xrightarrow{[\oplus_j \mathbf{TRI}(:,j)]} \begin{array}{|c|} 6 \\ 6 \\ 2 \\ 2 \\ 2 \end{array}$$

$$\mathbf{A} \qquad \mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} \oplus.\otimes \mathbf{A} \qquad \mathbf{tri}$$

Graph algorithms in GraphBLAS

Triangle count / Cohen’s algorithm

COHEN’S ALGORITHM: PSEUDOCODE

Input: adjacency matrix $\mathbf{A}$

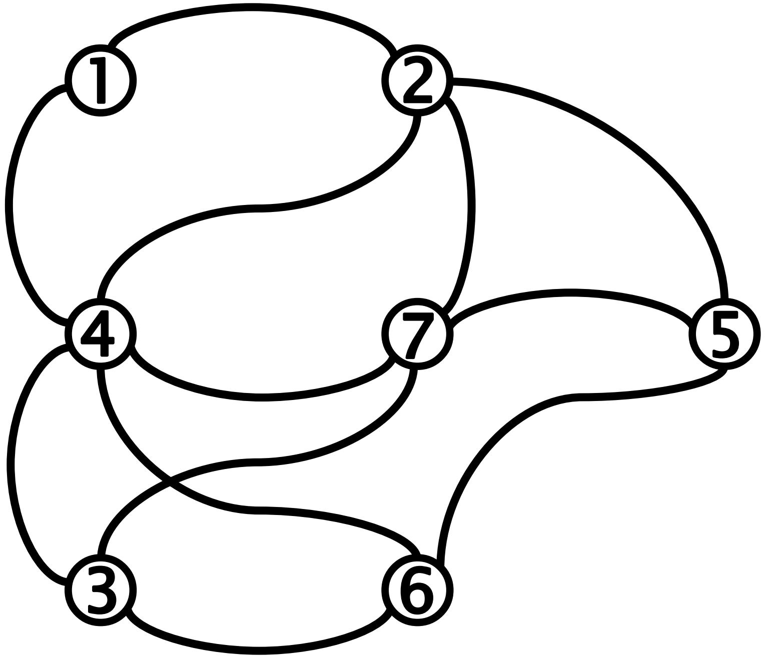
Output: triangle count t

Workspace: matrices $\mathbf{L}$, $\mathbf{U}$, $\mathbf{B}$, $\mathbf{C}$

```
1. $\mathbf{L} = \mathrm{tril}(\mathbf{A})$                    extract the lower triangle from A
2. $\mathbf{U} = \mathrm{triu}(\mathbf{A})$                    extract the upper triangle from A
3. $\mathbf{B} = \mathbf{L} \oplus.\otimes \mathbf{U}$
4. $\mathbf{C} = \mathbf{B} \otimes \mathbf{A}$
5. $t = \sum \mathbf{C} / \mathbf{2}$                          sum the values in C and divide by 2
```

J. Cohen: *Graph Twiddling in a MapReduce World,* Comput. Sci. Eng. 2009

COHEN’S ALGORITHM

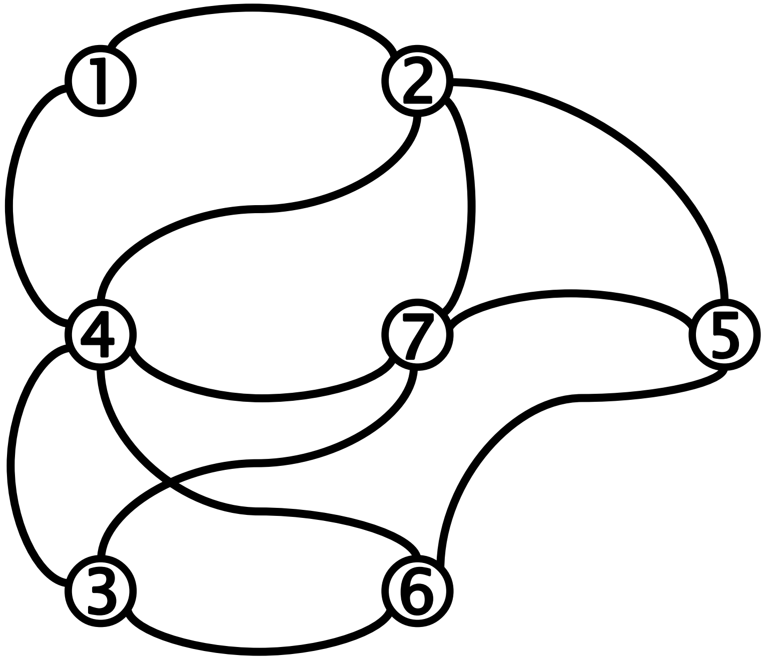


$$\mathbf{U}\;\begin{array}{c|ccccccc}
 & \text{①} & \text{②} & \text{③} & \text{④} & \text{⑤} & \text{⑥} & \text{⑦} \\ \hline
\text{①} & & 1 & & 1 & & & \\
\text{②} & & & & 1 & 1 & & 1 \\
\text{③} & & & & 1 & & 1 & 1 \\
\text{④} & & & & & & 1 & 1 \\
\text{⑤} & & & & & & 1 & 1 \\
\text{⑥} & & & & & & & \\
\text{⑦} & & & & & & &
\end{array}
\quad \otimes \quad
\mathbf{A}\;\begin{array}{ccccccc}
\text{①} & \text{②} & \text{③} & \text{④} & \text{⑤} & \text{⑥} & \text{⑦} \\ \hline
 & 1 & & 1 & & & \\
1 & & & 1 & 1 & & 1 \\
 & & & 1 & & 1 & 1 \\
1 & 1 & 1 & & & 1 & 1 \\
 & 1 & & & & 1 & 1 \\
 & & 1 & 1 & 1 & & \\
 & 1 & 1 & 1 & 1 & &
\end{array}$$

$$\mathbf{L}\;\begin{array}{c|ccccccc}
 & \text{①} & \text{②} & \text{③} & \text{④} & \text{⑤} & \text{⑥} & \text{⑦} \\ \hline
\text{①} & & & & & & & \\
\text{②} & 1 & & & & & & \\
\text{③} & & & & & & & \\
\text{④} & 1 & 1 & 1 & & & & \\
\text{⑤} & & 1 & & & & & \\
\text{⑥} & & & 1 & 1 & 1 & & \\
\text{⑦} & & 1 & 1 & 1 & 1 & &
\end{array}
\quad
\mathbf{B}\;\begin{array}{ccccccc}
 & & & & & & \\
 & 1 & & 1 & & & \\
 & & & & & & \\
 & 1 & & 3 & 1 & 1 & 2 \\
 & & & 1 & 1 & & 1 \\
 & & & 1 & & 3 & 3 \\
 & & & 2 & 1 & 3 & 4
\end{array}
\quad
\mathbf{C}\;\begin{array}{ccccccc}
 & & & & & & \\
 & & & 1 & & & \\
 & & & & & & \\
 & 1 & & & & 1 & 2 \\
 & & & & & & 1 \\
 & & & 1 & & & \\
 & & & 2 & 1 & &
\end{array}$$

$$t = \sum \mathbf{C} \,/2$$

COHEN'S ALGORITHM: MASKING



U	①	②	③	④	⑤	⑥	⑦
①		1		1			
②				1	1		1
③				1		1	1
④						1	1
⑤						1	1
⑥							
⑦							

L	①	②	③	④	⑤	⑥	⑦
①							
②	1						
③							
④	1	1	1				
⑤		1					
⑥			1	1	1		
⑦		1	1	1	1		

C

	▨		▨			
▨			1	▨		▨
			▨		▨	▨
▨	1	▨			1	2
	▨				▨	1
		▨	1	▨		
	▨	▨	2	1		

$$\mathbf{C}\langle\mathbf{A}\rangle = \mathbf{L} \oplus.\otimes \mathbf{U}$$

$$t = \sum \mathbf{C} \,/ \mathbf{2}$$

Graph algorithms in GraphBLAS

Triangle count / Sandia algorithm

SANDIA ALGORITHM

Input: adjacency matrix $\mathbf{A}$

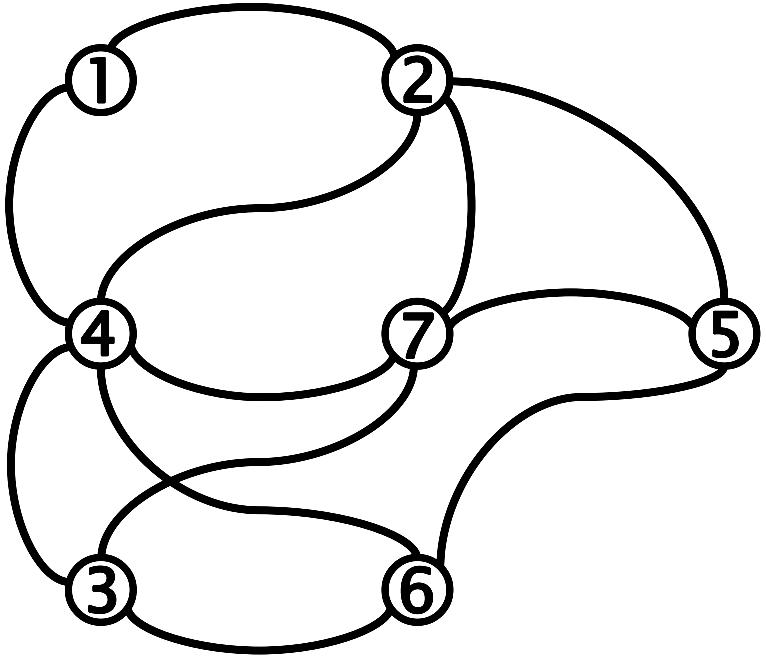
Output: triangle count t

Workspace: matrices $\mathbf{L}$, $\mathbf{U}$, $\mathbf{B}$, $\mathbf{C}$

1. $\mathbf{L} = \mathrm{tril}(\mathbf{A})$ — extract the lower triangle from A
2. $\mathbf{C}\langle\mathbf{L}\rangle = \mathbf{L} \oplus.\otimes \mathbf{L}$ — multiply matrices L and L using mask L
3. $t = \sum \mathbf{C}$ — sum the values in C

M.M. Wolf et al. (Sandia National Laboratories):
Fast linear algebra-based triangle counting with KokkosKernels, HPEC 2017

SANDIA ALGORITHM



L	①	②	③	④	⑤	⑥	⑦
①							
②	1						
③							
④	1	1	1				
⑤		1					
⑥			1	1	1		
⑦		1	1	1	1		

L	①	②	③	④	⑤	⑥	⑦
①							
②	1						
③							
④	1	1	1				
⑤		1					
⑥			1	1	1		
⑦		1	1	1	1		

C							
	▨						
	▨ 1	▨	▨				
		▨					
			▨ 1	▨	▨		
		▨ 2	▨ 1	▨	▨		

$$\mathbf{C}\langle\mathbf{L}\rangle = \mathbf{L} \oplus.\otimes\, \mathbf{L}$$

$$t = \sum \mathbf{C}$$

Graph algorithms in GraphBLAS

Triangle count / CMU algorithm

CMU ALGORITHM

- Iterates on the vertices of the graph, extracts corresponding submatrices and computes $t = t + a_{10}^{\top} \oplus.\otimes\ A_{20} \oplus.\otimes\ a_{12}$
- Tradeoffs:
  - does not require mxm, only vxm and mxv
  - slower than mxm-based algorithms

$$\begin{array}{c|c|c|c|}
\mathbf{A} & & i & \\
\hline
 & \mathbf{A}_{00} & \mathbf{a}_{01} & \mathbf{A}_{20}^{\top} \\
\hline
i & \mathbf{a}_{01}^{\top} & \mathbf{0} & \mathbf{a}_{21}^{\top} \\
\hline
 & \mathbf{A}_{20} & \mathbf{a}_{21} & \mathbf{A}_{22} \\
\hline
\end{array}$$

- The formula is derived using the matrix trace $\mathrm{tr}(\mathbf{A}) = \sum_{i=0}^{n-1} \mathbf{A}_{ii}$ and its invariant property under cyclic permutation, e.g. $\mathrm{tr}(\mathbf{ABC}) = \mathrm{tr}(\mathbf{BCA}) = \mathrm{tr}(\mathbf{CAB})$. See the paper for details.

T.M. Low et al. (Carnegie Mellon University):
First look: linear algebra-based triangle counting without matrix multiplication, HPEC 2017

CMU ALGORITHM: PSEUDOCODE

Input: adjacency matrix $\mathbf{A}$

Output: triangle count t

Workspace: matrices $\mathbf{A}_{20}$, $\mathbf{C}$, vectors $\mathbf{a}_{10}$, $\mathbf{a}_{12}^\top$

```
**for** $i = 2$ **to** $n - 1$
    $\mathbf{A}_{20} = \mathbf{A}[i + 1{:}\, n, 0{:}\, i - 1]$
    $\mathbf{a}_{10} = \mathbf{A}[0{:}\, i - 1, i]$
    $\mathbf{a}_{12} = \mathbf{A}[i, i + 1{:}\, n]$
    $t = t + a_{10}^\top \oplus.\otimes A_{20} \oplus.\otimes a_{12}$
```

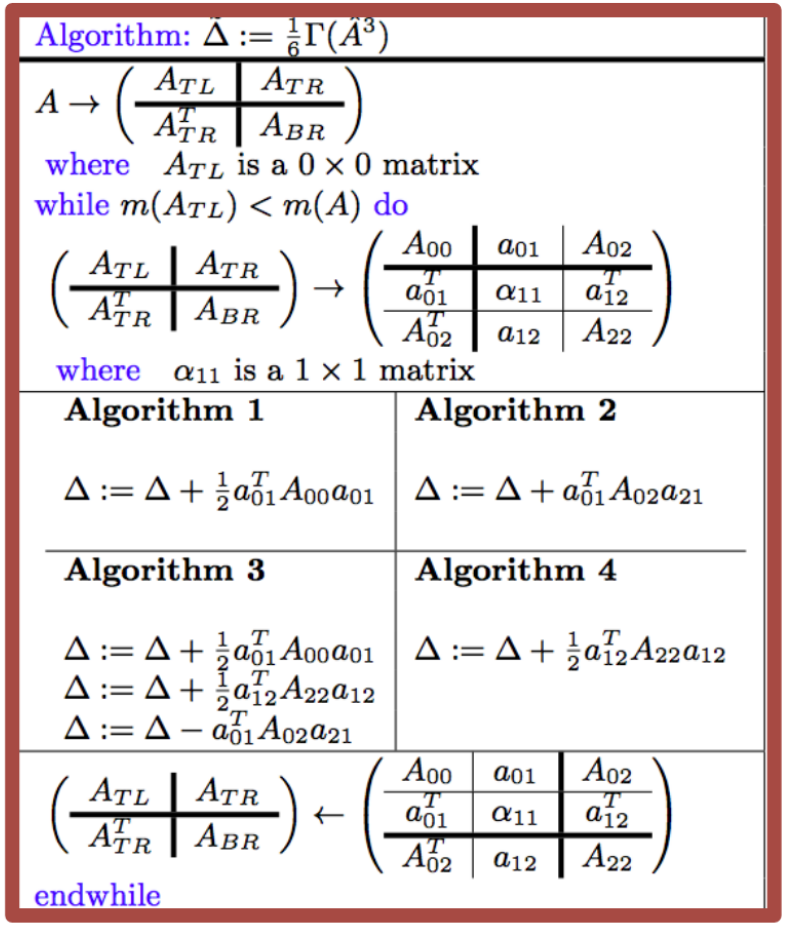
$$\begin{array}{c|c|c|c|}
\mathbf{A} & & i & \\
\hline
 & \mathbf{A}_{00} & \mathbf{a}_{01} & \mathbf{A}_{20}^\top \\
\hline
i & \mathbf{a}_{01}^\top & \mathbf{0} & \mathbf{a}_{21}^\top \\
\hline
 & \mathbf{A}_{20} & \mathbf{a}_{21} & \mathbf{A}_{22} \\
\hline
\end{array}$$

T.M. Low et al. (Carnegie Mellon University):
First look: linear algebra-based triangle counting without matrix multiplication, HPEC 2017

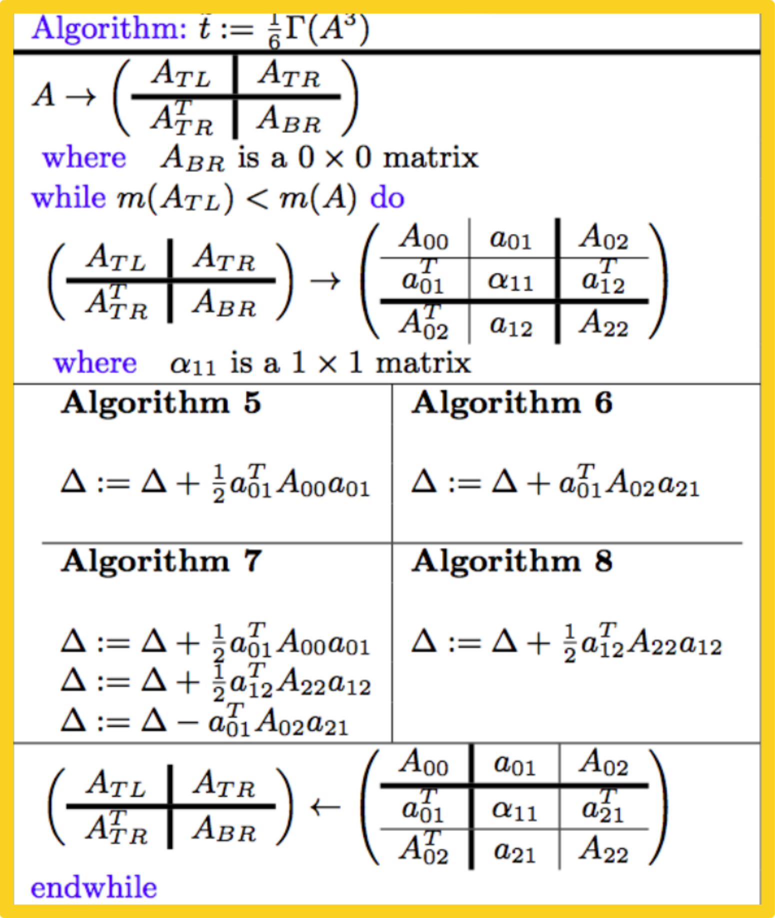
PROVABLY CORRECT ALGORITHMS

The “CMU algorithm” belongs to a family of algorithms which can be derived using the “FLAME approach”.

There are 8 similar algorithms in total, the one presented here is Algorithm 2.

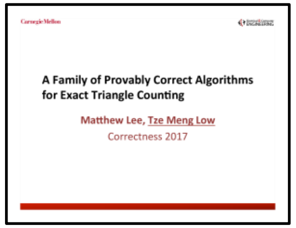


```
Algorithm: $\Delta := \frac{1}{6}\Gamma(A^3)$
$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{TR}^T & A_{BR} \end{array}\right)$
  where $A_{TL}$ is a $0 \times 0$ matrix
while $m(A_{TL}) < m(A)$ do
  $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{TR}^T & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{01}^T & \alpha_{11} & a_{12}^T \\ \hline A_{02}^T & a_{12} & A_{22} \end{array}\right)$
    where $\alpha_{11}$ is a $1 \times 1$ matrix
  Algorithm 1
    $\Delta := \Delta + \frac{1}{2} a_{01}^T A_{00} a_{01}$
  Algorithm 2
    $\Delta := \Delta + a_{01}^T A_{02} a_{21}$
  Algorithm 3
    $\Delta := \Delta + \frac{1}{2} a_{01}^T A_{00} a_{01}$
    $\Delta := \Delta + \frac{1}{2} a_{12}^T A_{22} a_{12}$
    $\Delta := \Delta - a_{01}^T A_{02} a_{21}$
  Algorithm 4
    $\Delta := \Delta + \frac{1}{2} a_{12}^T A_{22} a_{12}$
  $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{TR}^T & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{01}^T & \alpha_{11} & a_{12}^T \\ \hline A_{02}^T & a_{12} & A_{22} \end{array}\right)$
endwhile
```



```
Algorithm: $t := \frac{1}{6}\Gamma(A^3)$
$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{TR}^T & A_{BR} \end{array}\right)$
  where $A_{BR}$ is a $0 \times 0$ matrix
while $m(A_{TL}) < m(A)$ do
  $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{TR}^T & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{01}^T & \alpha_{11} & a_{12}^T \\ \hline A_{02}^T & a_{12} & A_{22} \end{array}\right)$
    where $\alpha_{11}$ is a $1 \times 1$ matrix
  Algorithm 5
    $\Delta := \Delta + \frac{1}{2} a_{01}^T A_{00} a_{01}$
  Algorithm 6
    $\Delta := \Delta + a_{01}^T A_{02} a_{21}$
  Algorithm 7
    $\Delta := \Delta + \frac{1}{2} a_{01}^T A_{00} a_{01}$
    $\Delta := \Delta + \frac{1}{2} a_{12}^T A_{22} a_{12}$
    $\Delta := \Delta - a_{01}^T A_{02} a_{21}$
  Algorithm 8
    $\Delta := \Delta + \frac{1}{2} a_{12}^T A_{22} a_{12}$
  $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{TR}^T & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{01}^T & \alpha_{11} & a_{21}^T \\ \hline A_{02}^T & a_{21} & A_{22} \end{array}\right)$
endwhile
```

M. Lee, T.M. Low (Carnegie Mellon University):
A family of provably correct algorithms for exact triangle counting,
CORRECTNESS @ SC 2017



Carnegie Mellon

A Family of Provably Correct Algorithms for Exact Triangle Counting

Matthew Lee, Tze Meng Low

Correctness 2017

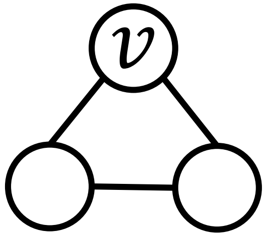
Source of the figure

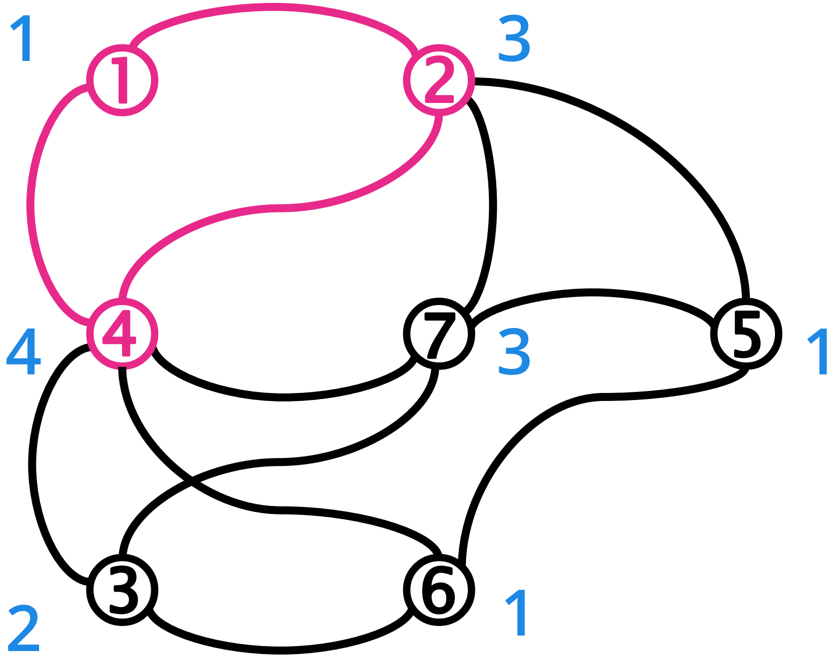
Graph algorithms in GraphBLAS

Vertex-wise triangle count

VERTEX-WISE TRIANGLE COUNT

Triangle: a set of three mutually adjacent vertices.



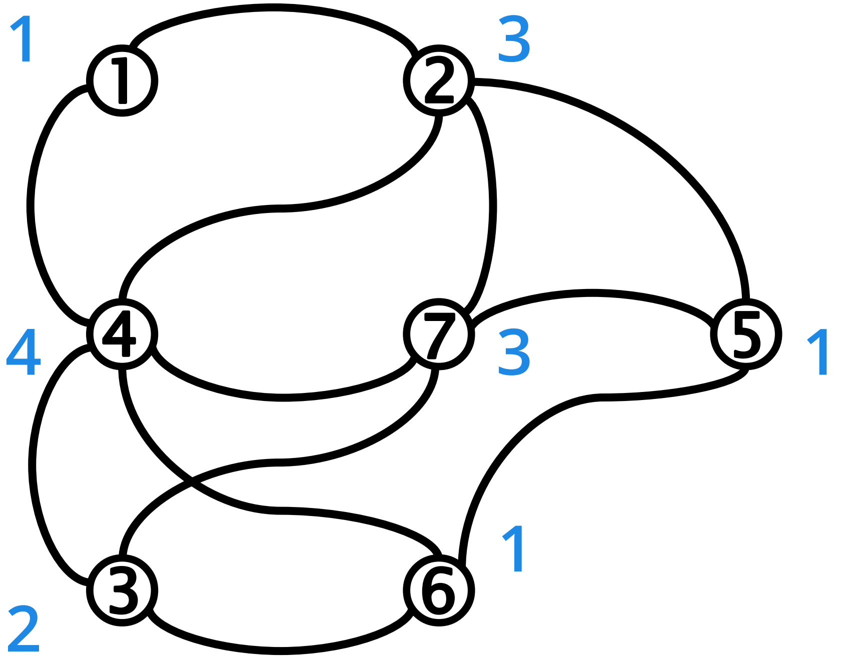


Usages:

- Global clustering coefficient
- Local clustering coefficient
- Finding communities

GraphChallenge.org: Raising the Bar on Graph Analytic Performance, HPEC 2018

TC: ELEMENT-WISE MULTIPLICATION



$$\mathbf{TRI} = \mathbf{A} \oplus.\otimes \mathbf{A} \otimes \mathbf{A}$$

$$\mathbf{tri} = \left[\oplus_j \mathbf{TRI}(:,j)\right]/2$$

$\mathbf{A} \oplus.\otimes \mathbf{A}$ is still very dense.

A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

2	1	1	1	1	1	2
1	4	2	2	1	2	2
1	2	3	2	2	1	1
1	2	2	5	3	1	2
1	1	2	3	3		1
1	2	1	1		3	3
2	2	1	2	1	3	4

$\xrightarrow{\otimes\ \mathbf{A}}$

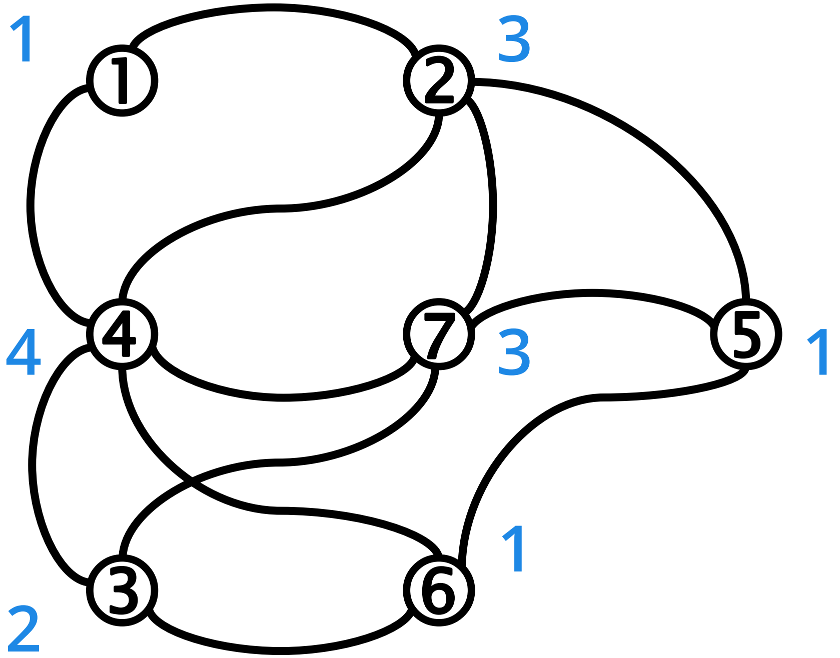
TRI

	1		1			
1			2	1		2
			2		1	1
1	2	2			1	2
	1					1
		1	1			
	2	1	2	1		

$\xrightarrow[/2]{[\oplus_j \cdots]}$

tri
1
3
2
4
1
1
3

TC: ELEMENT-WISE MULTIPLICATION



$$\begin{array}{c|ccccccc}
\mathbf{A} & ① & ② & ③ & ④ & ⑤ & ⑥ & ⑦ \\
\hline
① &   & 1 &   & 1 &   &   &   \\
② & 1 &   &   & 1 & 1 &   & 1 \\
③ &   &   &   & 1 &   & 1 & 1 \\
④ & 1 & 1 & 1 &   &   & 1 & 1 \\
⑤ &   & 1 &   &   &   & 1 & 1 \\
⑥ &   &   & 1 & 1 & 1 &   &   \\
⑦ &   & 1 & 1 & 1 & 1 &   &
\end{array}$$

Masking limits where the operation is computed. Here, we use $\mathbf{A}$ as a mask for $\mathbf{A} \oplus.\otimes \mathbf{A}$.

$$\mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} \oplus.\otimes \mathbf{A}$$

$$\mathbf{tri} = \left[\bigoplus_j \mathbf{TRI}(:,j)\right]/2$$

$$\begin{array}{c|ccccccc}
\mathbf{A} & ① & ② & ③ & ④ & ⑤ & ⑥ & ⑦ \\
\hline
① &   & 1 &   & 1 &   &   &   \\
② & 1 &   &   & 1 & 1 &   & 1 \\
③ &   &   &   & 1 &   & 1 & 1 \\
④ & 1 & 1 & 1 &   &   & 1 & 1 \\
⑤ &   & 1 &   &   &   & 1 & 1 \\
⑥ &   &   & 1 & 1 & 1 &   &   \\
⑦ &   & 1 & 1 & 1 & 1 &   &
\end{array}
\quad
\begin{array}{|ccccccc|}
\hline
  & 1 &   & 1 &   &   &   \\
1 &   &   & 2 & 1 &   & 2 \\
  &   &   & 2 &   & 1 & 1 \\
1 & 2 & 2 &   &   & 1 & 2 \\
  & 1 &   &   &   &   & 1 \\
  &   & 1 & 1 &   &   &   \\
  & 2 & 1 & 2 & 1 &   &   \\
\hline
\end{array}
\xrightarrow[/2]{\left[\bigoplus_j \cdots\right]}
\begin{array}{|c|}
\multicolumn{1}{c}{\mathbf{tri}} \\
\hline
1 \\ 3 \\ 2 \\ 4 \\ 1 \\ 1 \\ 3 \\
\hline
\end{array}$$

TC: ALGORITHM

Input: adjacency matrix $\mathbf{A}$

Output: vector $\mathbf{tri}$

Workspace: matrix $\mathbf{TRI}$

1. $\mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} \oplus.\otimes \mathbf{A}$ compute the triangle count matrix
2. $\mathbf{tri} = \left[\bigoplus_j \mathbf{TRI}(:, j)\right]/2$ compute the triangle count vector

Optimization: use $\mathbf{L}$, the lower triangular part of $\mathbf{A}$ to avoid duplicates.

$$\mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} \oplus.\otimes \mathbf{L}$$

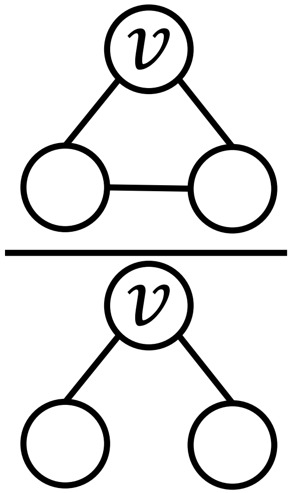
Worst-case optimal joins: There are deep theoretical connections between masked matrix multiplication and relational joins. It has been proven in 2013 that for the triangle query, binary joins always provide suboptimal runtime, which gave rise to new research on the family of worst-case optimal multi-way joins algorithms.

Graph algorithms in GraphBLAS

Local clustering coefficient

LCC: LOCAL CLUSTERING COEFFICIENT

$$\mathrm{LCC}(v) = \frac{\text{#edges between neighbours of } v}{\text{#possible edges between neighbours of } v} =$$



If $|N(v)| \leq 1$, $\mathrm{LCC}(v) = 0$

Important metric in social network analysis.

The numerator is the number of *triangles* in v, $\mathrm{tri}(v)$.

The denominator is the number of *wedges* in v, $\mathrm{wed}(v)$.

$$\mathrm{LCC}(v) = \frac{\mathrm{tri}(v)}{\mathrm{wed}(v)}$$

The difficult part is $\mathrm{tri}(v)$.

LCC: NUMBER OF WEDGES IN EACH VERTEX

$$\mathrm{LCC}(v) = \frac{\mathrm{tri}(v)}{\mathrm{wed}(v)}$$

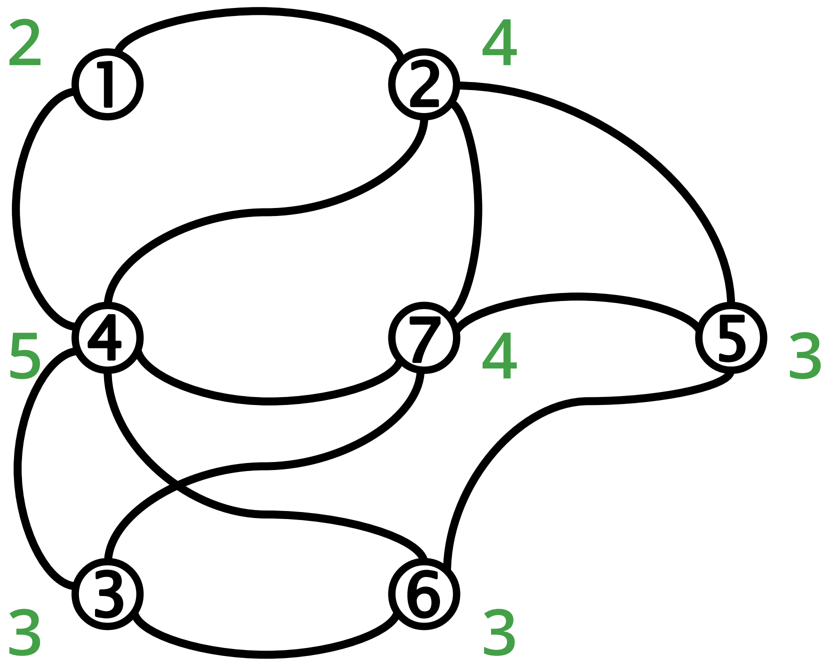
- For $\mathrm{wed}(v)$, we determine the #wedges for each vertex as the 2-combination of its degree:

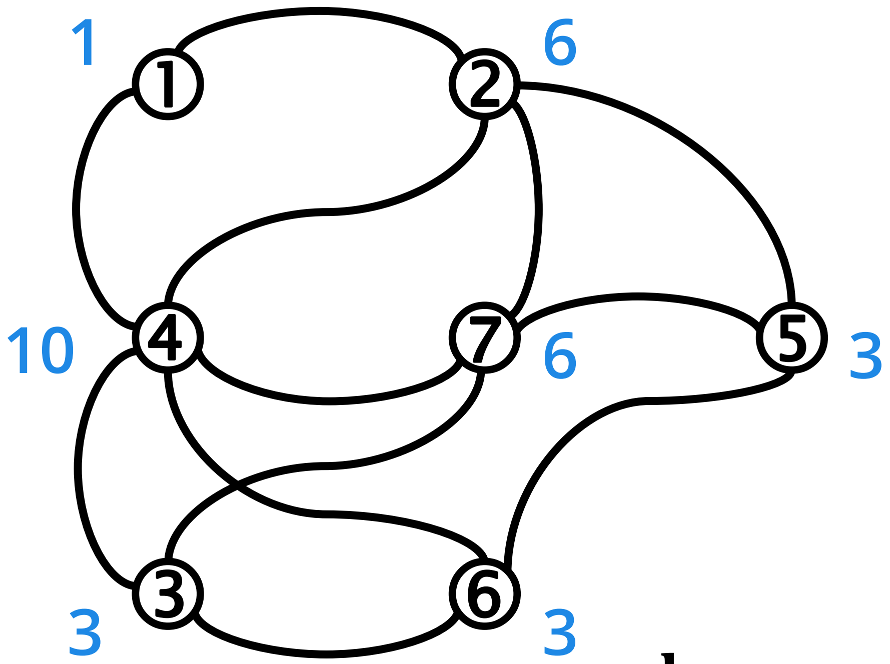
$$\mathrm{comb2}(x) = \frac{x \cdot (x-1)}{2}$$

- Given the degrees $\mathbf{deg} = \left[\bigoplus_j \mathbf{A}(:,j)\right]$, we compute $\mathbf{wed}$ by applying a unary function on the elements of the vector:

$$\mathbf{wed} = \mathrm{comb2}(\mathbf{deg})$$

LCC EXAMPLE: NUMBER OF WEDGES

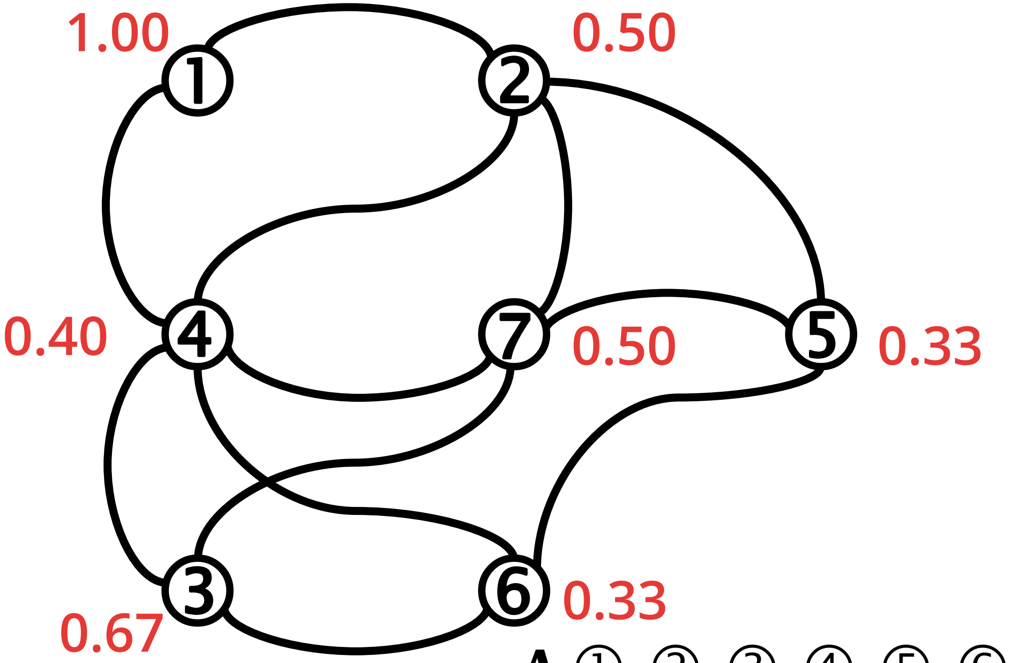




A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

$$\xrightarrow{\left[\bigoplus_j \mathbf{A}(:,j)\right]} \ \overset{\mathbf{deg}}{\begin{bmatrix} 2 \\ 4 \\ 3 \\ 5 \\ 3 \\ 3 \\ 4 \end{bmatrix}} \ \xrightarrow{\text{comb2}(\cdots)} \ \overset{\mathbf{wed}}{\begin{bmatrix} 1 \\ 6 \\ 3 \\ 10 \\ 3 \\ 3 \\ 6 \end{bmatrix}}$$

LCC EXAMPLE: COMPLETE ALGORITHM



$$\mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} \oplus.\otimes \mathbf{A}$$

A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

$$\xrightarrow{[\oplus_j \cdots]}$$

deg
2
4
3
5
3
3
4

$$\xrightarrow{\text{comb2}(\cdots)}$$

A	①	②	③	④	⑤	⑥	⑦
①		1		1			
②	1			1	1		1
③				1		1	1
④	1	1	1			1	1
⑤		1				1	1
⑥			1	1	1		
⑦		1	1	1	1		

	1		1			
1			2	1		2
			2		1	1
1	2	2			1	2
	1					1
		1	1			
	2	1	2	1		

$$\xrightarrow[/2]{[\oplus_j \cdots]}$$

tri		**wed**		**lcc**
1		1		1.00
3		6		0.50
2		3		0.67
4	⊘	10	=	0.40
1		3		0.33
1		3		0.33
3		6		0.50

LCC: ALGORITHM

Input: adjacency matrix $\mathbf{A}$

Output: vector $\mathbf{lcc}$

Workspace: matrix $\mathbf{TRI}$, vectors $\mathbf{tri}$, $\mathbf{deg}$, $\mathbf{wed}$, and $\mathbf{lcc}$

1. $\mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} \oplus.\otimes \mathbf{A}$ compute triangle count matrix
2. $\mathbf{tri} = \left[\bigoplus_j \mathbf{TRI}(:,j)\right]/2$ reduce to triangle count vector
3. $\mathbf{deg} = \left[\bigoplus_j \mathbf{A}(:,j)\right]$ reduce to vertex degree vector
4. $\mathbf{wed} = \mathrm{comb2}(\mathbf{deg})$ apply comb2 to get wedge count vector
5. $\mathbf{lcc} = \mathbf{tri} \oslash \mathbf{wed}$ LCC vector

M. Aznaveh, J. Chen, T.A. Davis, B. Hegyi, S.P. Kolodziej, T.G. Mattson, G. Szárnyas:
Parallel GraphBLAS with OpenMP, Workshop on Combinatorial Scientific Computing 2020

LCC: FURTHER OPTIMIZATIONS

Further optimization: use $\mathbf{L}$, the lower triangular part of $\mathbf{A}$.

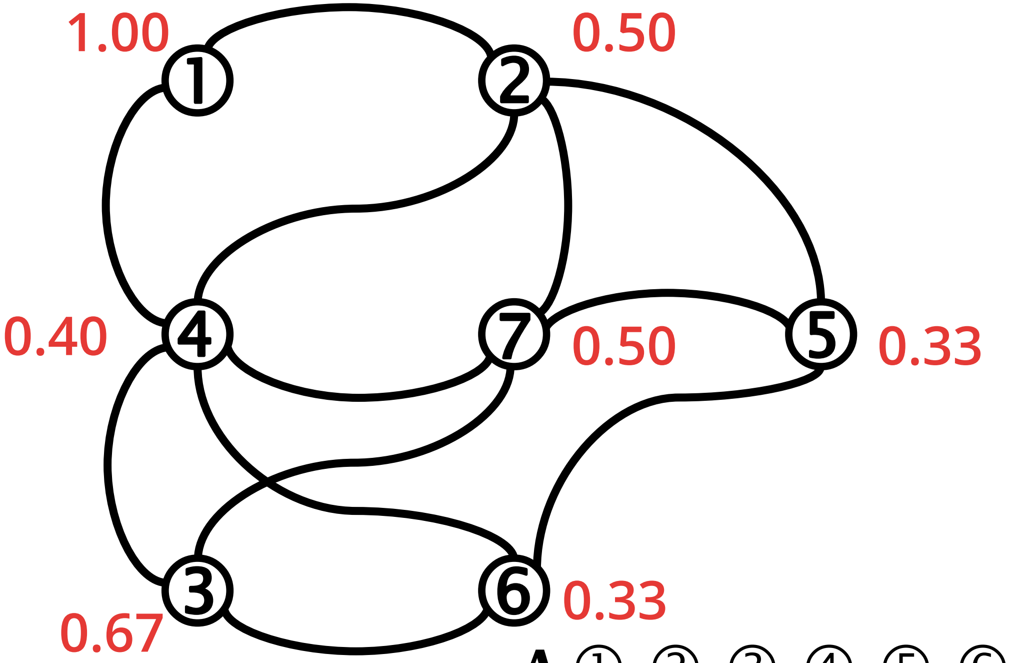
$$\mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} \oplus.\otimes \mathbf{L}$$

The number of wedges is now the 2-combination of $\mathbf{deg}$.

$$\mathrm{comb2}(x) = \frac{x \cdot (x-1)}{2}$$

Permuting the adjacency matrix allows further optimizations.

LCC EXAMPLE: LOWER TRIANGULAR PART OF MX.



$$\mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} \oplus.\otimes\, \mathbf{L}$$

$$\begin{array}{c|ccccccc|}
\mathbf{L} & ① & ② & ③ & ④ & ⑤ & ⑥ & ⑦ \\ \hline
① & & & & & & & \\
② & 1 & & & & & & \\
③ & & & & & & & \\
④ & 1 & 1 & 1 & & & & \\
⑤ & & 1 & & & & & \\
⑥ & & & 1 & 1 & 1 & & \\
⑦ & & 1 & 1 & 1 & 1 & & \\ \hline
\end{array}
\xrightarrow{[\oplus_j \cdots]}
\mathbf{deg}\ \begin{array}{|c|} \hline 2 \\ 4 \\ 3 \\ 5 \\ 3 \\ 3 \\ 4 \\ \hline \end{array}
\xrightarrow{\text{comb2}(\cdots)}
\mathbf{wed}$$

$$\begin{array}{c|ccccccc|}
\mathbf{A} & ① & ② & ③ & ④ & ⑤ & ⑥ & ⑦ \\ \hline
① & & 1 & & 1 & & & \\
② & 1 & & & 1 & 1 & & 1 \\
③ & & & & 1 & & 1 & 1 \\
④ & 1 & 1 & 1 & & & 1 & 1 \\
⑤ & & 1 & & & & 1 & 1 \\
⑥ & & & 1 & 1 & 1 & & \\
⑦ & & 1 & 1 & 1 & 1 & & \\ \hline
\end{array}
\quad
\begin{array}{|ccccccc|} \hline
 & 1 & & & & & \\
1 & & & 1 & 1 & & \\
 & & & 2 & & & \\
1 & 1 & 2 & & & & \\
 & 1 & & & & & \\
 & & 1 & & & & \\
 & 2 & 1 & & & & \\ \hline
\end{array}
\xrightarrow{[\oplus_j \cdots]}$$

$$\mathbf{tri}\ \begin{array}{|c|} \hline 1 \\ 3 \\ 2 \\ 4 \\ 1 \\ 1 \\ 3 \\ \hline \end{array}
\ \oslash\
\mathbf{wed}\ \begin{array}{|c|} \hline 1 \\ 6 \\ 3 \\ 10 \\ 3 \\ 3 \\ 6 \\ \hline \end{array}
\ =\
\mathbf{lcc}\ \begin{array}{|c|} \hline 1.00 \\ 0.50 \\ 0.67 \\ 0.40 \\ 0.33 \\ 0.33 \\ 0.50 \\ \hline \end{array}$$

Graph algorithms in GraphBLAS

PageRank

PAGERANK – DEFINITION (LDBC GRAPHALYTICS)

For $k = 1$ to t iterations:

$$\mathrm{PR}_0(v) = \frac{1}{n}$$

$$\mathrm{PR}_k(v) = \underbrace{\frac{1-\alpha}{n}}_{\text{teleport}} + \underbrace{\alpha \cdot \sum_{u \in N_{\mathrm{in}}(v)} \frac{\mathrm{PR}_{k-1}(u)}{|N_{\mathrm{out}}(u)|}}_{\text{influence}} + \underbrace{\frac{\alpha}{n} \cdot \sum_{w \in dng} \mathrm{PR}_{k-1}(w)}_{\text{dangling}}$$

α: damping factor

dng: dangling vertices, $dng = \{w \in V : |N_{\mathrm{out}}(w)| = 0\}$

There are dozens of PR definitions, some treat dangling vertices differently.

PAGERANK – IN LINEAR ALGEBRA

Initially:

$$\mathbf{pr}_0 = [1\;1\;\ldots\;1] \oslash n, \qquad \mathbf{outd} = \left[\oplus_j\, \mathbf{A}(:,j)\right]$$

In each iteration:

$$\mathrm{PR}_k(v) = \frac{1-\alpha}{n} + \alpha \cdot \sum_{u \in N_{\mathrm{in}}(v)} \frac{\mathrm{PR}_{k-1}(u)}{|N_{\mathrm{out}}(u)|} + \frac{\alpha}{n} \cdot \sum_{w \in dng} \mathrm{PR}_{k-1}(w)$$

$$\mathbf{pr}_k = \underbrace{\frac{1-\alpha}{n}}_{\text{constant}} \oplus \underbrace{\alpha \otimes \left(\frac{\mathbf{pr}_{k-1}}{\mathbf{outd}}\right) \oplus.\!\otimes \mathbf{A}}_{\text{SpMV}} \oplus \underbrace{\frac{\alpha}{n} \otimes \left[\oplus_i \left(\mathbf{pr}_k \otimes \overline{\mathbf{outd}}\right)(i)\right]}_{\text{element-wise sparse vector-dense vector multiplication}}$$

PAGERANK – ALGORITHM

Input: adjacency matrix $\mathbf{A}$, damping factor α, #iterations t, #vertices n

Output: PageRank vector $\mathbf{pr}$ (real); **Workspace:** vectors (real)

```
1.  $\mathbf{pr} = [1\ 1 \ldots 1] \oslash n$
2.  $\mathbf{outdegrees} = \left[\bigoplus_j \mathbf{A}(:, j)\right]$
3.  for $k = 1$ to $t$
4.      $\mathbf{importance} = \alpha \otimes (\mathbf{pr} \oslash \mathbf{outdegrees}) \oplus.\otimes\ \mathbf{A}$
5.      $\mathbf{danglingVertexRanks}\langle \neg\mathbf{outdegrees} \rangle = \mathbf{pr}(:)$
6.      $totalDanglingRank = \frac{\alpha}{n} \otimes [\bigoplus_i \mathbf{danglingVertexRanks}(i)]$
7.      $\mathbf{pr} = \frac{1-\alpha}{n} \oplus totalDanglingRank \oplus \mathbf{importance}$
```

Graph algorithms in GraphBLAS

k-truss

K-TRUSS

- **Definition:** the k-truss is a subset of the graph with the same number of vertices, where each edge appears in at least $k - 2$ triangles in the original graph.

K-TRUSS ALGORITHM

- **Input:** adjacency matrix $\mathbf{A}$, scalar k
- **Output:** k-truss adjacency matrix $\mathbf{C}$
- **Helper:** $f(x, support) = x \geq support$

```
$\mathbf{C} = \mathbf{A}$
for $i = 1$ to $n - 1$
    $\mathbf{C}\langle\mathbf{C}\rangle = \mathbf{C} \oplus.\wedge \mathbf{C}$        use the “plus-and” semiring
    $\mathbf{C} = f(\mathbf{C}, k - 2)$            drop entries in $\mathbf{C}$ less than $k - 2$
    terminate if the number of non-zero values in $\mathbf{C}$ did not change
```

T.A. Davis: *Graph algorithms via SuiteSparse:GraphBLAS: triangle counting and k-truss,* HPEC 2018

Graph algorithms in GraphBLAS

Community detection using label propagation

CDLP: COMMUNITY DETECTION USING LABEL PROPAGATION

Goal: assign a label to each vertex representing the community it belongs to. The algorithm (originally published in network science) is slightly altered to ensure deterministic execution. Initially:

$$L_0(v) = v$$

In the k^{th} iteration:

$$L_k(v) = \min(\arg\max_l |\{u \in N(v) : L_{k-1}(u) = l\}|),$$

where $N(v)$ is the set of neighbours of v.

Run for t iterations or until reaching a fixed point.

U.N. Raghavan, R. Albert, S. Kumara: *Near linear time algorithm to detect community structures in large-scale networks*, Phys. Rev. E, 2007

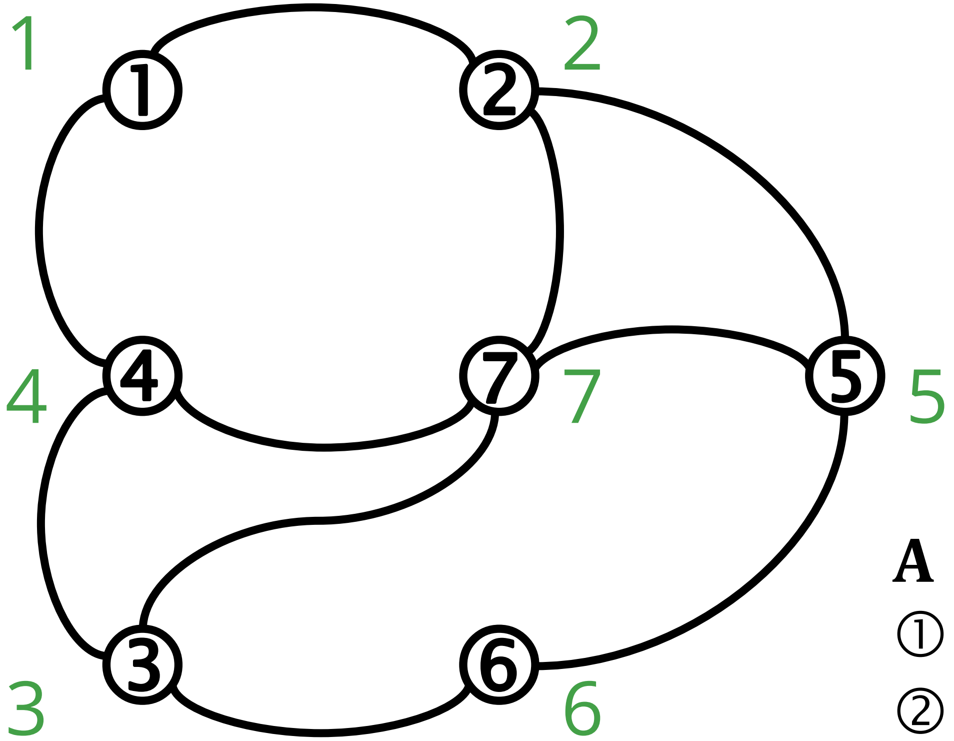
IDEA: CAPTURE CDLP IN PURE GRAPHBLAS

- Define a semiring that operates over **occurrence vectors**
- $\bigoplus$ operator: combines two occurrence vectors
  - $\{6 \rightarrow 1,\ 9 \rightarrow 1\} \bigoplus \{6 \rightarrow 1,\ 7 \rightarrow 2\} = \{6 \rightarrow 2,\ 7 \rightarrow 2,\ 9 \rightarrow 1\}$
- Convert each element in a row to an occurrence vector
  - $\{6 \rightarrow 1\}, \{6 \rightarrow 1\}, \{7 \rightarrow 1\}, \{7 \rightarrow 1\}, \{9 \rightarrow 1\}$
- Reduce each row into a single occurrence vector:
  - $\{6 \rightarrow 2, 7 \rightarrow 2, 9 \rightarrow 1\}$
- Select the min. mode element from the occurrence vector: 6
- Works on paper, but occurrence vectors need dynamic memory allocation, which leads to poor performance with the current GraphBLAS API

CDLP IN LINEAR ALGEBRA: FASTER ALGORITHM

- Extract each row from $\mathbf{F}$
  - Easy if the matrix is stored in CSR format
- Select the minimum mode value in each row
  - Sort elements using parallel merge sort
  - Pick the min value that has the longest run (done in a single pass)
- Sort each row $\mathbf{r}$
- Use the sorted list to compute $\mathrm{mode}(\mathbf{r})$
- The matrix multiplications are always performed with a diagonal matrix as the second operand so we never need the addition operator. Therefore, we set it to $\oplus \equiv \mathrm{any}$.

CDLP EXAMPLE



- Initially, $\mathbf{lab} = [1\ 2 \dots n]$
- Propagate labels to create a “frequency matrix”:

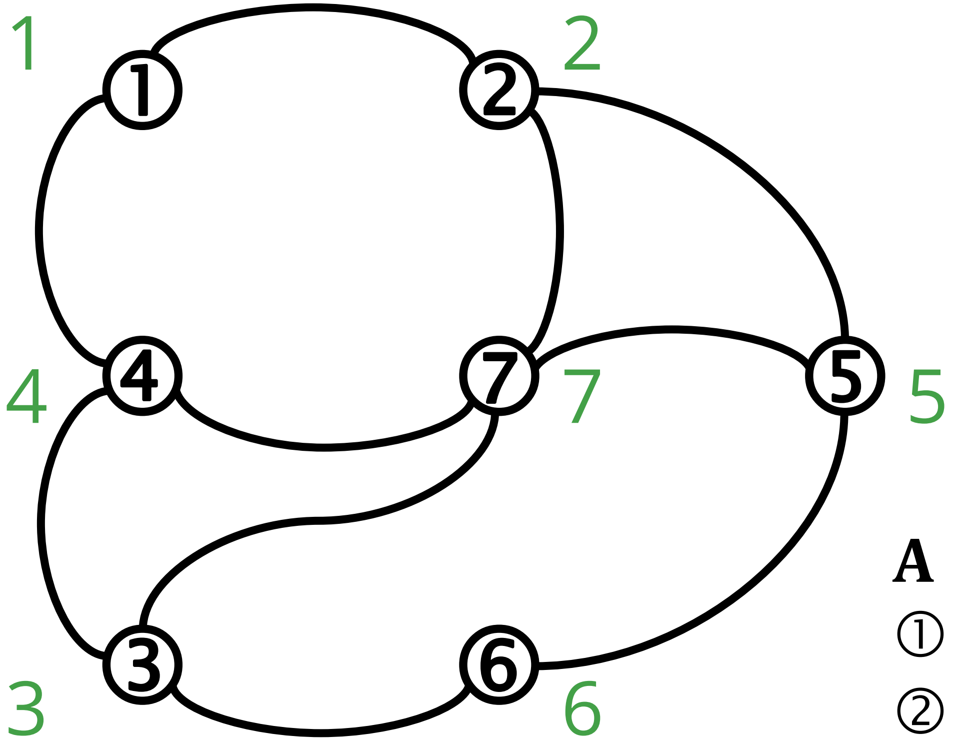
$$\mathbf{F} = \mathbf{A}\ \text{any}\,.\,\text{sel2nd}\ \text{diag}(\mathbf{lab})$$

$$\begin{array}{c|ccccccc}
\text{diag}(\mathbf{lab}) & \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} & \textcircled{7} \\
\hline
\textcircled{1} & 1 & & & & & & \\
\textcircled{2} & & 2 & & & & & \\
\textcircled{3} & & & 3 & & & & \\
\textcircled{4} & & & & 4 & & & \\
\textcircled{5} & & & & & 5 & & \\
\textcircled{6} & & & & & & 6 & \\
\textcircled{7} & & & & & & & 7
\end{array}$$

$$\begin{array}{c|ccccccc}
\mathbf{A} & \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} & \textcircled{7} \\
\hline
\textcircled{1} & & \bullet & & \bullet & & & \\
\textcircled{2} & \bullet & & & & \bullet & & \bullet \\
\textcircled{3} & & & & \bullet & & \bullet & \bullet \\
\textcircled{4} & \bullet & & \bullet & & & & \bullet \\
\textcircled{5} & & \bullet & & & & \bullet & \bullet \\
\textcircled{6} & & & \bullet & & \bullet & & \\
\textcircled{7} & & \bullet & \bullet & \bullet & \bullet & &
\end{array}$$

$$\begin{array}{ccccccc}
 & 2 & & 4 & & & \\
1 & & & & 5 & & 7 \\
 & & & 4 & & 6 & 7 \\
1 & & 3 & & & & 7 \\
 & 2 & & & & 6 & 7 \\
 & & 3 & & 5 & & \\
 & 2 & 3 & 4 & 5 & &
\end{array}$$

CDLP EXAMPLE



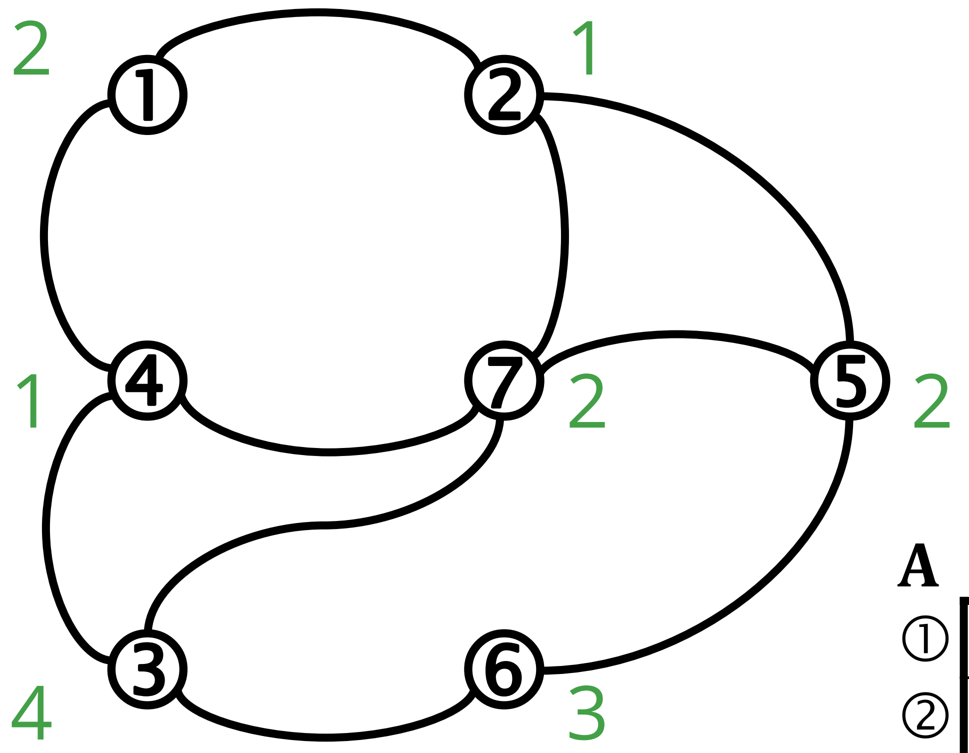
step: 1

$$
\begin{array}{c|ccccccc}
\mathrm{diag}(\mathbf{lab}) & \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} & \textcircled{7} \\
\hline
\textcircled{1} & \mathbf{1} & & & & & & \\
\textcircled{2} & & \mathbf{2} & & & & & \\
\textcircled{3} & & & \mathbf{3} & & & & \\
\textcircled{4} & & & & \mathbf{4} & & & \\
\textcircled{5} & & & & & \mathbf{5} & & \\
\textcircled{6} & & & & & & \mathbf{6} & \\
\textcircled{7} & & & & & & & \mathbf{7}
\end{array}
$$

$$
\begin{array}{c|ccccccc}
\mathbf{A} & \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} & \textcircled{7} \\
\hline
\textcircled{1} & & \bullet & & \bullet & & & \\
\textcircled{2} & \bullet & & & & \bullet & & \bullet \\
\textcircled{3} & & & & \bullet & & \bullet & \bullet \\
\textcircled{4} & \bullet & & \bullet & & & & \bullet \\
\textcircled{5} & & \bullet & & & & \bullet & \bullet \\
\textcircled{6} & & & \bullet & & \bullet & & \\
\textcircled{7} & & \bullet & \bullet & \bullet & \bullet & &
\end{array}
$$

$$
\begin{array}{|ccccccc|}
\hline
 & \mathbf{2} & & \mathbf{4} & & & \\
\mathbf{1} & & & & \mathbf{5} & & \mathbf{7} \\
 & & & \mathbf{4} & & \mathbf{6} & \mathbf{7} \\
\mathbf{1} & & \mathbf{3} & & & & \mathbf{7} \\
 & \mathbf{2} & & & & \mathbf{6} & \mathbf{7} \\
 & & \mathbf{3} & & \mathbf{5} & & \\
 & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & & \\
\hline
\end{array}
\xrightarrow{\text{min. mode}}
\begin{array}{c}
\mathbf{lab}' \\
\hline
\mathbf{2} \\
\mathbf{1} \\
\mathbf{4} \\
\mathbf{1} \\
\mathbf{2} \\
\mathbf{3} \\
\mathbf{2}
\end{array}
$$

CDLP EXAMPLE



step: 2

diag(**lab**)	①	②	③	④	⑤	⑥	⑦
①	**2**						
②		**1**					
③			**4**				
④				**1**			
⑤					**2**		
⑥						**3**	
⑦							**2**

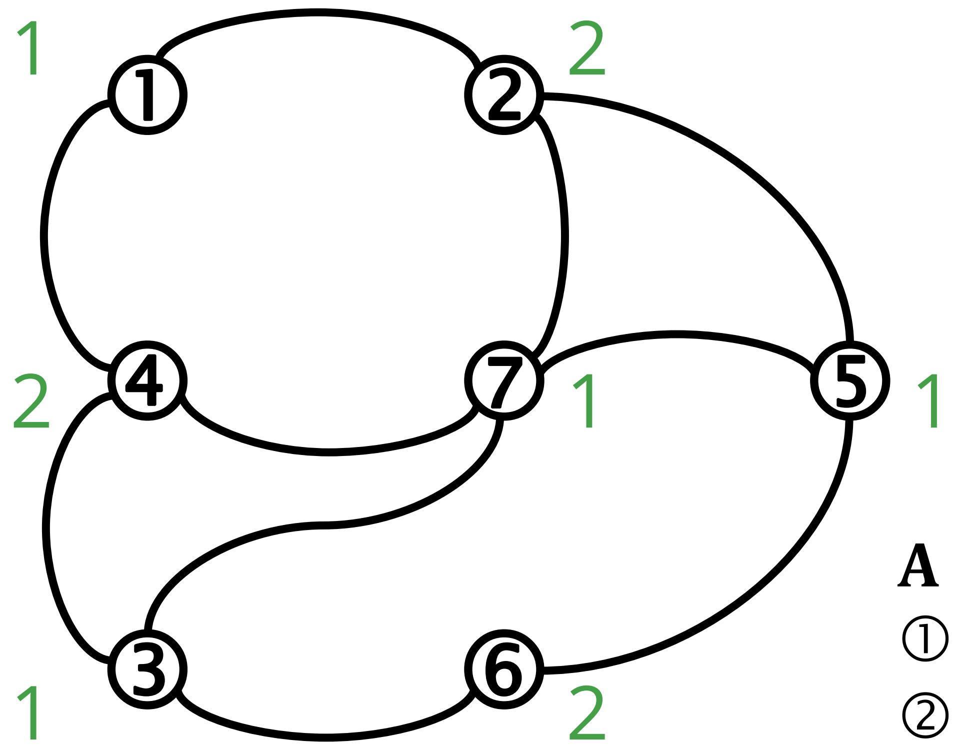
A	①	②	③	④	⑤	⑥	⑦
①		●		●			
②	●				●		●
③				●		●	●
④	●		●				●
⑤		●				●	●
⑥			●		●		
⑦		●	●	●	●		

	1		**1**			
2				**2**		**2**
			1		**3**	**2**
2		**4**				**2**
	1				**3**	**2**
		4		**2**		
	1	**4**	**1**	**2**		

min.
mode
⇨

lab′
1
2
1
2
1
2
1

CDLP EXAMPLE



step: 3

diag(lab)

	①	②	③	④	⑤	⑥	⑦
①	**1**						
②		**2**					
③			**1**				
④				**2**			
⑤					**1**		
⑥						**2**	
⑦							**1**

A

	①	②	③	④	⑤	⑥	⑦
①		●		●			
②	●				●		●
③				●		●	●
④	●		●				●
⑤		●				●	●
⑥			●		●		
⑦		●	●	●	●		

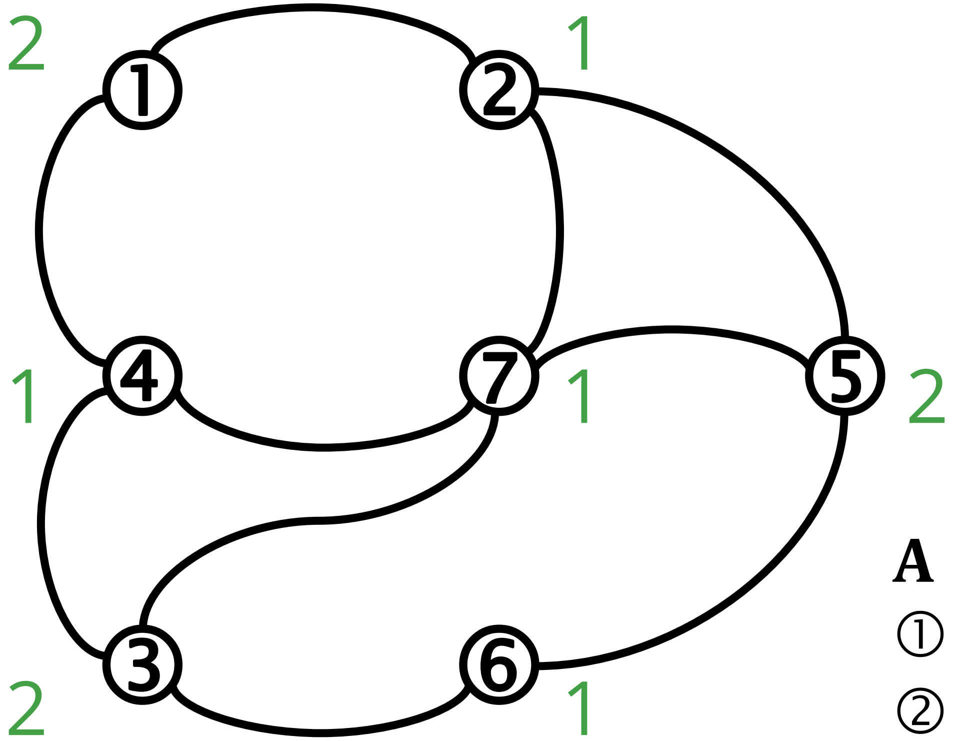
	2		**2**			
1				**1**		**1**
			2		**2**	**1**
1		**1**				**1**
	2				**2**	**1**
		1		**1**		
	2	**1**	**2**	**1**		

min.
mode ⇨

lab′

2
1
2
1
2
1
1

CDLP EXAMPLE



step: 4 – same result as in step 2
The original non-deterministic variant of the algorithm is better at avoiding such oscillations.

diag(lab)

	①	②	③	④	⑤	⑥	⑦
①	2						
②		1					
③			2				
④				1			
⑤					2		
⑥						1	
⑦							1

A

	①	②	③	④	⑤	⑥	⑦
①		●		●			
②	●				●		●
③				●		●	●
④	●		●				●
⑤		●				●	●
⑥			●		●		
⑦		●	●	●	●		

	1		1			
2				2		1
			1		1	1
2		2				1
	1				1	1
		2		2		
	1	2	1	2		

min. mode ⇨

lab′

1
2
1
2
1
2
1

CDLP: ALGORITHM

Input: adjacency matrix $\mathbf{A}$, #vertices n, #iterations t

Output: vector $\mathbf{lab}$

Workspace: matrix $\mathbf{F}$, vector $\mathbf{r}$

```
$\mathbf{lab} = [1\ 2\ \ldots\ n]$
for $k = 1$ to $t$
    $\mathbf{F} = \mathbf{A}$ any.sel2nd diag($\mathbf{lab}$)
    for $i = 1$ to $n$
        $\mathbf{r} = \mathbf{F}(i, :)$
        sort($\mathbf{r}$)                          Can be batched and parallelized
        $\mathbf{lab}(i) =$ select_min_mode($\mathbf{r}$)
```

CDLP: ALGORITHM

Input: adjacency matrix $\mathbf{A}$, #vertices n, #iterations t

Output: vector $\mathbf{lab}$

Workspace: matrix $\mathbf{F}$, vector $\mathbf{r}$, array of row indices $\mathbf{I}$, array of values $\mathbf{X}$

```
$\mathbf{lab} = [1\ 2\ \ldots\ n]$
for $k = 1$ to $t$
    $\mathbf{F} = \mathbf{A}$ any.sel2nd diag($\mathbf{lab}$)
    $\langle \mathbf{I}, \_, \mathbf{X} \rangle$ = extract_tuples($\mathbf{F}$)
    merge_sort_pairs($\langle \mathbf{I}, \mathbf{X} \rangle$)
    $\mathbf{lab}$ = for each row in $\mathbf{I}$, select min mode value from $\mathbf{X}$
```

CDLP ON DIRECTED GRAPHS

For directed graphs, we compute the labels $L_k(v)$ as:

$$\min(\arg\max_l [|\{u \in N_{\mathrm{in}}(v) : L_{k-1}(u) = l\}| + |\{u \in N_{\mathrm{out}}(v) : L_{k-1}(u) = l\}|])$$

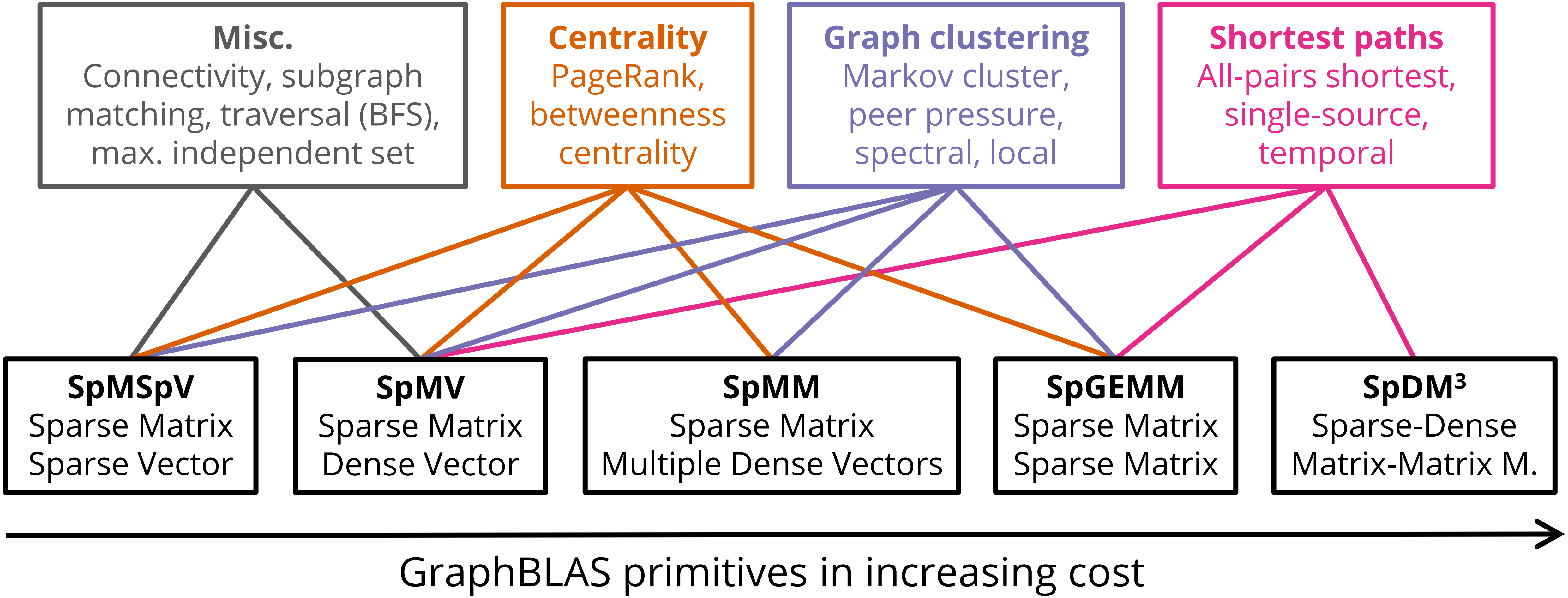
- In linear algebra, this can be expressed with two matrices:
  - $\mathbf{F}_{\mathrm{in}} = \mathbf{A}\ \mathrm{any.sel2nd}\ \mathrm{diag}(\mathbf{lab})$
  - $\mathbf{F}_{\mathrm{out}} = \mathbf{A}^{\top}\ \mathrm{any.sel2nd}\ \mathrm{diag}(\mathbf{lab})$
- Simultaneously iterate over rows $\mathbf{r}_{\mathrm{in}}$ of $\mathbf{F}_{\mathrm{in}}$ and $\mathbf{r}_{\mathrm{out}}$ of $\mathbf{F}_{\mathrm{out}}$
- For each row pair, sort $\mathbf{r}_{\mathrm{in}} \cup \mathbf{r}_{\mathrm{out}}$ and select the minimum mode value
- Batching also works:

$$\left.\begin{array}{l} \circ\ \langle \mathbf{I}_{\mathrm{in}}, \_, \mathbf{X}_{\mathrm{in}} \rangle = \mathrm{extract\_tuples}(\mathbf{F}_{\mathrm{in}}) \\ \circ\ \langle \mathbf{I}_{\mathrm{out}}, \_, \mathbf{X}_{\mathrm{out}} \rangle = \mathrm{extract\_tuples}(\mathbf{F}_{\mathrm{out}}) \end{array}\right\} \mathrm{merge\_sort\_pairs}(\langle \mathbf{I}_{\mathrm{in}} \cup \mathbf{I}_{\mathrm{out}}, \mathbf{X}_{\mathrm{in}} \cup \mathbf{X}_{\mathrm{out}} \rangle)$$

Graph algorithms in GraphBLAS

Graph algorithms & GraphBLAS primitives

GRAPH ALGORITHMS & GRAPHBLAS PRIMITIVES



Misc.
Connectivity, subgraph matching, traversal (BFS), max. independent set

Centrality
PageRank, betweenness centrality

Graph clustering
Markov cluster, peer pressure, spectral, local

Shortest paths
All-pairs shortest, single-source, temporal

SpMSpV
Sparse Matrix Sparse Vector

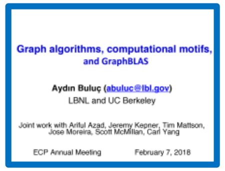
SpMV
Sparse Matrix Dense Vector

SpMM
Sparse Matrix Multiple Dense Vectors

SpGEMM
Sparse Matrix Sparse Matrix

SpDM^3
Sparse-Dense Matrix-Matrix M.

GraphBLAS primitives in increasing cost



Graph algorithms, computational motifs, and GraphBLAS

Aydın Buluç (abuluc@lbl.gov)
LBNL and UC Berkeley

Joint work with Ariful Azad, Jeremy Kepner, Tim Mattson, Jose Moreira, Scott McMillan, Carl Yang

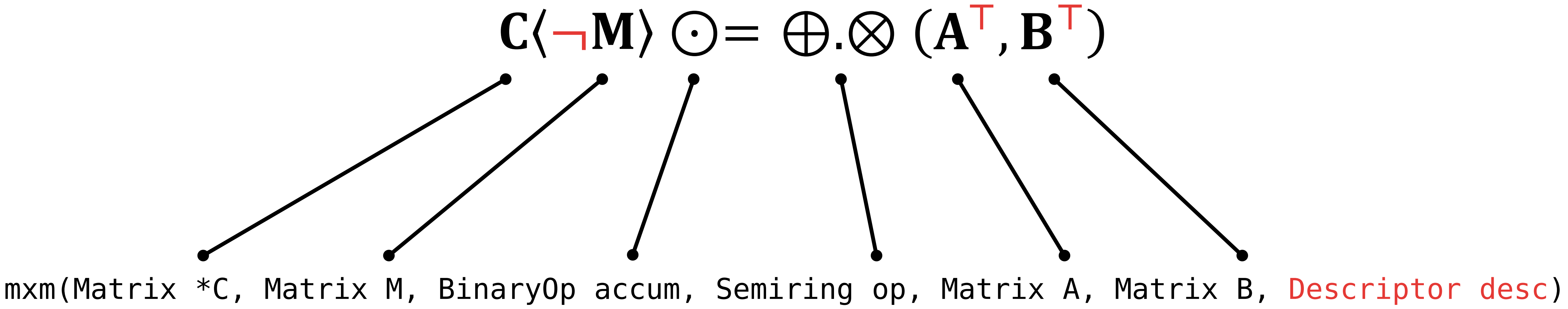
ECP Annual Meeting February 7, 2018

Based on the figure in A. Buluç:
Graph algorithms, computational motifs, and GraphBLAS, ECP Meeting 2018

GraphBLAS and SuiteSparse internals

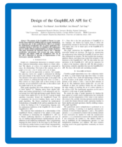
GRAPHBLAS C API

- “A crucial piece of the GraphBLAS effort is to translate the mathematical specification to an API that
  - is faithful to the mathematics as much as possible, and
  - enables efficient implementations on modern hardware.”



$$\mathbf{C}\langle\neg\mathbf{M}\rangle \odot= \oplus.\otimes\,(\mathbf{A}^{\top},\mathbf{B}^{\top})$$

```
mxm(Matrix *C, Matrix M, BinaryOp accum, Semiring op, Matrix A, Matrix B, Descriptor desc)
```



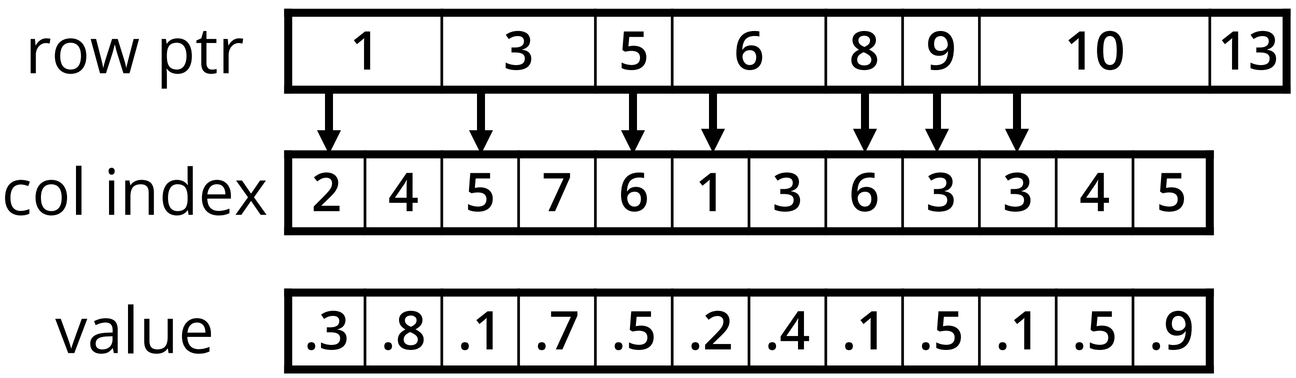
A. Buluç et al.: *Design of the GraphBLAS C API,* GABB@IPDPS 2017

GRAPHBLAS OBJECTS

- GraphBLAS objects are opaque: the matrix representation can be adjusted to suit the data distribution, hardware, etc.
- The typical representations compressed formats are:
  - CSR: Compressed Sparse Row (also known as CRS)
  - CSC: Compressed Sparse Column (also known as CCS)

A	①	②	③	④	⑤	⑥	⑦
①		.3		.8			
②					.1		.7
③						.5	
④	.2		.4				
⑤						.1	
⑥			.5				
⑦			.1	.5	.9		

CSR representation of **A**:



row ptr: 1 | 3 | 5 | 6 | 8 | 9 | 10 | 13

col index: 2 | 4 | 5 | 7 | 6 | 1 | 3 | 6 | 3 | 3 | 4 | 5

value: .3 | .8 | .1 | .7 | .5 | .2 | .4 | .1 | .5 | .1 | .5 | .9

SUITESPARSE:GRAPHBLAS INTERNALS

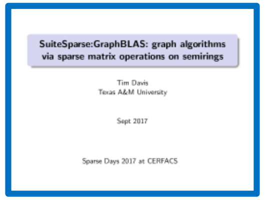
- Authored by Prof. Tim Davis at Texas A&M University, based on his SuiteSparse library (used in MATLAB).
- Design decisions, algorithms and data structures are discussed in the TOMS paper and in the User Guide.
- Extensions: methods and types prefixed with `GxB`.
- Sophisticated load balancer for multi-threaded execution.
- A GPU implementation is work-in-progress.

Algorithm 9xx: SuiteSparse:GraphBLAS: graph algorithms in the language of sparse linear algebra

TIMOTHY A. DAVIS, Texas A&M University, USA

1 INTRODUCTION

T.A. Davis: *Algorithm 1000: SuiteSparse:GraphBLAS: graph algorithms in the language of sparse linear algebra*, ACM TOMS, 2019



SuiteSparse:GraphBLAS: graph algorithms via sparse matrix operations on semirings

Tim Davis
Texas A&M University

Sept 2017

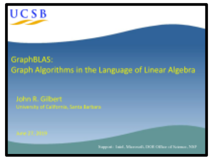
Sparse Days 2017 at CERFACS

T.A. Davis: *SuiteSparse:GraphBLAS: graph algorithms via sparse matrix operations on semirings*, Sparse Days 2017

Further reading and libraries

RESOURCES

Presentations and tutorials for learning GraphBLAS:



UCSB

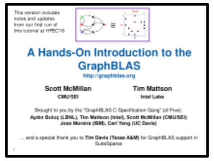
GraphBLAS:
Graph Algorithms in the Language of Linear Algebra

John R. Gilbert
University of California, Santa Barbara

June 27, 2019

Support : Intel , Microsoft, DOE Office of Science , NSF

J.R. Gilbert:
GraphBLAS: Graph Algorithms in the Language of Linear Algebra, Seminar talk since 2014



This version includes notes and updates from our first run of this tutorial at HPEC18

A Hands-On Introduction to the GraphBLAS
http://graphblas.org

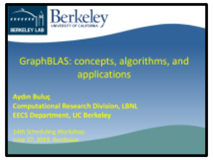
Scott McMillan
CMU/SEI

Tim Mattson
Intel Labs

Brought to you by the "GraphBLAS C Specification Gang" (of Five):
Aydin Buluç (LBNL), Tim Mattson (Intel), Scott McMillan (CMU/SEI) Jose Moreira (IBM), Carl Yang (UC Davis)

... and a special thank you to **Tim Davis (Texas A&M)** for GraphBLAS support in SuiteSparse

S. McMillan and T.G. Mattson:
A Hands-On Introduction to the GraphBLAS, Tutorial at HPEC since 2018



BERKELEY LAB

Berkeley
UNIVERSITY OF CALIFORNIA

GraphBLAS: concepts, algorithms, and applications

Aydin Buluç
Computational Research Division, LBNL
EECS Department, UC Berkeley

14th Scheduling Workshop
June 27, 2019, Bordeaux

A. Buluç:
GraphBLAS: Concepts, algorithms, and applications, Scheduling Workshop, 2019

GraphBLAS: Handling performance concerns in large graph analytics – *invited paper*

ABSTRACT

1 INTRODUCTION

CCS CONCEPTS

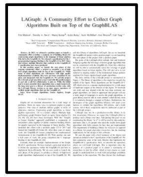
KEYWORDS

M. Kumar, J.E. Moreira, P. Pattnaik:
GraphBLAS: Handling performance concerns in large graph analytics,
Computing Frontiers 2018

List of GraphBLAS-related books, papers, presentations, posters, software, etc.
szarnyasg/graphblas-pointers

THE LAGRAPH LIBRARY

- Similar to the LAPACK library for BLAS
- Uses SuiteSparse:GraphBLAS
- Implementations of common algorithms
  - BFS, SSSP, LCC, PageRank, Boruvka
  - Triangle count, k-truss
  - CDLP (community detection using label propagation)
  - Weakly connected components, Strongly Connected Components
  - Betweenness centrality
  - Deep neural network



T.G. Mattson et al.: *LAGraph: A Community Effort to Collect Graph Algorithms Built on Top of the GraphBLAS,* GrAPL@IPDPS 2019

GraphBLAS/LAGraph

REQUIREMENTS BY GRAPH COMPUTATIONS

Libraries for linear-algebra based graph processing support the following features (prioritized):

1.	Sparse matrices	For reasonable performance
2.	Arbitrary semirings	For expressive power
3.	Masking	A big reduction in complexity for some algos
4.	Parallel execution	Constant speedup, ideally by #threads

Most libraries **only satisfy requirement #1:** Intel MKL, Eigen, Boost uBLAS, MTL4, Armadillo, NIST Sparse BLAS, GMM++, CUSP, Numpy

Exceptions are the Efficient Java Matrix Library (EJML) and Julia’s `SparseArrays` library, where arbitrary semirings can be used.

GRAPHBLAS PAPERS AND BOOKS

- *Standards for Graph Algorithm Primitives*
  - Position paper by 19 authors @ IEEE HPEC 2013

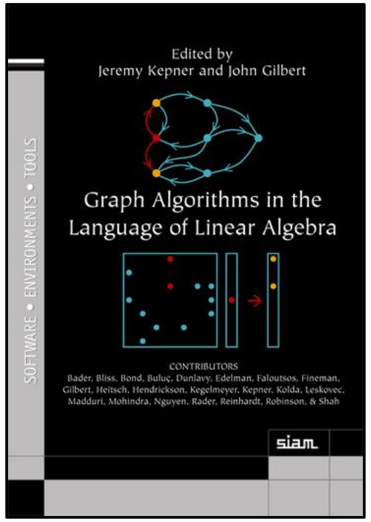
- *Novel Algebras for Advanced Analytics in Julia*
  - Technical paper on semirings in Julia @ IEEE HPEC 2013

- *Mathematical Foundations of the GraphBLAS*
  - Theory paper by 16 authors @ IEEE HPEC 2016

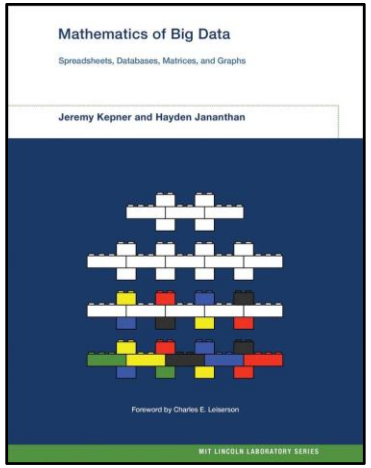
- *Design of the GraphBLAS C API*
  - Design decisions and overview of the C API @ GABB@IPDPS 2017

- *Algorithm 1000: SuiteSparse:GraphBLAS: graph algorithms in the language of sparse linear algebra*
  - Algorithms in the SuiteSparse implementation @ ACM TOMS 2019

BOOKS


Edited by
Jeremy Kepner and John Gilbert
Graph Algorithms in the Language of Linear Algebra
CONTRIBUTORS
Bader, Bliss, Bond, Buluç, Dunlavy, Edelman, Faloutsos, Fineman, Gilbert, Heitsch, Hendrickson, Kegelmeyer, Kepner, Kolda, Leskovec, Madduri, Mohindra, Nguyen, Rader, Reinhardt, Robinson, & Shah
siam
SOFTWARE • ENVIRONMENTS • TOOLS

- *Graph Algorithms in the Language of Linear Algebra*
  - Edited by J. Kepner and J.R. Gilbert, published by SIAM in 2011
  - Algorithms for connected components, shortest paths, max-flow, betwenness centrality, spanning tree, graph generation, etc.
  - Algorithms and data structure for fast matrix multiplication
  - Predates GraphBLAS: preliminary notation, no API usage


Mathematics of Big Data
Spreadsheets, Databases, Matrices, and Graphs
Jeremy Kepner and Hayden Jananthan
Foreword by Charles E. Leiserson
MIT LINCOLN LABORATORY SERIES

- *Mathematics of Big Data*
  - Authored by Jananthan & Kepner, published by MIT Press in 2018
  - Generalizes the semiring-based approach for associative arrays
  - Contains important papers, including the HPEC’16 paper above
  - Discusses D4M (Dynamic Distributed Dimensional Data Model)

GRAPHBLAS COMMUNITY

Wiki: graphblas.org | **Communication:** primarily mailing list

Annual events:

- May: IEEE IPDPS conference
  - GrAPL workshop (Graphs, Architectures, Programming and Learning), a merger of
    - GABB (Graph Algorithms Building Blocks)
    - GraML (Graph Algorithms and Machine Learning)
  - See graphanalysis.org for previous editions
- Sep: IEEE HPEC conference
  - GraphBLAS BoF meeting
- Nov: IEEE/ACM Supercomputing conference
  - GraphBLAS Working Group
  - IA^3 workshop (Workshop on Irregular Applications: Architectures and Algorithms)

Blog: AldenMath by Timothy Alden Davis

REDISGRAPH

- Graph database built on top of Redis with partial (but extending) support for the Cypher language
- Uses SuiteSparse:GraphBLAS for graph operations
- Preliminary benchmark results show good performance on traversal-heavy workloads

EVERYWHERE.

Graph Algebra

Graph operations in the language of linear algebra

redisconf18 #Redis #RedisConf redislabs

R. Lipman, T.A. Davis:
Graph Algebra – Graph operations in the language of linear algebra
RedisConf 2018

redisconf19

RedisGraph internals

Roi Lipman
RedisLabs, *Lead developer*

redislabs

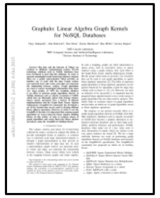
R. Lipman:
RedisGraph internals
RedisConf 2019

GRAPHBLAS IMPLEMENTATIONS

- SuiteSparse:GraphBLAS
  - v1.0.0: Nov 2017 – sequential
  - v3.0.1: July 2019 – parallel
  - v4.0.1draft: Dec 2020 – many optimizations, incl. bitmap format
- IBM GraphBLAS
  - Complete implementation in C++, released in May 2018
  - Concise but sequential
- GBTL (GraphBLAS Template Library): C++
  - v1.0: parallel but no longer maintained
  - v2.0, v3.0: sequential
- GraphBLAST: GPU implementation, based on GBTL

GRAPHULO

- Build on top of the Accumulo distributed key-value store
- Written in Java
- Focus on scalability



V. Gadepally et al.:
Graphulo: Linear Algebra Graph Kernels
for NoSQL Databases, GABB@IPDPS 2015

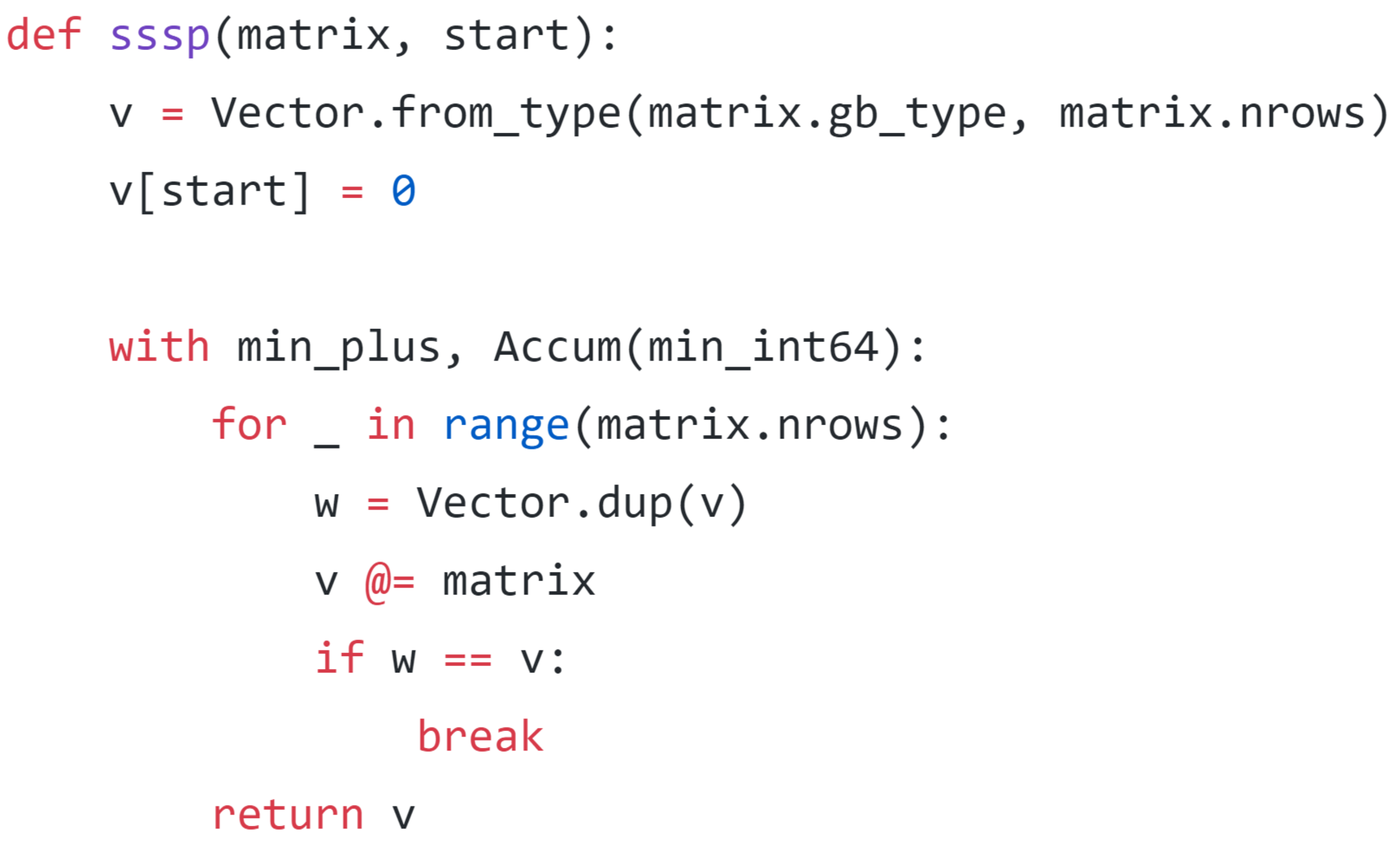
COMBBLAS: COMBINATORIAL BLAS

- “an extensible distributed memory parallel graph library offering a small but powerful set of linear algebra primitives”
- Not a GraphBLAS implementation but serves as an incubator for new ideas that may later find their way into GraphBLAS
- Scales to 250k+ CPU cores
- Used on supercomputers such as Cray

A. Buluç, J.R. Gilbert: *The Combinatorial BLAS: design, implementation, and application,* International Journal of High Performance Computing Applications, 2011

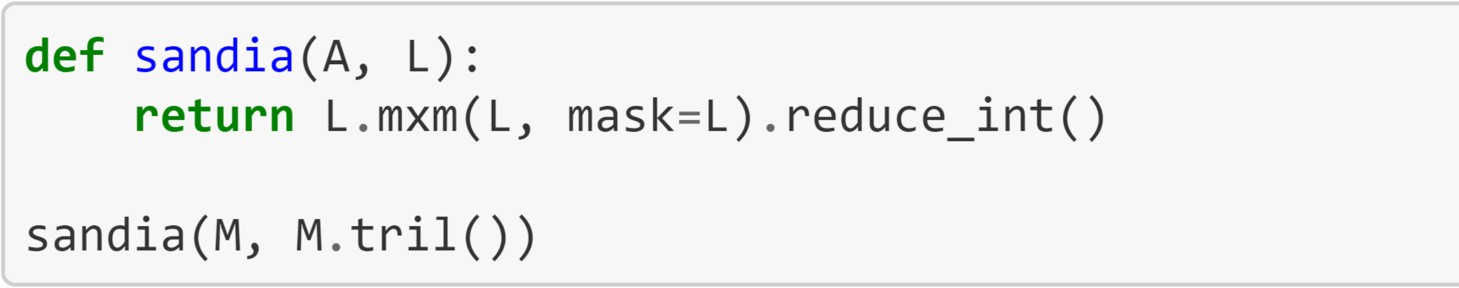
PYGRAPHBLAS: PYTHON WRAPPER

- **Goal:** Pythonic GraphBLAS wrapper, close to pseudo-code
- See example code for SSSP and triangle count
- Comes with Jupyter notebooks



```
def sssp(matrix, start):
    v = Vector.from_type(matrix.gb_type, matrix.nrows)
    v[start] = 0

    with min_plus, Accum(min_int64):
        for _ in range(matrix.nrows):
            w = Vector.dup(v)
            v @= matrix
            if w == v:
                break
        return v
```



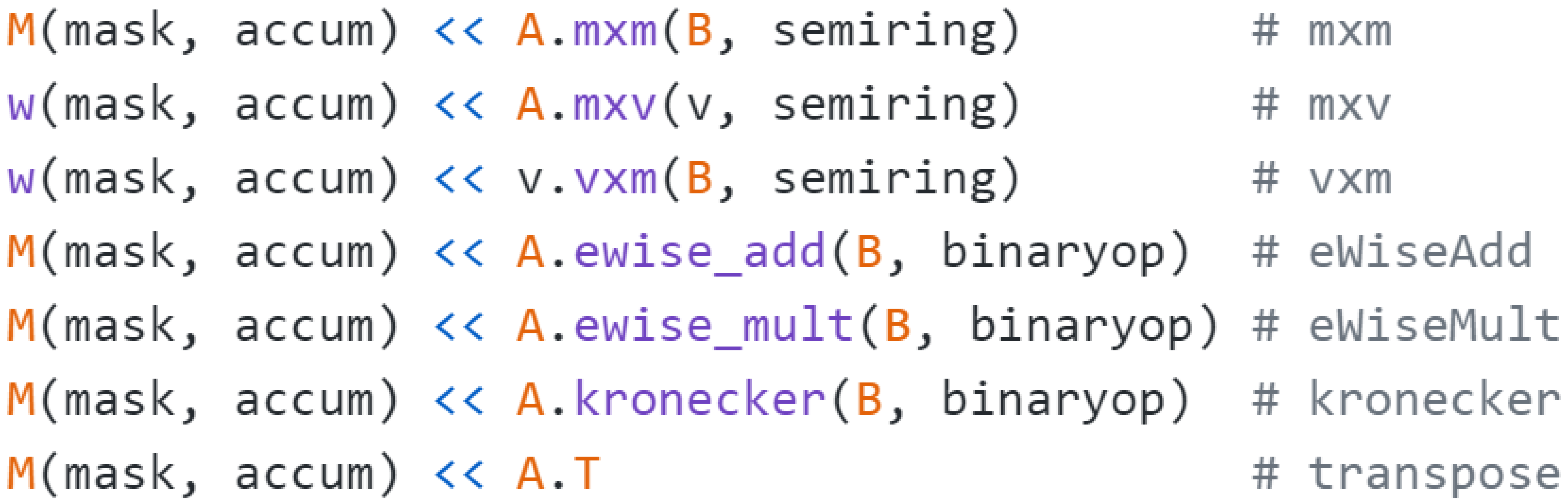
```
def sandia(A, L):
    return L.mxm(L, mask=L).reduce_int()

sandia(M, M.tril())
```

michelp/pygraphblas

GRBLAS: PYTHON WRAPPER

- **Goal:** wrapper with an almost 1-to-1 mapping to the GrB API
  - Comes with a Conda package
  - Compiles user-defined functions to C
  - Supports visualization

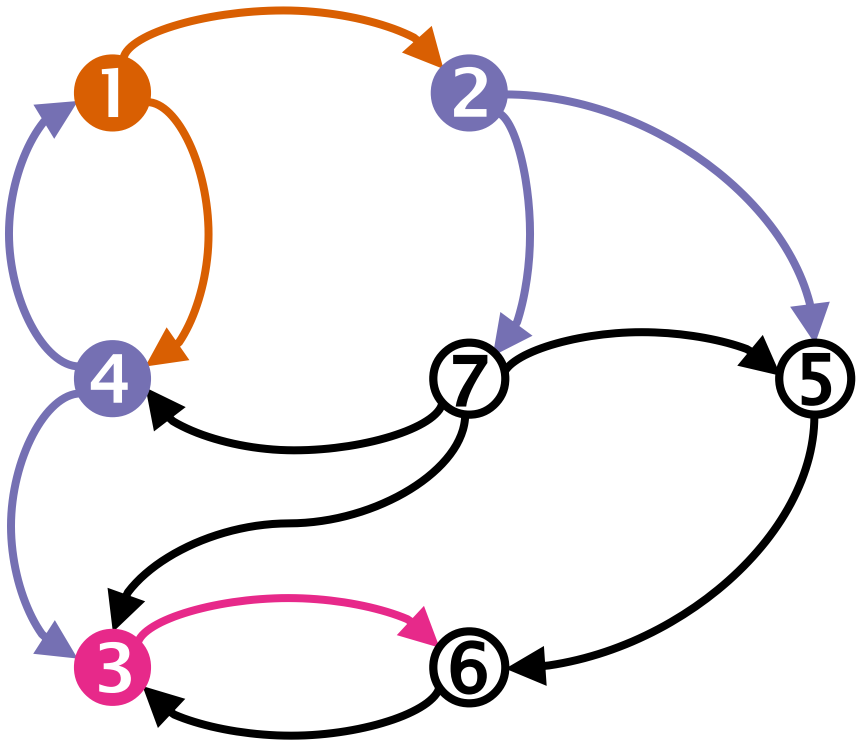


```
M(mask, accum) << A.mxm(B, semiring)        # mxm
w(mask, accum) << A.mxv(v, semiring)        # mxv
w(mask, accum) << v.vxm(B, semiring)        # vxm
M(mask, accum) << A.ewise_add(B, binaryop)  # eWiseAdd
M(mask, accum) << A.ewise_mult(B, binaryop) # eWiseMult
M(mask, accum) << A.kronecker(B, binaryop)  # kronecker
M(mask, accum) << A.T                       # transpose
```

metagraph-dev/grblas

Parallelism in GraphBLAS

PARALLELISM IN GRAPHBLAS



edges

A	①	②	③	④	⑤	⑥	⑦
①		●		●			
②					●		●
③						●	
④	●		●				
⑤						●	
⑥			●				
⑦			●	●	●		

vertices

traversals

F	①	②	③	④	⑤	⑥	⑦
t1	●						
t2			●				
t3		●		●			

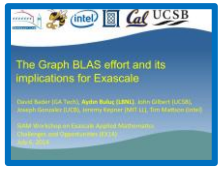
	●		●			
					●	
●		●		●		●

Frontier any . pair **A**

THE CASE FOR LINEAR ALGEBRA-BASED GRAPH ALGORITHMS

Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

Traditional graph computation	Graphs in the language of linear algebra
Data-driven, unpredictable communication	Fixed communication patterns
Irregular and unstructured, poor locality of reference	Operations on matrix blocks exploit memory hierarchy
Fine-grained data accesses, dominated by latency	Coarse-grained parallelism, bandwidth-limited



The Graph BLAS effort and its implications for Exascale

David Bader (GA Tech), **Aydin Buluç (LBNL)**, John Gilbert (UCSB), Joseph Gonzalez (UCB), Jeremy Kepner (MIT LL), Tim Mattson (Intel)

SIAM Workshop on Exascale Applied Mathematics Challenges and Opportunities (EX14)
July 6, 2014

D. Bader et al., *The GraphBLAS effort and its implications for Exascale*,
SIAM Workshop on Exascale Applied Mathematics Challenges and Opportunities, 2014

Summary

SUMMARY

- Linear algebra is a powerful abstraction
  - Good expressive power
  - Concise formulation of most graph algorithms
  - Good performance
  - Still lots of ongoing research
- Trade-offs:
  - Learning curve (maths, C programming, GraphBLAS API)
  - Some algorithms are difficult to formulate in linear algebra
  - Only a few GraphBLAS implementations (yet)
- **Overall:** GraphBLAS is a good abstraction layer for graph algorithms in the age of heterogeneous hardware

“*Nuances*” – Some important adjustments to the definitions

GRAPHBLAS SEMIRINGS*

The GraphBLAS specification defines semirings as follows:

$\langle D_{\mathrm{out}}, D_{\mathrm{in}_1}, D_{\mathrm{in}_2}, \oplus, \otimes, 0 \rangle$ structure is a *GraphBLAS semiring* defined by

D_{out}, D_{in_1}, and D_{in_2}	three domains
$\oplus : D_{\mathrm{out}} \times D_{\mathrm{out}} \rightarrow D_{\mathrm{out}}$	an associative and commutative addition operation
$\otimes : D_{\mathrm{in}_1} \times D_{\mathrm{in}_2} \rightarrow D_{\mathrm{out}}$	a multiplicative operation
$0 \in D_{\mathrm{out}}$	an identity element for $\oplus$

$A = \langle D_{\mathrm{out}}, \oplus, 0 \rangle$ is a commutative monoid.

$F = \langle D_{\mathrm{out}}, D_{\mathrm{in}_1}, D_{\mathrm{in}_2}, \otimes \rangle$ is a closed binary operator.

“It is expected that implementations will utilize IEEE-754 floating point arithmetic, which is not strictly associative.” (C API specification)

NOTATION*

- Symbols:
  - $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{M}$ – matrices
  - $\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{m}$ – vectors
  - s, k – scalar
  - i, j – indices
  - $\langle\mathbf{m}\rangle, \langle\mathbf{M}\rangle$ – masks
- Operators:
  - $\oplus$ – addition
  - $\otimes$ – multiplication, $\oslash$ – division
  - $^\top$ – transpose
  - $\odot$ – accumulator

Not included in the simplified table

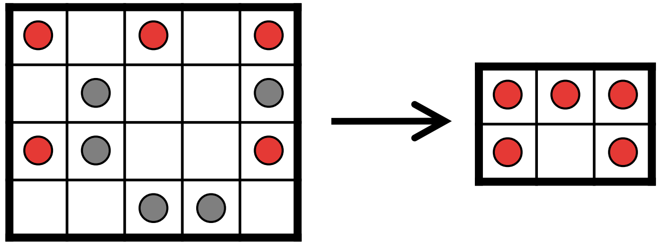
This table contains all GrB and GxB (SuiteSparse-specific) operations.

symbol	operation	notation
$\oplus.\otimes$	matrix-matrix multiplication	$\mathbf{C}\langle\mathbf{M}\rangle \odot= \mathbf{A} \oplus.\otimes \mathbf{B}$
	vector-matrix multiplication	$\mathbf{w}\langle\mathbf{m}\rangle \odot= \mathbf{v} \oplus.\otimes \mathbf{A}$
	matrix-vector multiplication	$\mathbf{w}\langle\mathbf{m}\rangle \odot= \mathbf{A} \oplus.\otimes \mathbf{v}$
$\otimes$	element-wise multiplication (set intersection of patterns)	$\mathbf{C}\langle\mathbf{M}\rangle \odot= \mathbf{A} \otimes \mathbf{B}$
		$\mathbf{w}\langle\mathbf{m}\rangle \odot= \mathbf{u} \otimes \mathbf{v}$
$\oplus$	element-wise addition (set union of patterns)	$\mathbf{C}\langle\mathbf{M}\rangle \odot= \mathbf{A} \oplus \mathbf{B}$
		$\mathbf{w}\langle\mathbf{m}\rangle \odot= \mathbf{u} \oplus \mathbf{v}$
f	apply unary operator	$\mathbf{C}\langle\mathbf{M}\rangle \odot= f(\mathbf{A})$
		$\mathbf{w}\langle\mathbf{m}\rangle \odot= f(\mathbf{v})$
$[\oplus \cdots]$	reduce to vector	$\mathbf{w}\langle\mathbf{m}\rangle \odot= \left[\oplus_j \mathbf{A}(:, j)\right]$
	reduce to scalar	$s \odot= \left[\oplus_{ij} \mathbf{A}(i, j)\right]$
$\mathbf{A}^\top$	transpose matrix	$\mathbf{C}\langle\mathbf{M}\rangle \odot= \mathbf{A}^\top$
–	extract submatrix	$\mathbf{C}\langle\mathbf{M}\rangle \odot= \mathbf{A}(\mathbf{i}, \mathbf{j})$
		$\mathbf{w}\langle\mathbf{m}\rangle \odot= \mathbf{v}(\mathbf{i})$
–	assign submatrix with submask for $\mathbf{C}(\mathbf{I}, \mathbf{J})$	$\mathbf{C}\langle\mathbf{M}\rangle(\mathbf{i}, \mathbf{j}) \odot= \mathbf{A}$
		$\mathbf{w}\langle\mathbf{m}\rangle(\mathbf{i}) \odot= \mathbf{v}$
–	assign submatrix with mask for $\mathbf{C}$	$\mathbf{C}(\mathbf{i}, \mathbf{j})\langle\mathbf{M}\rangle \odot= \mathbf{A}$
		$\mathbf{w}(\mathbf{i})\langle\mathbf{m}\rangle \odot= \mathbf{v}$
–	apply select operator (GxB)	$\mathbf{C}\langle\mathbf{M}\rangle \odot= f(\mathbf{A}, k)$
		$\mathbf{w}\langle\mathbf{m}\rangle \odot= f(\mathbf{v}, k)$
–	Kronecker product	$\mathbf{C}\langle\mathbf{M}\rangle \odot= \mathrm{kron}(\mathbf{A}, \mathbf{B})$

LINEAR ALGEBRAIC PRIMITIVES FOR GRAPHS #3*

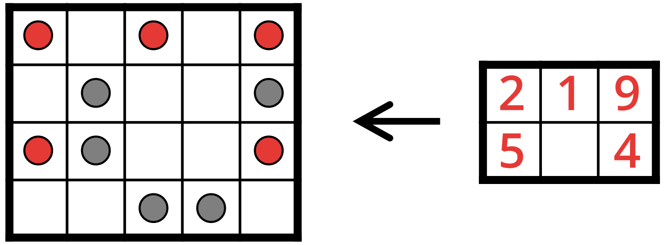
Sparse matrix extraction:

induced subgraph



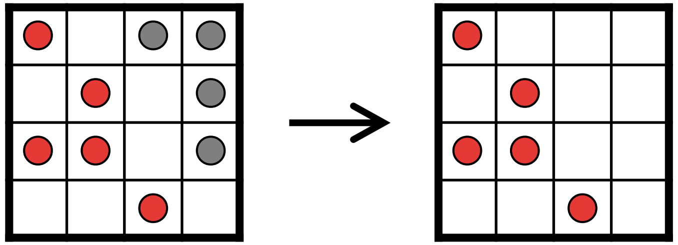
Sparse submatrix assignment:

replace subgraph



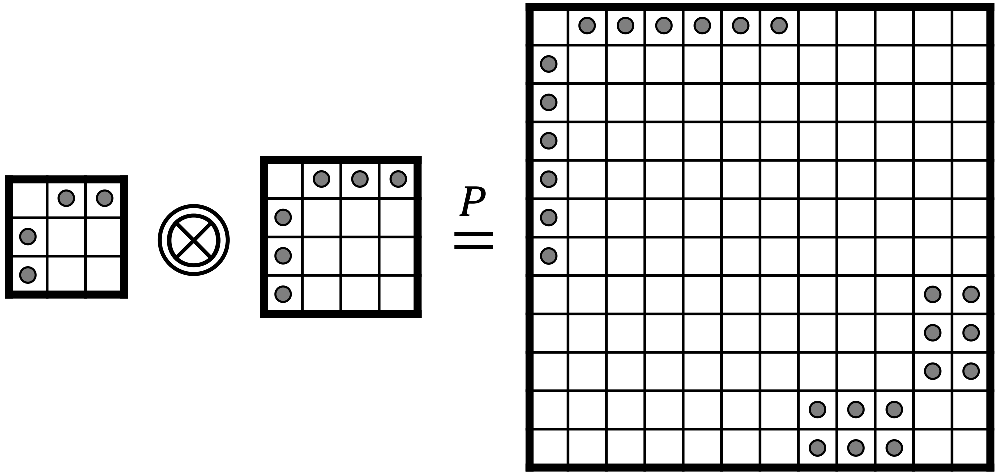
Sparse matrix selection:

filtering edges



Kronecker product:

graph generation



MATRIX-VECTOR MULTIPLICATION*

The operation $\mathbf{v} \oplus.\otimes \mathbf{A}$ gives the vertices reachable from the ones in $\mathbf{v}$. However, GraphBLAS publications and implementations often use $\mathbf{A}^\top \oplus.\otimes \mathbf{v}$ instead. The difference between these is that the former produces a row vector, while the latter produces a column vector:

$$\mathbf{v} \oplus.\otimes \mathbf{A} \equiv (\mathbf{A}^\top \oplus.\otimes \mathbf{v}^\top)^\top$$

The GraphBLAS does not distinguish row/column vectors, therefore the notations are (formally) equivalent:

$$\mathbf{v} \oplus.\otimes \mathbf{A} \equiv \mathbf{A}^\top \oplus.\otimes \mathbf{v}$$

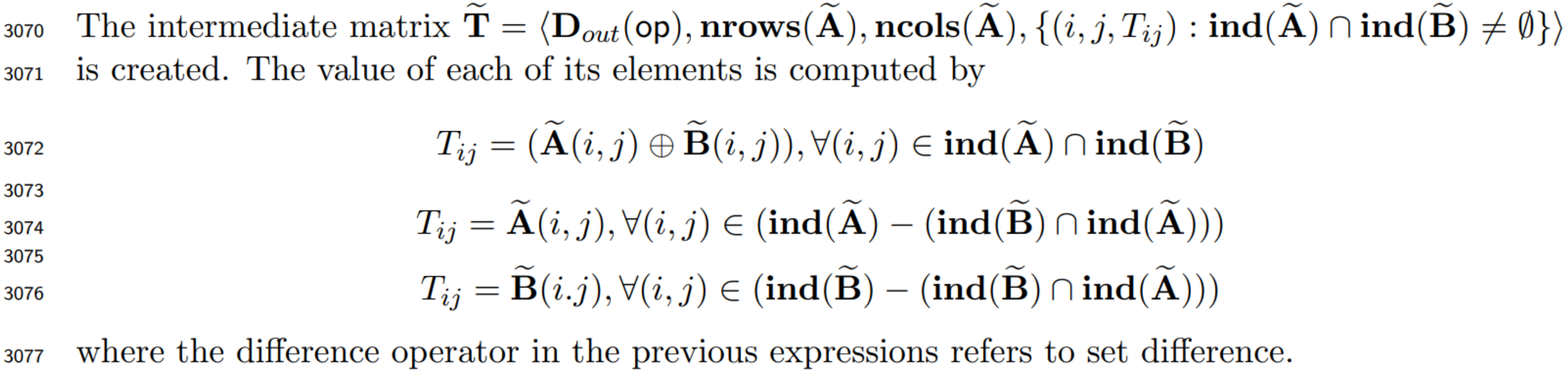
ELEMENT-WISE SUBTRACTION

Element-wise subtraction can be defined as an *element-wise addition* on the INT64_MINUS monoid. It has the following semantics $\mathbf{C} = \mathbf{A} \ominus \mathbf{B}$ is computed on the union of the patterns of the input matrices $\mathbf{A}$ and $\mathbf{B}$.

For cells where only one input matrix has a non-zero value but the other does not (e.g. $\mathbf{B}[0,0] = 1$ but $\mathbf{A}[0,0]$ is empty), the result is the non-zero value: $\mathbf{C}[0,0] = 1$.

$$\overset{\mathbf{A}}{\begin{bmatrix} & 1 \\ & \end{bmatrix}} \ominus \overset{\mathbf{B}}{\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}} = \overset{\mathbf{C}}{\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}}$$

This might come across as counter-intuitive first but it confirms the specification:



The intermediate matrix $\widetilde{\mathbf{T}} = \langle \mathbf{D}_{out}(\mathsf{op}), \mathbf{nrows}(\widetilde{\mathbf{A}}), \mathbf{ncols}(\widetilde{\mathbf{A}}), \{(i,j,T_{ij}) : \mathbf{ind}(\widetilde{\mathbf{A}}) \cap \mathbf{ind}(\widetilde{\mathbf{B}}) \neq \emptyset\}\rangle$ is created. The value of each of its elements is computed by

$$T_{ij} = (\widetilde{\mathbf{A}}(i,j) \oplus \widetilde{\mathbf{B}}(i,j)), \forall (i,j) \in \mathbf{ind}(\widetilde{\mathbf{A}}) \cap \mathbf{ind}(\widetilde{\mathbf{B}})$$

$$T_{ij} = \widetilde{\mathbf{A}}(i,j), \forall (i,j) \in (\mathbf{ind}(\widetilde{\mathbf{A}}) - (\mathbf{ind}(\widetilde{\mathbf{B}}) \cap \mathbf{ind}(\widetilde{\mathbf{A}})))$$

$$T_{ij} = \widetilde{\mathbf{B}}(i.j), \forall (i,j) \in (\mathbf{ind}(\widetilde{\mathbf{B}}) - (\mathbf{ind}(\widetilde{\mathbf{B}}) \cap \mathbf{ind}(\widetilde{\mathbf{A}})))$$

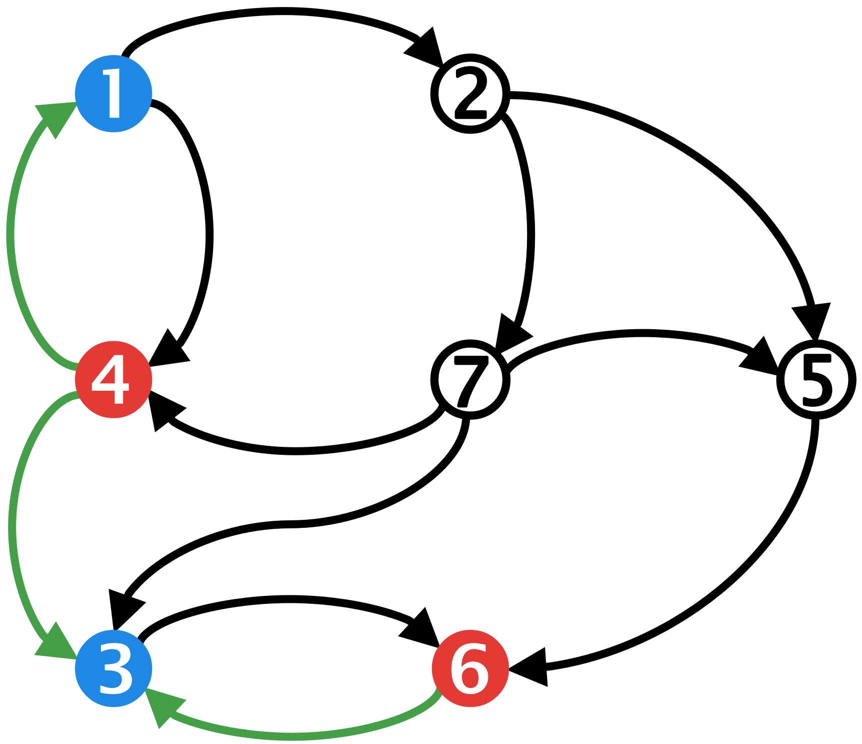
where the difference operator in the previous expressions refers to set difference.

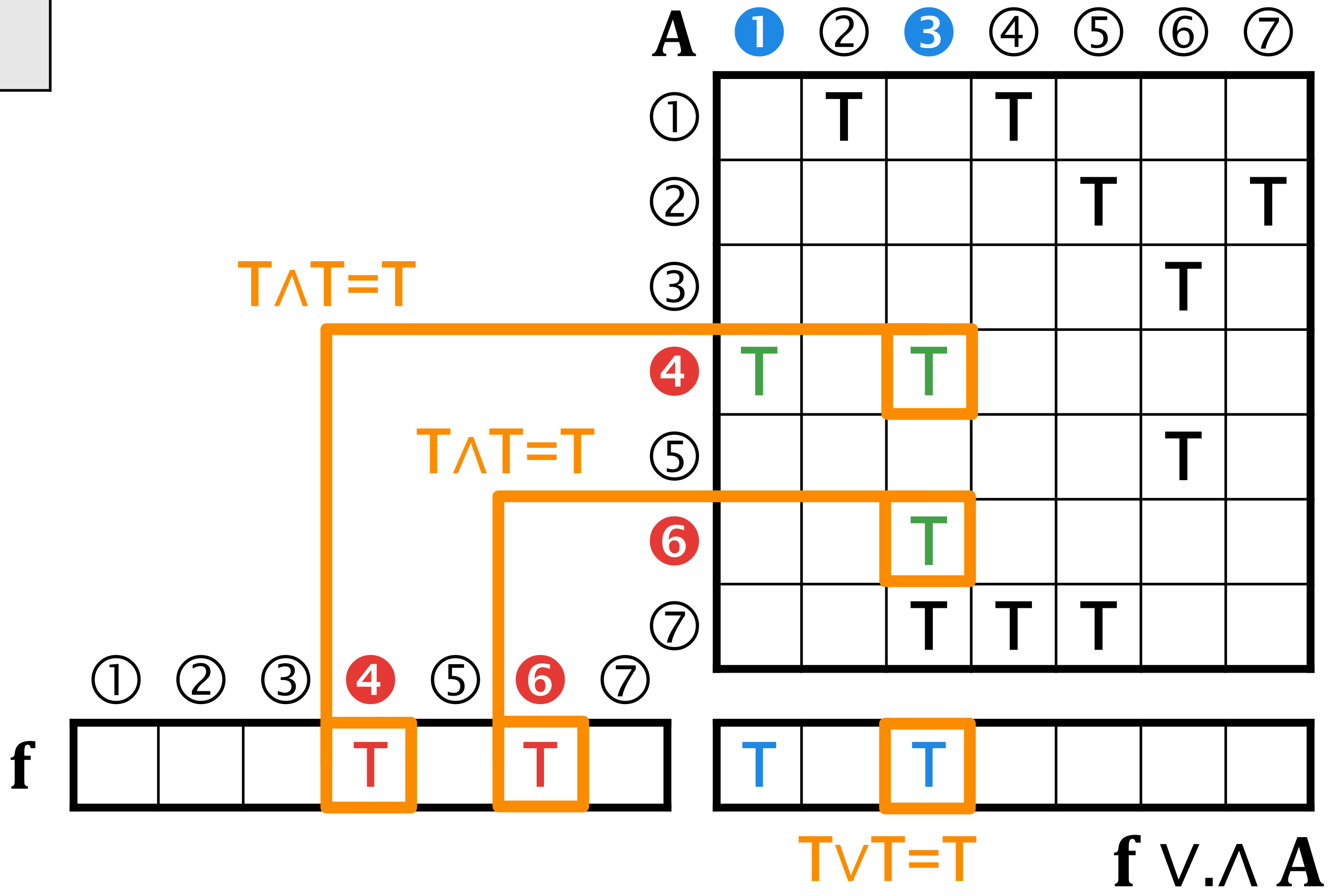
More semirings

MATRIX-VECTOR MULTIPLICATION SEMANTICS

semiring	set	⊕	⊗	0
lor-land	$\{T, F\}$	∨	∧	F

Semantics: **reachability**





A	❶	②	❸	④	⑤	⑥	⑦
①		T		T			
②					T		T
③						T	
❹	T		T				
⑤						T	
❻			T				
⑦			T	T	T		

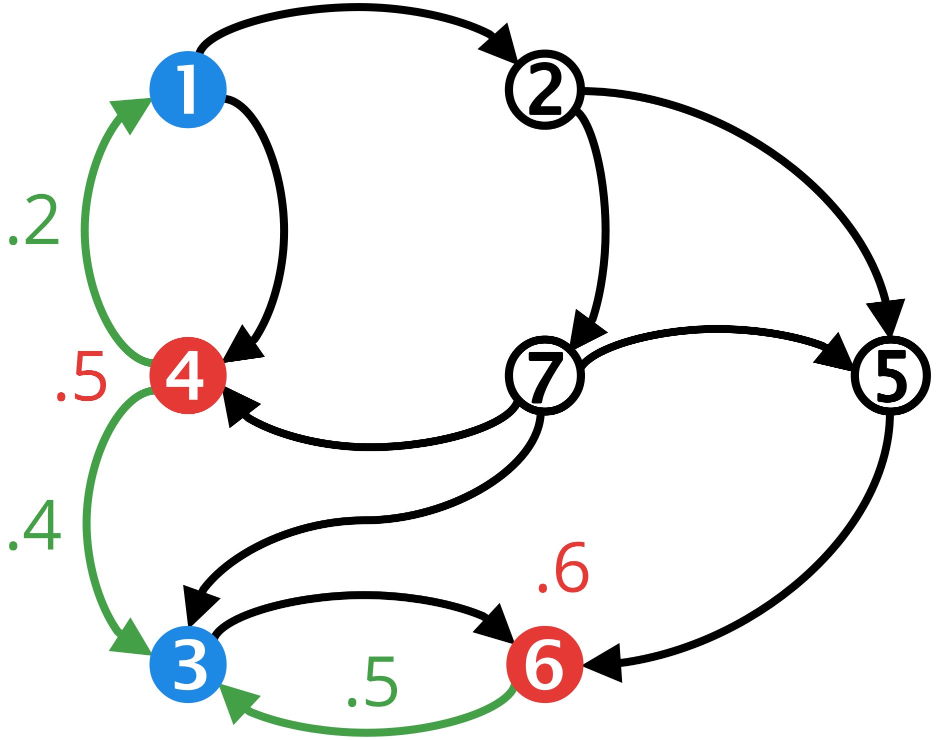
	①	②	③	❹	⑤	❻	⑦
f				T		T	

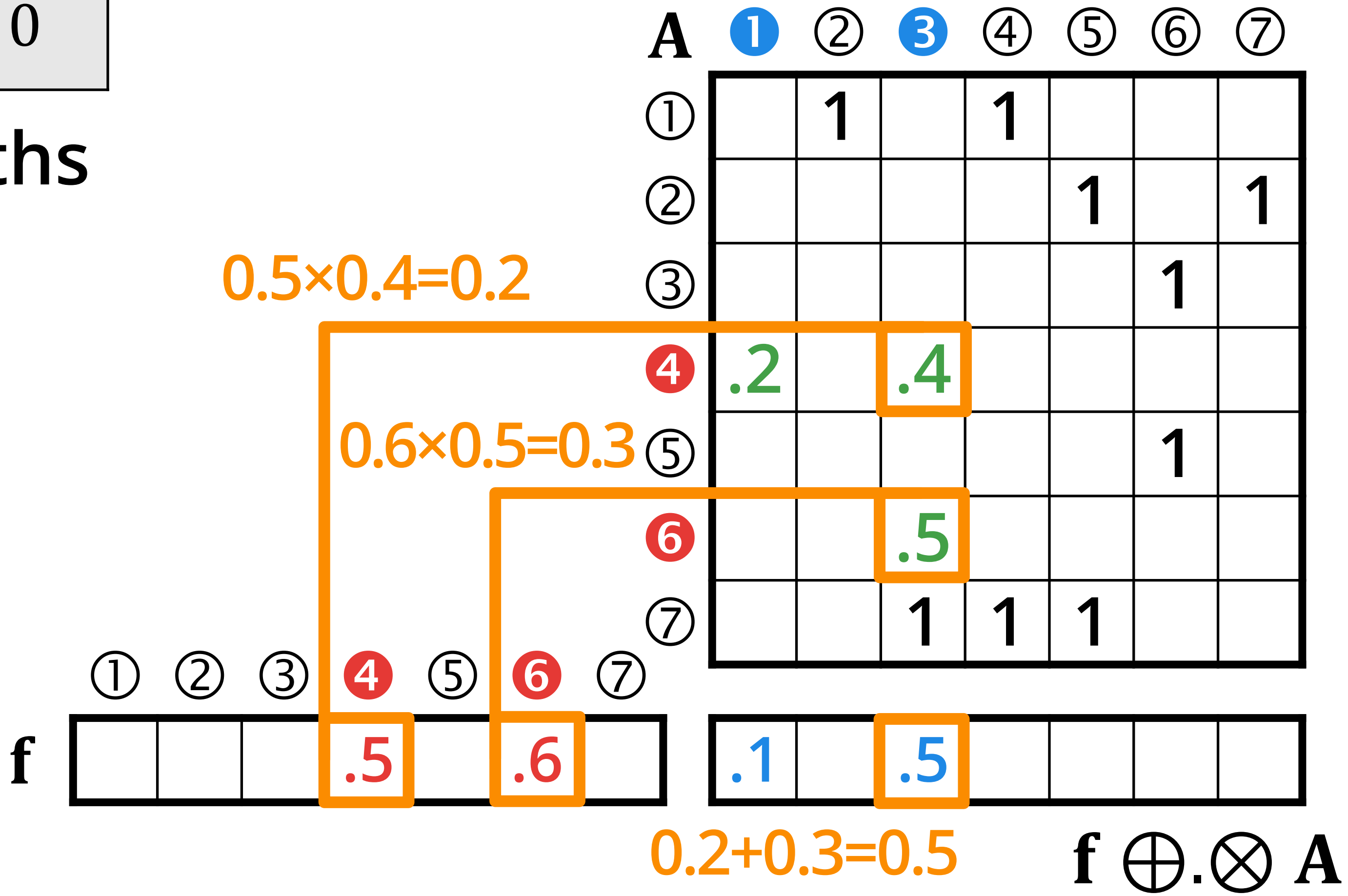
T		T				

MATRIX-VECTOR MULTIPLICATION SEMANTICS

semiring	set	⊕	⊗	0
real arithmetic	ℝ	+	×	0

Semantics: **strength of all paths**





A	❶	②	❸	④	⑤	⑥	⑦
①		1		1			
②					1		1
③						1	
❹	.2		.4				
⑤						1	
❻			.5				
⑦			1	1	1		

	①	②	③	❹	⑤	❻	⑦
f				.5		.6	

f ⊕.⊗ A	.1		.5				

0.5×0.4=0.2

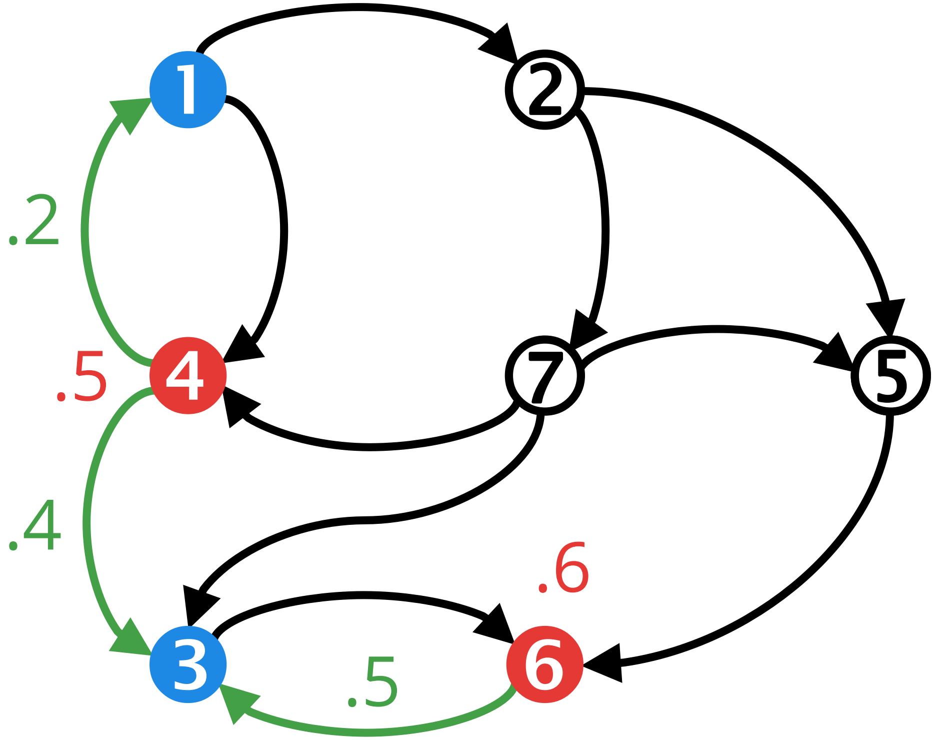
0.6×0.5=0.3

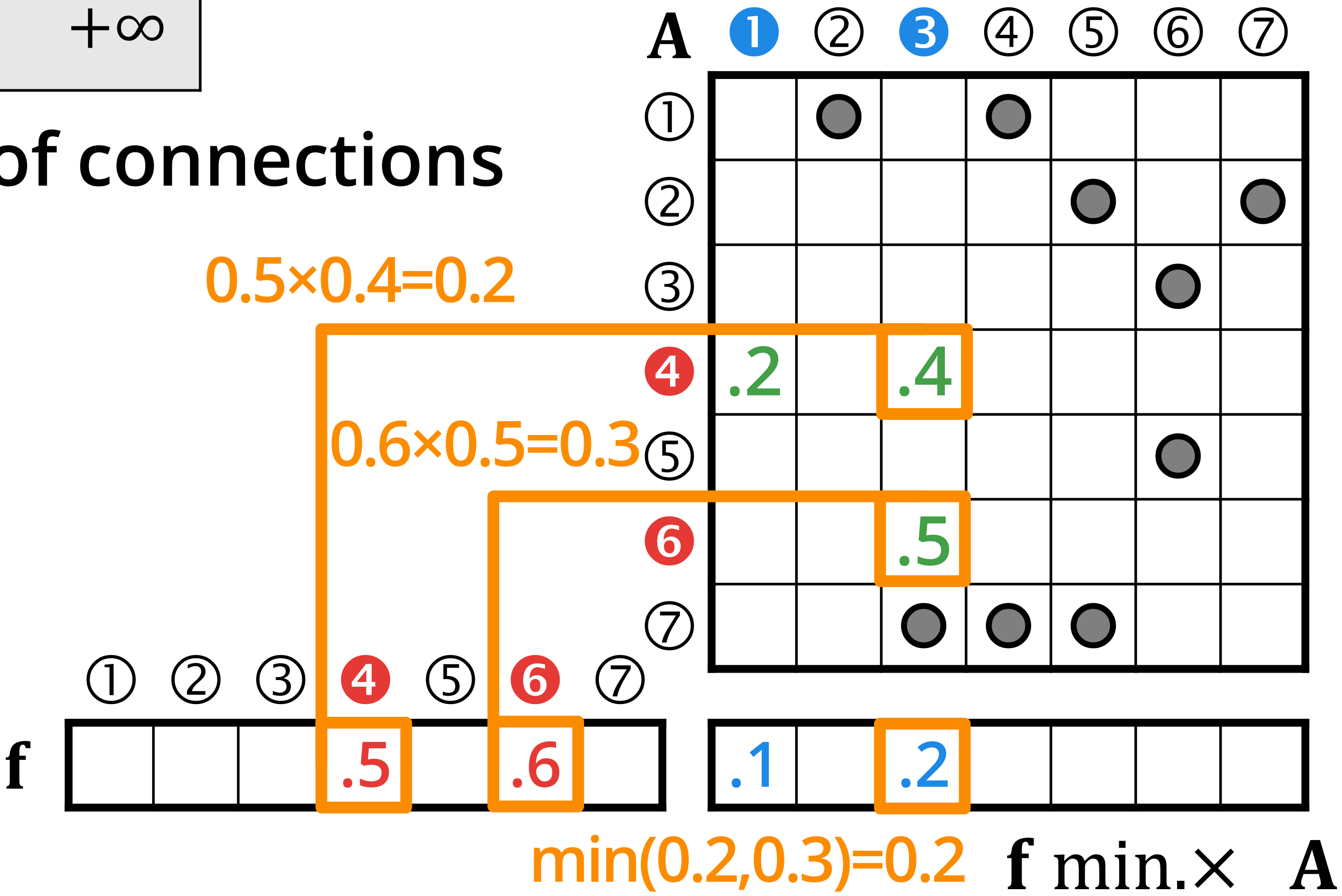
0.2+0.3=0.5

MATRIX-VECTOR MULTIPLICATION SEMANTICS

semiring	set	⊕	⊗	0
min-times	$\mathbb{R} \cup \{+\infty\}$	min	×	$+\infty$

Semantics: **shortest product of connections**

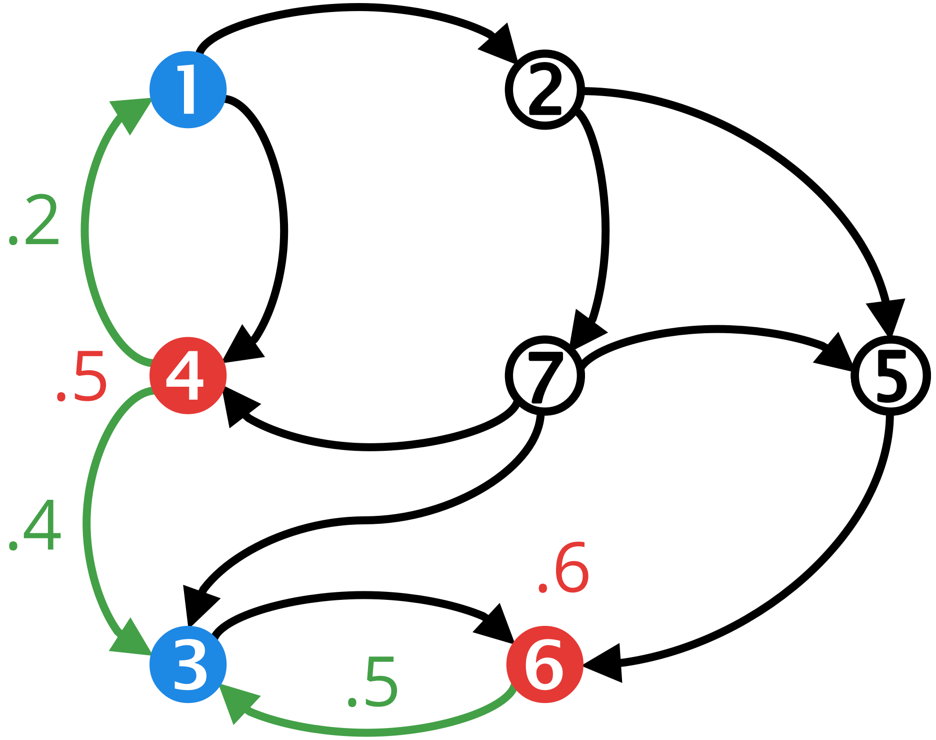


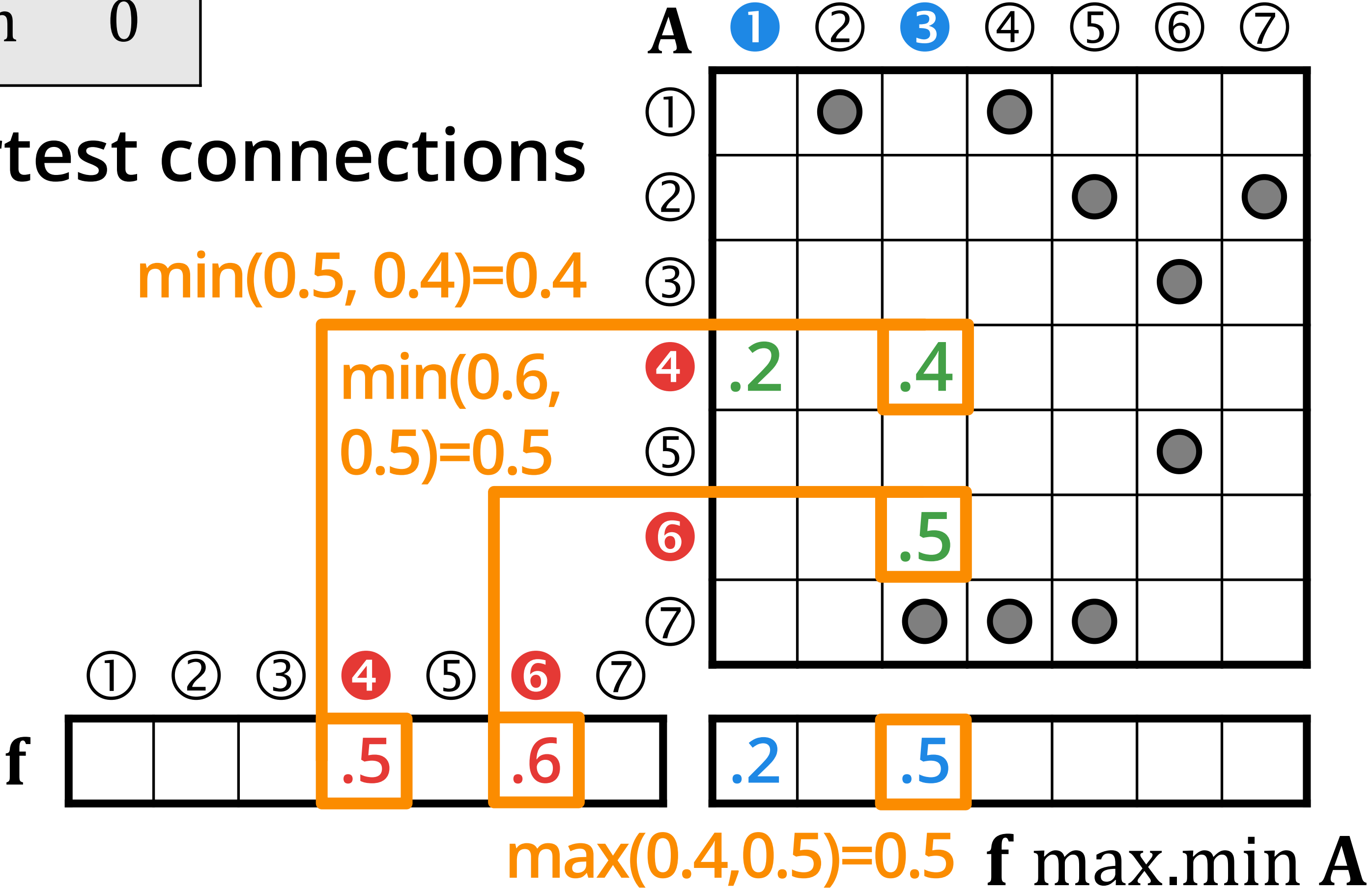


MATRIX-VECTOR MULTIPLICATION SEMANTICS

semiring	set	⊕	⊗	0
max-min	$\{0, +\infty\}$	max	min	0

Semantics: **longest of all shortest connections**

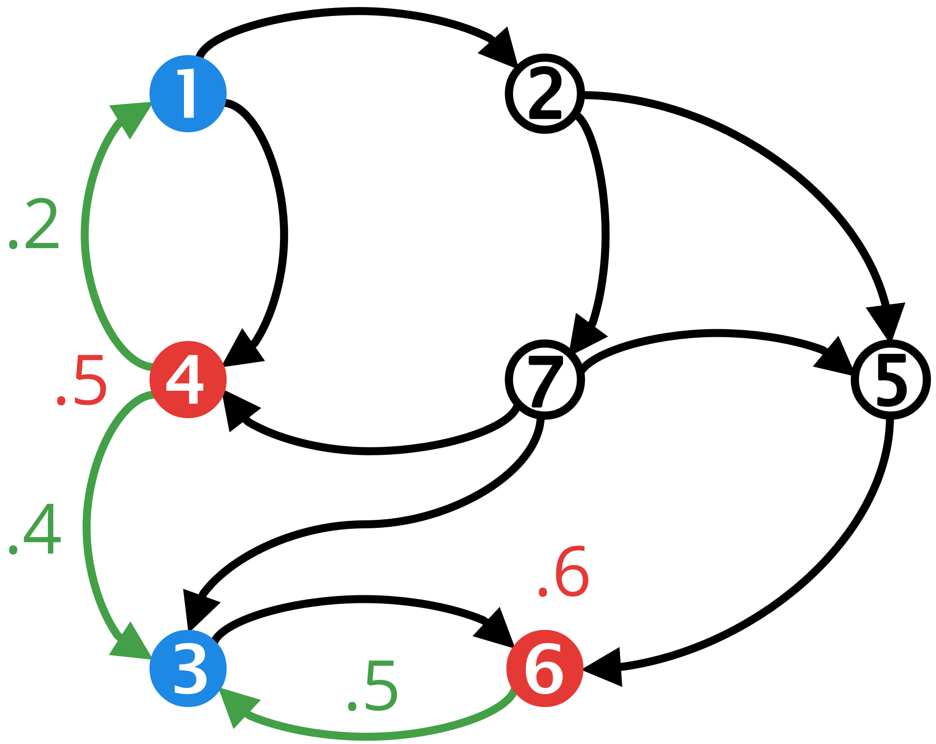


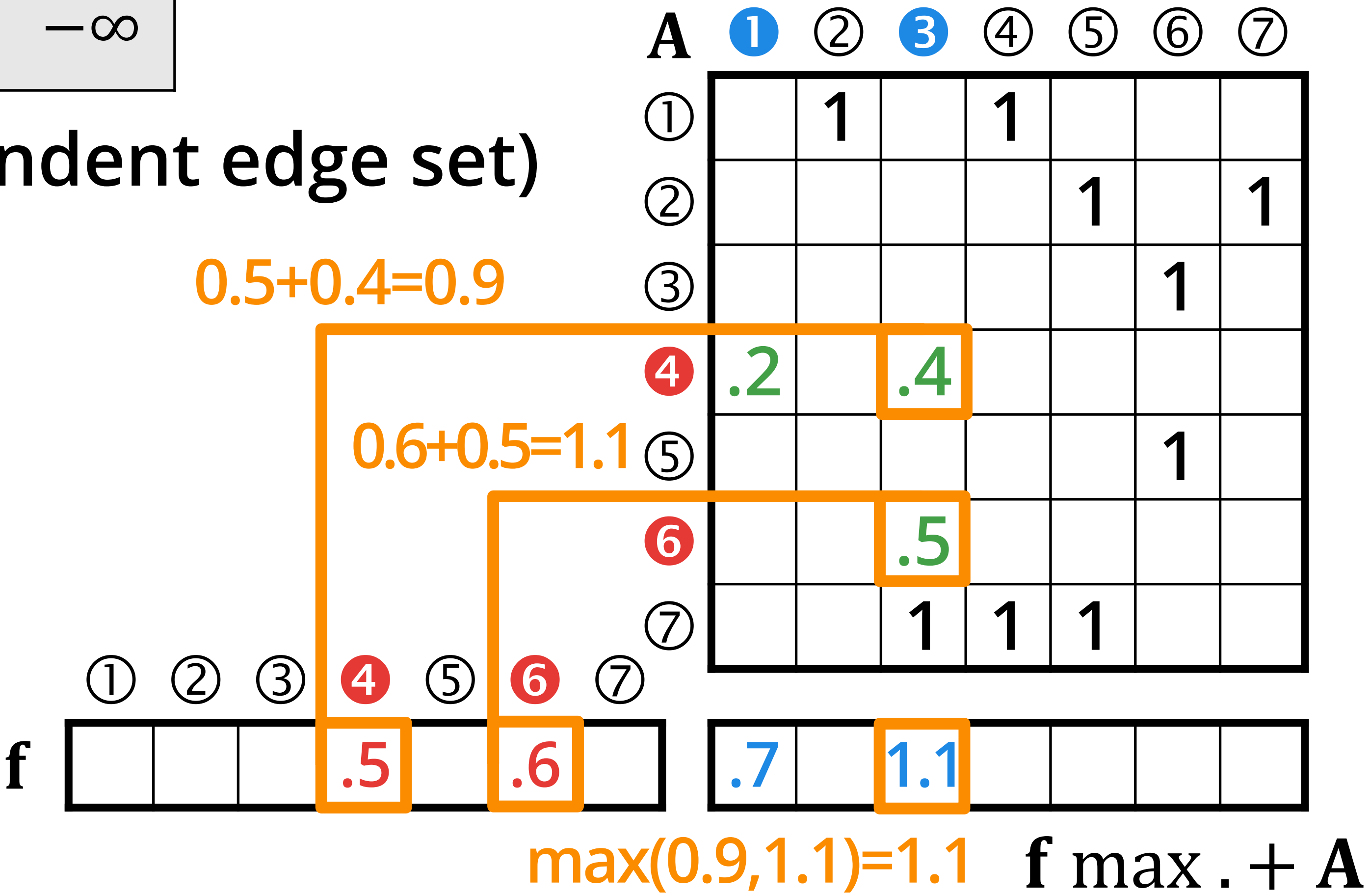


MATRIX-VECTOR MULTIPLICATION SEMANTICS

semiring	set	⊕	⊗	0
max-plus	$\mathbb{R} \cup \{-\infty\}$	max	$+$	$-\infty$

Semantics: **matching (independent edge set)**





0.5+0.4=0.9

0.6+0.5=1.1

A	❶	②	❸	④	⑤	⑥	⑦
①		1		1			
②					1		1
③						1	
❹	.2		.4				
⑤						1	
❻			.5				
⑦			1	1	1		

	①	②	③	❹	⑤	❻	⑦
f				.5		.6	

.7		1.1				

max(0.9,1.1)=1.1

$\mathbf{f}\ \mathrm{max}.+\ \mathbf{A}$

Case study: SIGMOD 2014 Contest

Overview

SIGMOD 2014 PROGRAMMING CONTEST

Annual contest

- Teams compete on database-related programming tasks
- Highly-optimized C++ implementations

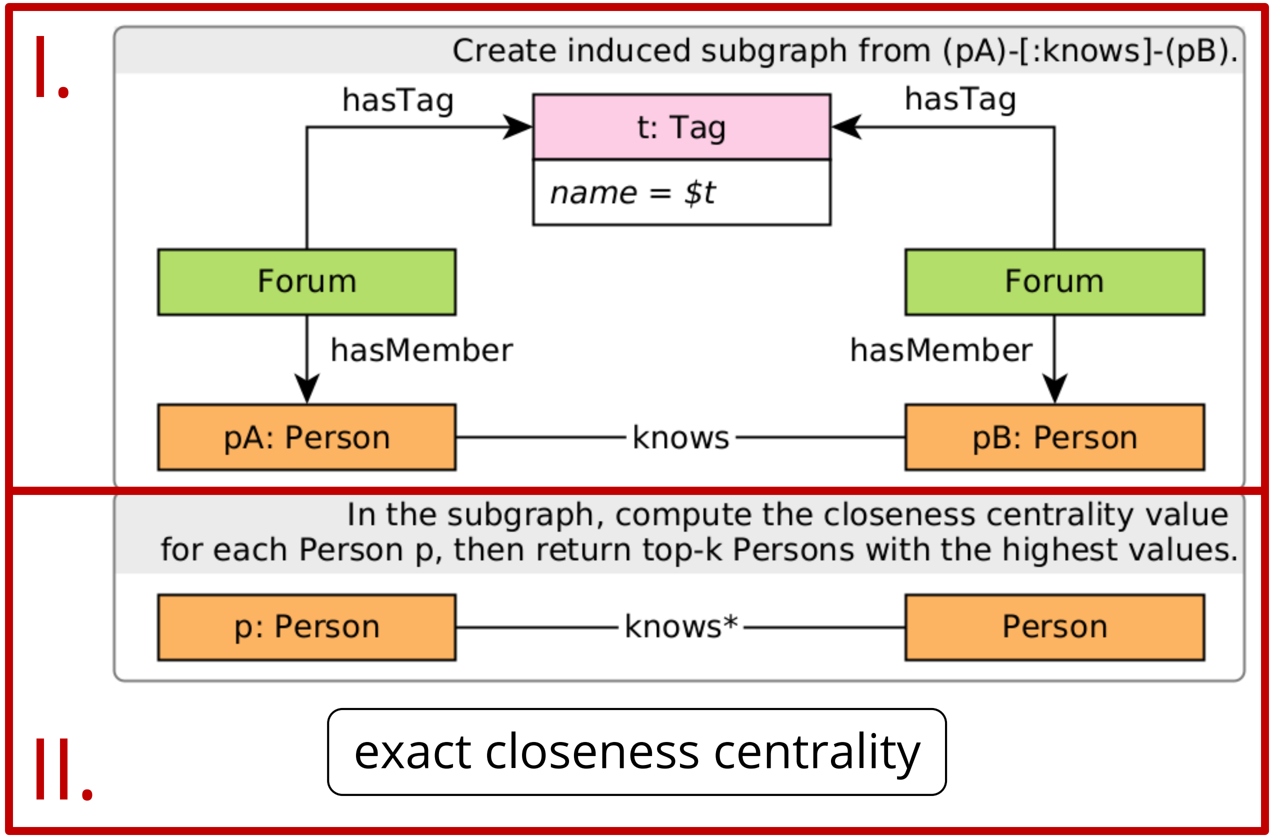
2014 event

- Tasks on the LDBC social network graph
  - Benchmark data set for property graphs
  - People, forums, comments, hashtags, etc.
- 4 queries
  - Mix of filtering operations and graph algorithms

QUERY TEMPLATE

I. Compute an induced subgraph over Person-knows-Person

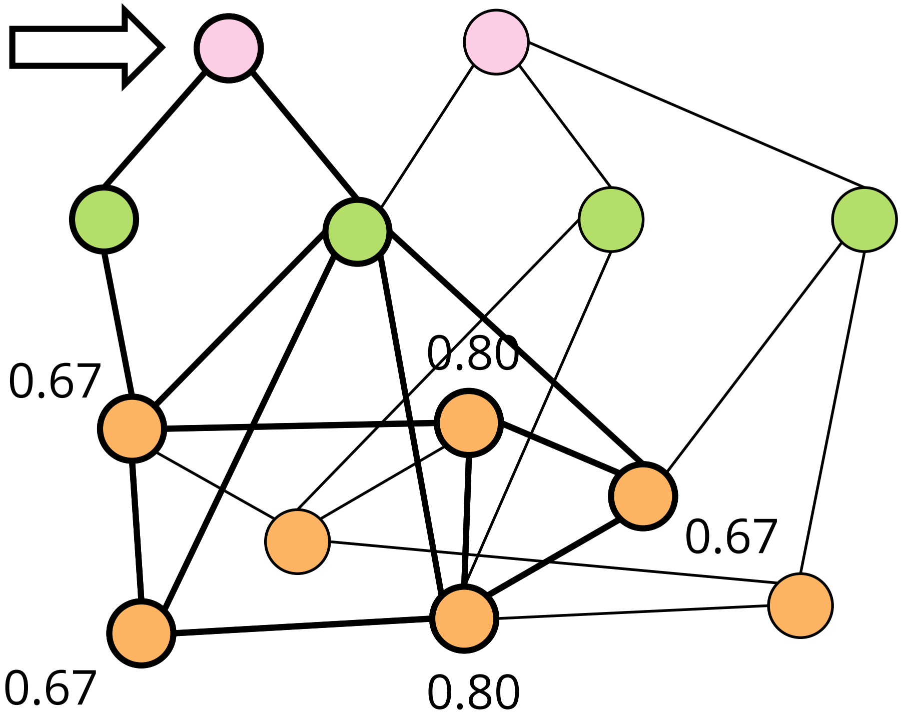
II. Run the graph algorithm on the subgraph



Create induced subgraph from (pA)-[:knows]-(pB).

In the subgraph, compute the closeness centrality value for each Person p, then return top-k Persons with the highest values.

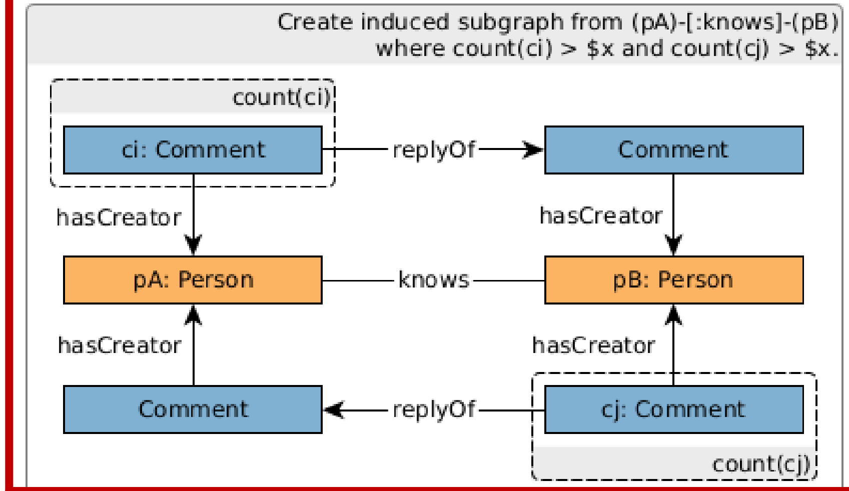
exact closeness centrality

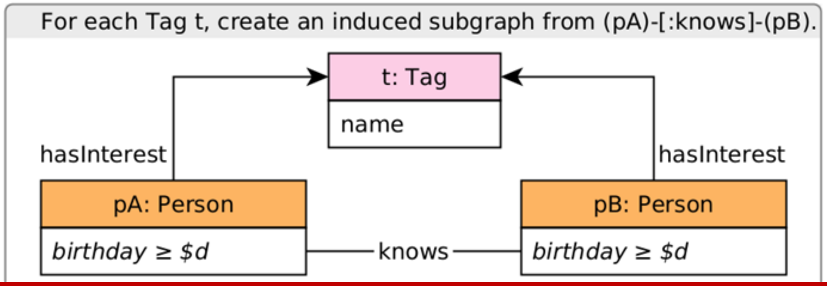


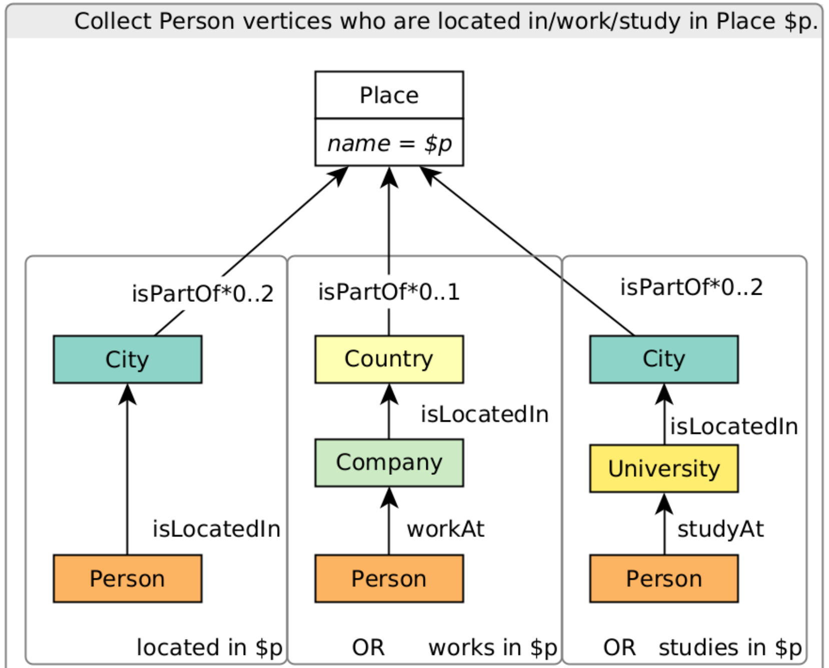
key kernel: **all-source BFS**

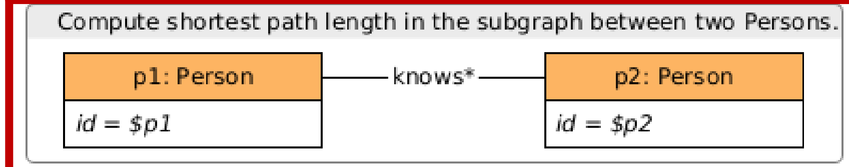
OVERVIEW OF QUERIES 1, 2, 3

I. Filter the induced subgraph(s)

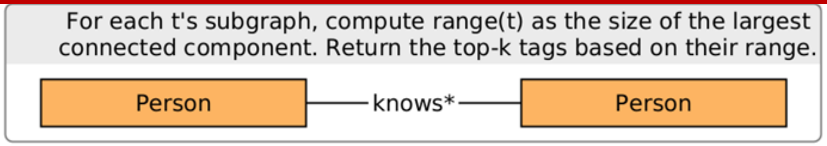

Create induced subgraph from (pA)-[:knows]-(pB) where count(ci) > $x and count(cj) > $x.


For each Tag t, create an induced subgraph from (pA)-[:knows]-(pB).

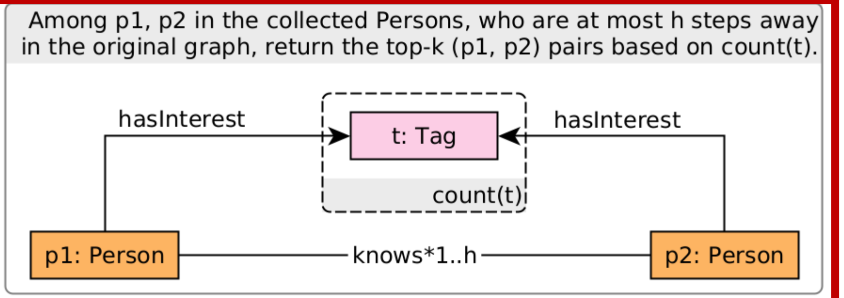

Collect Person vertices who are located in/work/study in Place $p.


Compute shortest path length in the subgraph between two Persons.

unweighted shortest path


For each t's subgraph, compute range(t) as the size of the largest connected component. Return the top-k tags based on their range.

connected components


Among p1, p2 in the collected Persons, who are at most h steps away in the original graph, return the top-k (p1, p2) pairs based on count(t).

pairwise reachability

II. Run the graph algorithm

GRAPHBLAS SOLUTION OF THE QUERIES

- **Loading** includes relabelling UINT64 vertex IDs to a contiguous sequence $0 \ldots N-1$.

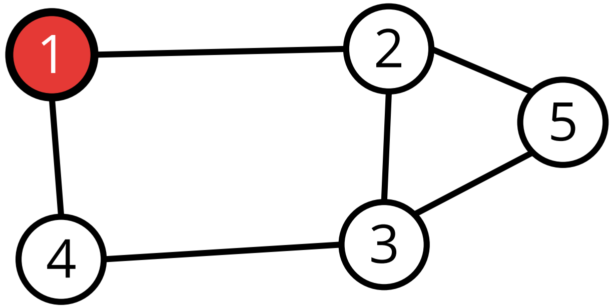
- **Filtering the induced subgraph** from the property graph is mostly straightforward and composable with the algorithms.

- **The algorithms** can be concisely expressed in GraphBLAS:
  - Connected components ✓ → FastSV [Zhang et al., PPSC’20]
  - BFS ✓
  - Bidirectional BFS
  - All-source BFS + bitwise optimization
  - Multi-source bidirectional BFS

Case study: SIGMOD 2014 Contest

BFS

BFS: BREADTH-FIRST SEARCH



Boolean matrices and vectors

frontier

①	●
②	
③	
④	
⑤	

⟨¬**seen**⟩ mask

seen

①	●
②	
③	
④	
⑤	

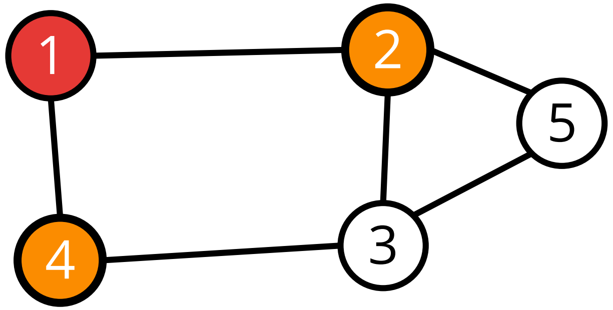
A	①	②	③	④	⑤
①		●		●	
②	●		●		●
③		●		●	●
④	●		●		
⑤		●	●		

☒

⊕: any
⊗: pair

next⟨¬**seen**⟩ =
A any . pair **frontier**

BFS: BREADTH-FIRST SEARCH



frontier

①	●
②	
③	
④	
⑤	

seen

①	●
②	
③	
④	
⑤	

A	①	②	③	④	⑤	
①		●		●		☒
②	●		●		●	●
③		●		●	●	
④	●		●			●
⑤		●	●			

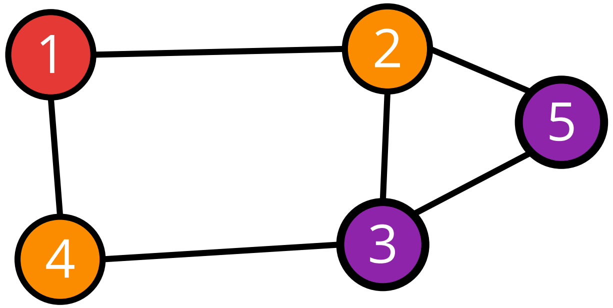
seen′

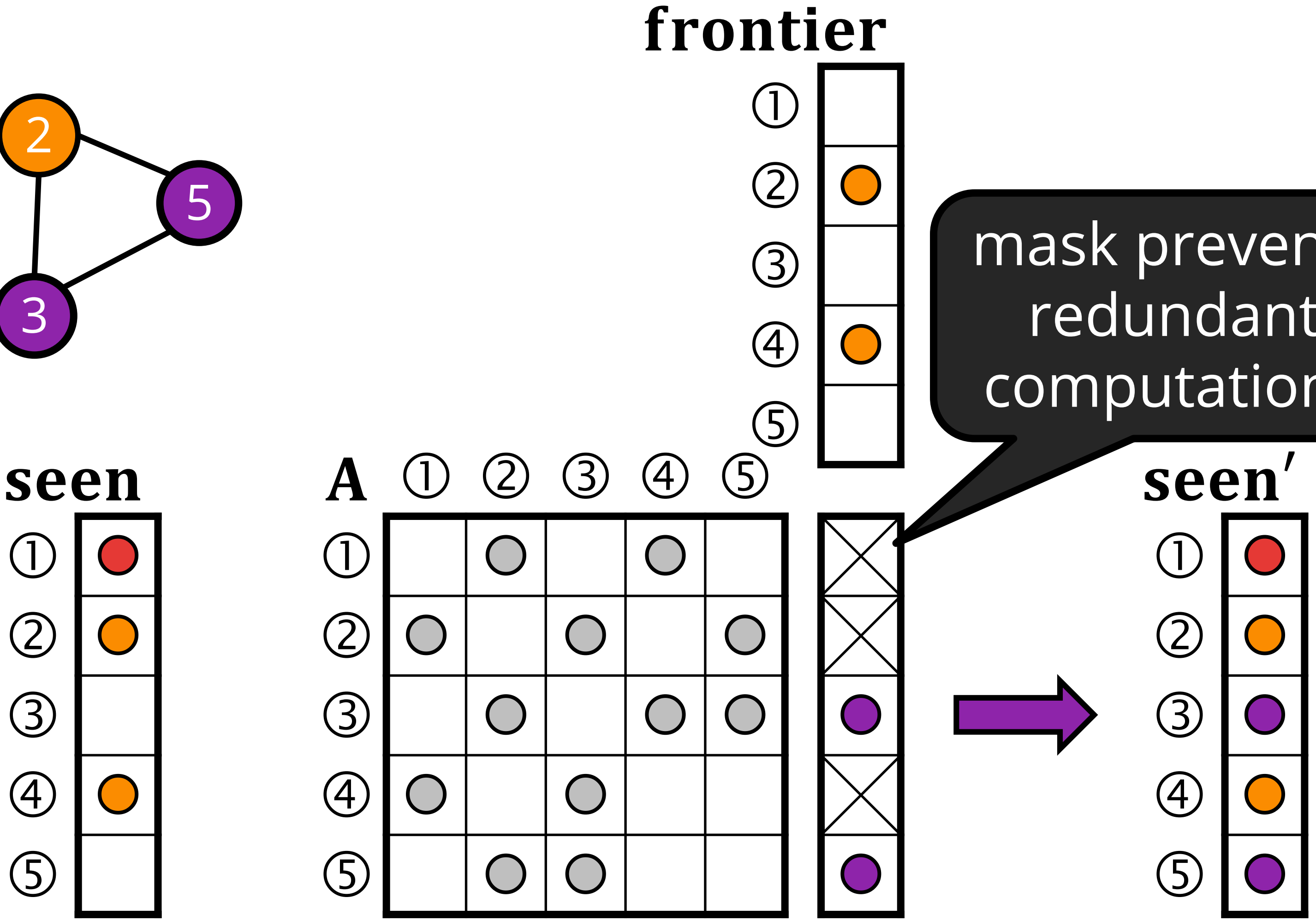
①	●
②	●
③	
④	●
⑤	

next⟨¬**seen**⟩ =
A any . pair **frontier**

seen′ =
seen any **next**

BFS: BREADTH-FIRST SEARCH





mask prevents redundant computations

next⟨¬**seen**⟩ =
A any . pair **frontier**

seen′ =
seen any **next**

Case study: SIGMOD 2014 Contest

All-source BFS

Q4: CLOSENESS CENTRALITY VALUES

Q4 computes the top-*k* Person vertices based on their exact *closeness centrality values*:

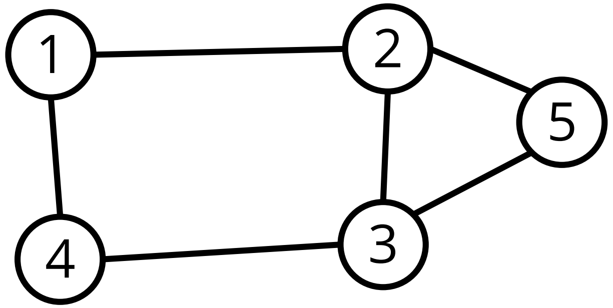
$$CCV(p) = \frac{(C(p) - 1)^2}{(n - 1) \cdot s(p)}$$

where

- $C(p)$ is the size of the connected component of vertex p,
- n is the number of vertices in the induced graph,
- $s(p)$ is the sum of geodesic distances to all other reachable persons from p.

$s(p)$ is challenging: needs unweighted all-pairs shortest paths.

BOOLEAN ALL-SOURCE BFS ALGORITHM



Frontier

traversals

	t1	t2	t3	t4	t5
①	🔴				
②		🔴			
③			🔴		
④				🔴	
⑤					🔴

Seen

	t1	t2	t3	t4	t5
①	🔴				
②		🔴			
③			🔴		
④				🔴	
⑤					🔴

A

	①	②	③	④	⑤
①		●		●	
②	●		●		●
③		●		●	●
④	●		●		
⑤		●	●		

☒	🟠		🟠	
🟠	☒	🟠		🟠
	🟠	☒	🟠	🟠
🟠		🟠	☒	
	🟠	🟠		☒

$$\mathbf{Next}\langle\neg\mathbf{Seen}\rangle = \mathbf{A}\ \text{any}\,.\,\text{pair}\ \mathbf{Frontier}$$

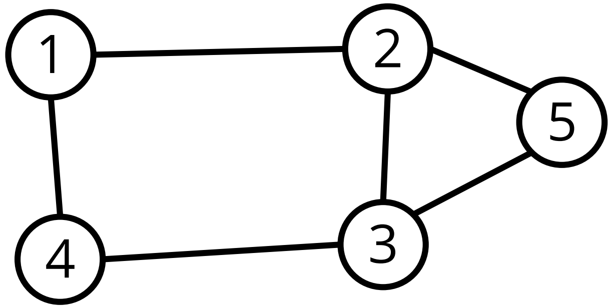
→

Seen′

	t1	t2	t3	t4	t5
①	🔴	🟠		🟠	
②	🟠	🔴	🟠		🟠
③		🟠	🔴	🟠	🟠
④	🟠		🟠	🔴	
⑤		🟠	🟠		🔴

$$\mathbf{Seen}' = \mathbf{Seen}\ \text{any}\ \mathbf{Next}$$

BOOLEAN ALL-SOURCE BFS ALGORITHM



Frontier

	t1	t2	t3	t4	t5
①		🟠		🟠	
②	🟠		🟠		🟠
③		🟠		🟠	🟠
④	🟠		🟠		
⑤		🟠	🟠		

Seen

	t1	t2	t3	t4	t5
①	🔴	🟠		🟠	
②	🟠	🔴	🟠		🟠
③		🟠	🔴	🟠	🟠
④	🟠		🟠	🔴	
⑤		🟠	🟠		🔴

A

	①	②	③	④	⑤
①		⚫		⚫	
②	⚫		⚫		⚫
③		⚫		⚫	⚫
④	⚫		⚫		
⑤		⚫	⚫		

☒	☒	🟣	☒	🟣
☒	☒	☒	🟣	☒
🟣	☒	☒	☒	☒
☒	🟣	☒	☒	🟣
🟣	☒	☒	🟣	☒

→

Seen′

	t1	t2	t3	t4	t5
①	🔴	🟠	🟣	🟠	🟣
②	🟠	🔴	🟠	🟣	🟠
③	🟣	🟠	🔴	🟠	🟠
④	🟠	🟣	🟠	🔴	🟣
⑤	🟣	🟠	🟠	🟣	🔴

Next⟨¬**Seen**⟩ =
A any . pair **Frontier**

Seen′ =
Seen any **Next**

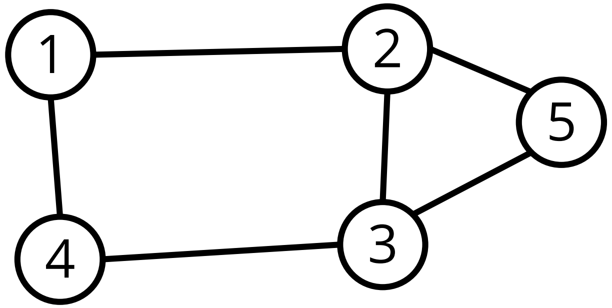
Case study: SIGMOD 2014 Contest

Bitwise all-source BFS

BITWISE ALL-SOURCE BFS ALGORITHM

- For large graphs, the all-source BFS algorithm might need to run 500k+ traversals
- Two top-ranking teams used bitwise operations to process traversals in batches of 64 [Then et al., VLDB’15]
- This idea can be adopted in the GraphBLAS algorithm by
  - using `UINT64` values
  - performing the multiplication on the $\mathrm{bor}\,.\,\mathrm{second}$ semiring, where bor is “bitwise or” and $\mathrm{second}(x, y) = y$
- 5-10x speedup compared to the Boolean all-source BFS

BITWISE ALL-SOURCE BFS ALGORITHM



Using UINT4s here

Frontier

	t1-t4	t5
①	1000	0000
②	0100	0000
③	0010	0000
④	0001	0000
⑤	0000	1000

Seen

	t1-t4	t5
①	1000	0000
②	0100	0000
③	0010	0000
④	0001	0000
⑤	0000	1000

A

	①	②	③	④	⑤
①		●		●	
②	●		●		●
③		●		●	●
④	●		●		
⑤		●	●		

0101	0000
1010	1000
0101	1000
1010	0000
0110	0000

Next =
A bor . second **Frontier**

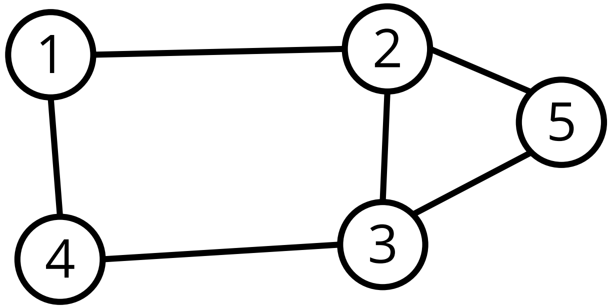
➡

Seen′

	t1-t4	t5
①	0101	0000
②	1010	1000
③	0101	1000
④	1010	0000
⑤	0110	0000

Seen′ =
Seen bor **Next**

BITWISE ALL-SOURCE BFS ALGORITHM



Frontier

	t1-t4	t5
①	0101	0000
②	1010	1000
③	0101	1000
④	1010	0000
⑤	0110	0000

Full VLDB paper on this algorithm

vs.

9 GrB operations

Seen

	t1-t4	t5
①	0101	0000
②	1010	1000
③	0101	1000
④	1010	0000
⑤	0110	0000

A	①	②	③	④	⑤
①		●		●	
②	●		●		●
③		●		●	●
④	●		●		
⑤		●	●		

0010	1000
0001	0000
1000	0000
0100	1000
1001	0000

$\mathbf{Next} =$

$\mathbf{A}\ \text{bor.second}\ \mathbf{Frontier}$

Seen′

	t1-t4	t5
①	1111	1000
②	1111	1000
③	1111	1000
④	1111	1000
⑤	1111	1000

$\mathbf{Seen'} =$

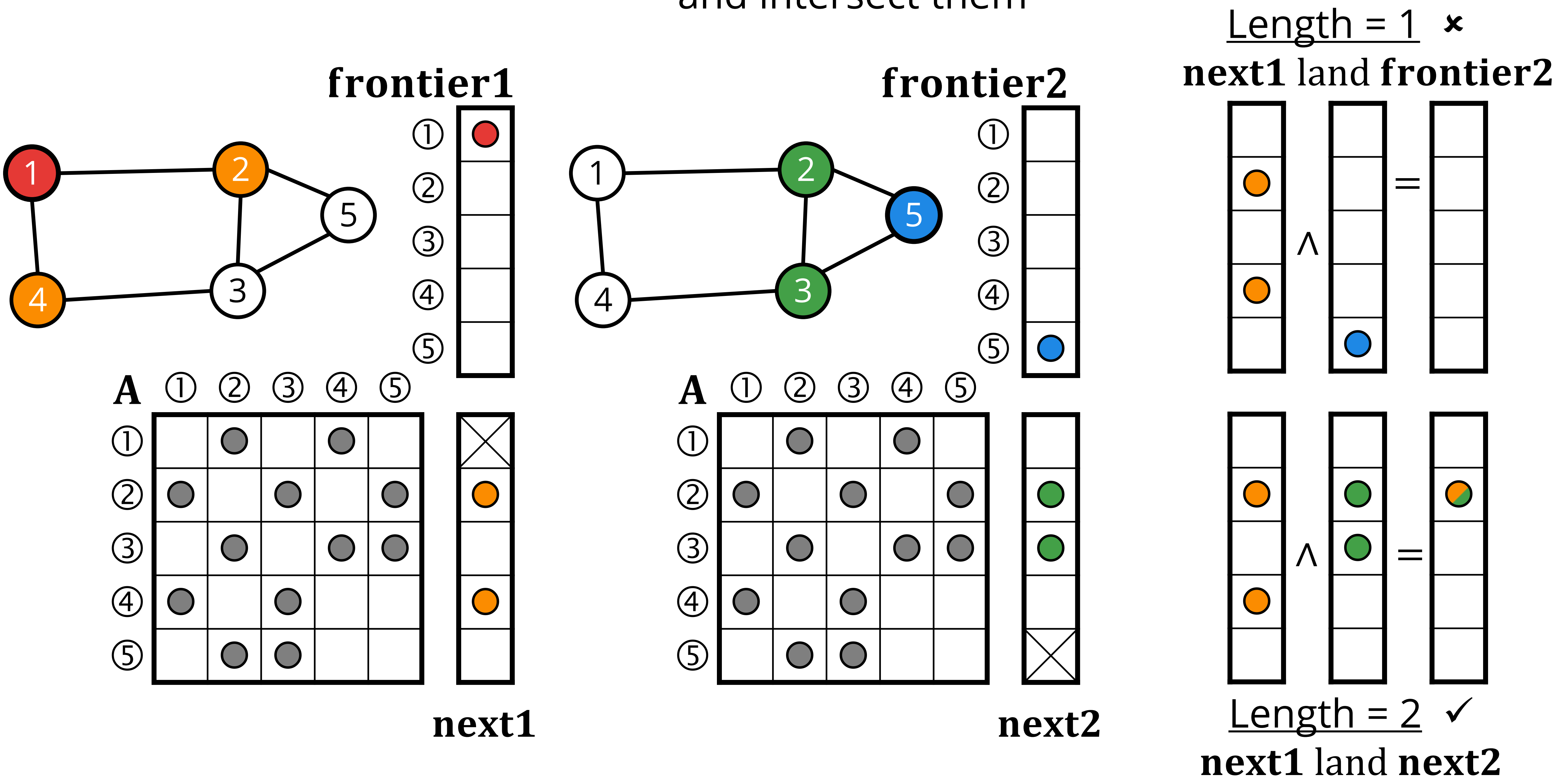
$\mathbf{Seen}\ \text{bor}\ \mathbf{Next}$

Case study: SIGMOD 2014 Contest

Bidirectional BFS

BIDIRECTIONAL BFS

Advance frontiers alternately
and intersect them



Length = 1 ✗

next1 land **frontier2**

frontier1

frontier2

A	①	②	③	④	⑤
①		●		●	
②	●		●		●
③		●		●	●
④	●		●		
⑤		●	●		

A	①	②	③	④	⑤
①		●		●	
②	●		●		●
③		●		●	●
④	●		●		
⑤		●	●		

next1

next2

Length = 2 ✓

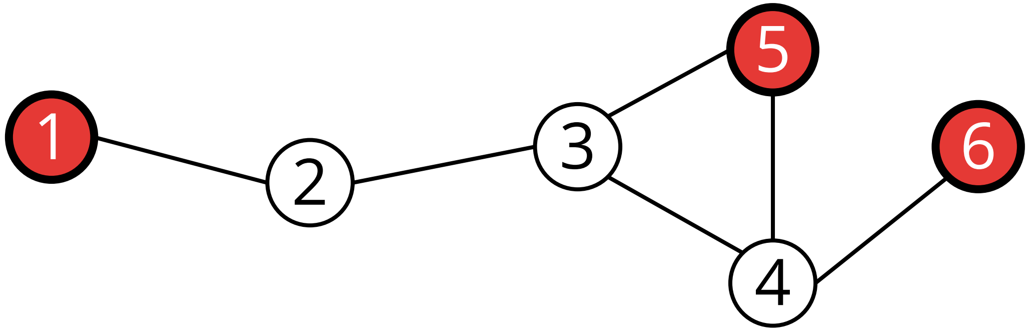
next1 land **next2**

Case study: SIGMOD 2014 Contest

Bidirectional MSBFS

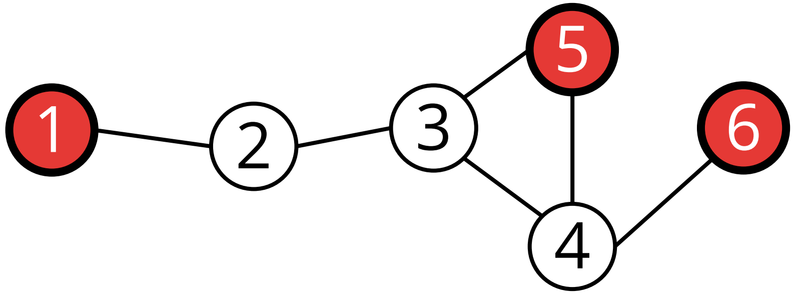
BIDIRECTIONAL MSBFS ALGORITHM

- **Pairwise reachability problem:**
  From a given set of k vertices, which pairs of vertices are reachable from each other with at most h hops?
- **Naïve solution:**
  Run a k-source MSBFS for h steps and check reachability.
  The frontiers get large as they grow exponentially.
- **Better solution:**
  Advance all frontiers simultaneously for $\lceil h/2 \rceil$ iterations.



BIDIRECTIONAL MSBFS

Seen[1]: reachability with ≤ 1 hops



A	①	②	③	④	⑤	⑥
①		●				
②	●		●			
③		●		●	●	
④			●		●	●
⑤			●	●		
⑥				●		

S[0]	①	②	③	④	⑤	⑥
①	●					
②						
③						
④						
⑤					●	
⑥						●

F	①	②	③	④	⑤	⑥
①	●					
②						
③						
④						
⑤					●	
⑥						●

☒	●				
		●	●	☒	
			●		☒

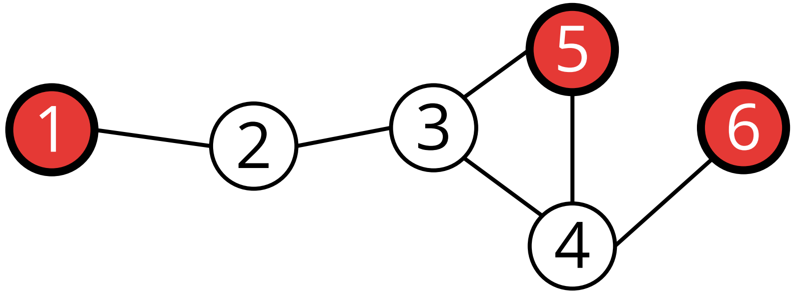
Next[1]

①	●	●				
②						
③						
④						
⑤			●	●	●	
⑥				●		●

Seen[1]

BIDIRECTIONAL MSBFS

Seen[2]: reachability with ≤ 2 hops



A	①	②	③	④	⑤	⑥
①		●				
②	●		●			
③		●		●	●	
④			●		●	●
⑤			●	●		
⑥				●		

S[1]	①	②	③	④	⑤	⑥
①	🔴	🟠				
②						
③						
④						
⑤			🟠	🟠	🔴	
⑥				🟠		🔴

F	①	②	③	④	⑤	⑥
①		🟠				
②						
③						
④						
⑤			🟠	🟠		
⑥				🟠		

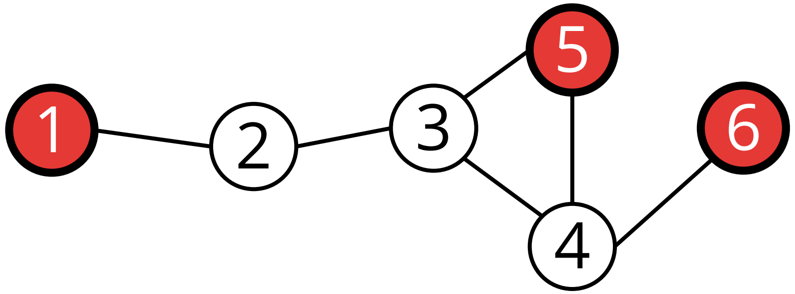
☒	☒	🟣			
		☒	☒	☒	
		🟣	☒		☒

Next[2]⟨**Seen**[1]⟩

	①	②	③	④	⑤	⑥
①	🔴	🟠	🟣			
②						
③						
④						
⑤			🟠	🟠	🔴	
⑥			🟣	🟠		🔴

Seen[2]

BIDIRECTIONAL MSBFS: PATHS OF LENGTH ≤ 4



$\mathbf{Seen}[2]^\mathsf{T}$

	①	②	③	④	⑤	⑥
①	🔴					
②	🟠					
③	🟣				🟠	🟣
④					🟠	🟠
⑤					🔴	
⑥						🔴

To get paths of at most 4 hops, we compute

$\mathbf{Seen}[2]$

	①	②	③	④	⑤	⑥
①	🔴	🟠	🟣			
②						
③						
④						
⑤			🟠	🟠	🔴	
⑥			🟣	🟠		🔴

From vertex 5, we could get to these vertices with ≤ 2 hops

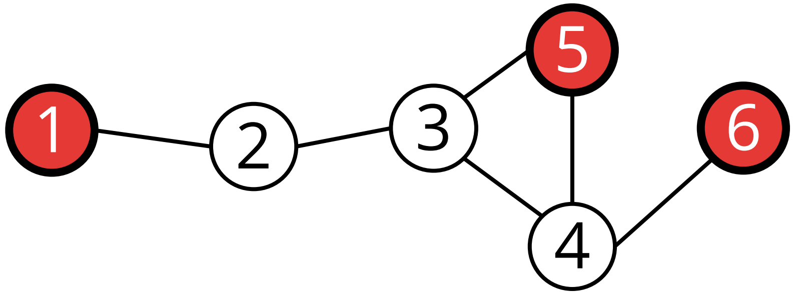
$\mathbf{Seen}[2]$ any.pair $\mathbf{Seen}[2]^\mathsf{T}$

☒				⚫	⚫
	☒				
		☒			
			☒		
⚫				☒	⚫
⚫				⚫	☒

Here, we found paths between all pairs:

- from ① to ⑤,
- from ① to ⑥,
- from ⑤ to ⑥.

BIDIRECTIONAL MSBFS: PATHS OF LENGTH = 3



Next$[2]^T$

	①	②	③	④	⑤	⑥
①						
②						
③	●					●
④						
⑤						
⑥						

From vertex 1, we could get to this vertex with 2 hops

To get exactly 3-length paths we compute

Next$[1]$

	①	②	③	④	⑤	⑥
①		●				
②						
③						
④						
⑤			●	●		
⑥				●		

From vertex 5, we could get to these vertices with 1 hop

Next$[1]$ any.pair **Next**$[2]^T$

☒					
	☒				
		☒			
			☒		
●				☒	●
					☒

We found two 3-length paths:

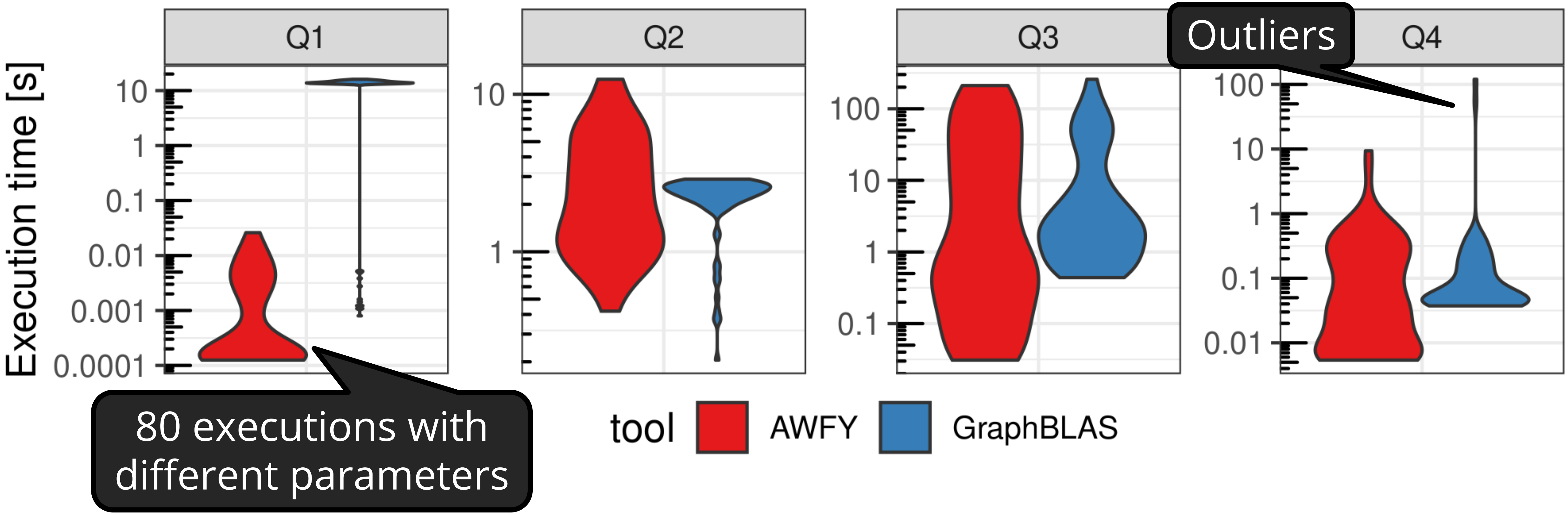
- from ⑤ to ①
- from ⑤ to ⑥.

Case study: SIGMOD 2014 Contest

Results

BENCHMARK RESULTS

- The top solution of AWFY vs. SuiteSparse:GraphBLAS v3.3.3
- AWFY's solution uses SIMD instructions → difficult to port
- GraphBLAS load times are slow (see details in paper)



Outliers

80 executions with different parameters

SUMMARY

- An interesting case study, see technical report

- GraphBLAS can capture mixed workloads
  - Induced subgraph computations are simple to express
  - Algorithms are concise, bitwise optimizations can be adopted
  - Performance is *sometimes* on par with specialized solutions

- Future optimizations
  - Q1: filter the induced subgraph on-the-fly
  - Q4: use more sophisticated unweighted shortest path algorithms

sigmod2014-contest-graphblas

M. Elekes et al., *A GraphBLAS solution to the SIGMOD 2014 Programming Contest using multi-source BFS,* HPEC 2020

ACKNOWLEDGEMENTS

- Thanks for feedback on these slides to Tim Mattson, Tim Davis, Jeremy Kepner, John Gilbert, Aydin Buluc, Tamás Nyíri, Gyula Katona, Dávid Sándor, Attila Nagy, János Benjamin Antal, Márton Elekes, Florentin Dörre.
- Thanks to the LDBC Graphalytics task force for creating the benchmark and assisting in the measurements.
- Thanks to the Master’s students at BME for exploring the area of GraphBLAS: Bálint Hegyi, Petra Várhegyi, Lehel Boér

Notes

ABOUT THIS PRESENTATION

- This presentation is intended to serve as an introduction to semiring-based graph processing and the GraphBLAS.
- Common graph algorithms (BFS, shortest path, PageRank, etc.) are used to demonstrate the features of GraphBLAS. Many of the algorithms presented are part of the LAGraph library.
- The presentation complements existing technical talks on GraphBLAS and can form a basis of anything from a short 20min overview to 2×90min lectures on the GraphBLAS.
- The slides contain numerous references to papers and talks.
- There are detailed textual explanations on some slides to make them usable as a learning material.

TECHNICAL DETAILS

- The slides were created with PowerPoint 2016 using the Open Sans and DejaVu Sans Mono font families (embedded in the presentation).
- The mathematical expressions are typeset with PowerPoint's built-in Equation Editor.
- The circled numbers (denoting graph vertices) are rendered using the standard Wingdings font.
- The text is written in Oxford English.
- The icons for referenced papers and talks are clickable and will lead to an open-access / preprint / author's copy version of the referred work (if such copy exists). The icons depict the first page of the cited document.