Introduction to GraphBLAS-based Graph Analytics

Gábor Szárnyas

Hungarian Academy of Sciences

GRAPH PROCESSING

- **Graph use cases:** Focus on the connections between entities rather than attribute values.
  - Social network — friend recommendation
  - Financial transactions — fraud detection
  - Logistics — routing, shortest paths
  - HW/SW models — validation
  - runtime models — verification
  - ontologies — inferencing

  } ftsrg

- Graph processing covers a **broad set of problems with a rich family of algorithms**. Is there a generic framework?

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

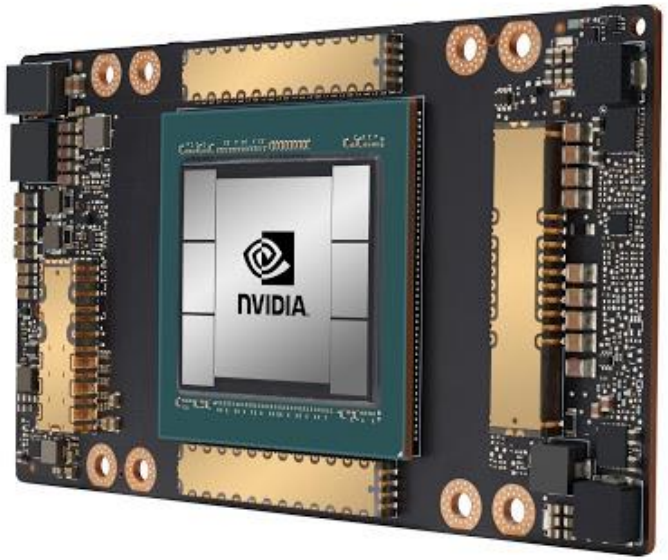
Graph algorithms have
“a high communication to computation ratio” [ICCS’15].
Computing is cheap, finding *what to compute* is expensive

Speedup with a CPU that has better arithmetic performance:

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

What can we do about this?

HETEROGENEOUS COMPUTER ARCHITECTURE



[1]

DATA PROCESSING PIPELINES

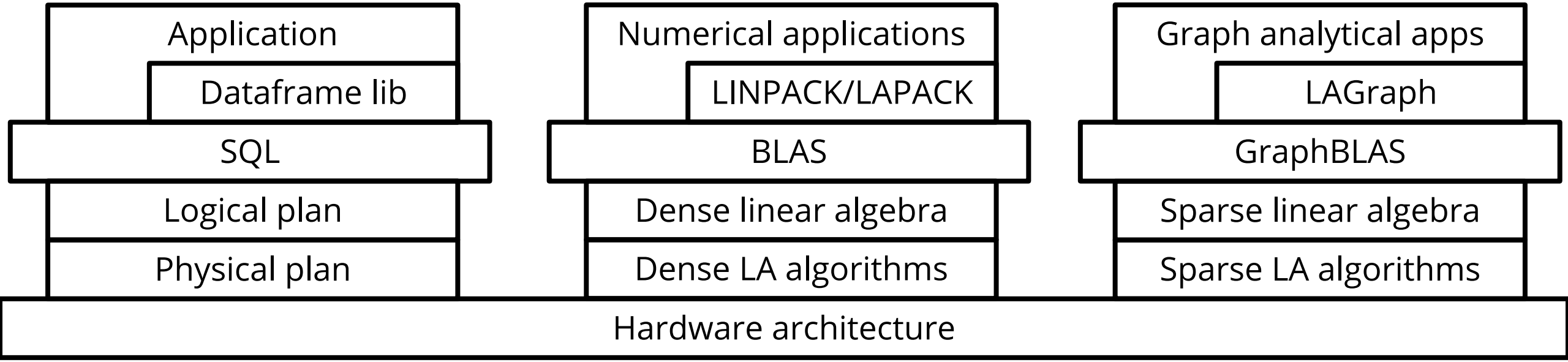
Recurring problems in CS often form a “cluster” that can be tackled efficiently using the right tools.

Tabular data processing	relational databases	*relational algebra*
Machine learning	ML frameworks	*linear algebra*
String manipulation	Unix tools & pipeline	*finite automata algebras*

These algebras have proven to be composable, portable, and expressive. They have well-optimized implementations.

Catch: the underlying algebras are always the most intuitive ones.

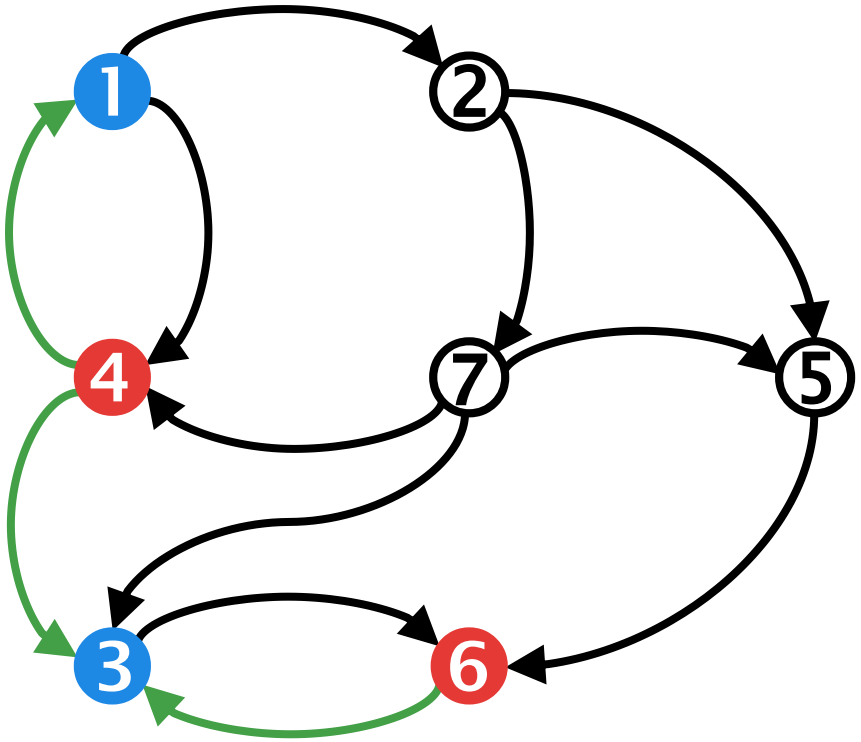
ABSTRACTION LAYERS



The GraphBLAS standard

LINEAR ALGEBRA-BASED GRAPH PROCESSING

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



frontier

	①	②	③	④	⑤	⑥	⑦
f				1		1	

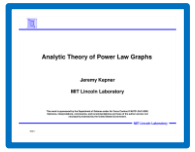
A	❶	②	❸	④	⑤	⑥	⑦
①		1		1			
②					1		1
③						1	
❹	1		1				
⑤						1	
❻			1				
⑦			1	1	1		

							fA
1		2					

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	$m + n \log n$	$m \log n$



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

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:** supported on x86, Arm (WIP), GPUs (WIP)
- **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:

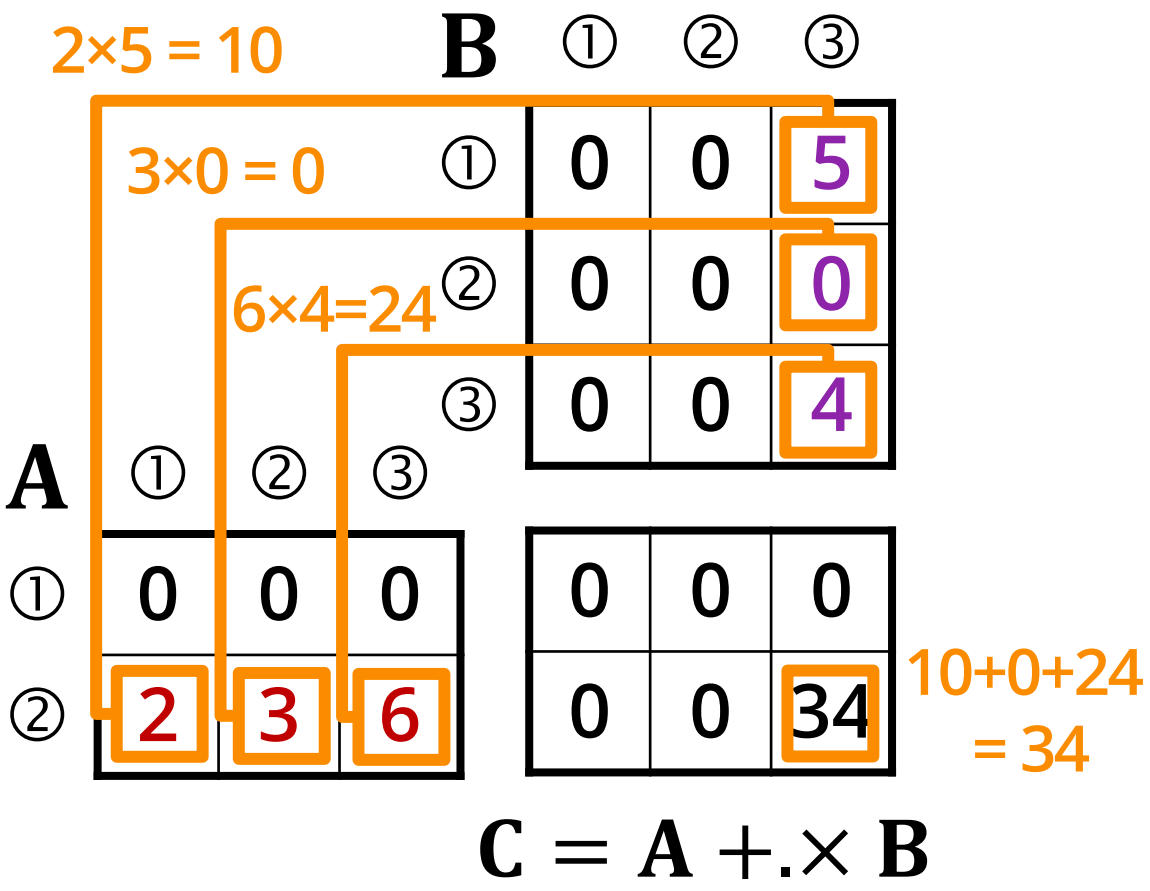
$$\mathbf{C} = \mathbf{AB}$$

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

Example:

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

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



SPARSE MATRIX MULTIPLICATION

Definition:

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

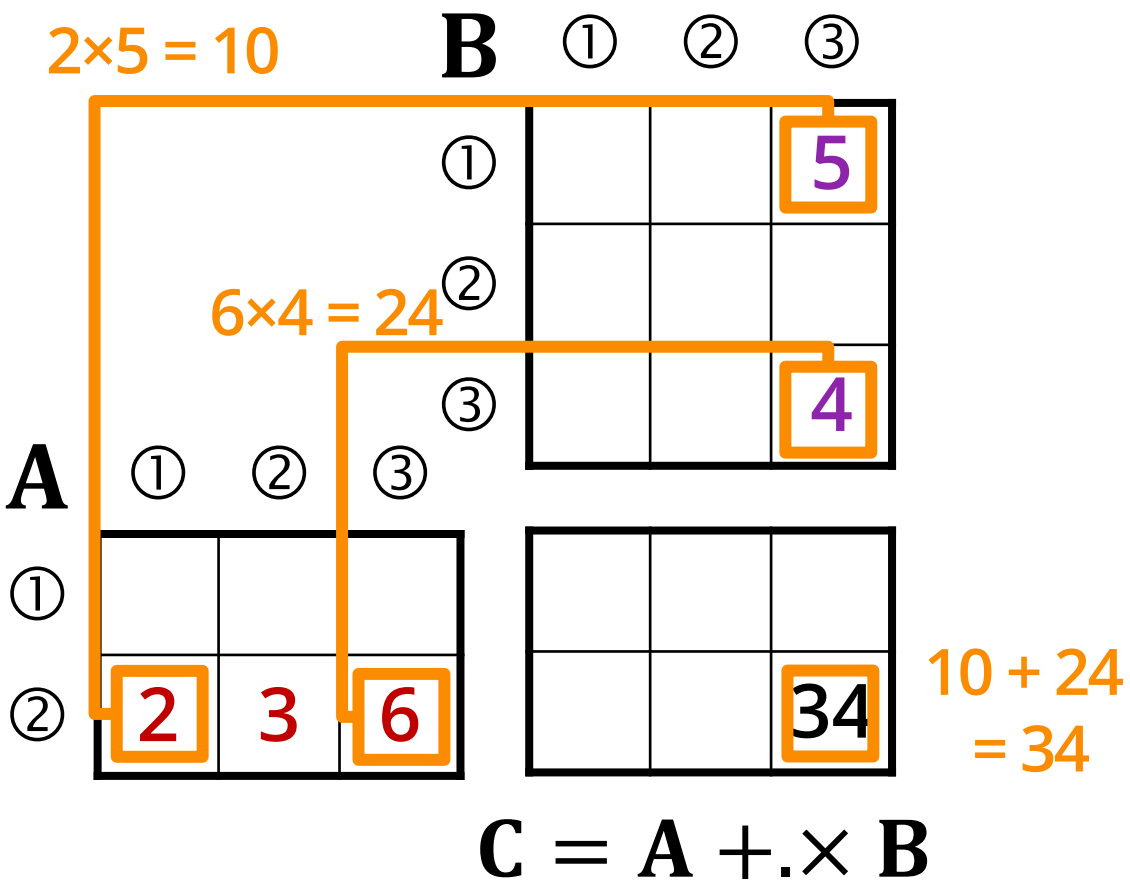
$$\mathbf{C}(i,j) = \bigoplus_{k \in \mathrm{ind}(\mathbf{A}(i,:)) \cap \mathrm{ind}(\mathbf{B}(:,j))} \mathbf{A}(i,k) \otimes \mathbf{B}(k,j)$$

Only evaluate the multiplication operator $\otimes$ for positions where there is a non-zero element in both $\mathbf{A}(i,k)$ and $\mathbf{B}(k,j)$.

Example:

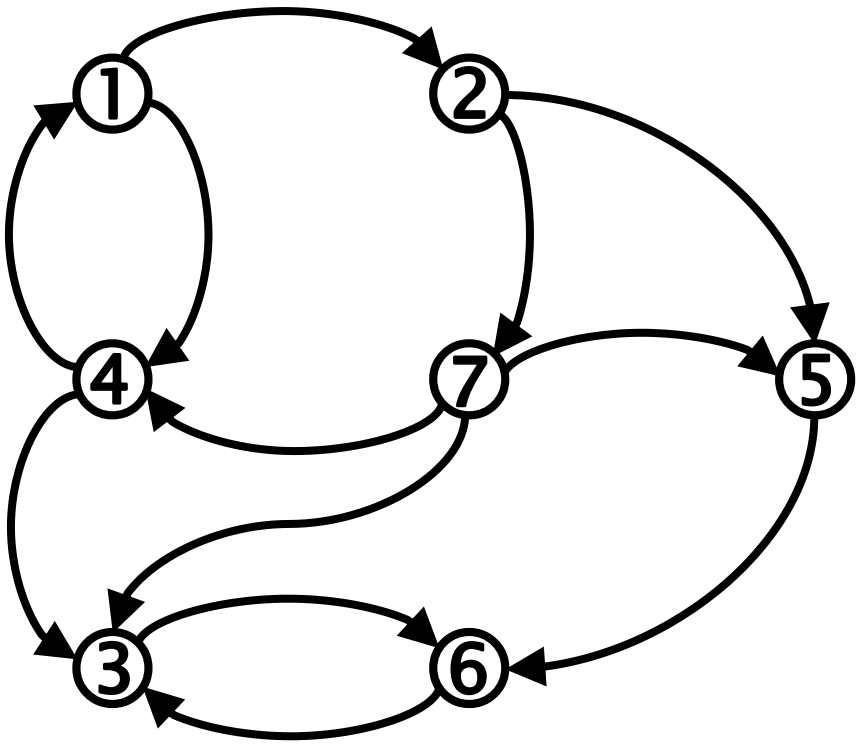
$$\mathbf{C}(2,3) = \mathbf{A}(2,1) \times \mathbf{B}(1,3) + \mathbf{A}(2,3) \times \mathbf{B}(3,3)$$

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



ADJACENCY MATRIX

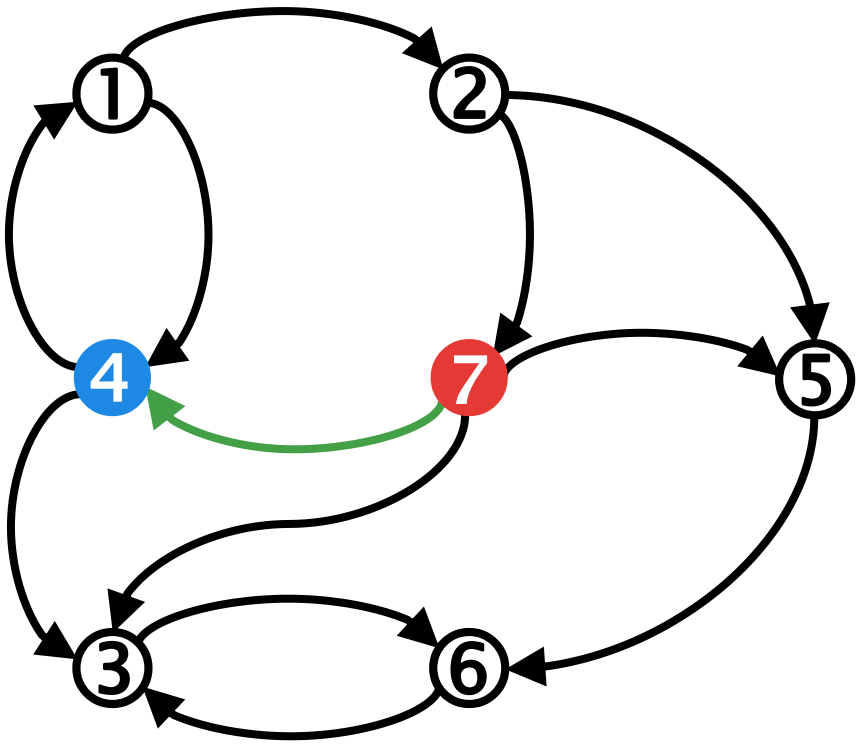
$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{if } (v_i, v_j) \notin E \end{cases}$$



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

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

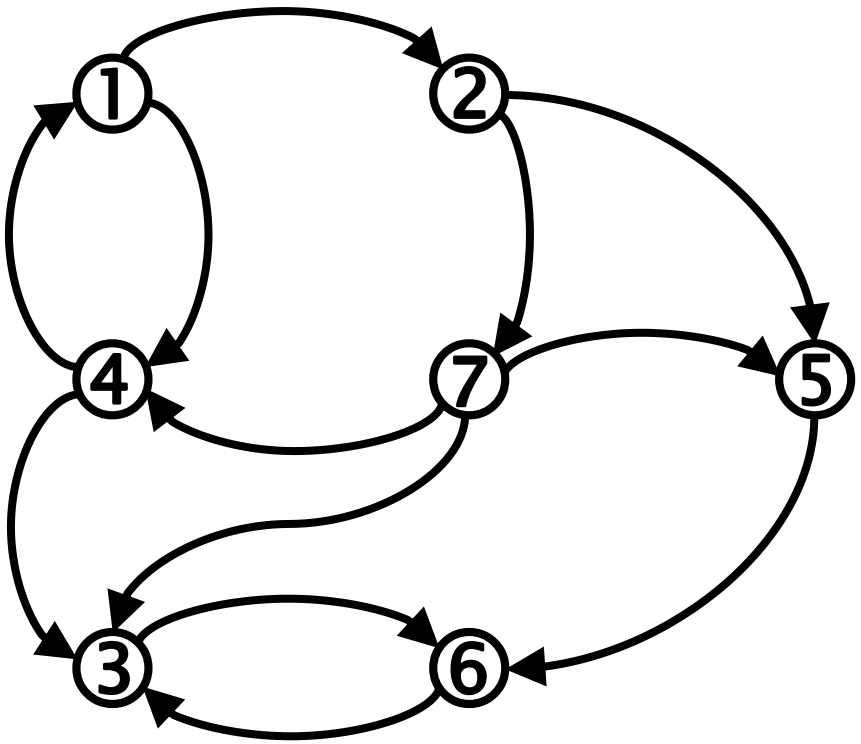


target

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

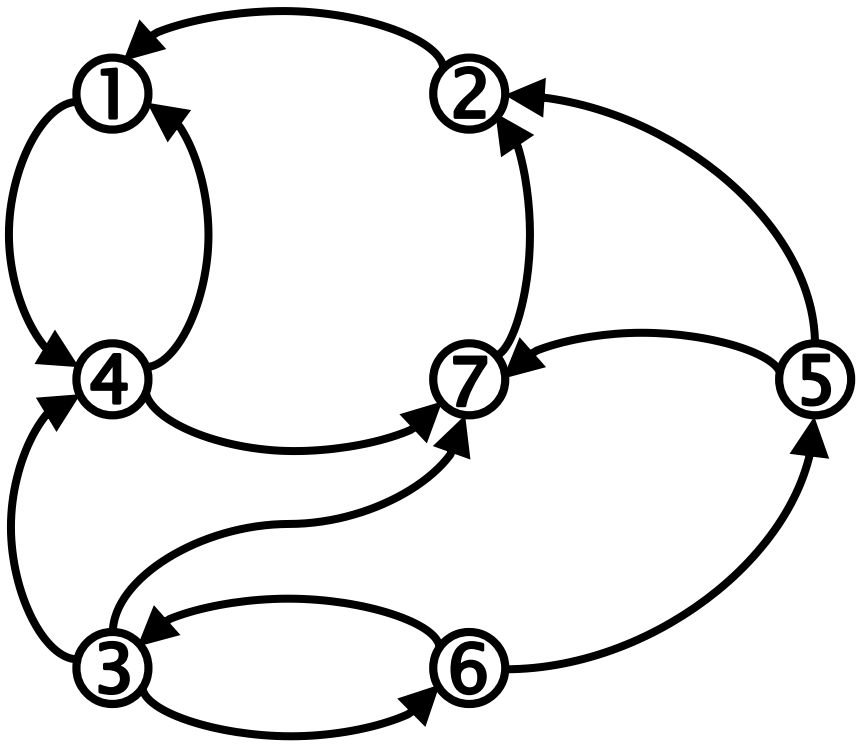
$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{if } (v_i, v_j) \notin E \end{cases}$$



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

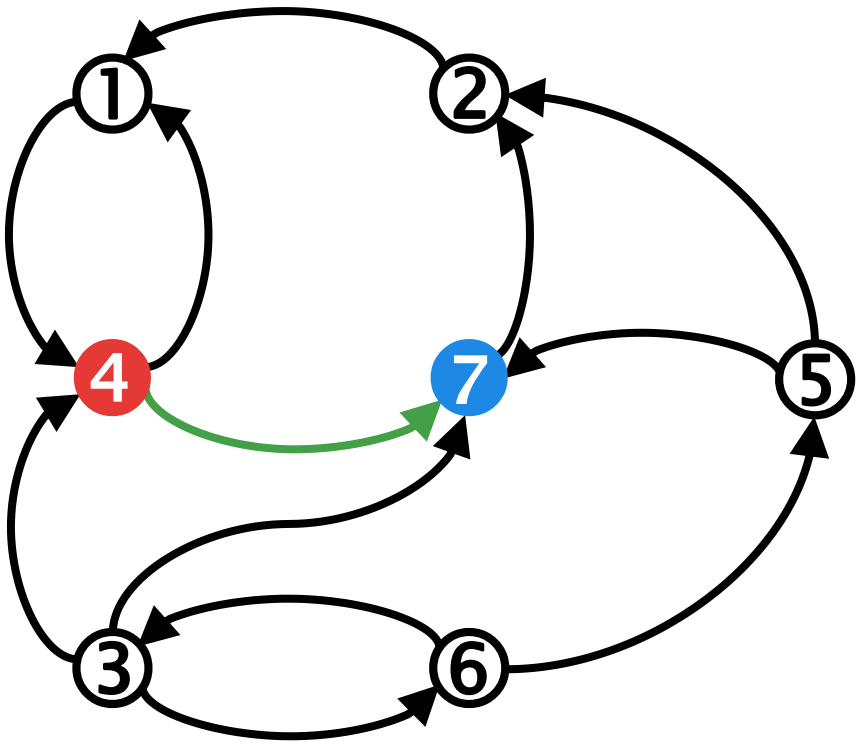
$$\mathbf{A}^{\top}_{ij} = \begin{cases} 1 & \text{if } (v_j, v_i) \in E \\ 0 & \text{if } (v_j, v_i) \notin E \end{cases}$$



$\mathbf{A}^{\top}$	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

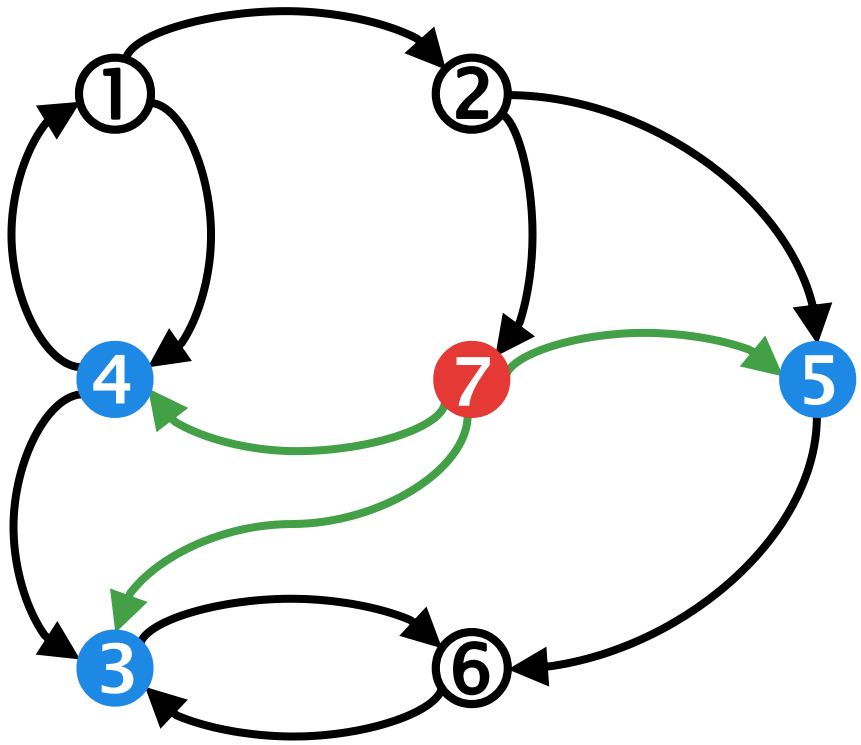
$$\mathbf{A}^{\top}_{ij} = \begin{cases} 1 & \text{if } (v_j, v_i) \in E \\ 0 & \text{if } (v_j, v_i) \notin E \end{cases}$$



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

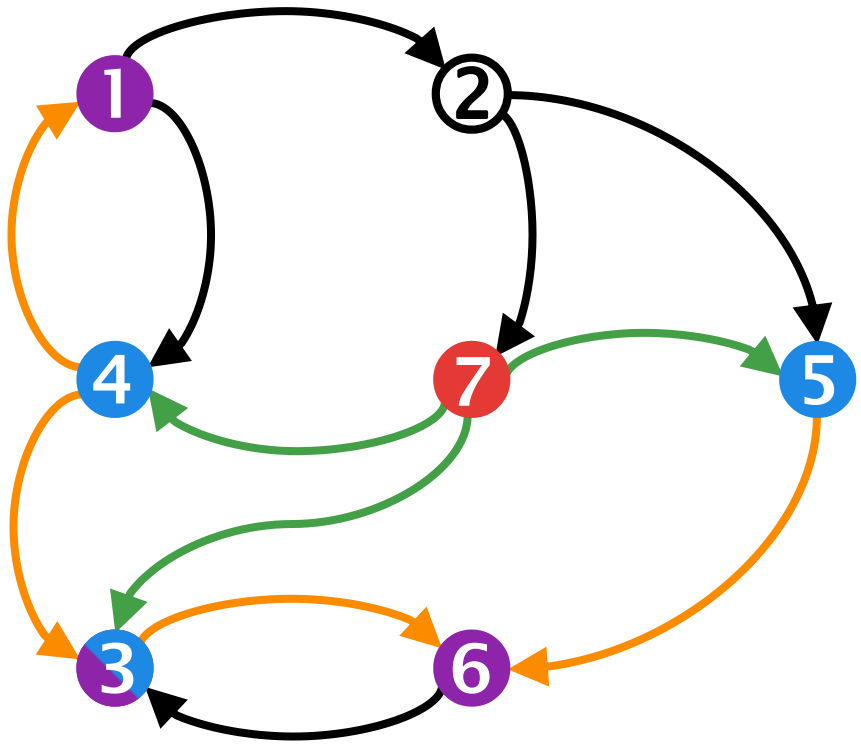
$\mathbf{A}$	1	2	3	4	5	6	7
1		1		1			
2					1		1
3						1	
4	1		1				
5						1	
6			1				
7			1	1	1		

$\mathbf{f}$	1	2	3	4	5	6	7
							1

	1	2	3	4	5	6	7
			1	1	1		

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

GRAPH TRAVERSAL WITH MATRIX MULTIPLICATION



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

$\mathbf{A}$	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		

$\mathbf{A}$	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		

	①	②	③	④	⑤	⑥	⑦
$\mathbf{f}$							1

①	②	③	④	⑤	⑥	⑦
		1	1	1		

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

①	②	③	④	⑤	⑥	⑦
1		1			2	

two hops: $\mathbf{f}\mathbf{A}^2$

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

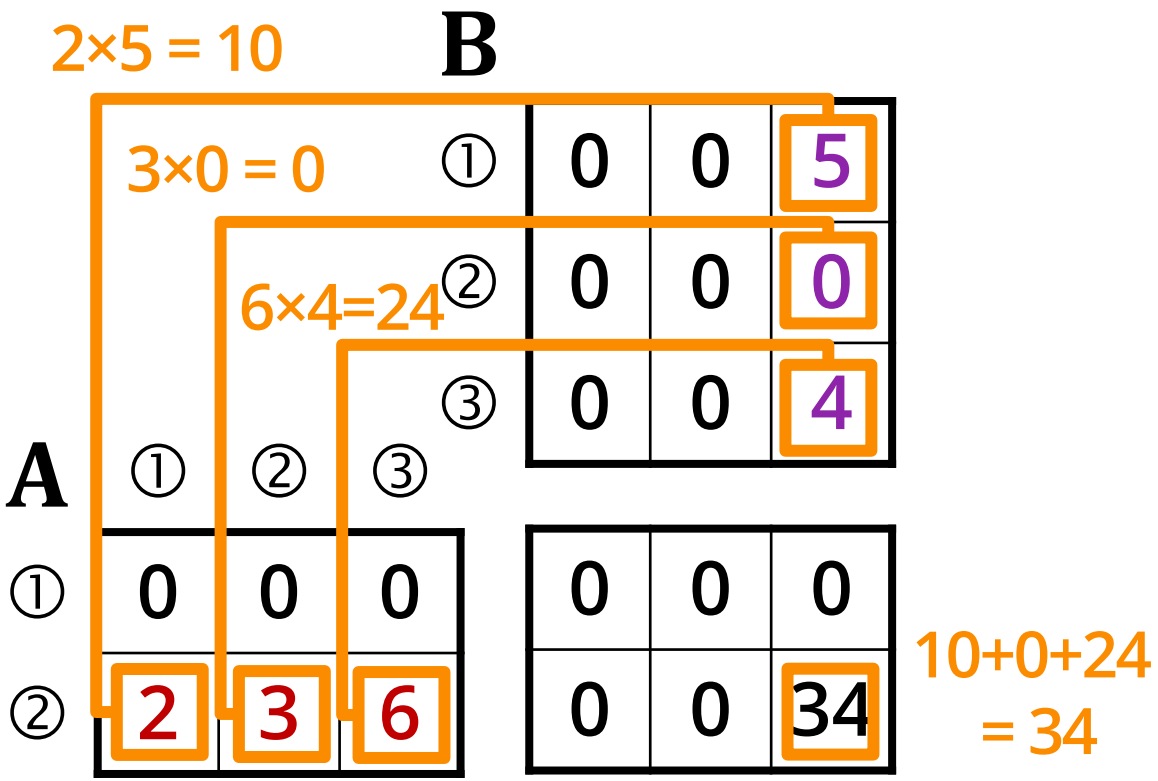
Multiplication on dense matrices

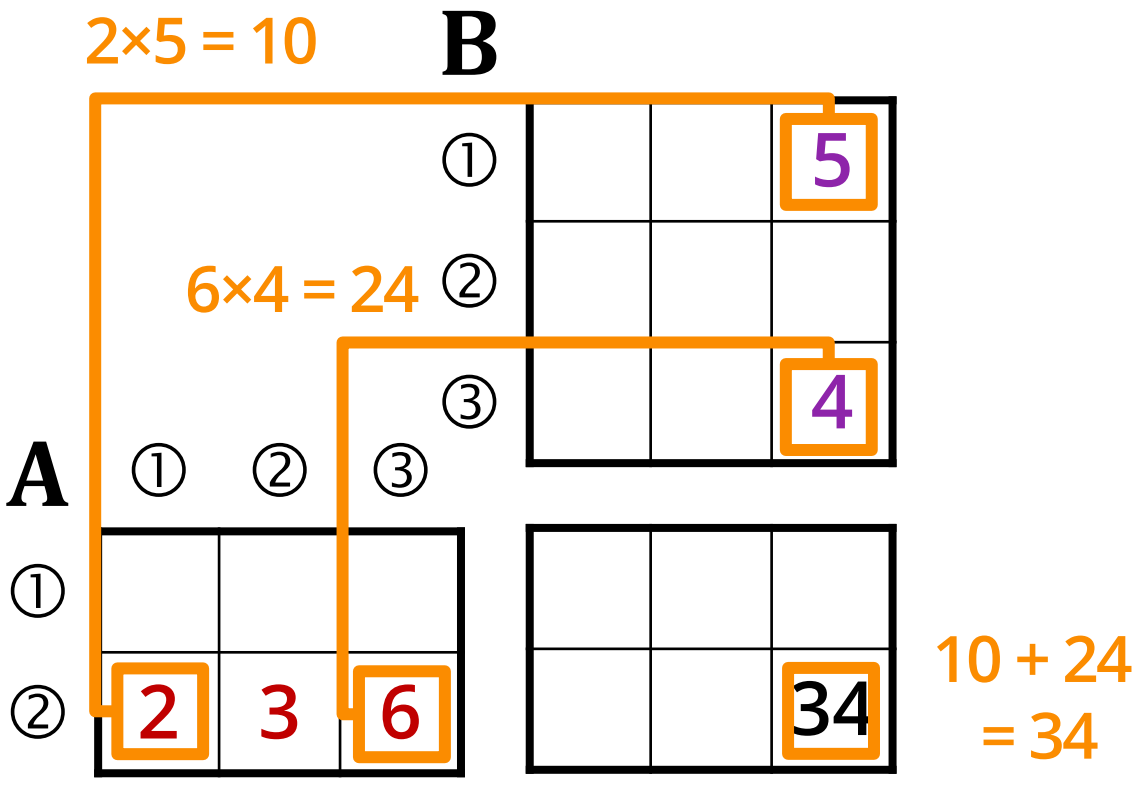
$$\mathbf{C}(i,j) = \bigoplus_{j} \mathbf{A}(i,k) \otimes \mathbf{B}(k,j)$$

Multiplication on sparse matrices

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

Example: $\mathbf{C} = \mathbf{A} +.\times \mathbf{B}$





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 $\forall a, b, c \in D$:
  - 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	set	$\oplus$	$\otimes$	0	graph semantics
any-pair	arbitrary	any	pair	F	connectivity
integer arithmetic	$a \in \mathbb{N}$	$+$	$\times$	0	number of paths
min-plus	$a \in \mathbb{R} \cup \{+\infty\}$	$\min$	$+$	$+\infty$	shortest path

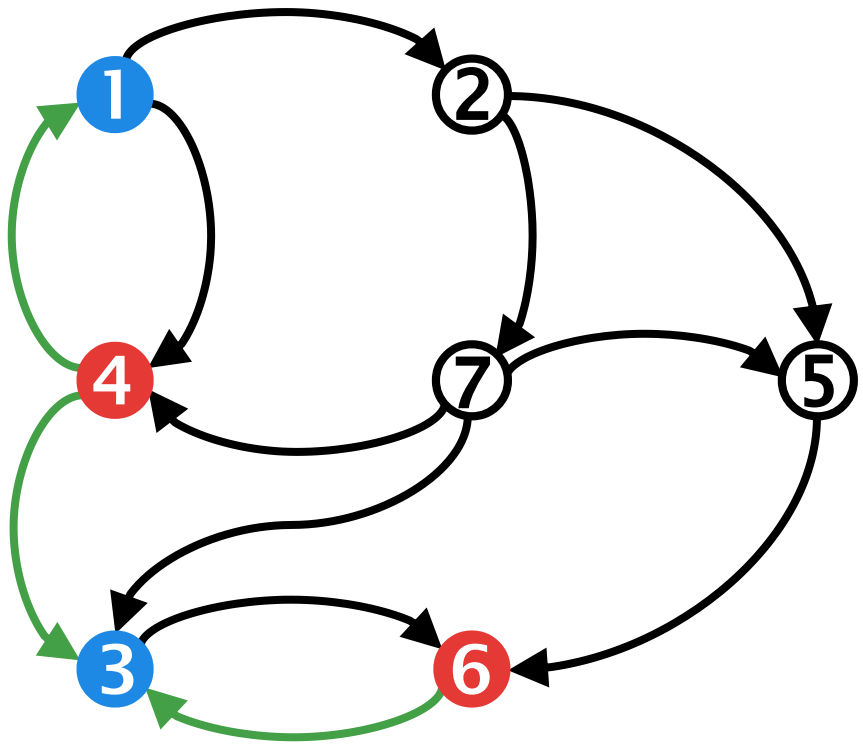
The default semiring is the conventional one:

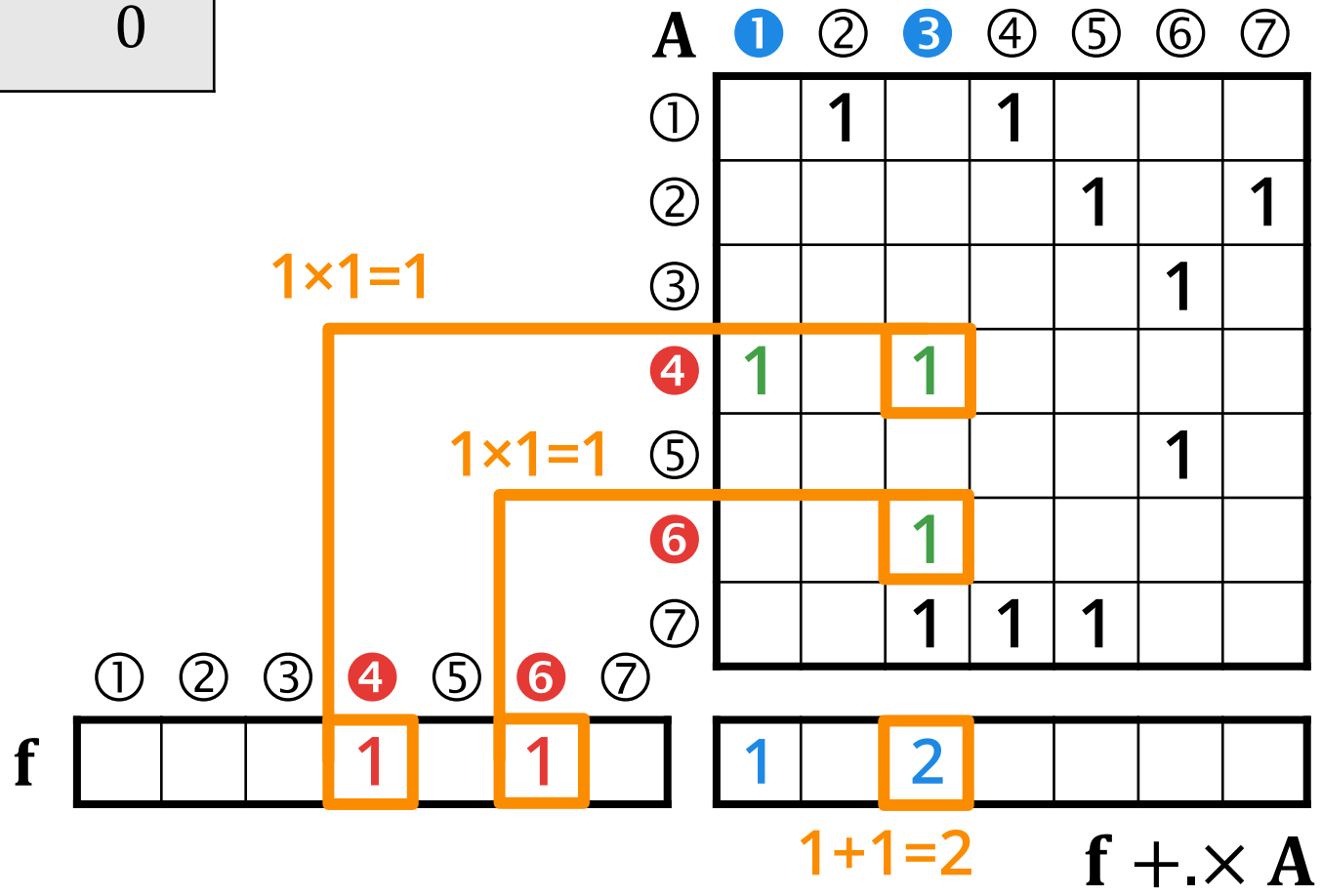
- Operator $\otimes$ defaults to floating point multiplication.
- Operator $\oplus$ defaults to floating point addition.

MATRIX-VECTOR MULTIPLICATION SEMANTICS

semiring	set	$\oplus$	$\otimes$	0
integer arithmetic	$a \in \mathbb{N}$	$+$	$\times$	0

Semantics: **number of paths**

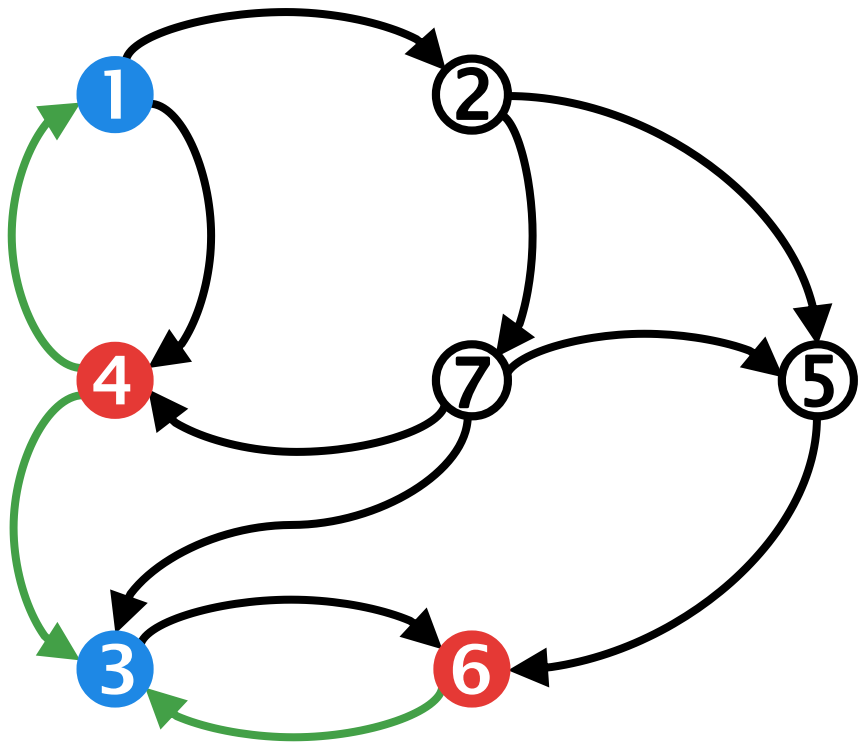


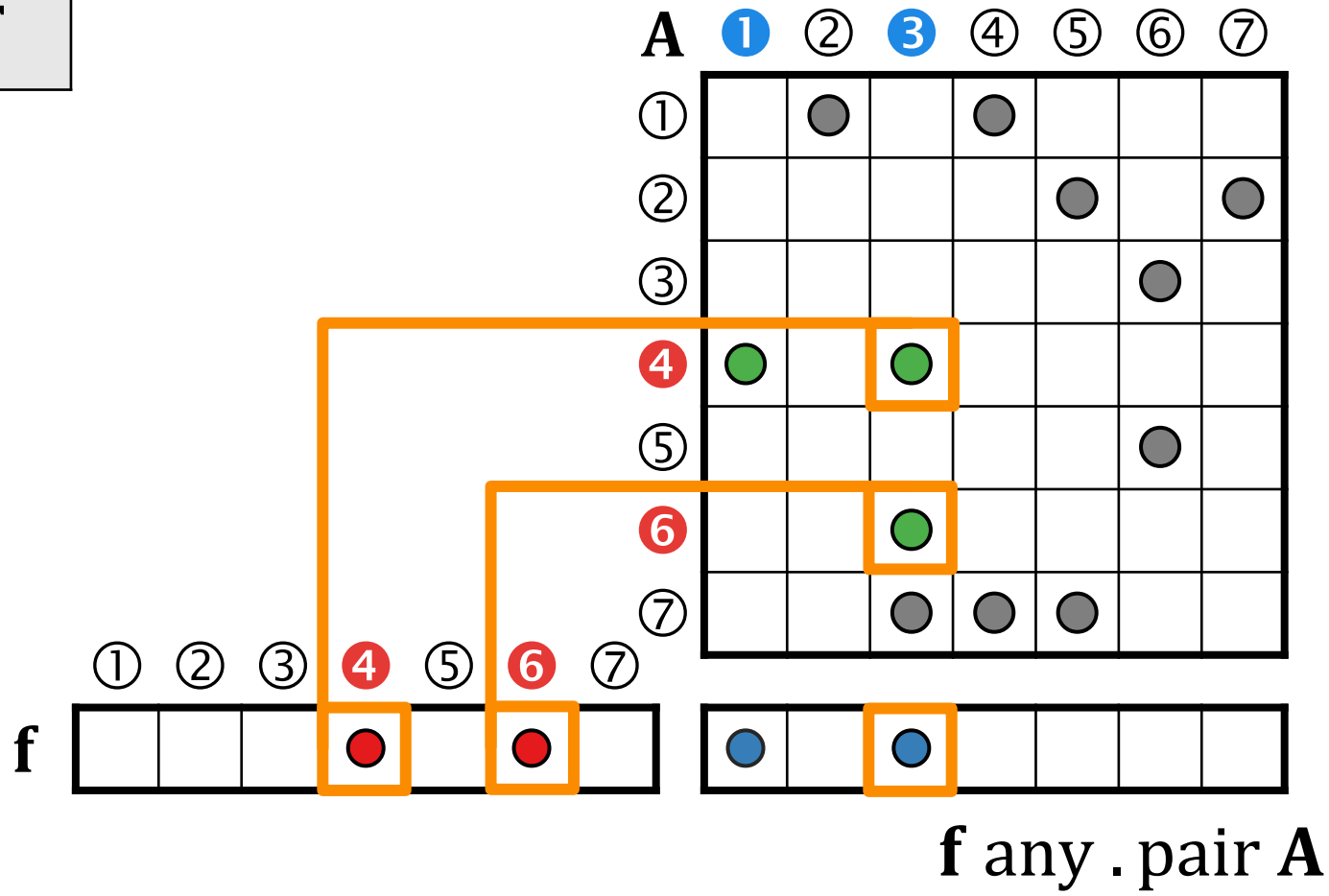


MATRIX-VECTOR MULTIPLICATION SEMANTICS

semiring	set	$\oplus$	$\otimes$	0
any-pair	arbitrary	any	pair	F

Semantics: **reachability**

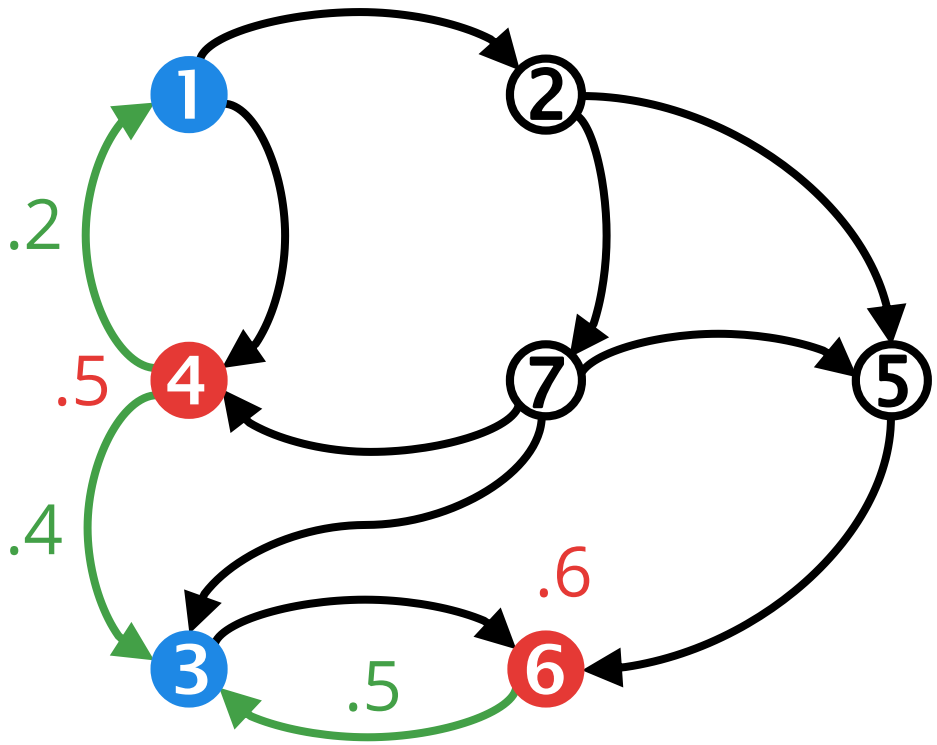


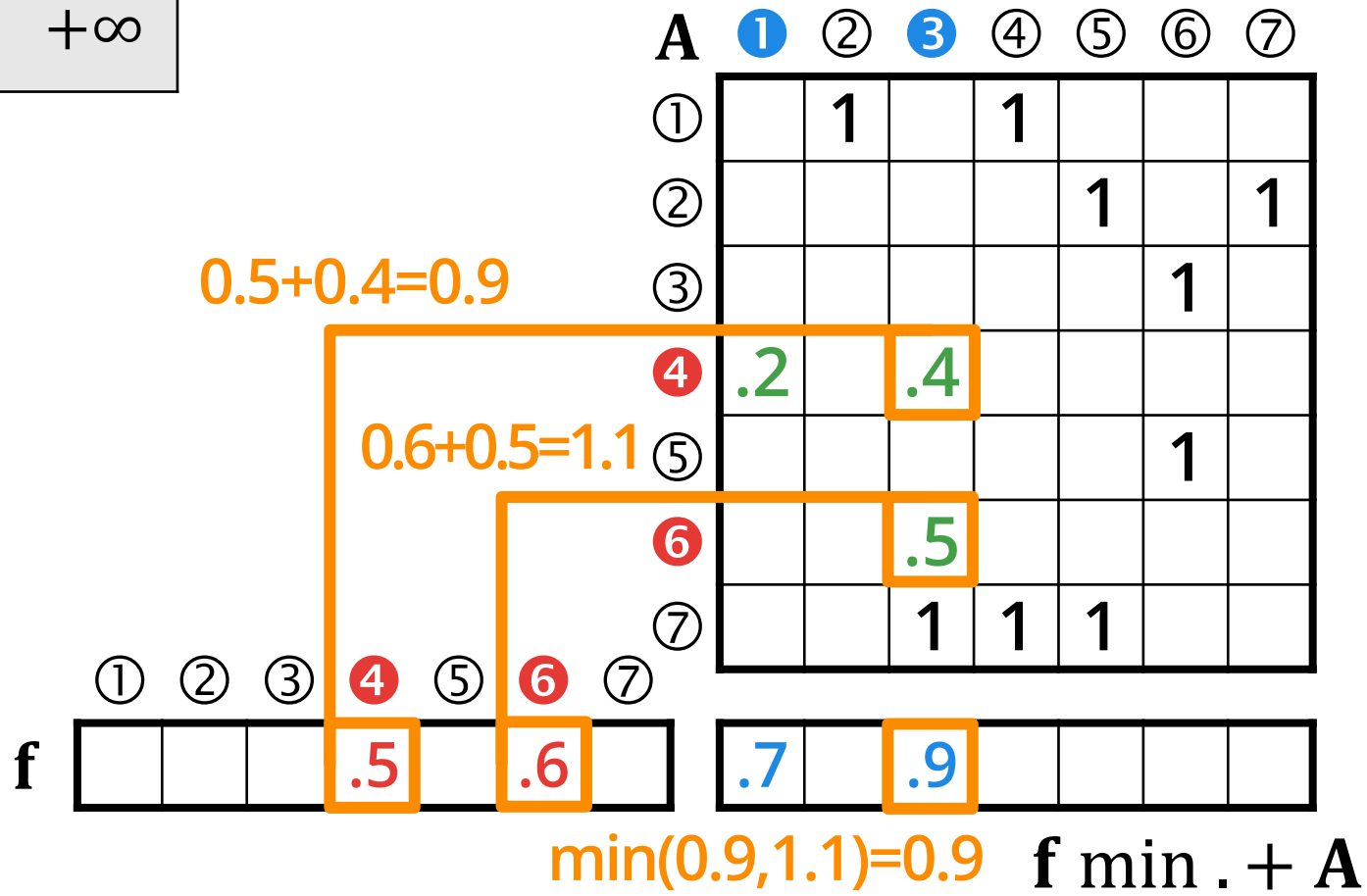


MATRIX-VECTOR MULTIPLICATION SEMANTICS

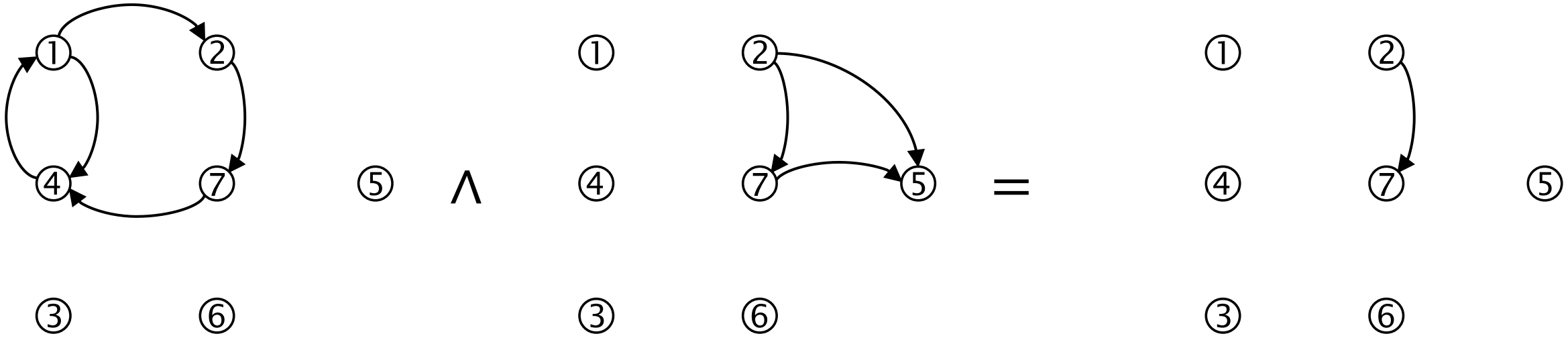
semiring	set	$\oplus$	$\otimes$	0
min-plus	$a \in \mathbb{R} \cup \{+\infty\}$	min	$+$	$+\infty$

Semantics: **shortest path**





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



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

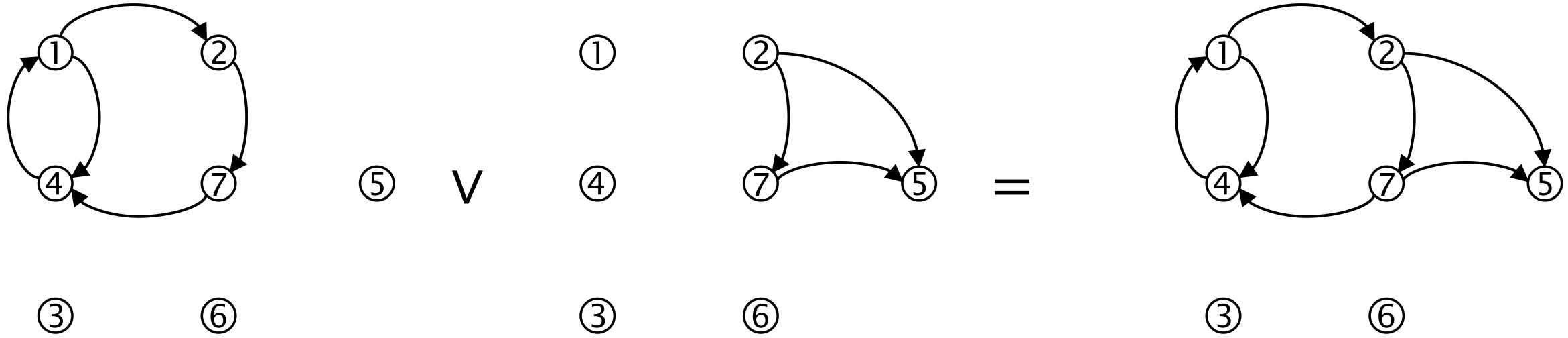
∧

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

=

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

ELEMENT-WISE ADDITION: $\mathbf{A} \vee \mathbf{B}$



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

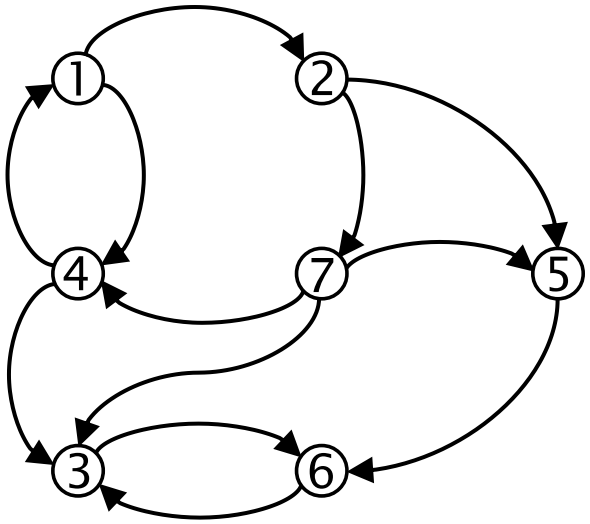
∨

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

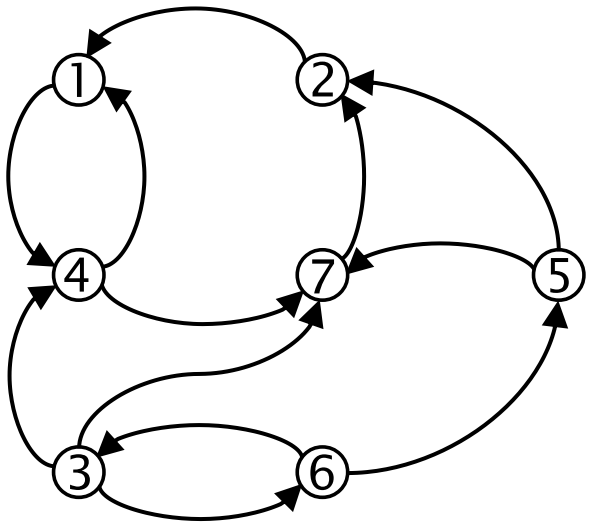
=

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

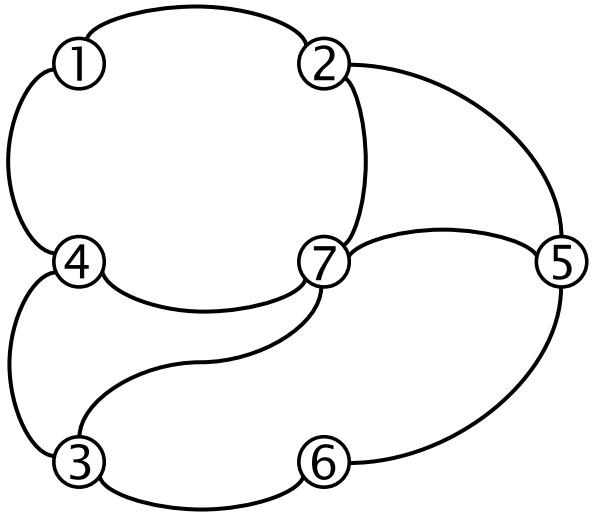
TURNING A GRAPH INTO UNDIRECTED: $\mathbf{A} \vee \mathbf{A}^T$



$\vee$



$=$



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

$\vee$

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

$=$

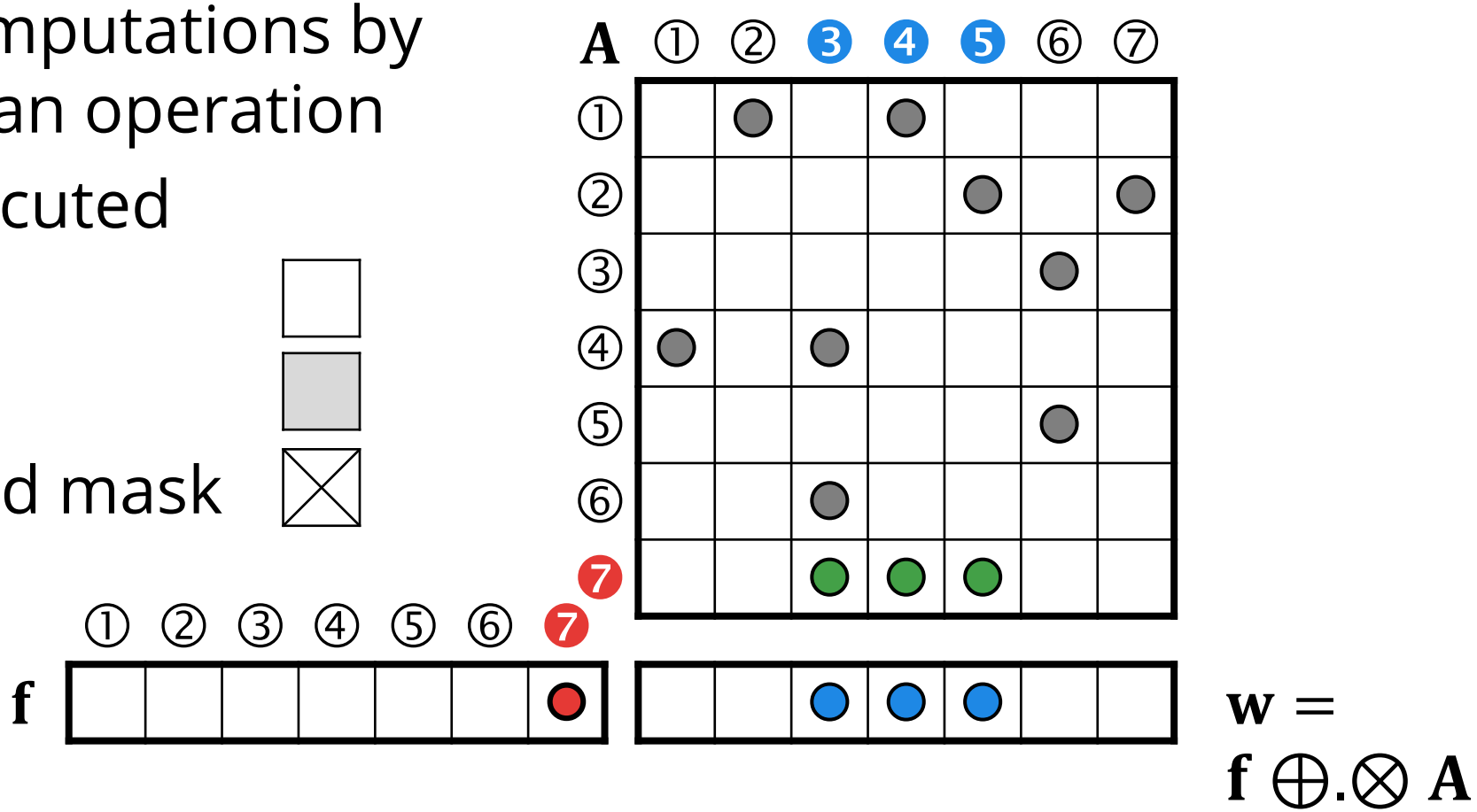
$\mathbf{A} \vee \mathbf{A}^T$	①	②	③	④	⑤	⑥	⑦
①		●		●			
②	●				●		●
③				●		●	●
④	●		●				●
⑤		●				●	●
⑥			●		●		
⑦		●	●	●	●		

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

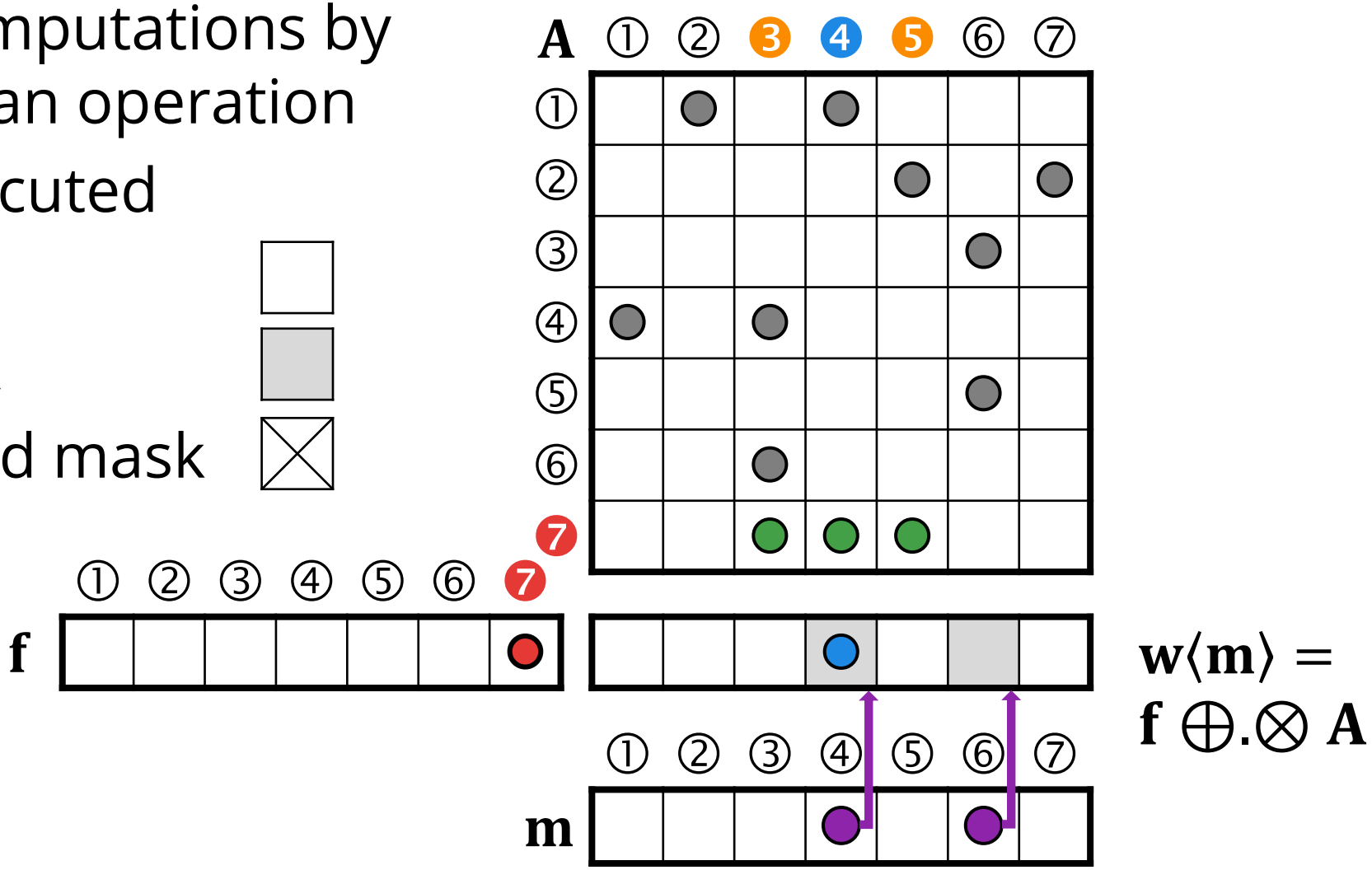


MASKING

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

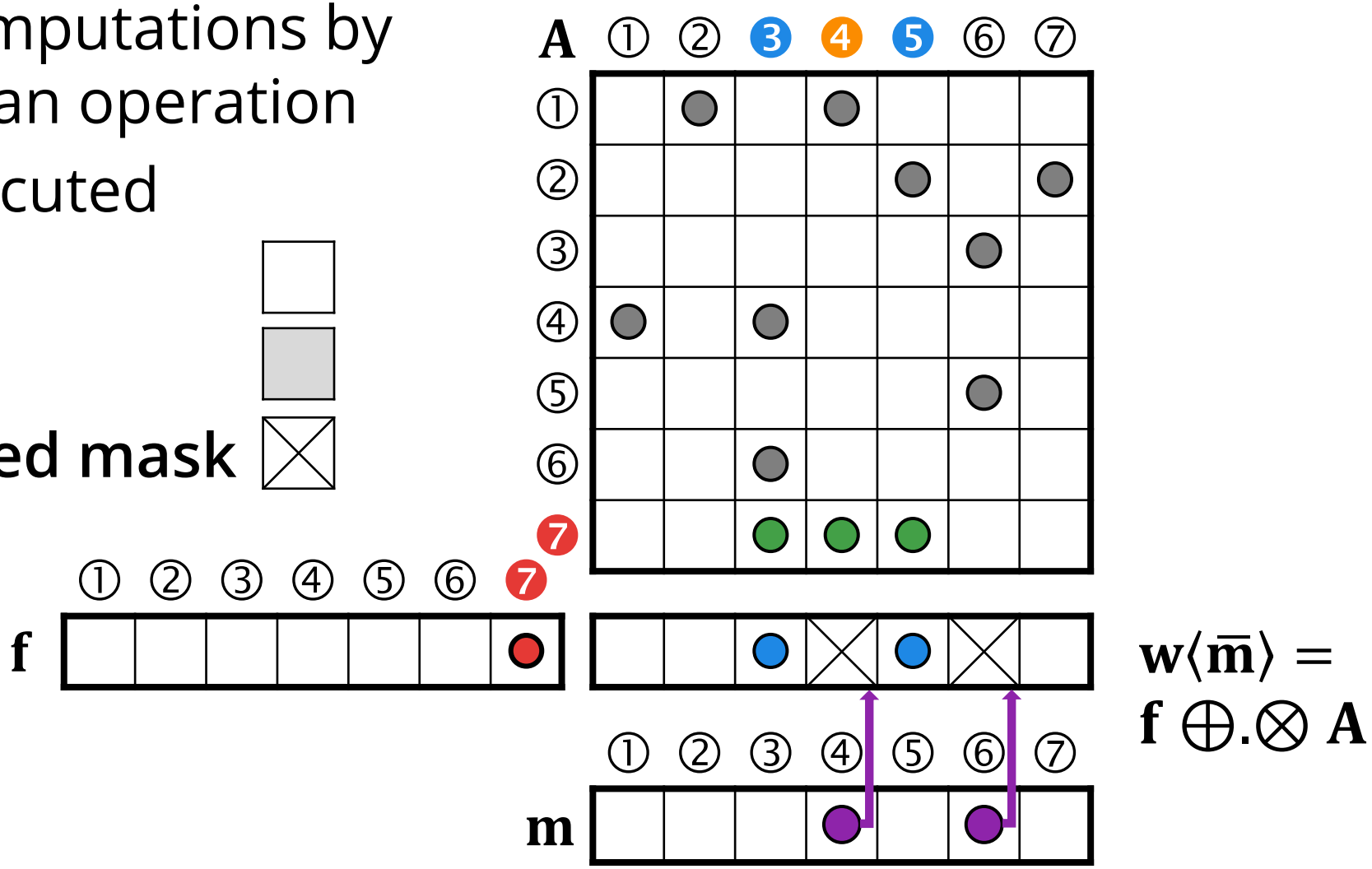


MASKING

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**



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
  - $^\top$ – transpose
  - $\oslash$ – element-wise division

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	$\mathbf{w}\langle \mathbf{m} \rangle = \left[\oplus_j \mathbf{A}(:, j)\right]$
	reduce to scalar	$s = \left[\oplus_{ij} \mathbf{A}(i, j)\right]$
$\mathbf{A}^\top$	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…)

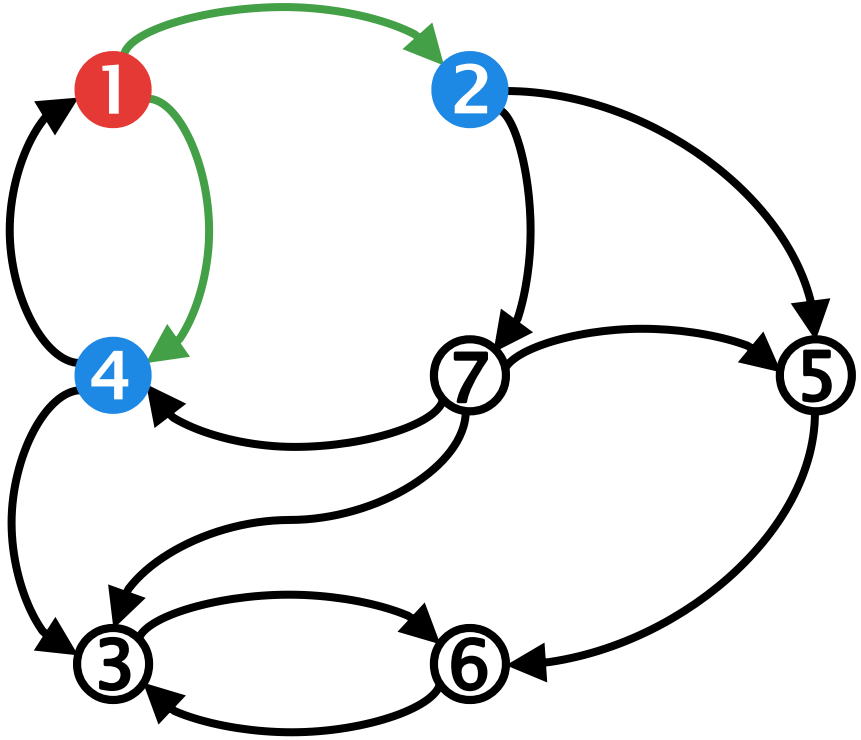
Graph algorithms in GraphBLAS

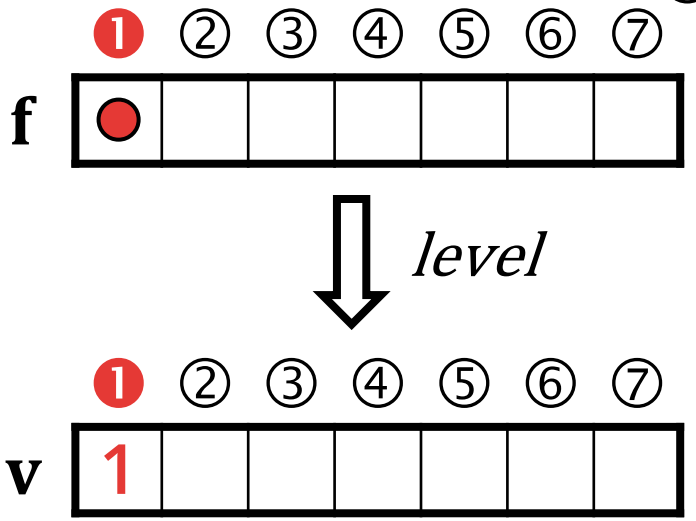
Breadth-first search / Levels

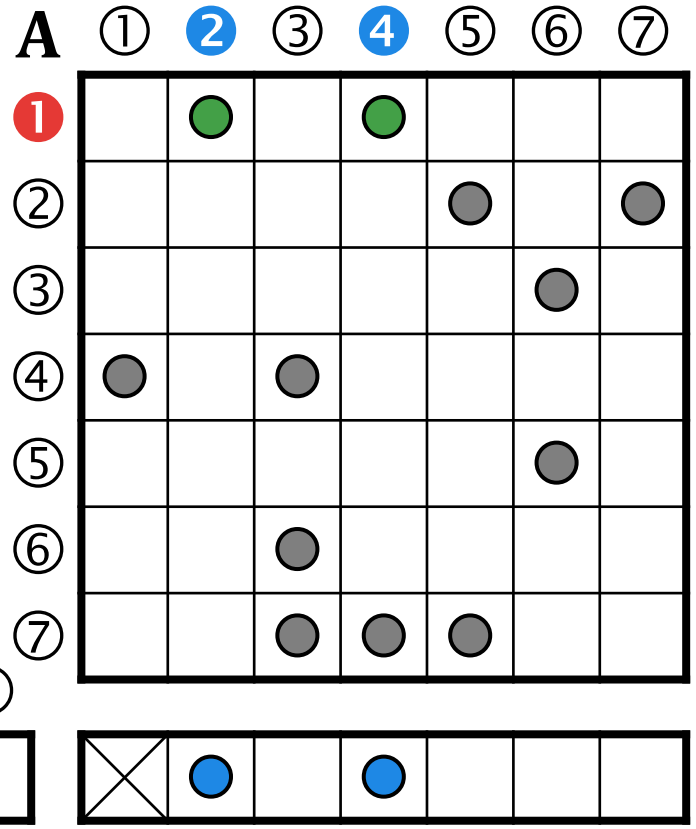
BFS – LEVELS

semiring	set	$\oplus$	$\otimes$	0
lor-land	$a \in \{\mathrm{F}, \mathrm{T}\}$	any	pair	F

$level = 1$





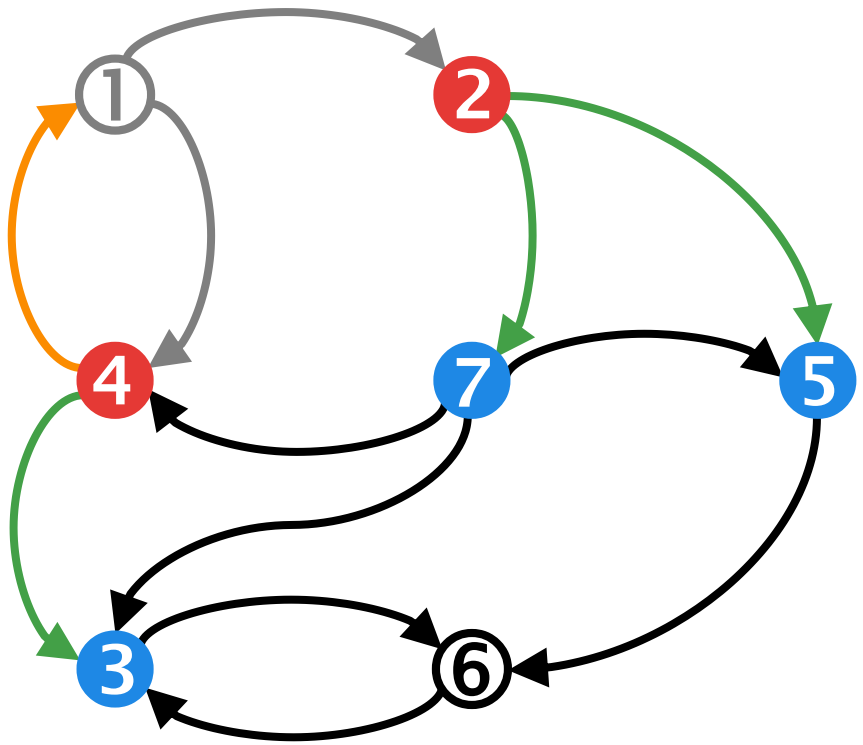


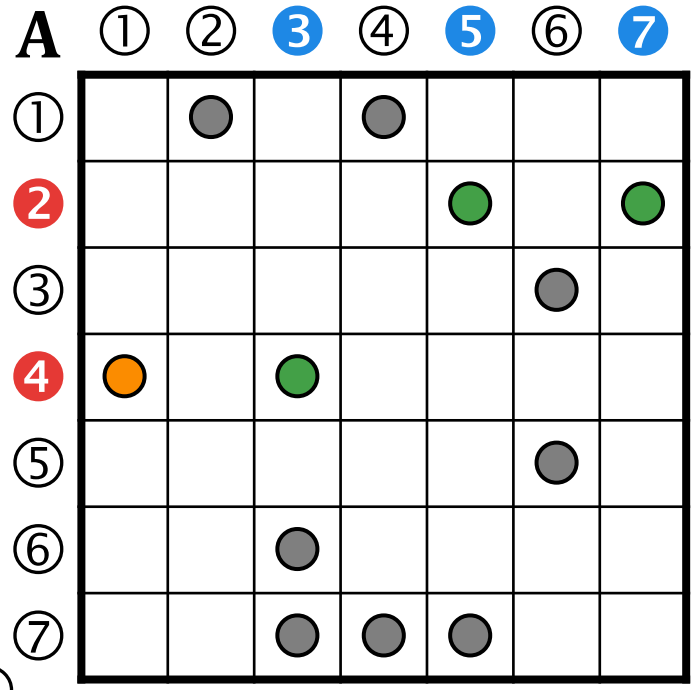
$\mathbf{f}\langle\bar{\mathbf{v}}\rangle = \mathbf{f}$ any. pair $\mathbf{A}$

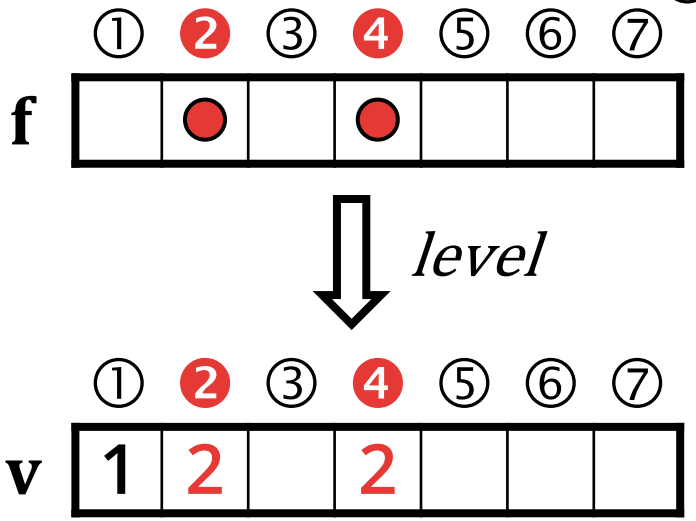
BFS – LEVELS

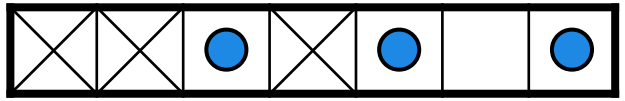
semiring	set	$\oplus$	$\otimes$	0
lor-land	$a \in \{\mathrm{F}, \mathrm{T}\}$	any	pair	F

$level = 2$







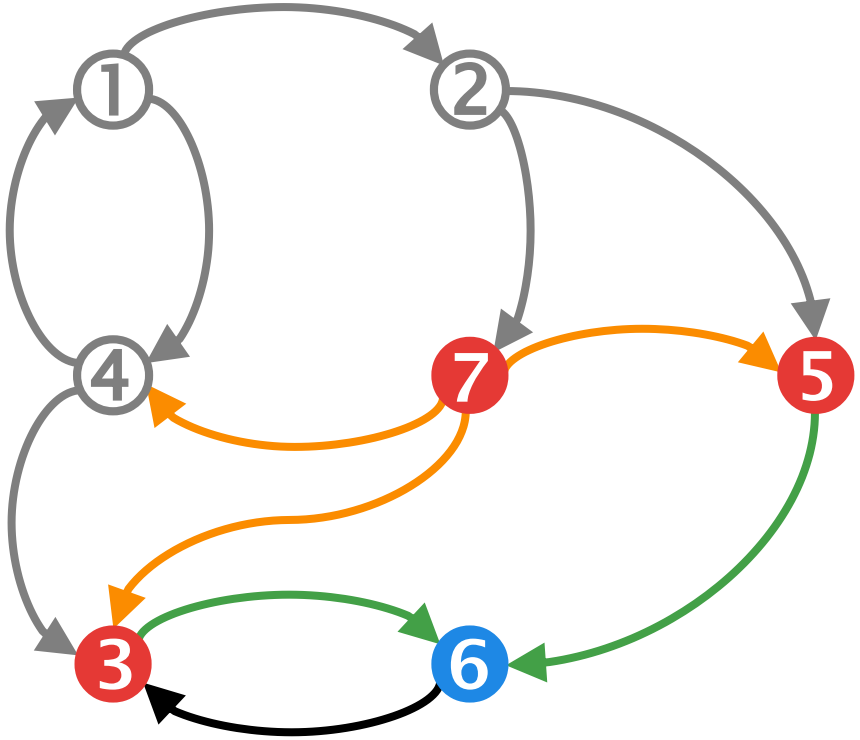


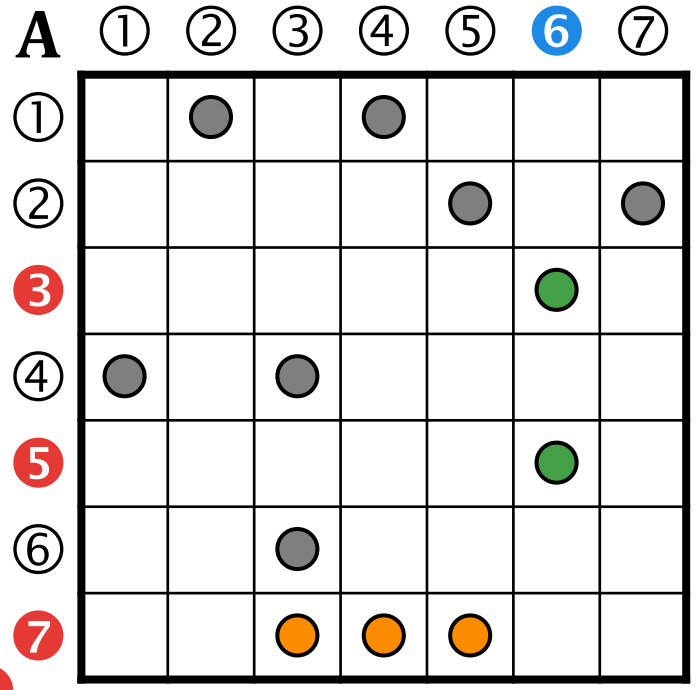
$\mathbf{f}\langle\bar{\mathbf{v}}\rangle = \mathbf{f}$ any. pair $\mathbf{A}$

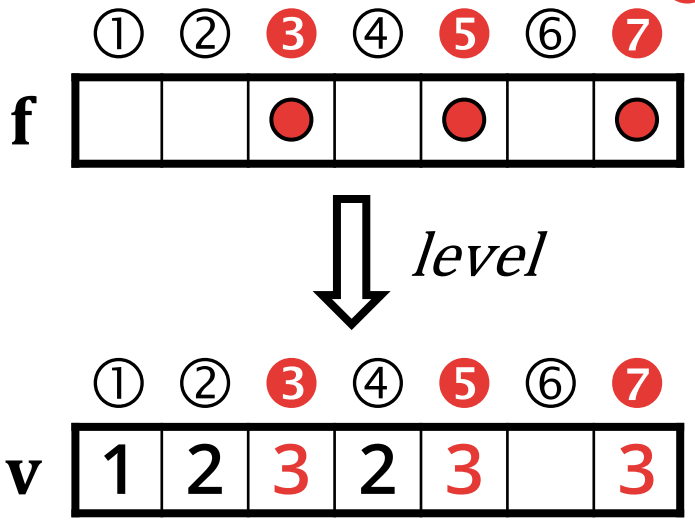
BFS – LEVELS

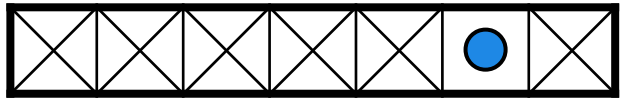
semiring	set	$\oplus$	$\otimes$	0
lor-land	$a \in \{\mathrm{F}, \mathrm{T}\}$	any	pair	F

$level = 3$







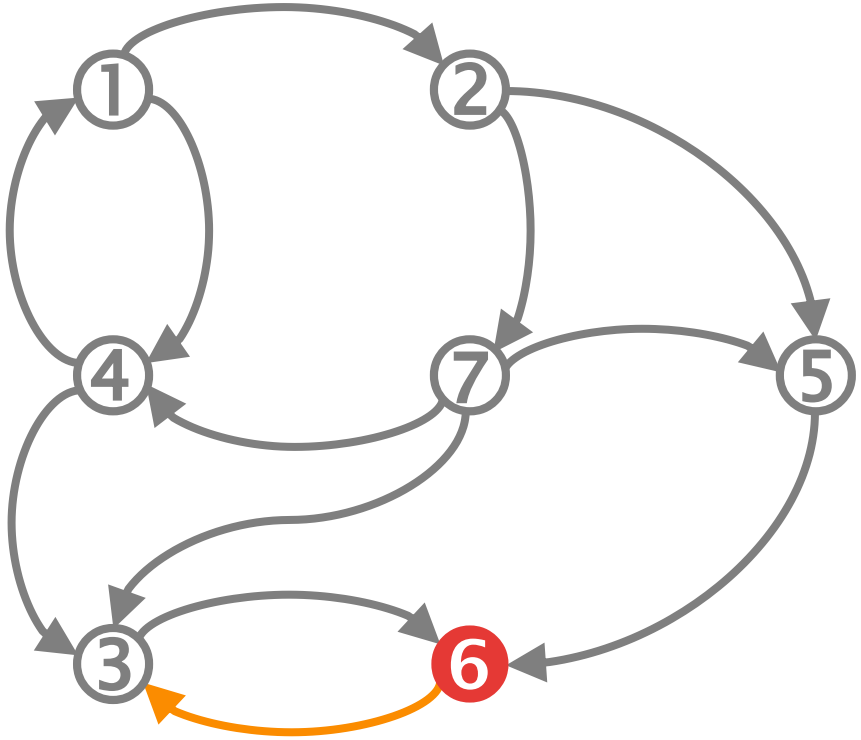


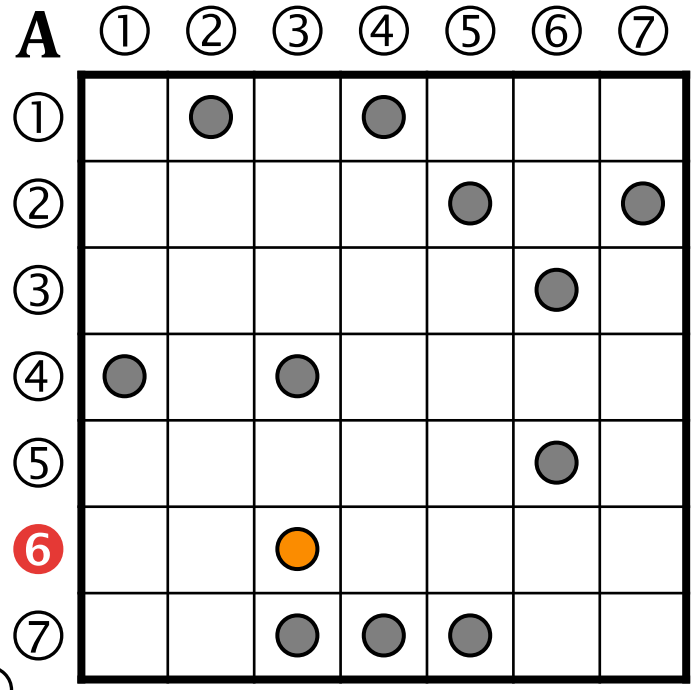
$\mathbf{f}\langle\bar{\mathbf{v}}\rangle = \mathbf{f}$ any. pair $\mathbf{A}$

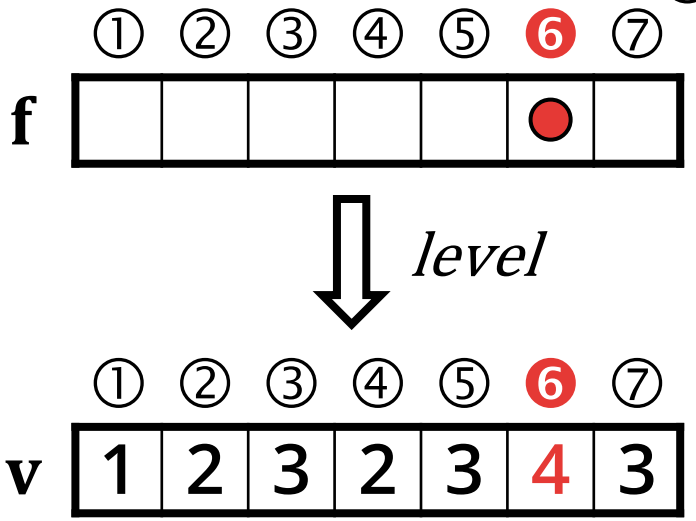
BFS – LEVELS

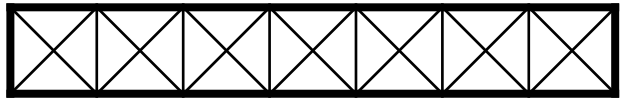
semiring	set	$\oplus$	$\otimes$	0
lor-land	$a \in \{\mathrm{F}, \mathrm{T}\}$	any	pair	F

$level = 4$









$\mathbf{f}\langle\bar{\mathbf{v}}\rangle = \mathbf{f}$ any. pair $\mathbf{A}$

$\mathbf{f}$ is empty
$\rightarrow$ 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)

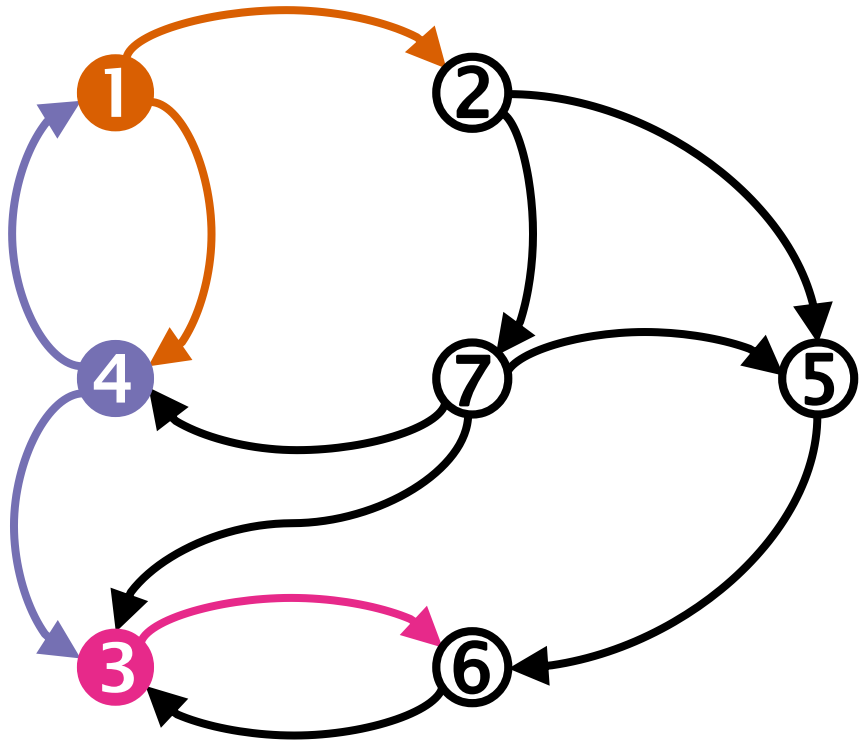
1. $\mathbf{f}(s) = \mathrm{T}$
2. for $\mathit{level} = 1$ to $n - 1$ — *terminate earlier if $\mathbf{f}$ is empty
3. $\quad \mathbf{v}\langle\mathbf{f}\rangle = \mathit{level}$ — assign level value to the vertices in the frontier
4. $\quad \mathrm{clear}(\mathbf{f})$ — clear the frontier $\mathbf{f}$ (all elements set to FALSE)
5. $\quad \mathbf{f}\langle\bar{\mathbf{v}}\rangle = \mathbf{f}\ \mathrm{any.pair}\ \mathbf{A}$ — use the any-pair semiring to advance the frontier

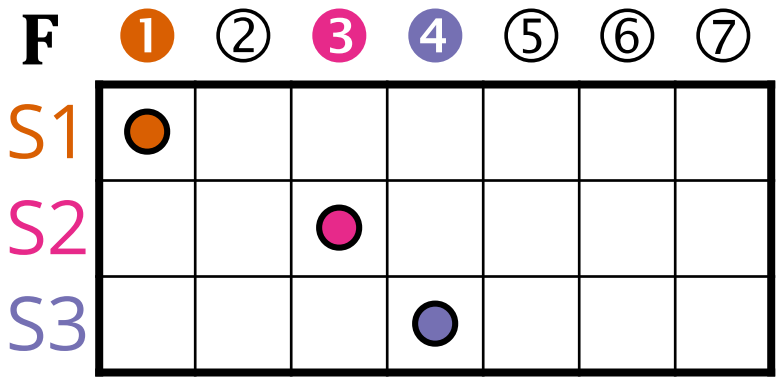
Graph algorithms in GraphBLAS

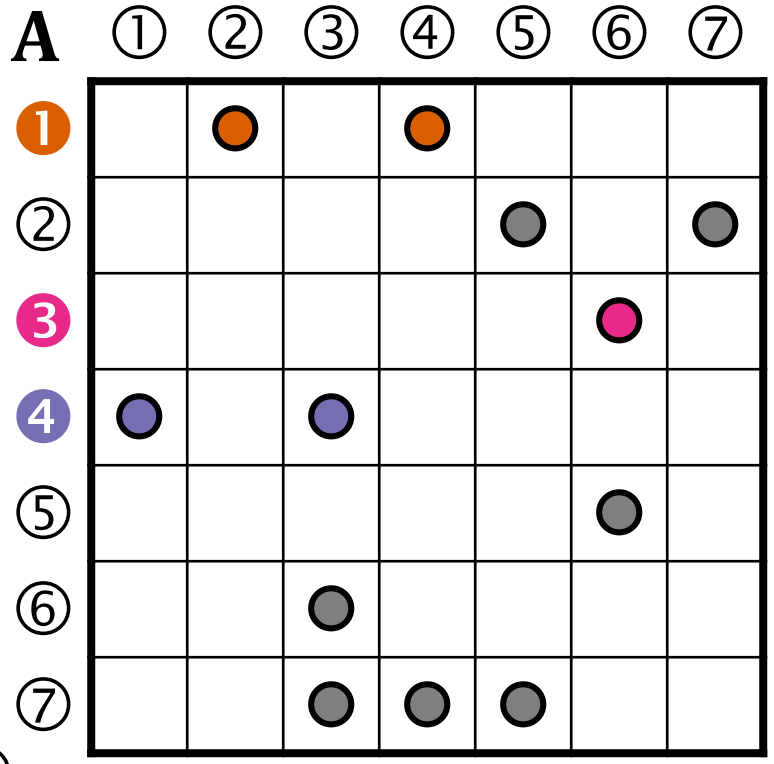
Multi-source BFS

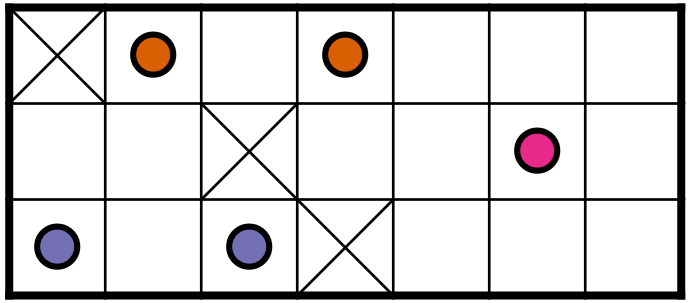
MULTI-SOURCE BFS – LEVELS

semiring	set	$\oplus$	$\otimes$	0
any-pair	$a \in \{\mathrm{F}, \mathrm{T}\}$	any	pair	F









$$\mathbf{F}\langle\overline{\mathbf{V}}\rangle = \mathbf{F} \text{ any.pair } \mathbf{A}$$

Graph algorithms in GraphBLAS

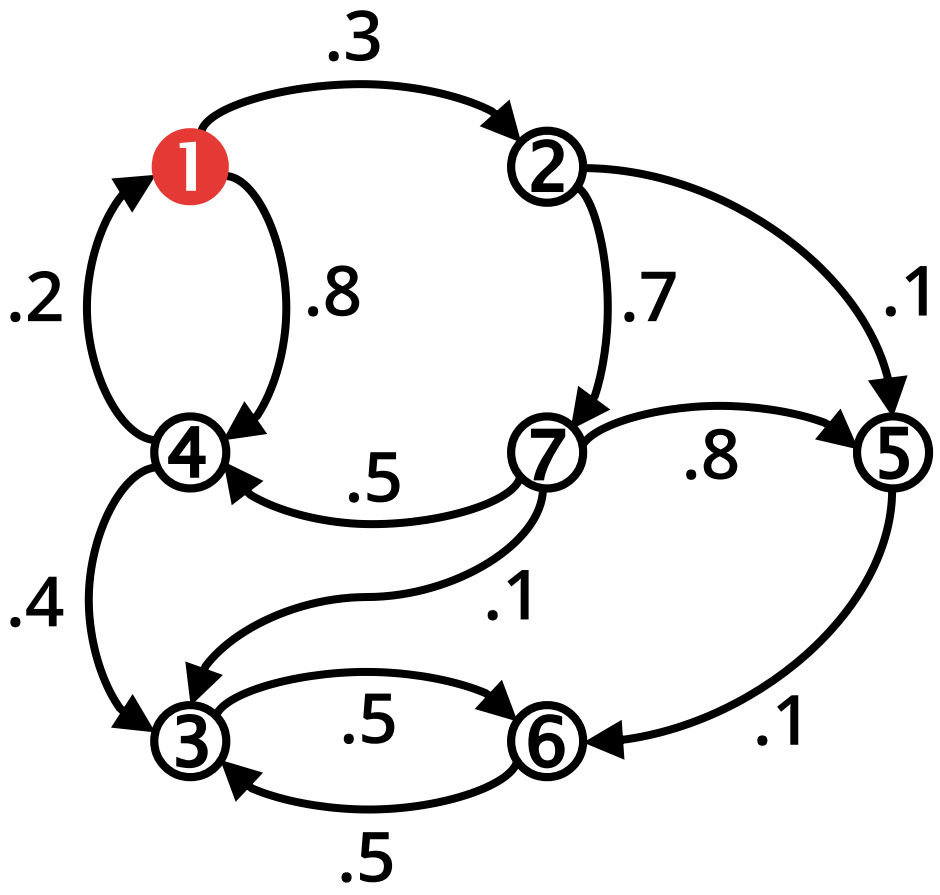
Single-source shortest path

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	set	$\oplus$	$\otimes$	0
min-plus	$a \in \mathbb{R} \cup \{+\infty\}$	min	$+$	$+\infty$



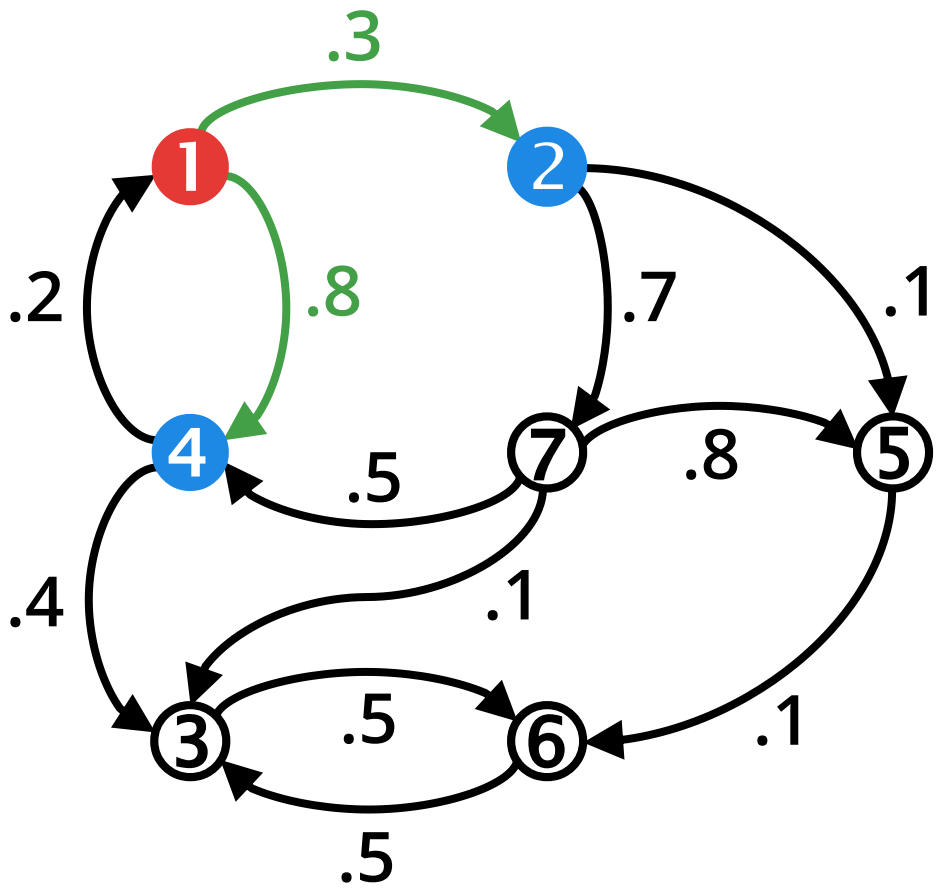
A	①	②	③	④	⑤	⑥	⑦
①	0	.3		.8			
②		0			.1		.7
③			0			.5	
④	.2		.4	0			
⑤					0	.1	
⑥			.5			0	
⑦			.1	.5	.9		0

	1	②	③	④	⑤	⑥	⑦
d	0						

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	set	$\oplus$	$\otimes$	0
min-plus	$a \in \mathbb{R} \cup \{+\infty\}$	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

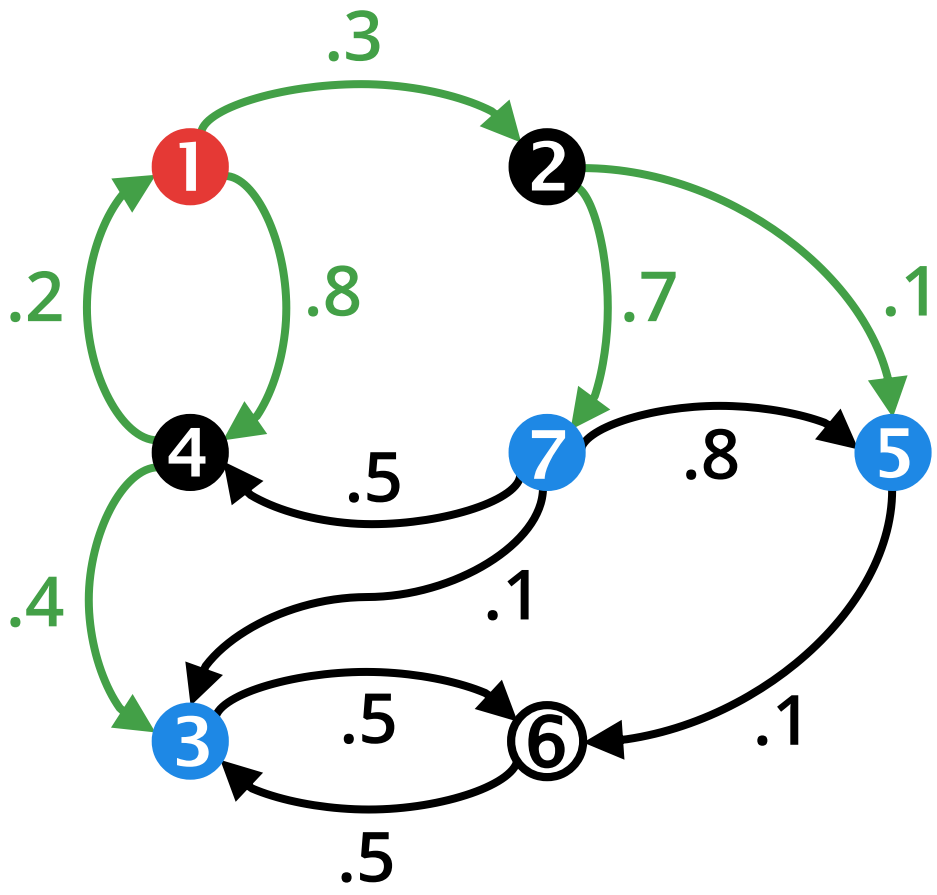
	❶	②	③	④	⑤	⑥	⑦
d	0						

0	.3		.8			

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	set	$\oplus$	$\otimes$	0
min-plus	$a \in \mathbb{R} \cup \{+\infty\}$	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

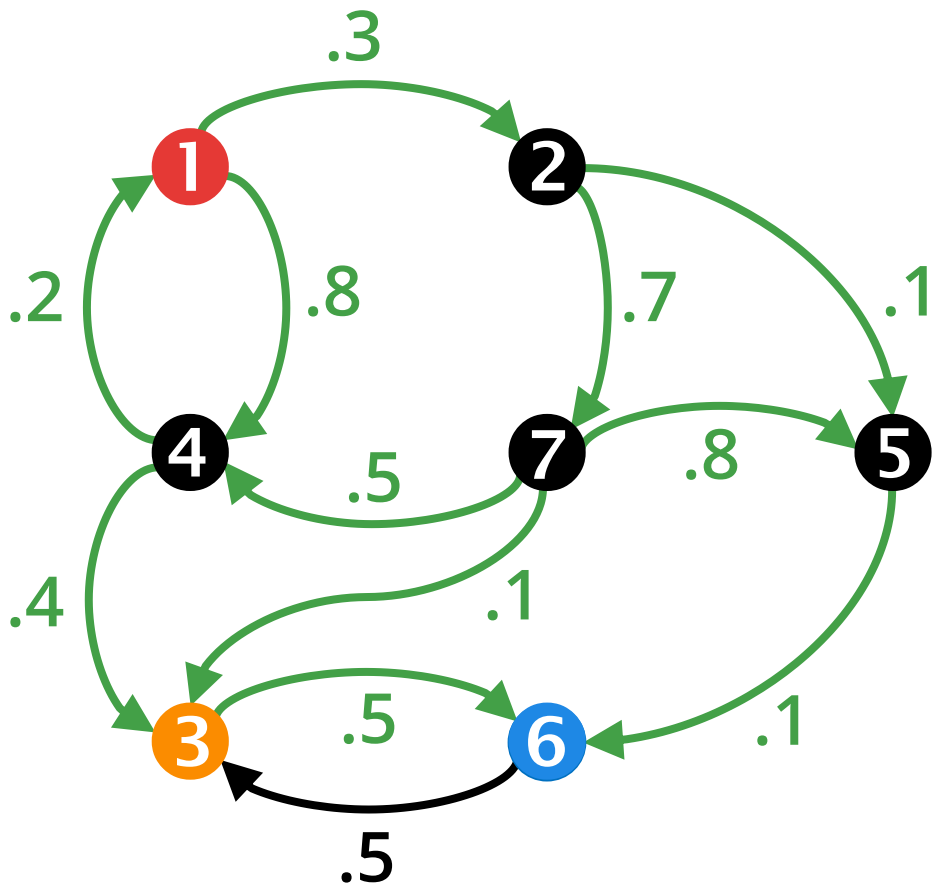
	❶	❷	③	❹	⑤	⑥	⑦
d	0	.3		.8			

0	.3	1.2	.8	.4		1

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	set	$\oplus$	$\otimes$	0
min-plus	$a \in \mathbb{R} \cup \{+\infty\}$	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

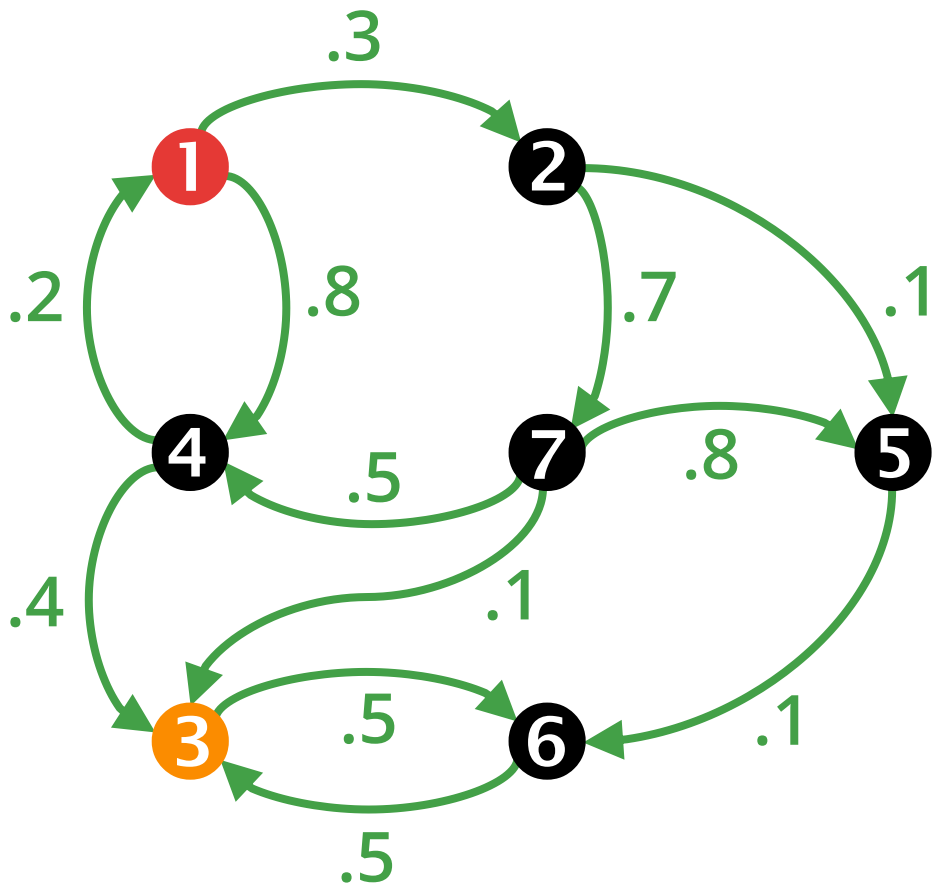
	❶	❷	❸	❹	❺	⑥	❼
d	0	.3	1.2	.8	.4		1

0	.3	1.1	.8	.4	.5	1

d min.+ **A**

SSSP – ALGEBRAIC BELLMAN-FORD

semiring	set	$\oplus$	$\otimes$	0
min-plus	$a \in \mathbb{R} \cup \{+\infty\}$	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	
❻			.5			0	
❼			.1	.5	.9		0

	❶	❷	❸	❹	❺	❻	❼
d	0	.3	1.1	.8	.4	.5	1

0	.3	1	.8	.4	.5	1

d min.+ **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)

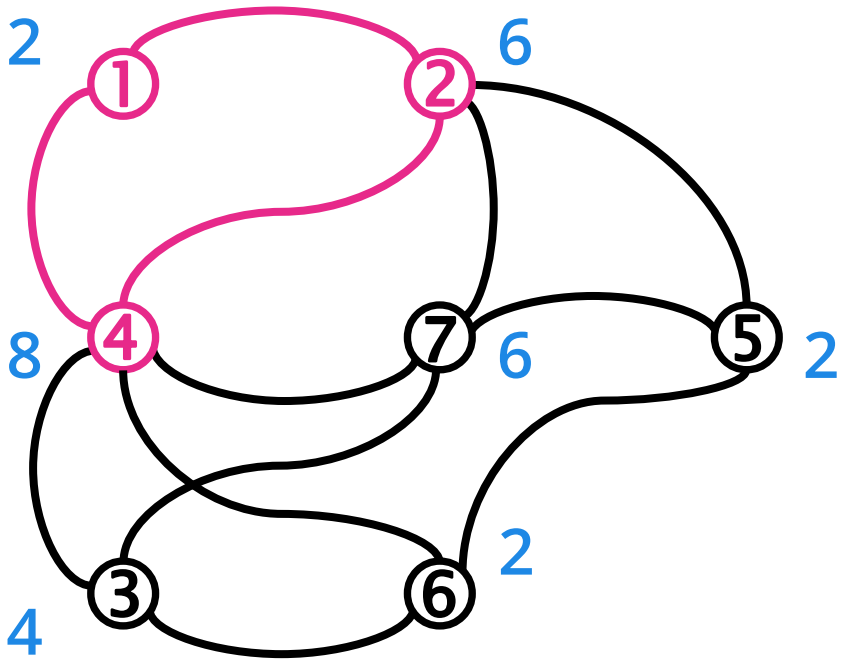
1. $\mathbf{d} = [\infty\ \infty \ldots \infty]$
2. $\mathbf{d}(s) = 0$
3. for $k = 1$ to $n - 1$ *terminate earlier if we reach a fixed point
4. $\quad \mathbf{d} = \mathbf{d}\ \min.+\ \mathbf{A}$

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:

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

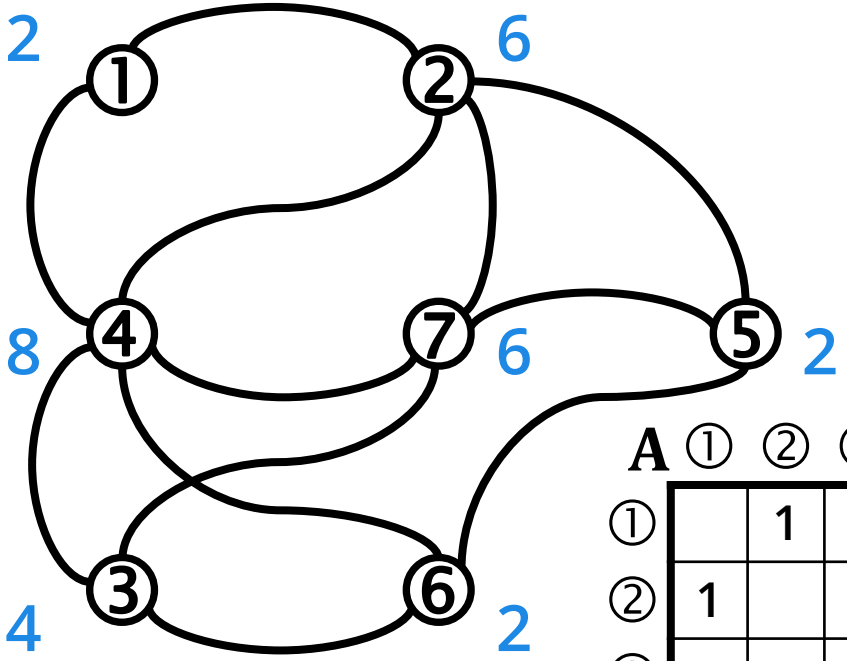
GraphChallenge.org: Raising the Bar on Graph Analytic Performance, HPEC 2018

Graph algorithms in GraphBLAS

Triangle count / Naïve algorithm

TC EXAMPLE: NAÏVE APPROACH

$$\mathbf{tri2} = \mathrm{diag}^{-1}(\mathbf{A} \mathbin{+.\times} \mathbf{A} \mathbin{+.\times} \mathbf{A})$$



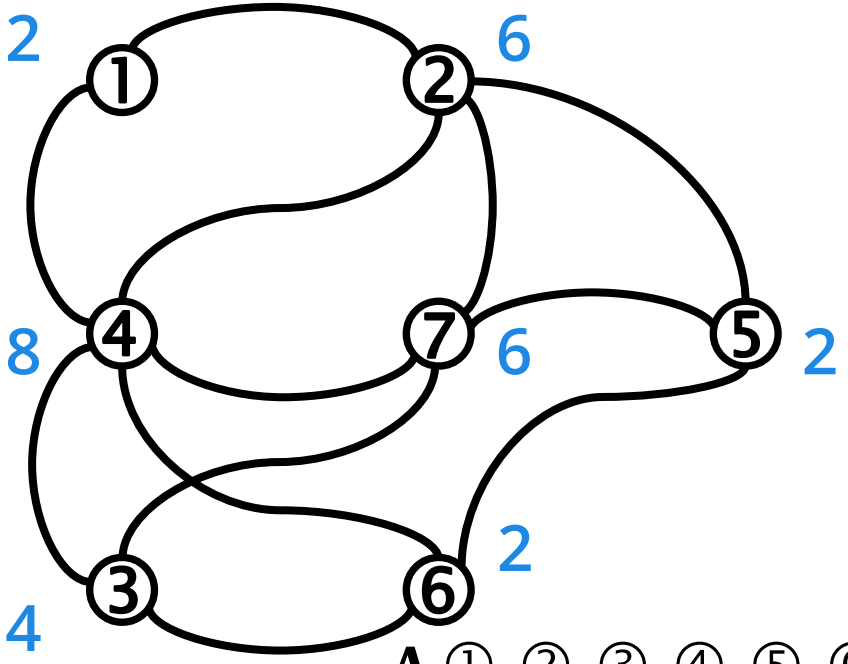
$$\begin{array}{c|ccccccc}
\mathbf{A} & ① & ② & ③ & ④ & ⑤ & ⑥ & ⑦ \\
\hline
① & & 1 & & 1 & & & \\
② & 1 & & & 1 & 1 & & 1 \\
③ & & & & 1 & & 1 & 1 \\
④ & 1 & 1 & 1 & & & 1 & 1 \\
⑤ & & 1 & & & & 1 & 1 \\
⑥ & & & 1 & 1 & 1 & & \\
⑦ & & 1 & 1 & 1 & 1 & &
\end{array}
\qquad
\begin{array}{c|ccccccc}
\mathbf{A}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

$$\begin{array}{c|ccccccc}
\mathbf{A} & ① & ② & ③ & ④ & ⑤ & ⑥ & ⑦ \\
\hline
① & & 1 & & 1 & & & \\
② & 1 & & & 1 & 1 & & 1 \\
③ & & & & 1 & & 1 & 1 \\
④ & 1 & 1 & 1 & & & 1 & 1 \\
⑤ & & 1 & & & & 1 & 1 \\
⑥ & & & 1 & 1 & 1 & & \\
⑦ & & 1 & 1 & 1 & 1 & &
\end{array}
\quad
\begin{array}{ccccccc}
2 & 1 & 1 & 1 & 1 & 1 & 2 \\
1 & 4 & 2 & 2 & 1 & 2 & 2 \\
1 & 2 & 3 & 2 & 2 & 1 & 1 \\
1 & 2 & 2 & 5 & 3 & 1 & 2 \\
1 & 1 & 2 & 3 & 3 & & 1 \\
1 & 2 & 1 & 1 & & 3 & 3 \\
2 & 2 & 1 & 2 & 1 & 3 & 4
\end{array}
\quad
\begin{array}{ccccccc}
2 & 6 & 4 & 7 & 4 & 3 & 4 \\
6 & 6 & 6 & 11 & 8 & 5 & 9 \\
4 & 6 & 4 & 8 & 4 & 7 & 9 \\
7 & 11 & 8 & 8 & 5 & 10 & 12 \\
4 & 8 & 4 & 5 & 2 & 8 & 9 \\
3 & 5 & 7 & 10 & 8 & 2 & 4 \\
4 & 9 & 9 & 12 & 9 & 4 & 6
\end{array}
\quad
\begin{array}{c}
\mathbf{tri2} \\
\hline
2 \\ 6 \\ 4 \\ 8 \\ 2 \\ 2 \\ 6
\end{array}$$

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		

$$\mathbf{TRI} = \mathbf{A} +.\times \mathbf{A} \times \mathbf{A}$$

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

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

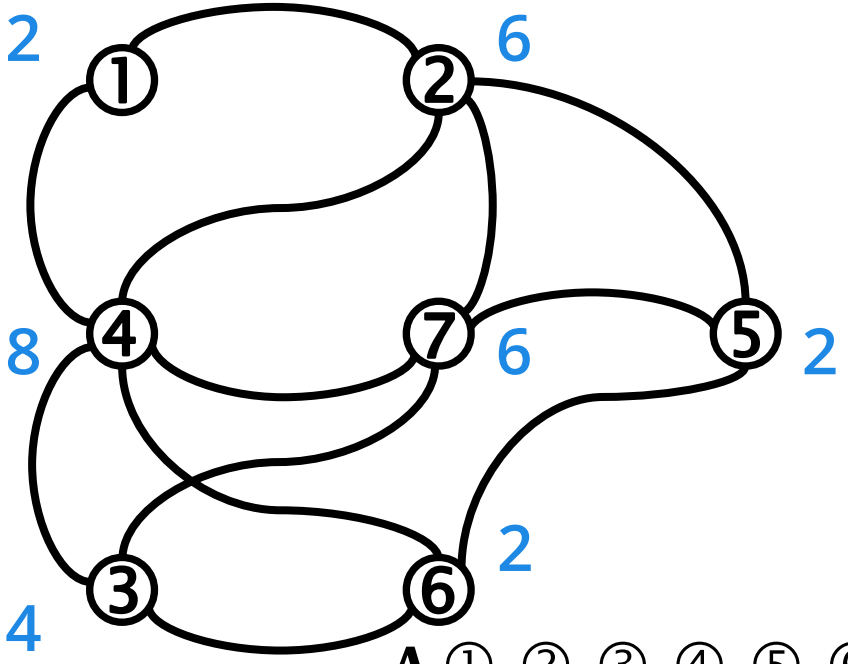
	1		1			
1			2	1		2
			2		1	1
1	2	2			1	2
	1					1
		1	1			
	2	1	2	1		

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

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 $\mathbf{A}$ as a mask for $\mathbf{A} +.\times \mathbf{A}$.

$$\mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} +.\times \mathbf{A}$$
$$\mathbf{tri2} = \left[+_j \mathbf{TRI}(:,j)\right]$$

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	②	③	④	⑤	⑥	⑦
		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

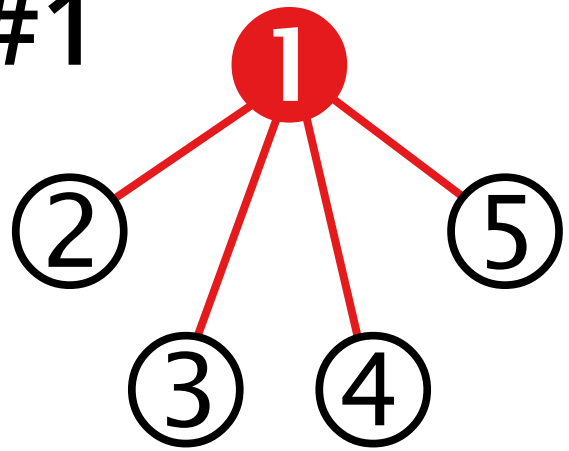
The importance of masking

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



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

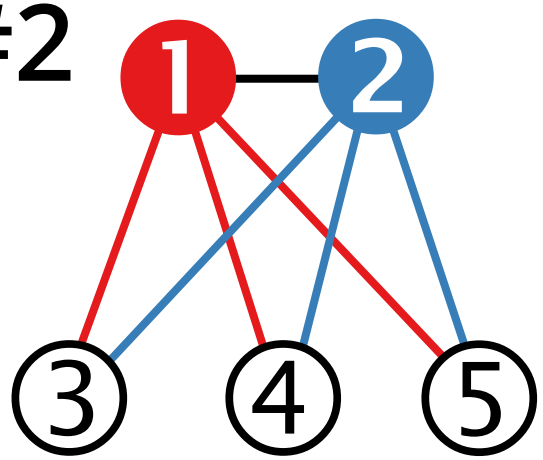
$$\begin{array}{c|ccccc} \mathbf{A} & \text{❶} & \text{②} & \text{③} & \text{④} & \text{⑤} \\ \hline \text{❶} & & 1 & 1 & 1 & 1 \\ \text{②} & 1 & & & & \\ \text{③} & 1 & & & & \\ \text{④} & 1 & & & & \\ \text{⑤} & 1 & & & & \end{array} \qquad \begin{array}{|ccccc|} \hline 4 & & & & \\ & 1 & 1 & 1 & 1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

$\mathbf{A}$ $\qquad$ $\mathbf{A} +.\times \mathbf{A}$ $\qquad$ $\mathbf{A} +.\times \mathbf{A} +.\times \mathbf{A}$

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).

#2



A	❶	❷	③	④	⑤
❶		1	1	1	1
❷	1		1	1	1
③	1	1			
④	1	1			
⑤	1	1			

A	❶	❷	③	④	⑤
❶		1	1	1	1
❷	1		1	1	1
③	1	1			
④	1	1			
⑤	1	1			

A	❶	❷	③	④	⑤
❶		1	1	1	1
❷	1		1	1	1
③	1	1			
④	1	1			
⑤	1	1			

A

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

$\mathbf{A} +.\times \mathbf{A}$

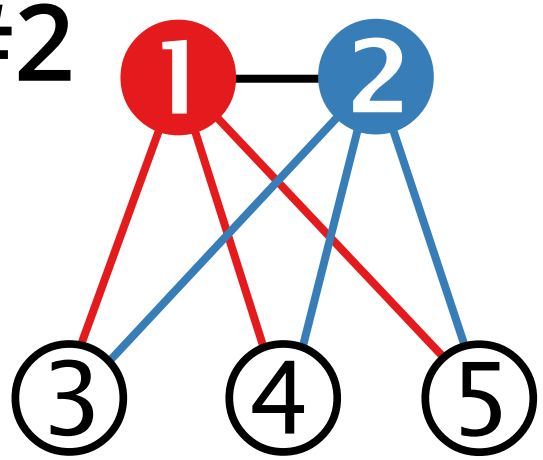
	3	1	1	1
3		1	1	1
1	1			
1	1			
1	1			

$\mathbf{A} +.\times \mathbf{A} +.\times \mathbf{A}$

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, $\mathbf{A} +.\times \mathbf{A}$ enumerates all wedges starting and ending in the bottom partition, thus performing a lot of unnecessary work.

#2



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

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

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

A

	3	1	1	1
3		1	1	1
1	1			
1	1			
1	1			

$\mathbf{TRI}\langle\mathbf{A}\rangle = \mathbf{A} +.\times \mathbf{A}$

$$\xrightarrow{[+_j \mathbf{TRI}(:,j)]}$$

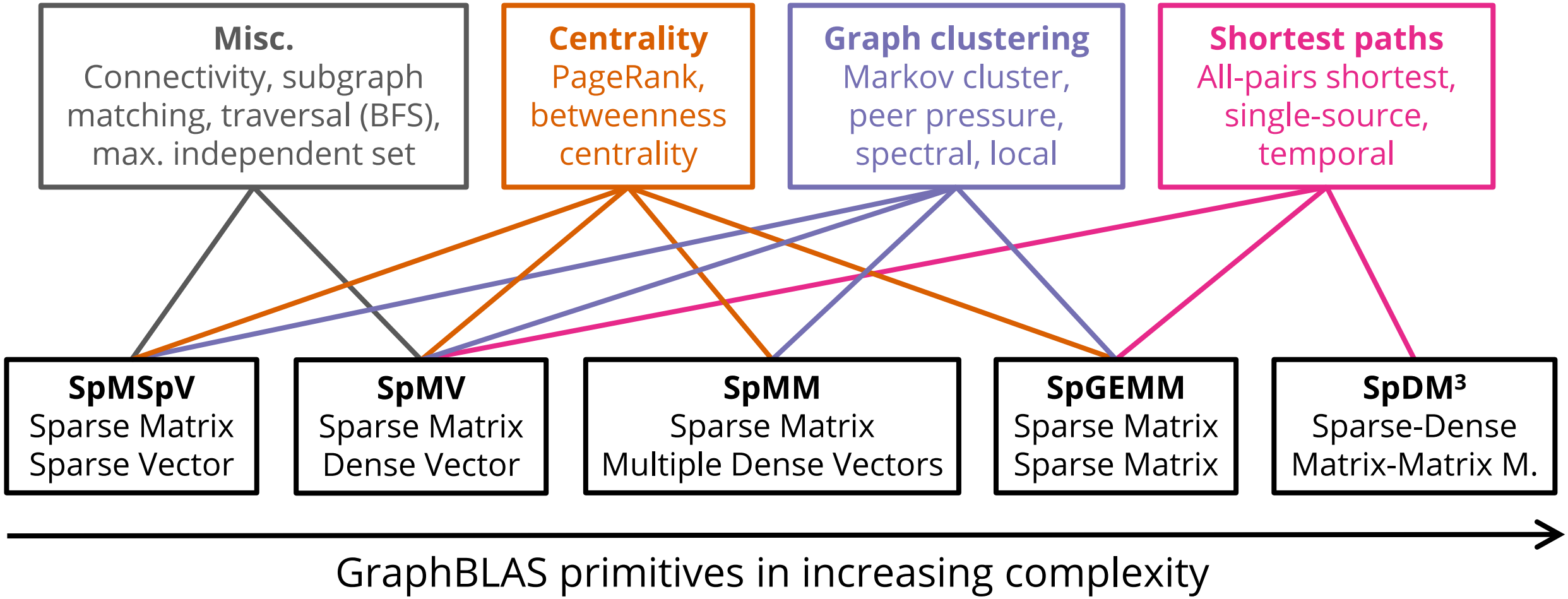
6
6
2
2
2

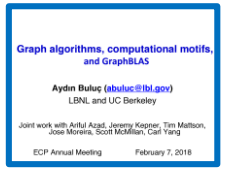
tri

Graph algorithms in GraphBLAS

Graph algorithms & GraphBLAS primitives

GRAPH ALGORITHMS & GRAPHBLAS PRIMITIVES





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

Complex case study

SIGMOD 2014 PROGRAMMING CONTEST

Annual contest

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

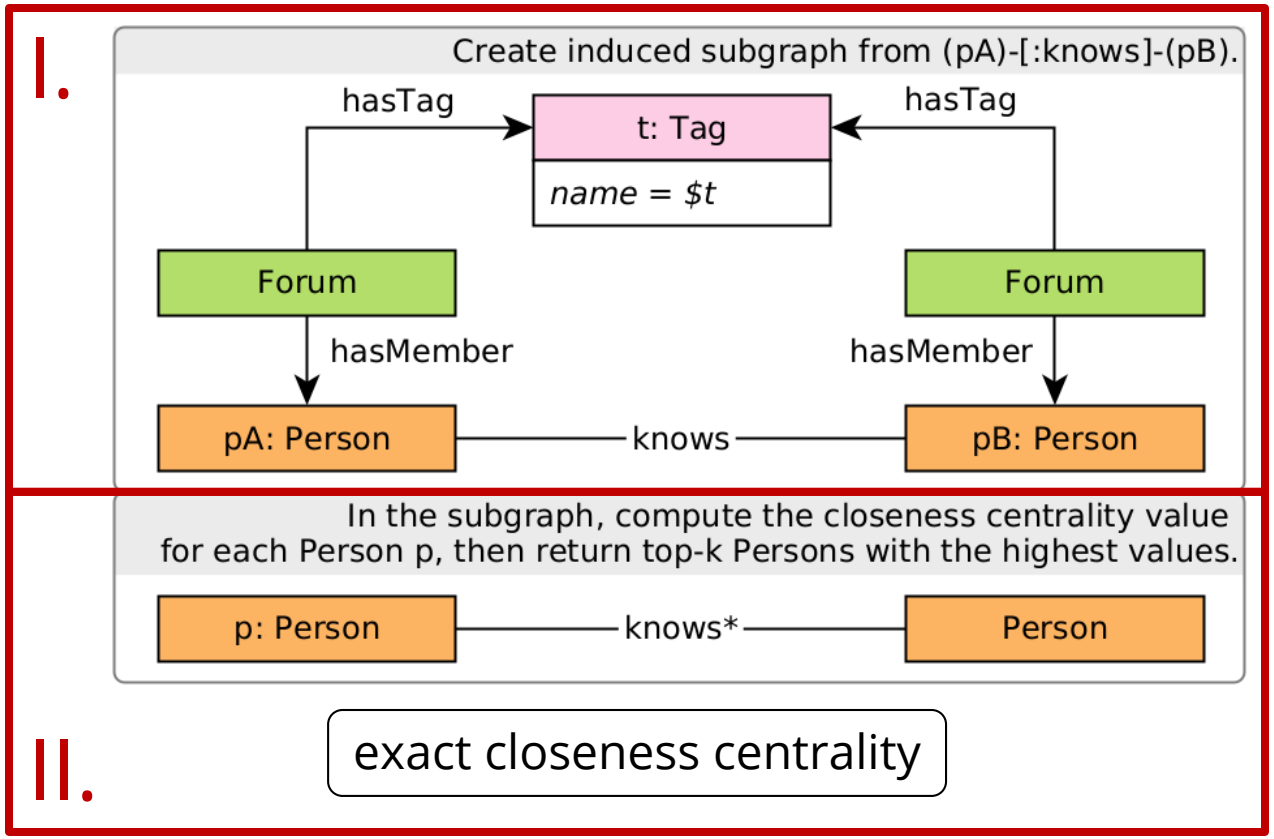
2014 event

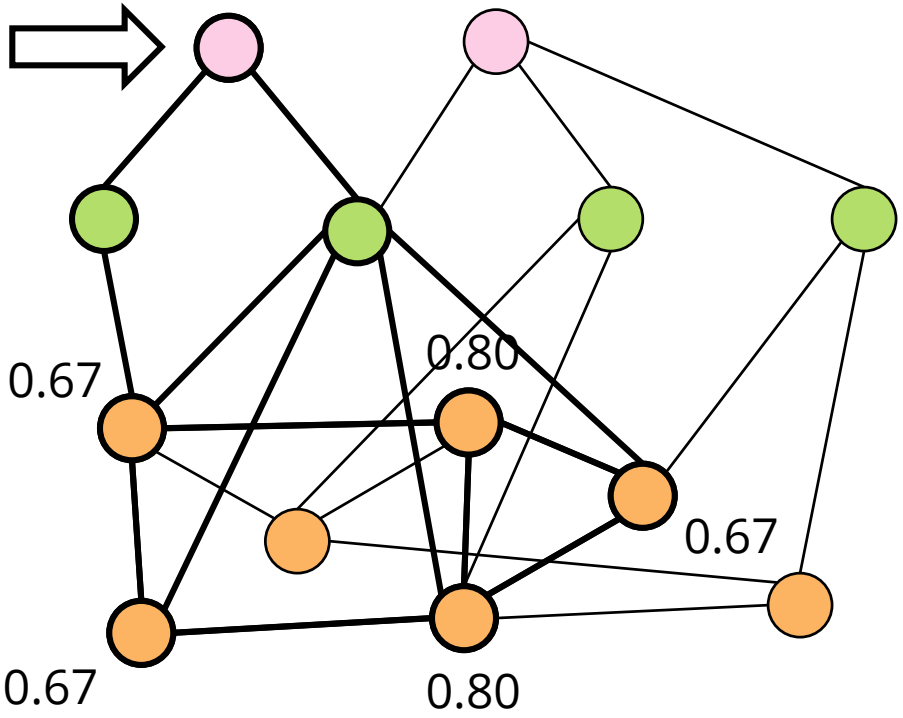
- Tasks on the LDBC social network graph
  - Benchmark data set for property graphs
  - People, forums, comments, hashtags, etc.
- 4 queries
  - Mix of filtering operations and graph algorithms

LDBC

QUERY TEMPLATE

I. Compute an induced subgraph over Person-knows-Person
II. Run the graph algorithm on the subgraph





key kernel: **all-source BFS**

Q4: CLOSENESS CENTRALITY VALUES

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

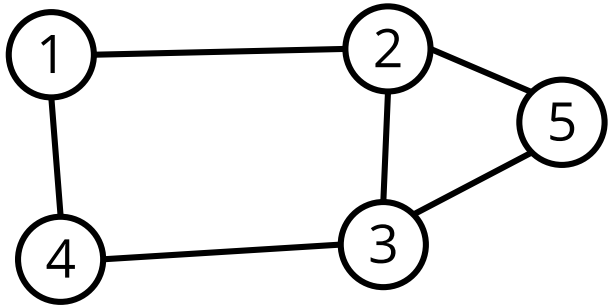
$$CCV(p) = \frac{(C(p) - 1)^2}{(n - 1) \cdot s(p)}$$

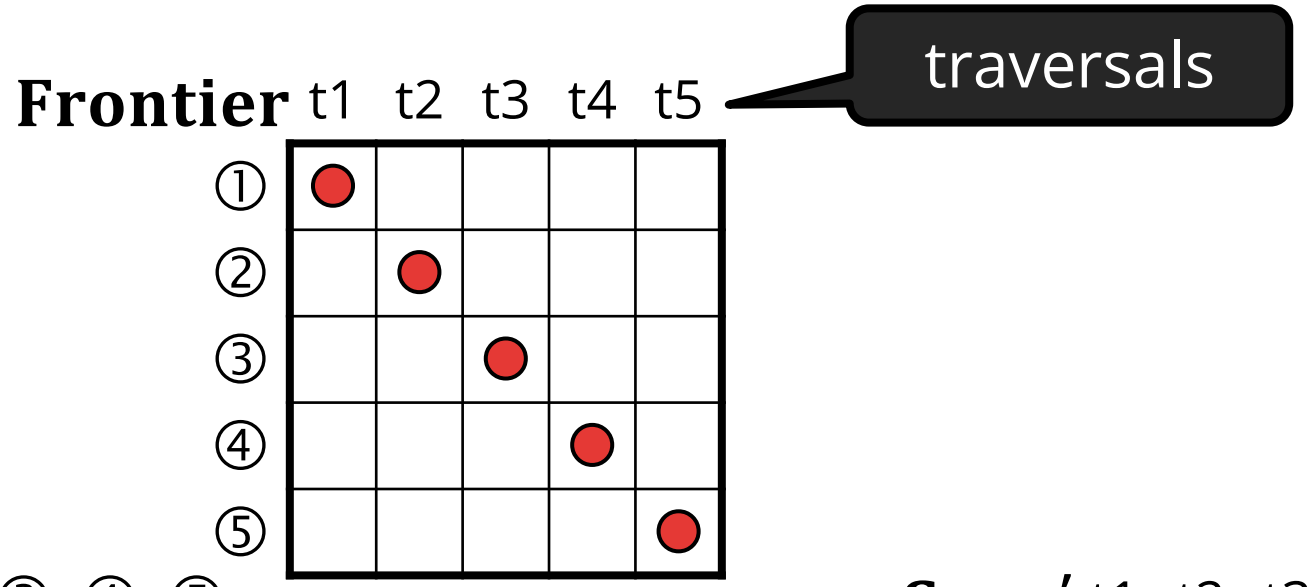
where

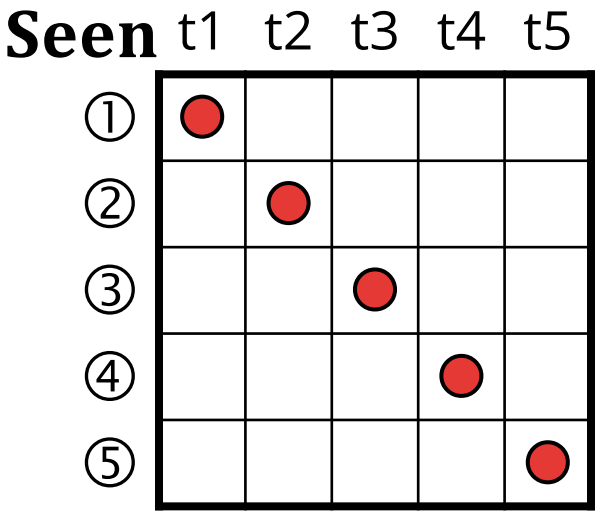
- $C(p)$ is the size of the connected component of vertex p,
- n is the number of vertices in the induced graph,
- $s(p)$ is the sum of geodesic distances to all other reachable persons from p.

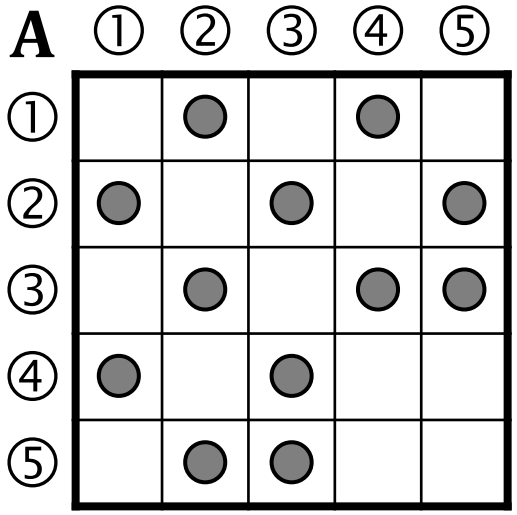
$s(p)$ is challenging: needs unweighted all-pairs shortest paths.

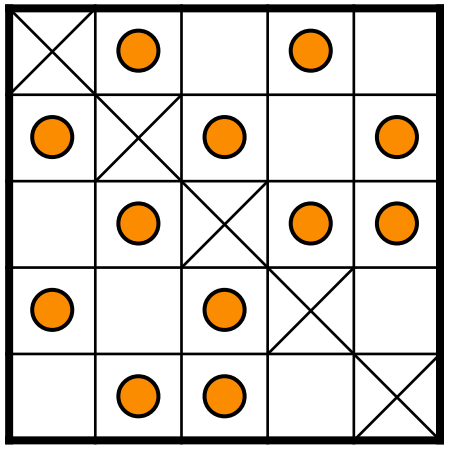
BOOLEAN ALL-SOURCE BFS ALGORITHM

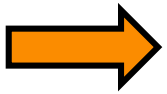


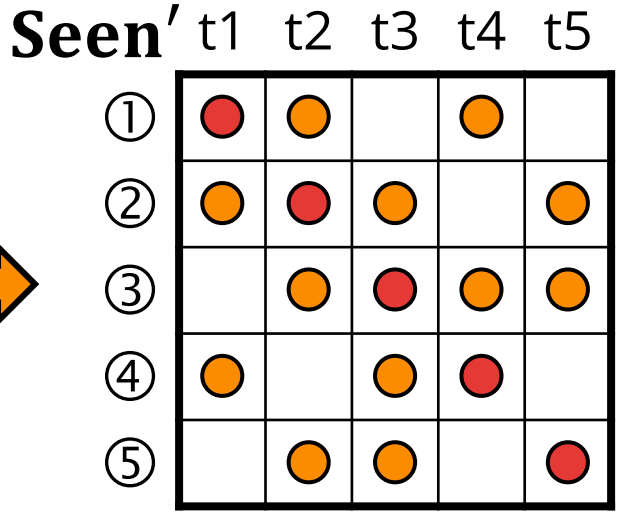








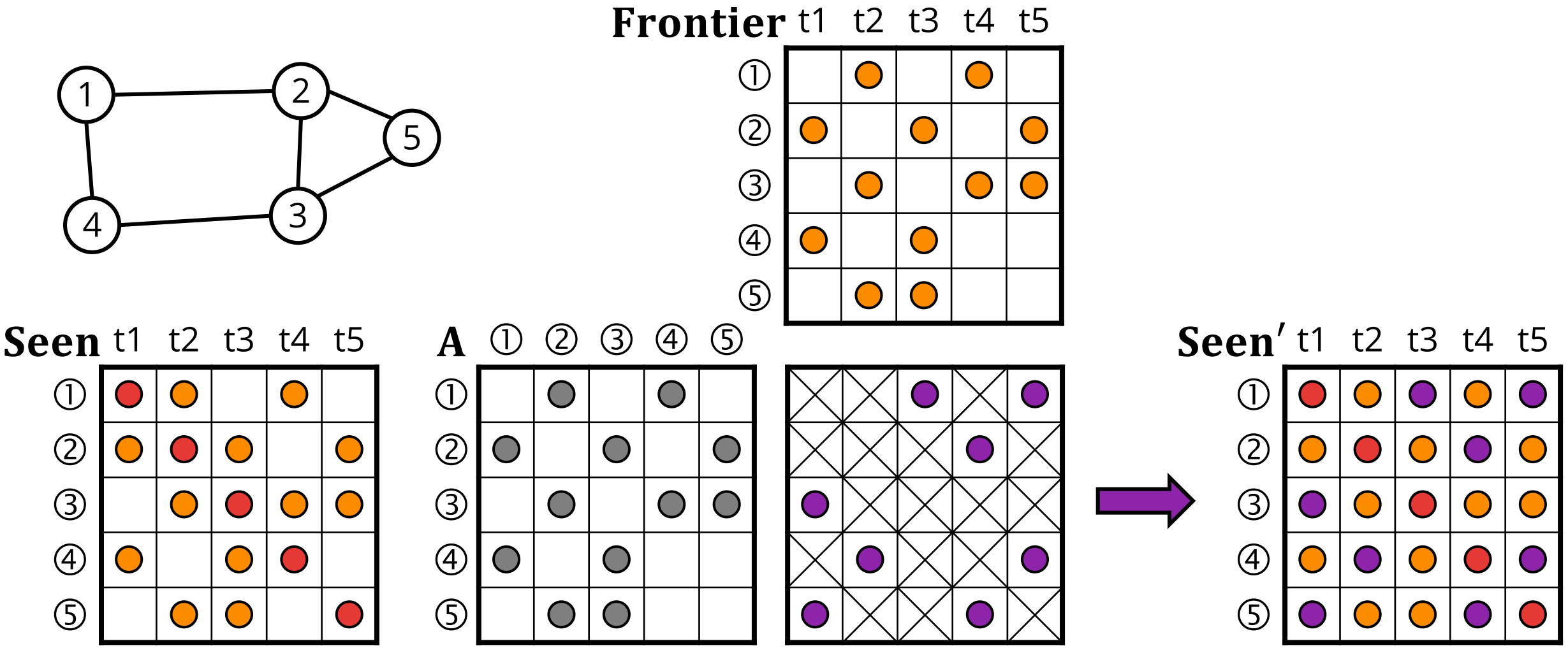




$$\mathbf{Next}\langle\neg\mathbf{Seen}\rangle = \mathbf{A}\ \text{any}\ .\ \text{pair}\ \mathbf{Frontier}$$

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

BOOLEAN ALL-SOURCE BFS ALGORITHM



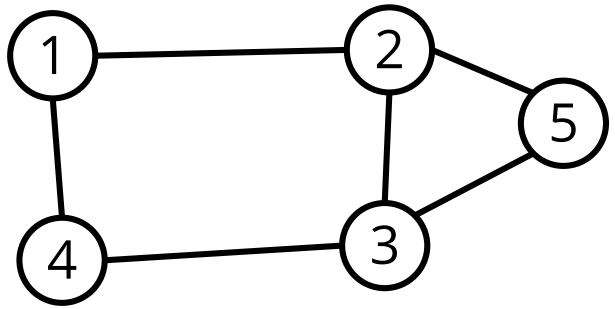
$\mathbf{Next}\langle\neg\mathbf{Seen}\rangle =$
$\mathbf{A}$ any . pair $\mathbf{Frontier}$

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

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

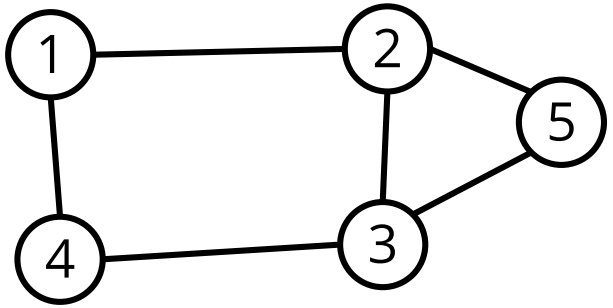
Next =
A bor . second **Frontier**

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

Next =

A bor . second **Frontier**

Seen′

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

Seen′ =

Seen bor **Next**

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 are compressed formats are:
  - CSR: Compressed Sparse Row (also known as CRS)
  - CSC: Compressed Sparse Column (also known as CCS)

$\mathbf{A}$	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		

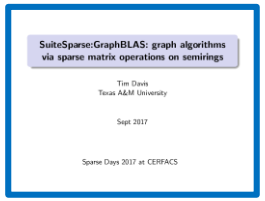
CSR representation of $\mathbf{A}$:

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

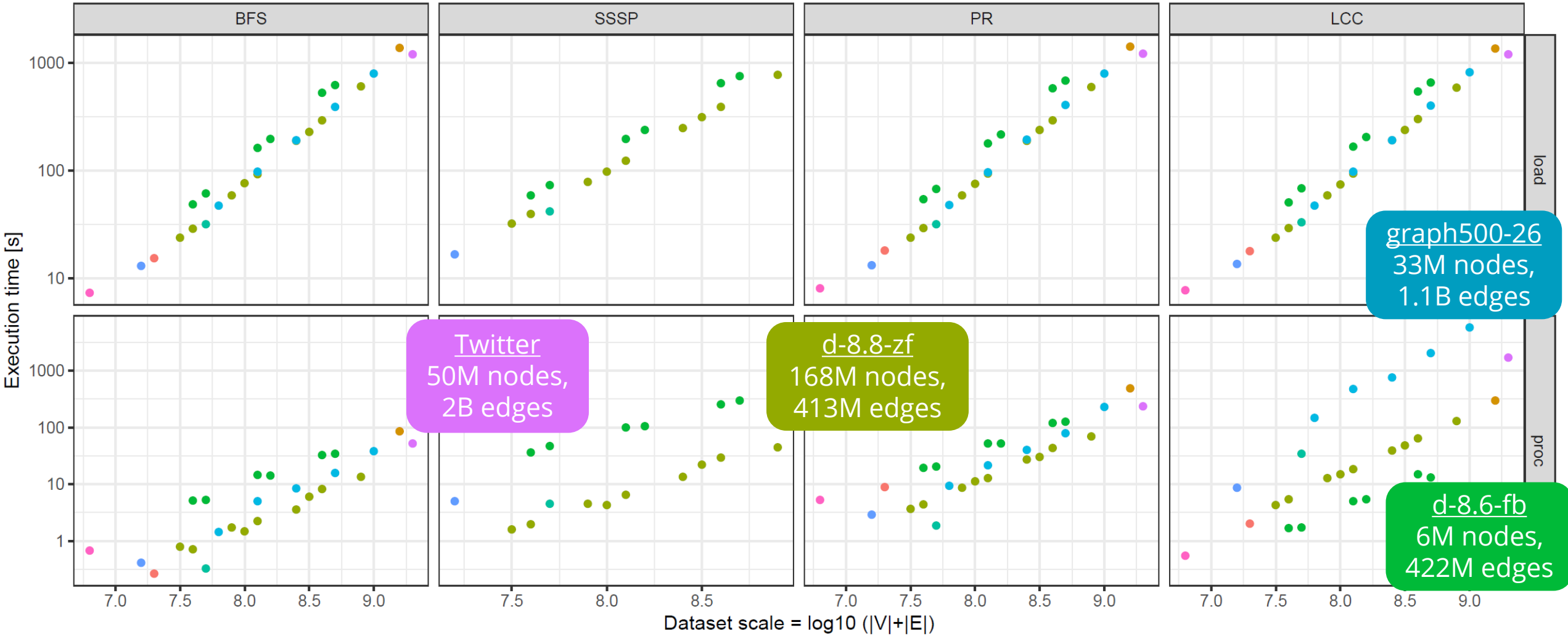
- Authored by Prof. Tim Davis at Texas A&M University, based on his SuiteSparse library (used in MATLAB).
- Design decisions, algorithms and data structures are discussed in the TOMS paper and in the User Guide.
- Extensions: methods and types prefixed with `GxB`.
- Sophisticated load balancer for multi-threaded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

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



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

SUITESPARSE:GRAPHBLAS / LDBC GRAPHALYTICS



Ubuntu Server, 512 GB RAM,
64 CPU cores, 128 threads

Results from late 2019,
new version is even faster

Summary

SUMMARY

- Linear algebra is a powerful abstraction
- GraphBLAS has good expressive power, allows concise formulation of algorithms and offers good performance

- Lots of ongoing research on the underpinning & usage

- **Overall:** sparse linear algebra as an abstraction layer for graph algorithms fits the age of heterogeneous hardware

Performance engineering case study: Grakn

GRAKN

- **Knowledge graph:**
  - is a highly heterogeneous graph built from multiple sources
  - models an open domain
- Basically: a graph with a non-trivial schema

- Storage/processing without a schema is slow.
- **Grakn** offers
  - A rich type system for defining a schema
  - The Graql query language
  - Automated reasoning (“knowledge discovery”)

Rules

Grakn allows you to define rules in your knowledge schema, which extends the expressivity of your model as it enables the system to derive new conclusions when a certain logical form in your dataset is satisfied.

Like functions in programming, that rules can chain itself to another, creating abstractions of behaviour at the data level.

1 2 3 4 5

Rules.gql

```
# Schema Rules

define

transitive-location sub rule,
when {
  (located: $x, locating: $y);
  (located: $y, locating: $z);
},
then {
  (located: $x, locating: $z);
};

commit
```

Graql

GRAKN PERFORMANCE

Haikal Pribadi
@haikalpribadi

1/ 2020 has been one helluva year for #engineering at @GraknLabs: we rewrote #grakn and #graql. We took the best of our #design, #math and #compsci from our work over the past 5 years that the #community loved, and we #rebuilt it with the best #software #architecture and #mindset

2:10 PM · Nov 15, 2020 · Twitter Web App

PARALLELISM IN GRAKN

Haikal Pribadi
@haikalpribadi

Replying to @haikalpribadi

11/ Right now the team is in the midst of completing the new #reasoning engine, designed ground up using our very own #multithreaded #eventloop and #concurrent #actormodel. The new #reasoner #computation will be natively #parallel! @GraknLabs

4:18 PM · Nov 15, 2020 · Twitter Web App