Carnegie Mellon University
Software Engineering Institute

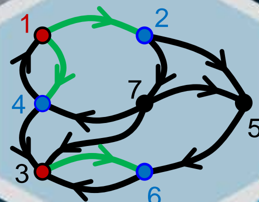
[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Patterns in GraphBLAS Algorithms:
Tales from the Trenches

21 May 2018

Scott McMillan (with Tze Meng Low)

Carnegie Mellon University
Software Engineering Institute



Copyright 2018 Carnegie Mellon University. All Rights Reserved.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM18-0662

A Brief History…

- Sep. 2013: HPEC "GraphBLAS position paper"
- Oct. 2014: Work on C++ GraphBLAS Template Library (GBTL) begins

Standards for Graph Algorithm Primitives

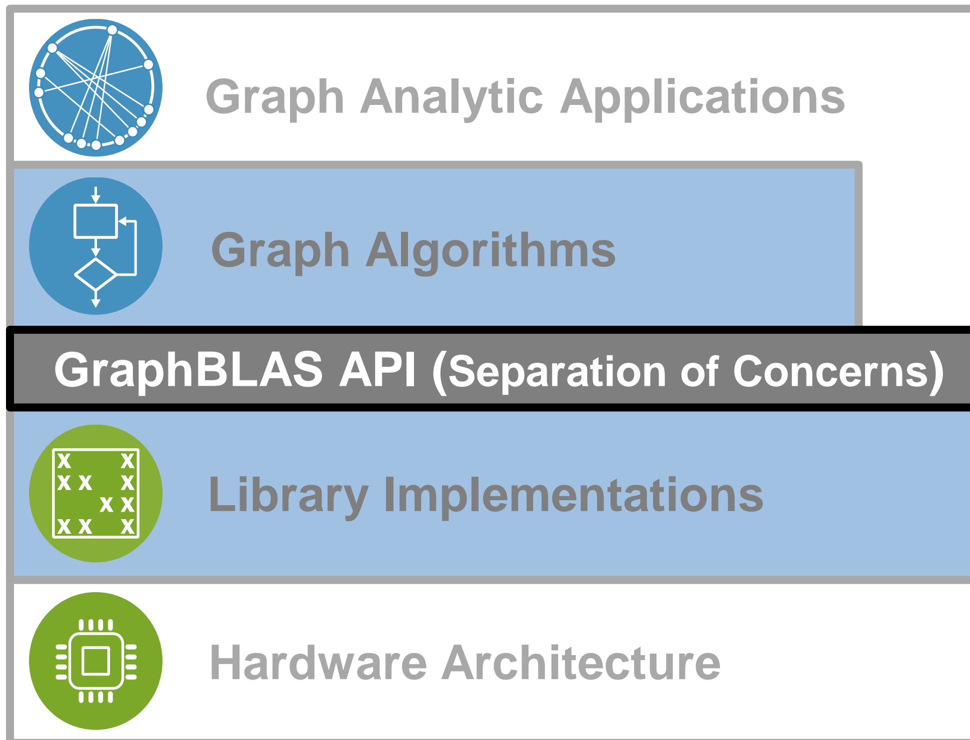
Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Melon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

- Jun. 2015: GraphBLAS "east coast/west coast" kickoff meeting
- Nov. 2015: GBTL v1.0 released
- Dec. 2015: Formation of the "GraphBLAS Signature Proposal Subcommittee"
- May 2017: GraphBLAS C API Specification v1.0 released ("provisional")
- Nov. 2017: SuiteSparse GraphBLAS v1.0 released
- May 2018: IBM GraphBLAS released, **"provisional" removed from C API spec. (v1.2.0)**
- May 2018: GBTL v2.0 released (C++, mathematically equivalent to C API spec)

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Library Design Goals

- GraphBLAS API
  - Achieve a ***separation of concerns*** between algorithm development and hardware optimizations
  - SuiteSparse/IBM C API
  - GBTL C++ “frontend”
  - pyGB python DSL



[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

GraphBLAS Primitives (the Math)

Operation	Description	Mathematical Description
mxm	Matrix multiplication (bread-first traversal)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^T \oplus.\otimes \mathbf{B}^T)$
mxv, (vxm)		$\mathbf{c}\langle\neg\mathbf{m}, z\rangle = \mathbf{c} \odot (\mathbf{A}^T \oplus.\otimes \mathbf{b})$
eWiseMult	Element-wise 'multiplication' (graph intersection)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^T \otimes \mathbf{B}^T)$
eWiseAdd	Element-wise 'addition' (graph union)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^T \oplus \mathbf{B}^T)$
reduce (row/col)	Reduce along rows/cols (vertex degree)	$\mathbf{c}\langle\neg\mathbf{m}, z\rangle = \mathbf{c} \odot [\oplus_j \mathbf{A}^T(:,j)]$
apply	Apply unary function to each element (edge modification)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot f(\mathbf{A}^T)$
transpose	Swap rows and columns (reverse directed edges)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot \mathbf{A}^T$
extract	Extract a sub-matrix (sub-graph selection)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot \mathbf{A}^T(\mathbf{i},\mathbf{j})$
assign	Assign to a sub-matrix (sub-graph assignment)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle (\mathbf{i},\mathbf{j}) = \mathbf{C}(\mathbf{i},\mathbf{j}) \odot \mathbf{A}^T$
build (meth.)	Build a matrix from row, column, value tuples	$\mathbf{C} = \mathbb{S}^{mxn}(\mathbf{i},\mathbf{j},\mathbf{v},\odot)$
extractTuples (meth.)	Extract row, column, value tuples from a matrix	$(\mathbf{i},\mathbf{j},\mathbf{v}) = \mathbf{A}$

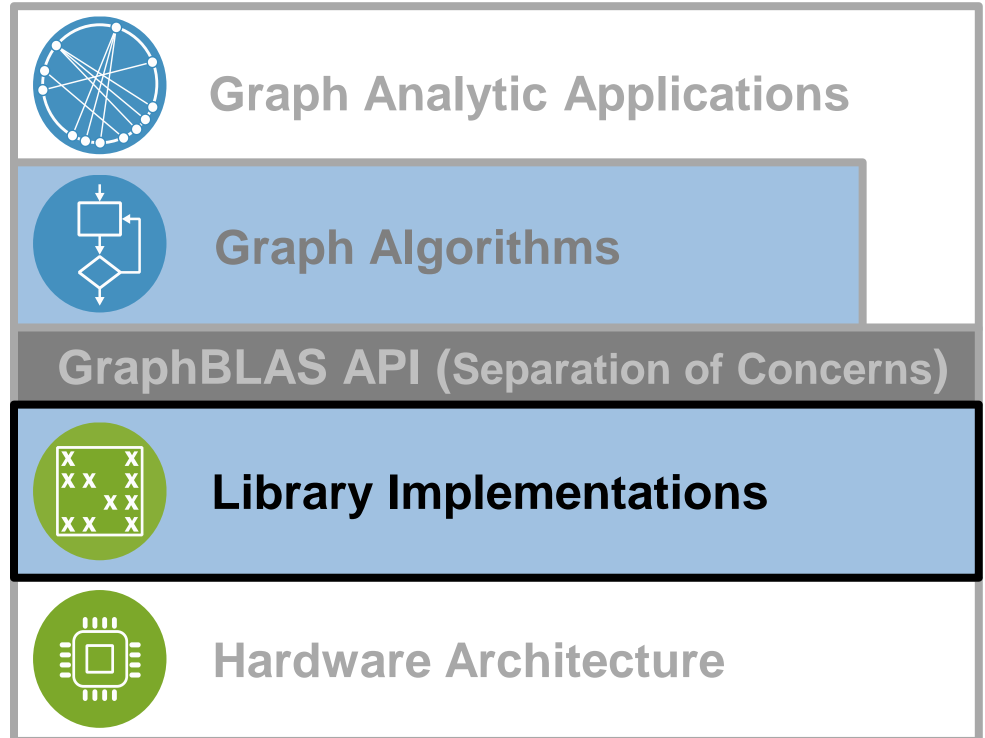
S. McMillan, et al., "Design and Implementation of the GraphBLAS Template Library (GBTL)," SIAM Annual Meeting (AN16), July 2016.
Updated IEEE HPEC Conference, Sep 2017.

Notation: **i**,**j** – index arrays, **v** – scalar array, **m** – 1D mask, **other bold-lower** – vector (column), **M** – 2D mask, **other bold-caps** – matrix, **T** – transpose, ¬ - structural complement, z – clear output, ⊕ monoid/binary function, ⊕.⊗ semiring, **blue** – optional parameters, **red** – optional modifiers

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Library Design Goals

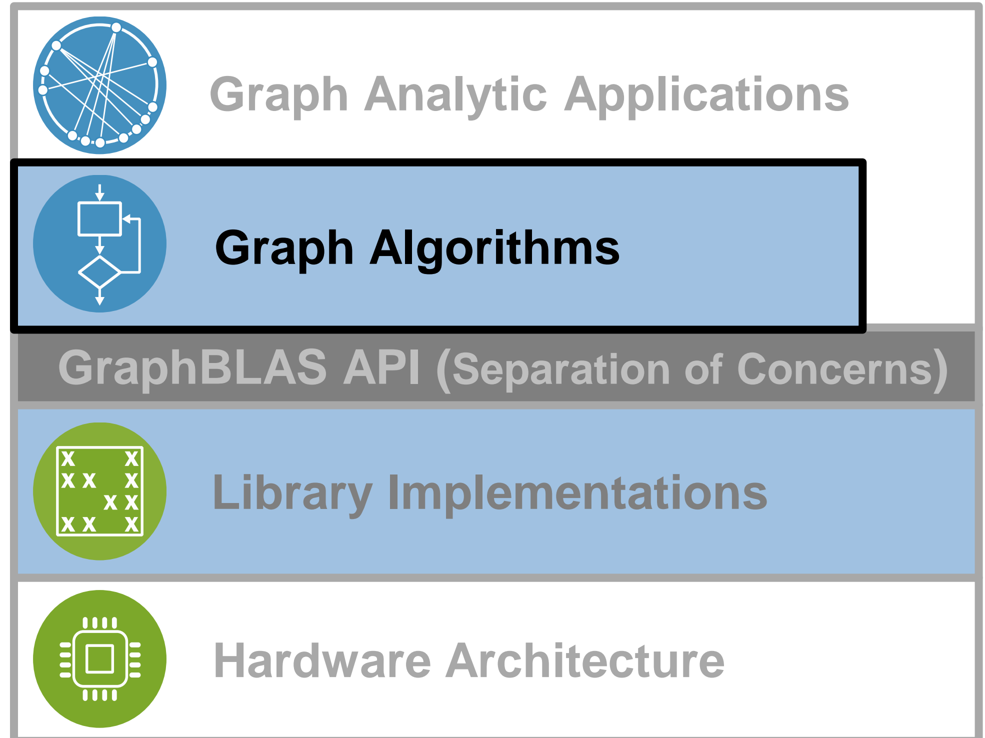
- GraphBLAS API
  - Achieve a ***separation of concerns*** between algorithm development and hardware optimizations
  - SuiteSparse/IBM C API
  - GBTL C++ “frontend”
  - pyGB python DSL
- Library Implementations
  - Tune primitives for specific architecture
  - SuiteSparse GraphBLAS (some opt.)
  - IBM GraphBLAS (reference)
  - GBTL “backend” (seq. CPU reference)



[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Library Design Goals

- GraphBLAS API
  - Achieve a ***separation of concerns*** between algorithm development and hardware optimizations
  - SuiteSparse/IBM C API
  - GBTL C++ “frontend”
  - pyGB python DSL
- Library Implementations
  - Tune primitives for specific architecture
  - SuiteSparse GraphBLAS (some opt.)
  - IBM GraphBLAS (reference)
  - GBTL “backend” (seq. CPU reference)
- Algorithms
  - Written as simply as possible without concern for optimization



[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Algorithms Implemented in GBTL

- Metrics: triangle count, diameter, radius, eccentricity, degree, etc.
- Traversals: Breadth-First Search (BFS), level and parent variants
- Shortest Path/Cost Minimization (SSSP, APSP)
- Maximal Independent Set
- Betweenness Centrality
- Connected Components
- Minimum Spanning Tree
- Clustering/Community Detection
- PageRank
- K-truss Enumeration, incidence and adjacency matrix variants
- **MaxFlow**

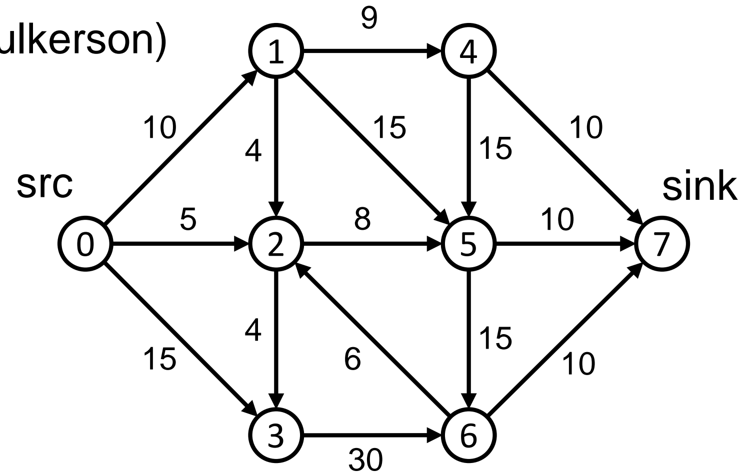
[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

“Stuff” I’ve learned while implementing algorithms…so far

- Semiring algebra looks like linear algebra…but isn’t…you will have to learn the math
  - Beware of non-commutative binary operators (e.g., division, subtraction, second)
  - There is a right way and a wrong way to perform subtraction
- Switching semirings in the middle of your algorithm can…
  - …lead to unexpected results
  - …happen without you realizing it
- Mask logic seems inconsistent but very useful for “annihilating” elements
  - Used in Maximal Independent Set, Maxflow, K-truss enumeration, etc.
- Getting parent IDs for BFS is not straightforward (points to possible future features)
- And a few other “tricks”

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm (derived from Ford-Fulkerson)



A – the directed adjacency matrix (orig. capacity)
F – the flow matrix (used capacity)
R – the residual matrix (remaining capacity)
P – adjacency matrix constituting edges in path from source to sink

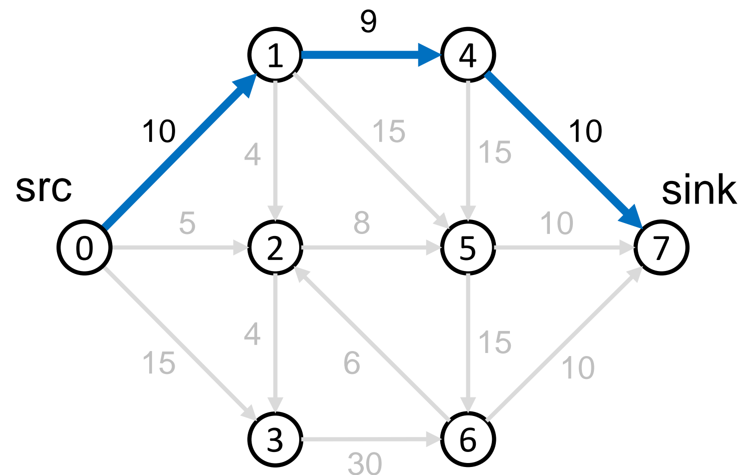
```
F = 0;     // flow initialized to zero
R = A;     // residual = capacity

while ("there is a path, P, in R from src to sink"){
    gamma = min(P .* R)
    F = F + gamma*P
    R = R - gamma*P
}
```

$$A = \begin{bmatrix} - & 10 & 5 & 15 & - & - & - & - \\ - & - & 4 & - & 9 & 15 & - & - \\ - & - & - & 4 & - & 8 & - & - \\ - & - & - & - & - & - & 30 & - \\ - & - & - & - & - & 15 & - & 10 \\ - & - & - & - & - & - & 15 & 10 \\ - & - & 6 & - & - & - & - & 10 \\ - & - & - & - & - & - & - & - \end{bmatrix}$$

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm

A – the directed adjacency matrix (orig. capacity)
F – the flow matrix (used capacity)
R – the residual matrix (remaining capacity)
P – adjacency matrix constituting
edges in path from source to sink

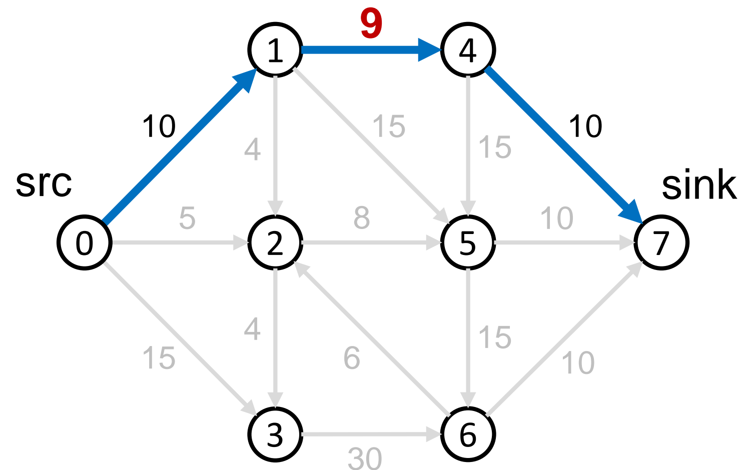



```
F = 0;    // flow initialized to zero
R = A;    // residual = capacity

while ("there is a path, P, in R from src to sink"){
    gamma = min(P .* R)
    F = F + gamma*P
    R = R - gamma*P
}
```


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm

A – the directed adjacency matrix (orig. capacity)
F – the flow matrix (used capacity)
R – the residual matrix (remaining capacity)
P – adjacency matrix constituting
    edges in path from source to sink

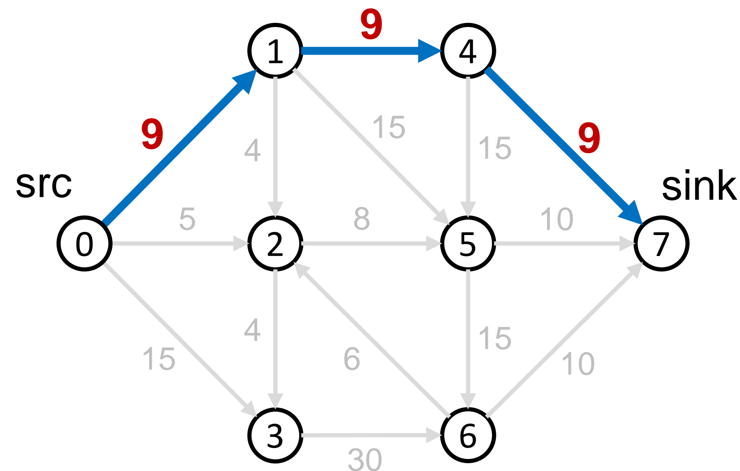



```
F = 0;     // flow initialized to zero
R = A;     // residual = capacity

while ("there is a path, P, in R from src to sink"){
    gamma = min(P .* R)
    F = F + gamma*P
    R = R - gamma*P
}
```


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm

A – the directed adjacency matrix (orig. capacity)
F – the flow matrix (used capacity)
R – the residual matrix (remaining capacity)
P – adjacency matrix constituting
edges in path from source to sink



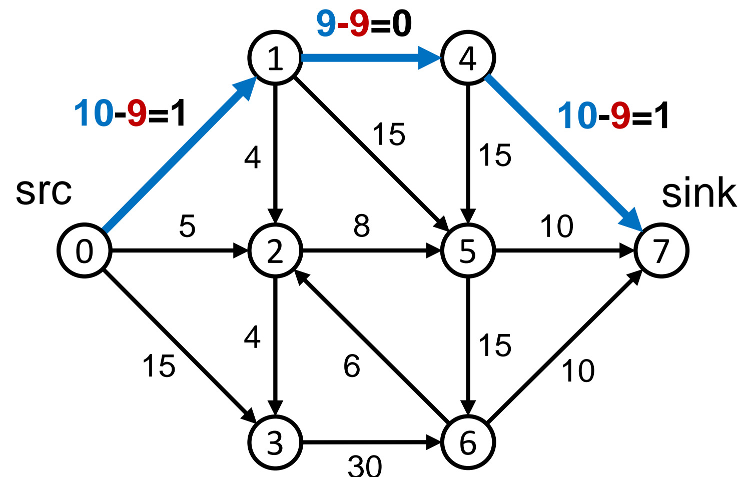

```
F = 0;     // flow initialized to zero
R = A;     // residual = capacity

while (“there is a path, P, in R from src to sink”){
    gamma = min(P .* R)
    F = F + gamma*P
    R = R - gamma*P
}
```


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm

A – the directed adjacency matrix (orig. capacity)
F – the flow matrix (used capacity)
R – the residual matrix (remaining capacity)
P – adjacency matrix constituting
edges in path from source to sink

```
F = 0;     // flow initialized to zero
R = A;     // residual = capacity

while ("there is a path, P, in R from src to sink"){
     gamma = min(P .* R)
     F = F + gamma*P
     R = R - gamma*P
}
```



[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Part 0: GraphBLAS API Primer

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

GraphBLAS Signatures: mxm

$$C\langle \neg M, z\rangle = C \odot (A^T \oplus.\otimes B^T)$$

```
// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix                C,
                 GrB_Matrix      const     M,
                 GrB_BinaryOp    const     accum,
                 GrB_Semiring    const     op,
                 GrB_Matrix      const     A,
                 GrB_Matrix      const     B,
                 GrB_Descriptor  const     desc);
```

```
// GBTL C++ API
namespace GraphBLAS
{
    template <typename CMatrixT,
              typename MaskT,
              typename AccumT,
              typename SemiringT,
              typename AMatrixT,
              typename BMatrixT>
    void mxm(CMatrixT           &C,
             MaskT        const &M,
             AccumT              accum,
             SemiringT           op,
             AMatrixT     const &A,
             BMatrixT     const &B,
             bool                replace_flag);
}
```


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

GraphBLAS Signatures: mxm

$$C\langle \neg M, z\rangle = C \odot (A^T \oplus . \otimes B^T)$$

```
// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix               C,
                 GrB_Matrix      const   M,
                 GrB_BinaryOp    const   accum,
                 GrB_Semiring    const   op,
                 GrB_Matrix      const   A,
                 GrB_Matrix      const   B,
                 GrB_Descriptor  const   desc);
```

- **C** stores the result
- **C** is also used as input if an optional accumulation operator ($\odot$) is specified.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

GraphBLAS Signatures: mxm

$$C\langle \neg M, z\rangle = C \odot (A^T \oplus . \otimes B^T)$$

```
// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix                C,
                 GrB_Matrix       const    M,
                 GrB_BinaryOp     const    accum,
                 GrB_Semiring     const    op,
                 GrB_Matrix       const    A,
                 GrB_Matrix       const    B,
                 GrB_Descriptor   const    desc);
```

- Mask, **M**, is optional.
- If not specified (GrB_NULL), the entire **C** matrix is overwritten.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

GraphBLAS Signatures: mxm

$$C\langle \neg M, z\rangle = C \odot (A^T \oplus . \otimes B^T)$$

```
// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix                C,
                 GrB_Matrix       const    M,
                 GrB_BinaryOp     const    accum,
                 GrB_Semiring     const    op,
                 GrB_Matrix       const    A,
                 GrB_Matrix       const    B,
                 GrB_Descriptor   const    desc);
```

- The accumulation operator, $\odot$, is optional.
- If not specified (GrB_NULL), the **C** matrix is used as output only (i.e., does not appear on the right hand side).

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

GraphBLAS Signatures: mxm

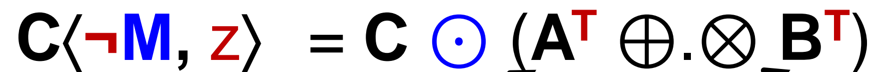
$$C\langle \neg M, z\rangle = C \odot (A^T \underbrace{\oplus . \otimes} B^T)$$

```
// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix               C,
                 GrB_Matrix       const   M,
                 GrB_BinaryOp     const   accum,
                 GrB_Semiring     const   op,
                 GrB_Matrix       const   A,
                 GrB_Matrix       const   B,
                 GrB_Descriptor   const   desc);
```

- The semiring used in the matrix multiply.
  - $\oplus$, a commutative monoid, replaces "plus"
  - $\otimes$, a binary operator, replaces "times"
- More on this is a minute…

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

GraphBLAS Signatures: mxm




```
// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix                C,
                 GrB_Matrix      const     M,
                 GrB_BinaryOp    const     accum,
                 GrB_Semiring    const     op,
                 GrB_Matrix      const     A,
                 GrB_Matrix      const     B,
                 GrB_Descriptor  const     desc);
```

- Input matrices

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

GraphBLAS Signatures: mxm

$$C\langle\neg M, z\rangle = C \odot (A^T \oplus.\otimes B^T)$$

```
// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix                   C,
                 GrB_Matrix       const   M,
                 GrB_BinaryOp     const   accum,
                 GrB_Semiring     const   op,
                 GrB_Matrix       const   A,
                 GrB_Matrix       const   B,
                 GrB_Descriptor const   desc);
```

- Optional Descriptor can specify any or all of the following:
  - Complement the mask, ¬
  - Clear the output matrix before writing final result, z
  - Transpose any input matrix, ^{T}

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm

A – the directed adjacency matrix (orig. capacity)
F – the flow matrix (used capacity)
R – the residual matrix (remaining capacity)
P – adjacency matrix constituting edges in path from source to sink

```
F = 0;    // flow initialized to zero
R = A;    // residual = capacity

while ("there is a path, P, in R from src to sink"){
    gamma = min(P .* R)
    F = F + gamma*P
    R = R - gamma*P
}
```

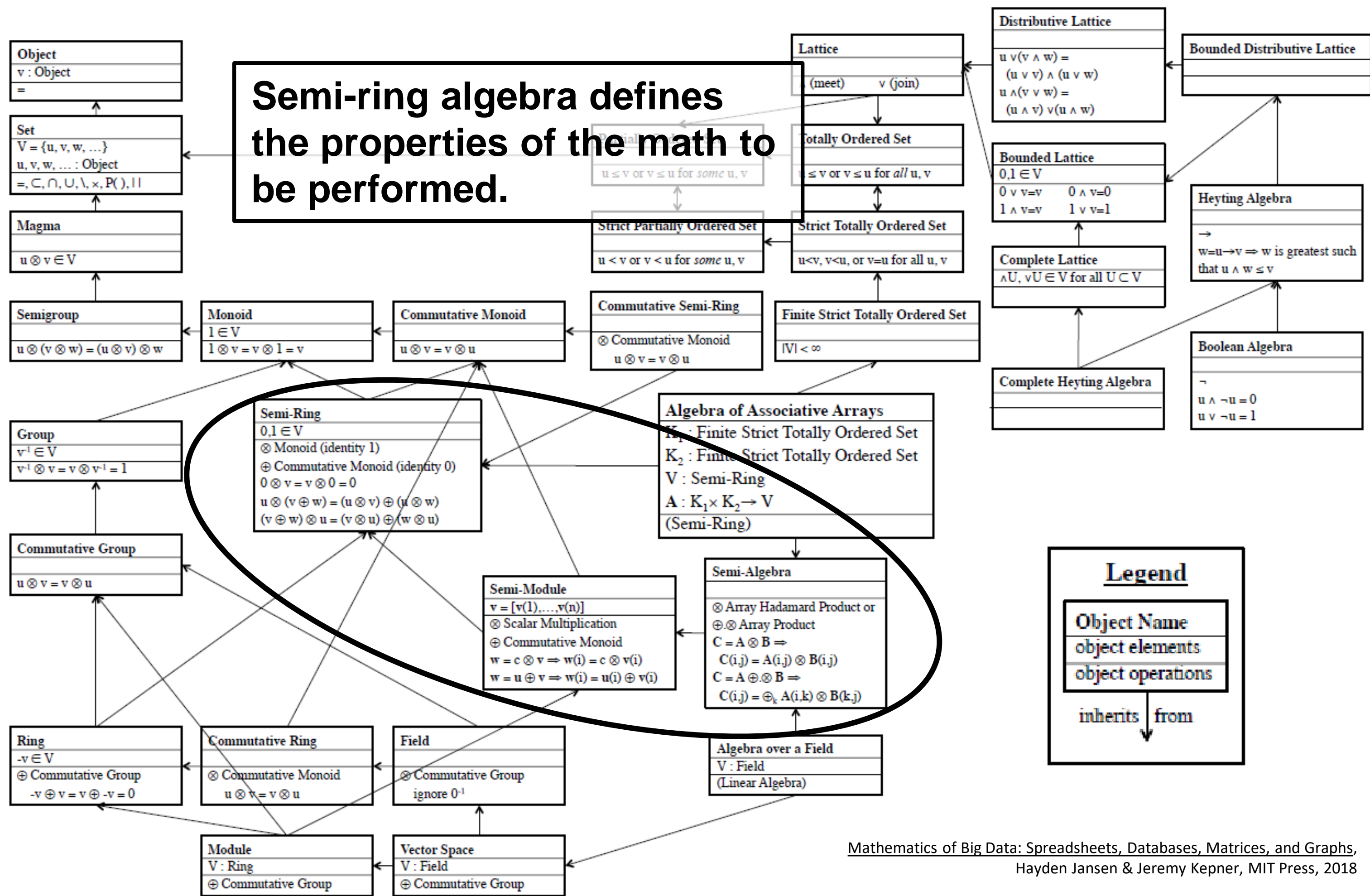
```
while (getPath(P, R, src, sink))
{
   // Compute gamma, minimum capacity on path, P
   // P is binary, R is double
   GrB_eWiseMult(PR, GrB_NULL, GrB_NULL,
                 GrB_TIMES_FP64, P, R, GrB_NULL);
   GrB_reduce(gamma, GrB_NULL,
              GrB_MIN_FP64, PR, GrB_NULL);

   // Compute gamma*P
   // ...
   //    GrB_UnaryOp scalarMultGamma(gamma);
   // ...
   GrB_apply(gammaP, GrB_NULL, GrB_NULL,
             scalarMultGamma, P, GrB_NULL);

   // Update Flow Graph:  F = F + gamma*P
   GrB_ewiseAdd(F, GrB_NULL, GrB_NULL,
                GrB_PLUS_FP64, F, gammaP, GrB_NULL);

   // Update Residual Graph: R = R - gamma*P
   GrB_eWiseAdd(R, GrB_NULL, GrB_NULL,
                GrB_MINUS_FP64, R, gammaP, GrB_NULL);
}
```


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Part 1: A Semiring Primer

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Algebra

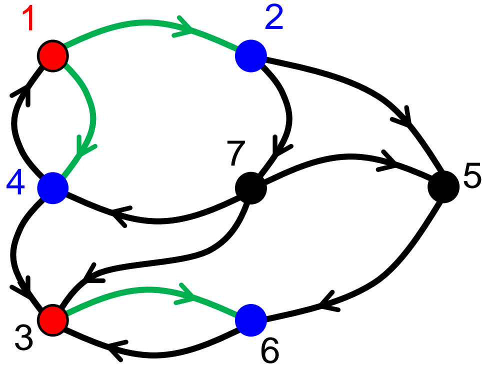


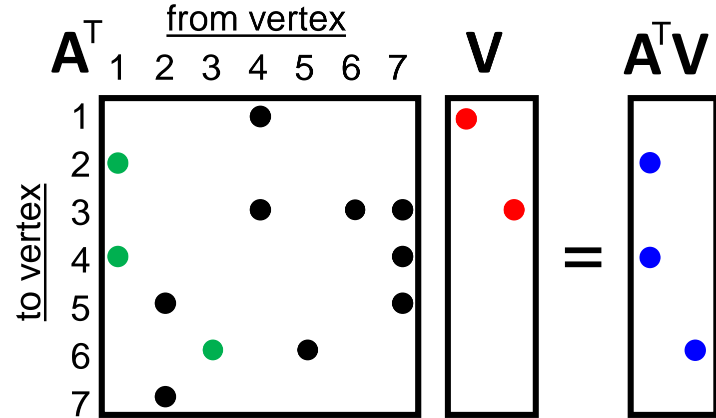
Mathematics of Big Data: Spreadsheets, Databases, Matrices, and Graphs, Hayden Jansen & Jeremy Kepner, MIT Press, 2018

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Matrix Multiply using Semirings

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





- The Semiring (⊕.⊗) determines how this computation is carried out.
- Consists of two Monoids (binary operator, identity)
  - ⊕, Commutative Monoid: e.g., (add, 0)
  - ⊗, Monoid: e.g., (multiply, 1)

where ⊕'s identity = ⊗'s annihilator

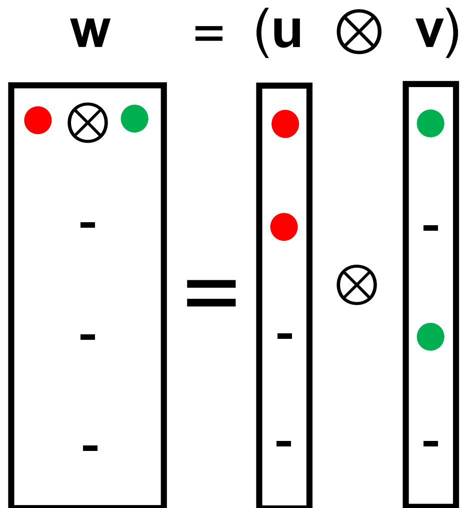
$$c_{i,j} = \sum_{l=1}^{k} a_{i,l} \times b_{l,j}$$

- In GraphBLAS:
  - ⊕, is a Commutative Monoid with identity (same as above)
  - ⊗, is a Binary Function (i.e., no identity required)
  - No enforcement that ⊕'s identity = ⊗'s annihilator, e.g. ⊕=(min, ∞) and ⊗='second'

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

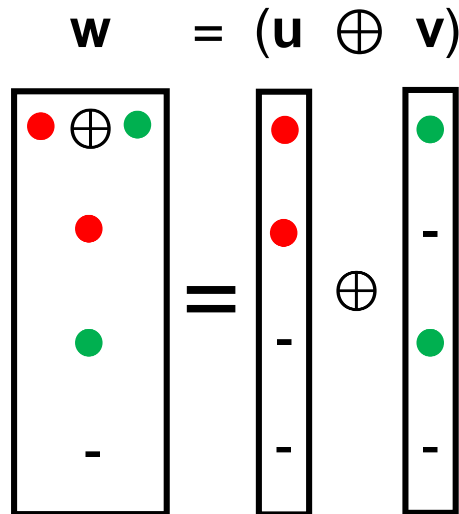
Behavior of the Binary Operators: ⊕ and ⊗

⊗ assumes unstored values (-) are the binary operator's ***annihilator***:



Examples: (x,0), (and, false), (+, ∞)

⊕ assumes unstored values (-) are the binary operator's ***identity***:



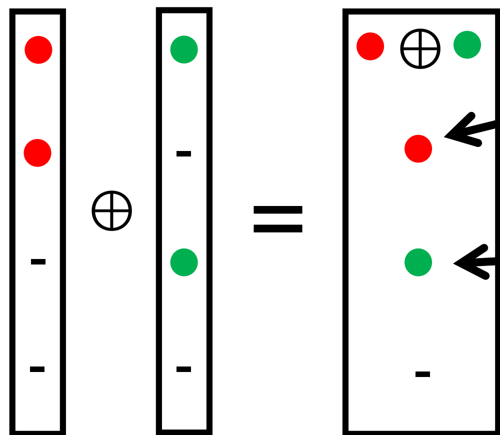
Examples: (+,0), (or, false), (min, ∞)

This also applies to eWiseMult and eWiseAdd

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Non-commutative operators with ⊕

- Non-commutative operators don't have an identity
- Specification does not enforce commutativity of ⊕ in mxm, mxv, vxm
- Specification allows the use of non-commutative operators in eWiseAdd and accum
- Almost always leads to "unintuitive" results
- Examples: Second, Minus



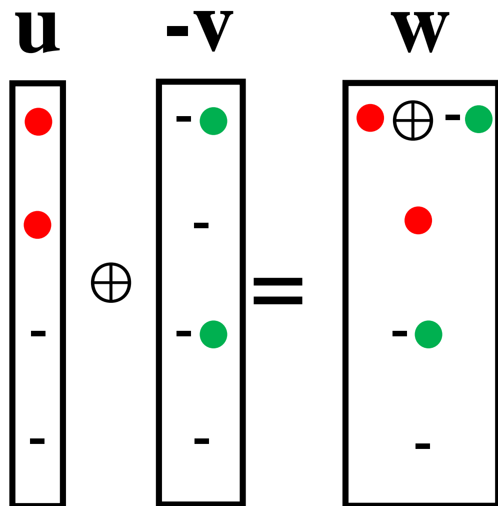
Is this incorrect for ⊕ = "Second"?
Should it be "-".

Incorrect for ⊕ = "Minus".
It should be "-●"

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Performing arithmetic subtraction

- Use “apply” operation with additive inverse (GrB_AINV) to the second argument
  - Apply is similar to ⊗ semantics (no identity or commutativity required).
- Use “eWiseAdd” operation with addition (GrB_PLUS)




```
// Compute the additive inverse of v
GrB_apply(negv, GrB_NULL, GrB_NULL,
          GrB_AINV, v, GrB_NULL);

// Add the vectors
GrB_eWiseAdd(w, GrB_NULL, GrB_NULL,
             GrB_PLUS_FP64, u, negv,
             GrB_NULL);
```

negv = *AINV*(**v**)

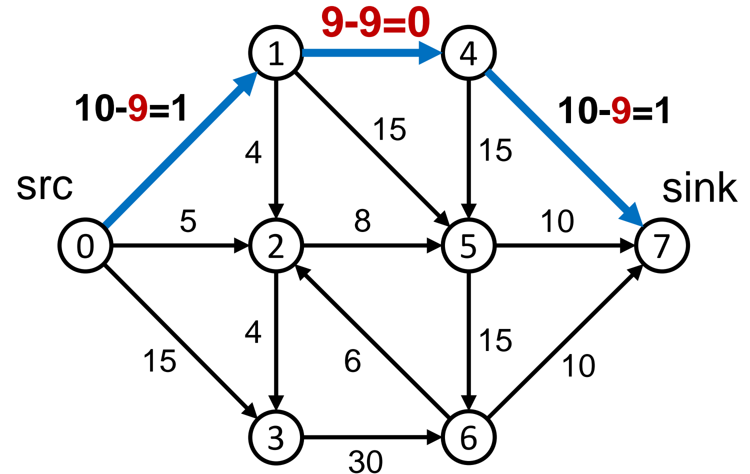
w = u + negv

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm (revisited)

A – the directed adjacency matrix (orig. capacity)
F – the flow matrix (used capacity)
R – the residual matrix (remaining capacity)
P – adjacency matrix constituting
edges in path from source to sink

```
F = 0;     // flow initialized to zero
R = A;     // residual = capacity

while (“there is a path, P, in R from src to sink”){
    gamma = min(P .* R)
    F = F + gamma*P
    R = R - gamma*P
}
```

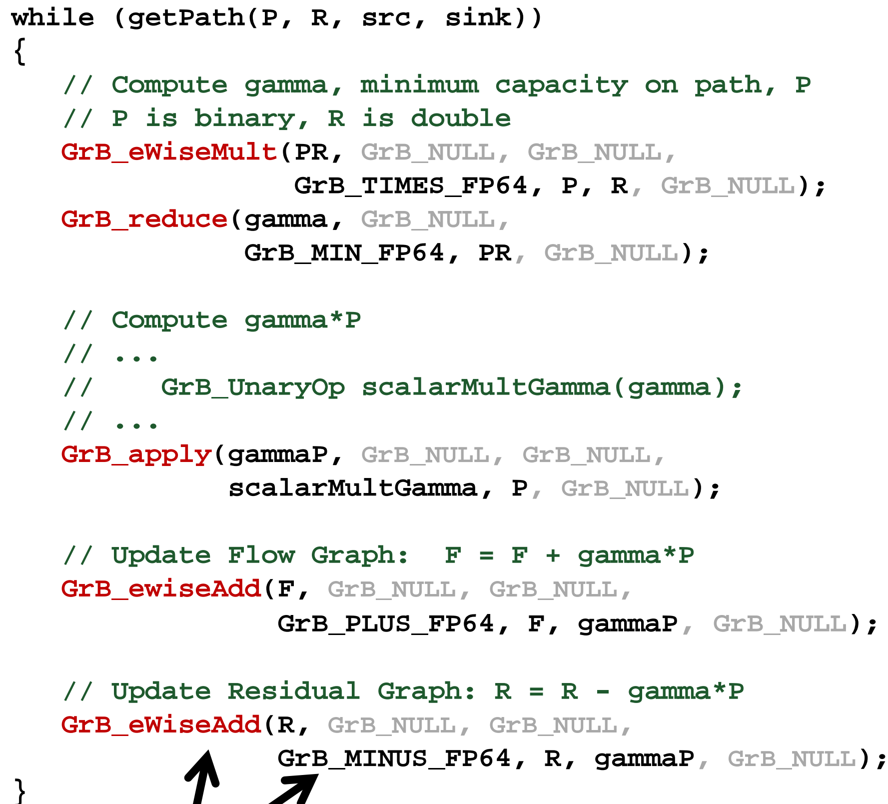


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm (revisited)

A – the directed adjacency matrix (orig. capacity)
F – the flow matrix (used capacity)
R – the residual matrix (remaining capacity)
P – adjacency matrix constituting
edges in path from source to sink

```
F = 0;     // flow initialized to zero
R = A;     // residual = capacity

while ("there is a path, P, in R from src to sink"){
    gamma = min(P .* R)
    F = F + gamma*P
    R = R - gamma*P
}
```



```
while (getPath(P, R, src, sink))
{
    // Compute gamma, minimum capacity on path, P
    // P is binary, R is double
    GrB_eWiseMult(PR, GrB_NULL, GrB_NULL,
                  GrB_TIMES_FP64, P, R, GrB_NULL);
    GrB_reduce(gamma, GrB_NULL,
               GrB_MIN_FP64, PR, GrB_NULL);

    // Compute gamma*P
    // ...
    //    GrB_UnaryOp scalarMultGamma(gamma);
    // ...
    GrB_apply(gammaP, GrB_NULL, GrB_NULL,
              scalarMultGamma, P, GrB_NULL);

    // Update Flow Graph:  F = F + gamma*P
    GrB_ewiseAdd(F, GrB_NULL, GrB_NULL,
                 GrB_PLUS_FP64, F, gammaP, GrB_NULL);

    // Update Residual Graph: R = R - gamma*P
    GrB_eWiseAdd(R, GrB_NULL, GrB_NULL,
                 GrB_MINUS_FP64, R, gammaP, GrB_NULL);
}
```


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm (revisited)

A – the directed adjacency matrix (orig. capacity)
F – the flow matrix (used capacity)
R – the residual matrix (remaining capacity)
P – adjacency matrix constituting
edges in path from source to sink

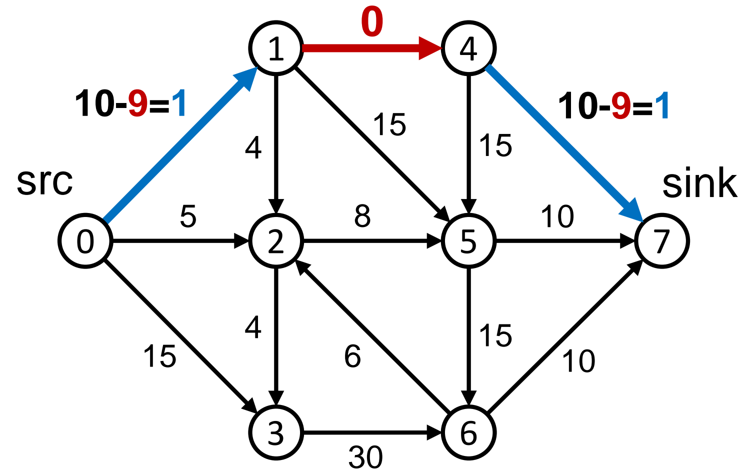
```
F = 0;     // flow initialized to zero
R = A;     // residual = capacity

while ("there is a path, P, in R from src to sink"){
    gamma = min(P .* R)
    F = F + gamma*P
    R = R + (-gamma*P)
}
```

```
while (getPath(P, R, src, sink))
{
    // Compute gamma, minimum capacity on path, P
    // P is binary, R is double
    GrB_eWiseMult(PR, GrB_NULL, GrB_NULL,
                  GrB_TIMES_FP64, P, R, GrB_NULL);
    GrB_reduce(gamma, GrB_NULL,
               GrB_MIN_FP64, PR, GrB_NULL);

    // Compute gamma*P
    // ...
    //    GrB_UnaryOp scalarMultGamma(gamma);
    // ...
    GrB_apply(gammaP, GrB_NULL, GrB_NULL,
              scalarMultGamma, P, GrB_NULL);

    // Update Flow Graph:  F = F + gamma*P
    GrB_ewiseAdd(F, GrB_NULL, GrB_NULL,
                 GrB_PLUS_FP64, F, gammaP, GrB_NULL);

    // Update Residual Graph: R = R + (-gamma*P)
    GrB_apply(R, GrB_NULL, GrB_PLUS_FP64,
              GrB_AINV_FP64, gammaP, GrB_NULL);
}
```


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm (revisited)

A – the directed adjacency matrix (orig. capacity)
F – the flow matrix (used capacity)
R – the residual matrix (remaining capacity)
P – adjacency matrix constituting
edges in path from source to sink




```
F = 0;     // flow initialized to zero
R = A;     // residual = capacity

while ("there is a path, P, in R from src to sink"){
    gamma = min(P .* R)
    F = F + gamma*P
    R = R + (-gamma*P)
}
```

$$
R = \begin{bmatrix}
- & 1 & 5 & 15 & - & - & - & - \\
- & - & 4 & - & 0 & 15 & - & - \\
- & - & - & 4 & - & 8 & - & - \\
- & - & - & - & - & - & 30 & - \\
- & - & - & - & - & 15 & - & 1 \\
- & - & - & - & - & - & 15 & 10 \\
- & - & 6 & - & - & - & - & 10 \\
- & - & - & - & - & - & - & -
\end{bmatrix}
$$

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

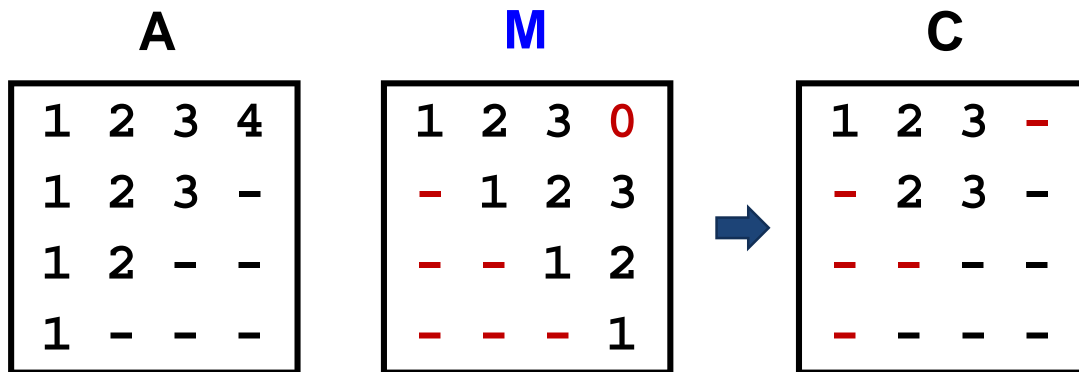
Part 2: A Mask Primer

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Masks

$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{A}$

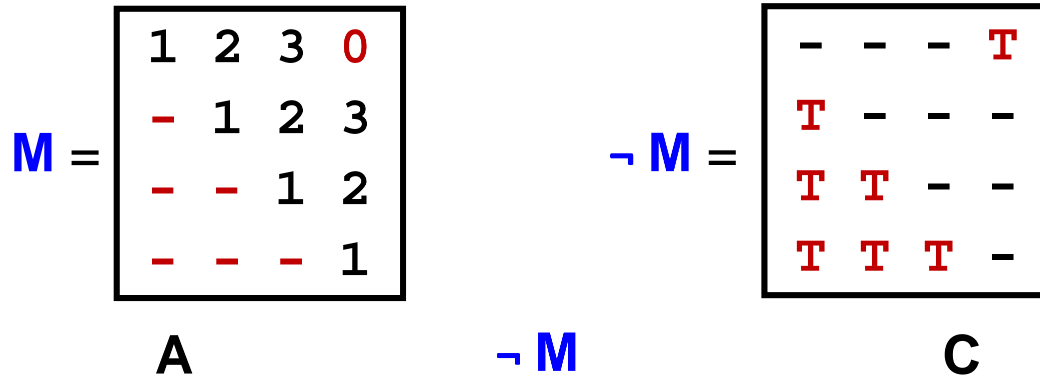
- Masks control what is written to the final result matrix/vector.
- Masks are passed as matrix/vector whose scalar type is convertible to bool.
- Elements corresponding to mask locations that "evaluate to true" are written to the output.
- Elements corresponding to mask locations that are not true are **annihilated**.

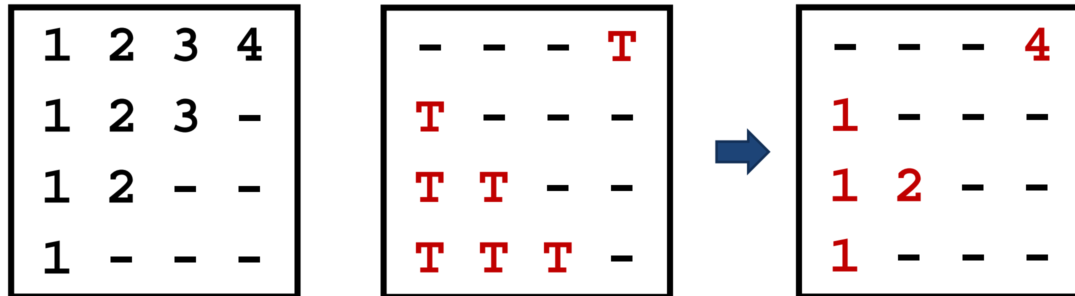


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Masks Structural Complement $C\langle\neg M\rangle = A$

- Inverts the logic determining which elements are written

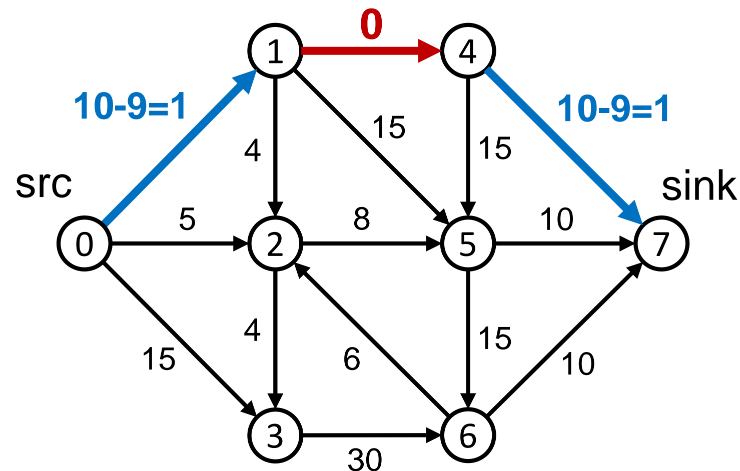




[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm (revisited)

$$
R = \begin{bmatrix}
- & 1 & 5 & 15 & - & - & - & - \\
- & - & 4 & - & 0 & 15 & - & - \\
- & - & - & 4 & - & 8 & - & - \\
- & - & - & - & - & - & 30 & - \\
- & - & - & - & - & 15 & - & 1 \\
- & - & - & - & - & - & 15 & 10 \\
- & - & 6 & - & - & - & - & 10 \\
- & - & - & - & - & - & - & -
\end{bmatrix}
$$

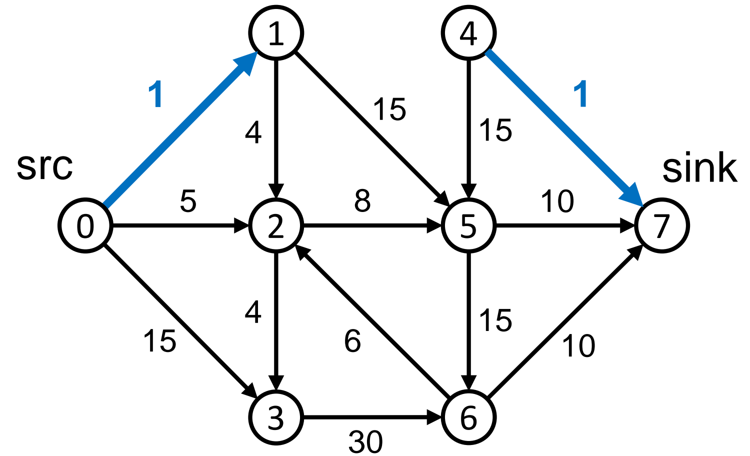


- Any zero elements in R need to be removed
- Use R as a Mask to annihilate zero edges

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm (revisited)

$$R = \begin{bmatrix} - & 1 & 5 & 15 & - & - & - & - \\ - & - & 4 & - & - & 15 & - & - \\ - & - & - & 4 & - & 8 & - & - \\ - & - & - & - & - & - & 30 & - \\ - & - & - & - & - & 15 & - & 1 \\ - & - & - & - & - & - & 15 & 10 \\ - & - & 6 & - & - & - & - & 10 \\ - & - & - & - & - & - & - & - \end{bmatrix}$$




```
// Annihilate zero entries from self
GrB_apply(R, R, GrB_NULL, GrB_IDENTITY_FP64, R, GrB_NULL);
```


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Part 3: Operating on structure

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm (revisited)

A – the directed adjacency matrix (orig. capacity)
F – the flow matrix (used capacity)
R – the residual matrix (remaining capacity)
P – adjacency matrix constituting
edges in path from source to sink

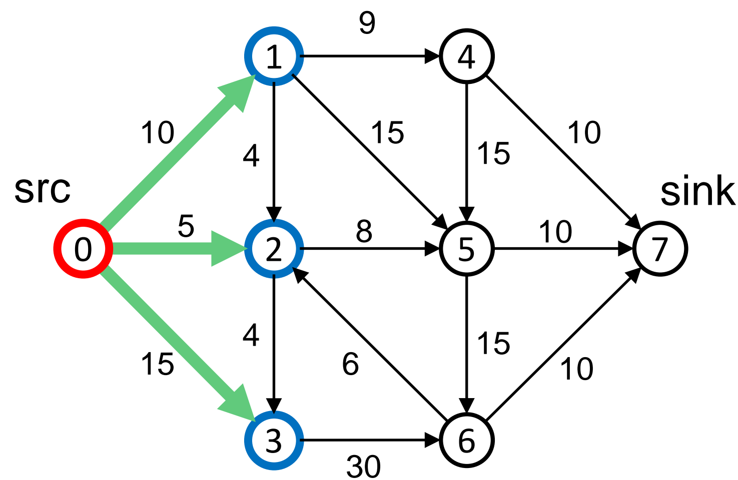
```
F = 0;     // flow initialized to zero
R = A;     // residual = capacity

while (“there is a path, P, in R from src to sink”){
    gamma = min(P .* R)
    F = F + gamma*P
    R = R + (-gamma*P)
}
```

```
while (getPath(P, R, src, sink))
{
    // Compute gamma, minimum capacity on path, P
    // P is binary, R is double
    GrB_eWiseMult(PR, GrB_NULL, GrB_NULL,
                  GrB_TIMES_FP64, P, R, GrB_NULL);
    GrB_reduce(gamma, GrB_NULL,
               GrB_MIN_FP64, PR, GrB_NULL);

    // Compute gamma*P
    // ...
    //    GrB_UnaryOp scalarMultGamma(gamma);
    // ...
    GrB_apply(gammaP, GrB_NULL, GrB_NULL,
              scalarMultGamma, P, GrB_NULL);

    // Update Flow Graph:  F = F + gamma*P
    GrB_ewiseAdd(F, GrB_NULL, GrB_NULL,
                 GrB_PLUS_FP64, F, gammaP, GrB_NULL);

    // Update Residual Graph: R = R + (-gamma*P)
    GrB_apply(R, GrB_NULL, GrB_PLUS_FP64,
              GrB_AINV_FP64, gammaP, GrB_NULL);

    // Annihilate zero entries from self
    GrB_apply(R, R, GrB_NULL,
              GrB_IDENTITY_FP64, R, GrB_NULL);
}
```


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Using BFS to find the path…

- Goal: compute the parent vertex ID’s
- What values do you put in the wavefront, $\mathbf{w}_i$?
- What semiring should be used for mxv?



$$\mathbf{R}^T \oplus.\otimes \mathbf{w}_0 = \mathbf{w}_1$$

$\mathbf{R}^T$:

-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-
5	4	-	-	-	-	6	-
15	-	4	-	-	-	-	-
-	9	-	-	-	-	-	-
-	15	8	-	15	-	-	-
-	-	-	30	-	15	-	-
-	-	-	-	10	10	10	-

$\mathbf{w}_0$: o, -, -, -, -, -, -, -

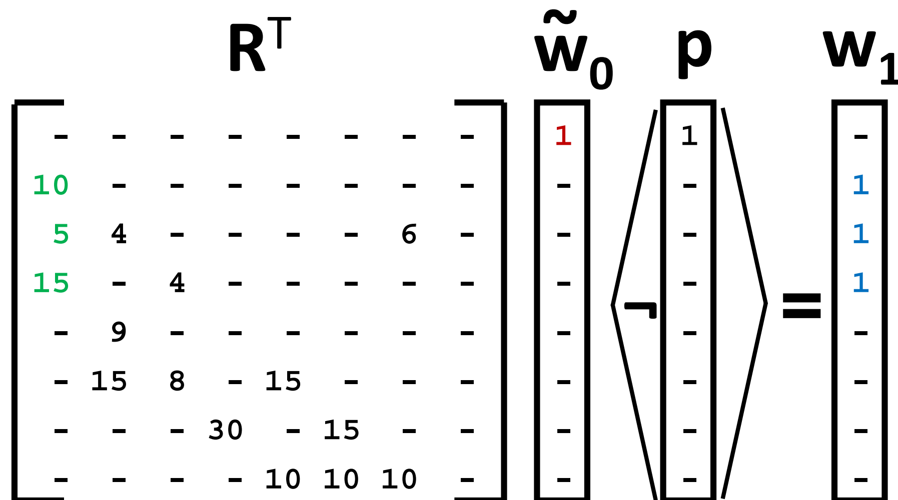
$\mathbf{w}_1$: -, o, o, o, -, -, -, -

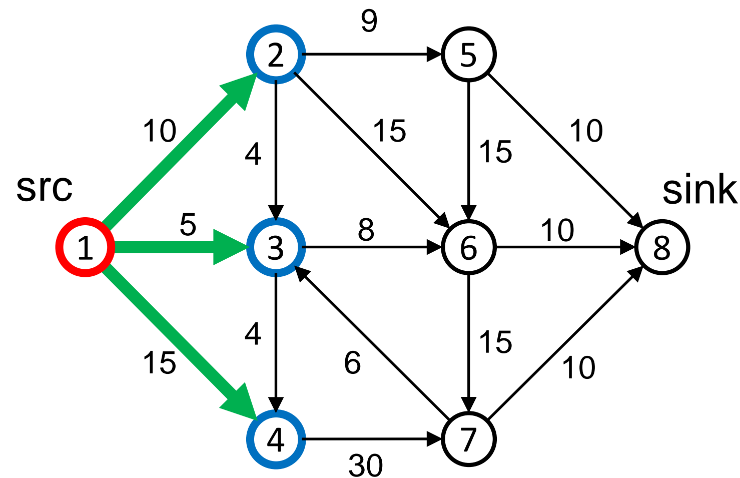
$$\mathbf{w}_{i+1} = \mathbf{R}^T \mathbf{w}_i$$

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

CombBLAS Approach…

- "Semiring:" $\oplus.\otimes$ = Min.Second
- $\tilde{\mathbf{w}}_i$ contains **1-based** index of stored values in $\mathbf{w}_i$ (indexOf).
- Use parent list, **p**, as mask (complemented)





while ($\mathbf{w}_i$ not empty):

$\tilde{\mathbf{w}}_i$ = "indexOf"($\mathbf{w}_i$)

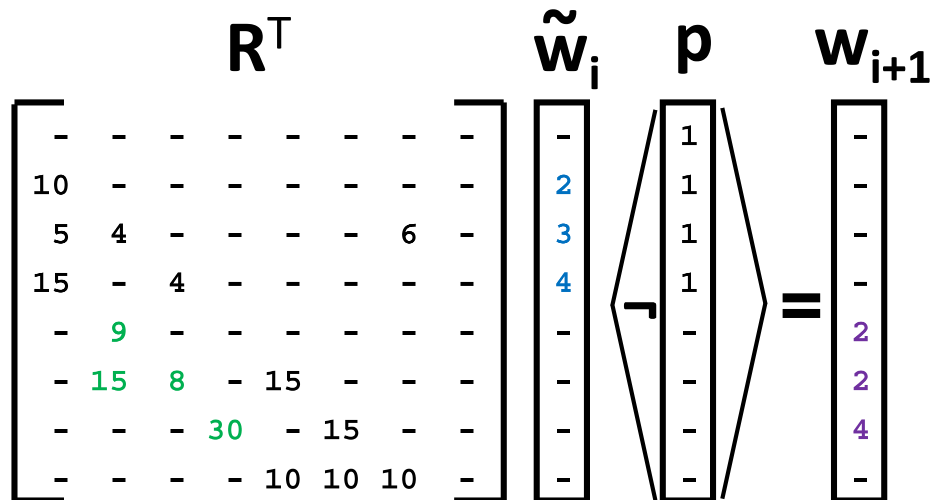
$\mathbf{w}_{i+1}\langle\neg\mathbf{p}\rangle = \mathbf{R}^T \tilde{\mathbf{w}}_i$

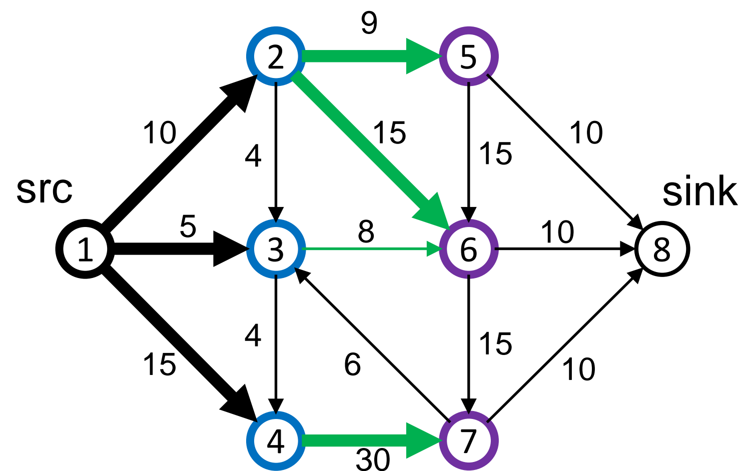
$\mathbf{p} \mathrel{+}= \mathbf{w}_{i+1}$

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

CombBLAS Approach…

- "Semiring:" $\oplus.\otimes$ = Min.Second
- $\mathbf{w_i}$ contains **1-based** index of stored values in $\mathbf{w_i}$ (indexOf).
- Use parent list, **p**, as mask (complemented)





while ($\mathbf{w_i}$ not empty):

$\tilde{\mathbf{w}}_i$ = "indexOf"($\mathbf{w_i}$)

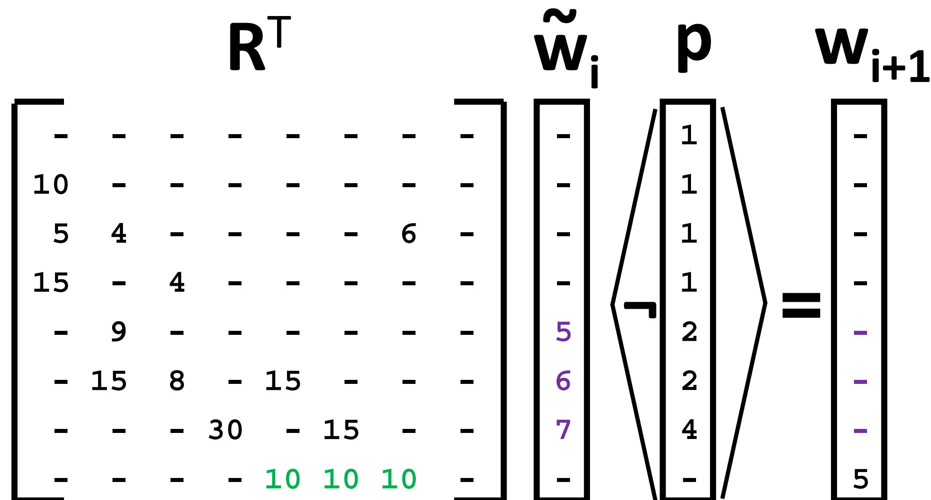
$\mathbf{w_{i+1}}$<¬**p**> = $\mathbf{R}^T \tilde{\mathbf{w}}_i$

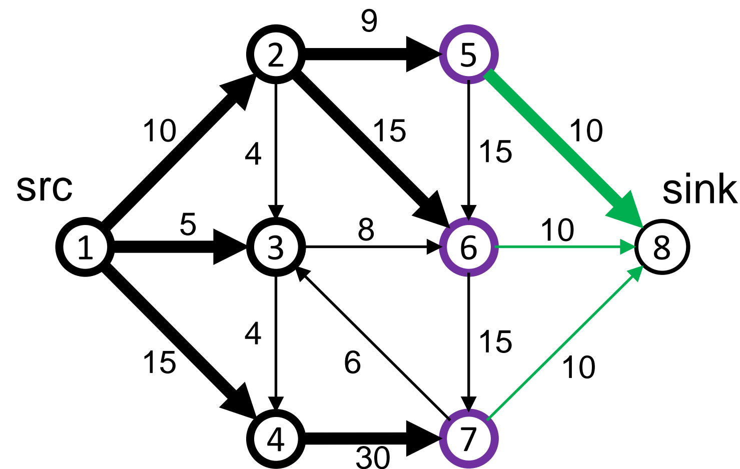
p += $\mathbf{w_{i+1}}$

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

CombBLAS Approach…

- "Semiring:" $\oplus.\otimes$ = Min.Second
- $\mathbf{w_i}$ contains **1-based** index of stored values in $\mathbf{w_i}$ (indexOf).
- Use parent list, **p**, as mask (complemented)





while ($\mathbf{w_i}$ not empty):

$\mathbf{\tilde{w}_i}$ = "indexOf"$(\mathbf{w_i})$

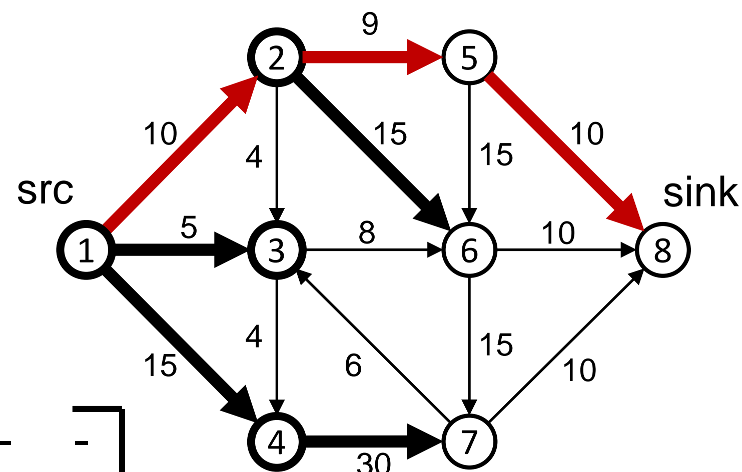
$\mathbf{w_{i+1}}$<¬**p**> = $\mathbf{R}^T \mathbf{\tilde{w}_i}$

p += $\mathbf{w_{i+1}}$

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Extracting the path and creating P.

- "Walk" parent list backward from sink (8) to source (1)
  - Extract an edge list,
  - Matrix_build() or Matrix_setElement() methods
  - Remember to restore 0-based indexing



p = [1, 1, 1, 1, 2, 2, 4, 5] ← 8

(1,2)

(2,5)

(5,8)

-	1	-	-	-	-	-	-
-	-	-	-	1	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	1
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

= P

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Alternative approaches to dealing with structure needed:

- New GraphBLAS method: “indexOf” operation (shown)
  - Replace stored values in vectors with their index locations
  - For Matrices: columnIndexOf and rowIndexOf
- Use a more complex data type that stores both scalar values and location
  - Uses more memory
  - More complex operators
- New GraphBLAS behavior: change the inner loop of matrix multiplication
  - Don’t just pass the two scalar values to the semirings binary operator
  - Also pass the scalars’ locations (index values)
  - Requires more complex operators

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Summary

- Semiring algebra looks like linear algebra…but isn't
  - Take care when using non-commutative binary operators for $\oplus$
  - Know when you are switching semirings in the middle of your algorithms and "clean up stored zeros" if necessary
- Mask logic seems inconsistent but very useful for "annihilating" elements
  - Used in Maxflow, Maximal Independent Set, K-truss enumeration, etc.
- Opportunity for extensions to the specification to deal with structure better
  - E.g., getting parent IDs for BFS, (e.g. triangle enumeration)
  - Structure only matrices (e.g. masks, triangle counting)

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Work in Progress

- Continue to explore new algorithms
- Create optimized backends for GBTL
  - Active run-time for distributed systems
  - Specialized hardware: GPU, FPGA, Logic-In-Memory, etc.
- GraphBLAS Unit Test Framework
  - Reasoned and Reasonable set of tests for GraphBLAS implementers
- Move from GBTL to a C++ Specification
- Developing higher level abstractions for expressing the primitives
  - Math-like Domain-Specific Language (e.g., pyGB)
  - Automated code generation for targeted hardware: Math → Machine Code
- Developing tutorials and coursework
  - Enlist more users and researchers to help inform new features and changes to the Spec.

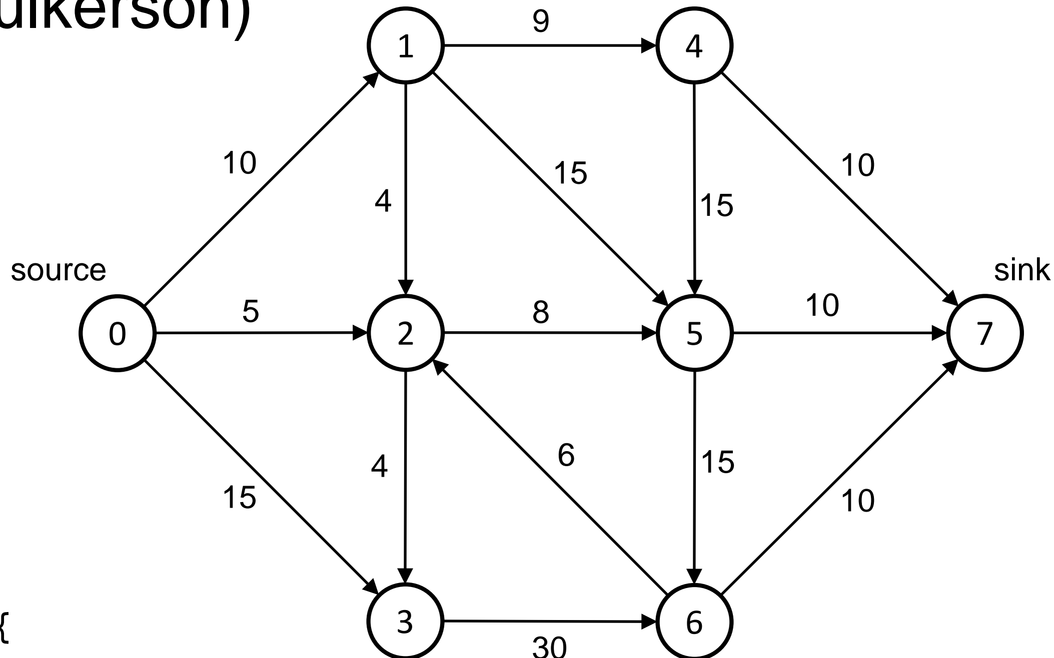
[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Backups

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm (Ford-Fulkerson)

Let F be the flow matrix
Let A be the adjacency matrix (or capacity)
Let R be the residual matrix
Let M be a mask matrix constituting
edges in path from source to sink.




```
R = A;       // capacity = adjacency
F = 0;

while (M = BFS(R, src, sink) && M != 0){
     gamma = min(M .* R)
     F = F + gamma(M - M')
     R = A - F                // Possible optimization R = R - gamma(M - M')
                              // or even R = R - (M .* gamma(M - M')) to avoid fill-in
}                             // or even R = R + (M .* gamma(M’ + (-M))
```


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

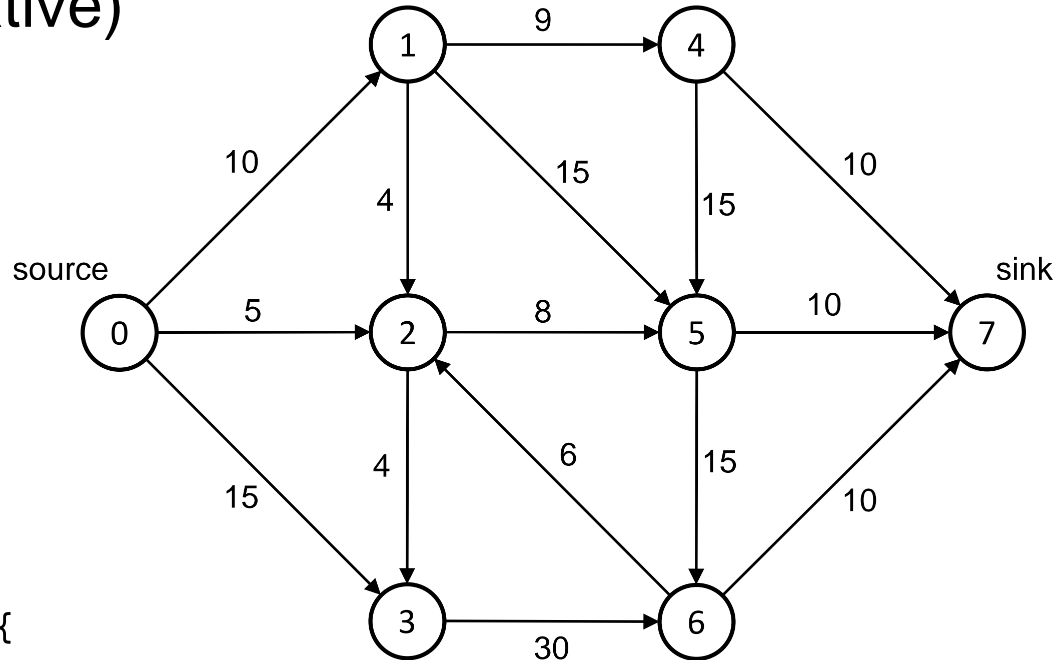
Maxflow in GBTL

```
double maxflow_v1(Matrix<double> const &graph, IndexType src, IndexType sink)
{
  IndexType num_nodes(graph.nrows());
  Matrix<double> R(graph);                              // R = graph, initialized to capacity graph
  Matrix<double> F(num_nodes, num_nodes);               // F = 0
  Matrix<bool>   M(num_nodes, num_nodes);  Matrix<double> G(...), GM(...), mF(...);
  double gamma, flow;

  while (maxflow_BFS(R, source, sink, M)) { // iterate while path found to sink in residual
    // Step 3. Find the minimum capacity on the path: gamma = min(M .* R)
    double gamma;
    eWiseMult(G, NoMask, NoAccumulate, Times<double>, M, R);
    reduce(gamma, NoAccumulate, MinMonoid<double>, G);

    // Step 4. Apply minimum capacity as flow across entire path: F = F + gamma(M - M')
    BinaryOp_Bind2nd<double,Times<double>> multiply_gamma(gamma);
    apply(GM, NoMask, NoAccumulate, multiply_gamma, M);                    // GM = gamma * M
    apply(GM, NoMask, Plus<double>, AdditiveInverse<double>, transpose(GM)); // GM += (-GM')
    eWiseAdd(F, NoMask, NoAccumulate, Plus<double>, F, GM);                // F = F + (GM + (-GM'))

    // Step 5. Remove flow from capacity: R = graph - F
    apply(mF, NoMask, Plus<double>, AdditiveInverse<double>, F);           // mF = (-F)
    eWiseAdd(R, NoMask, NoAccumulate, Plus<double>, graph, mF);            // R = graph + (-F)

    // Step 5b. Remove zero edges: R<R> = R
    apply(R, R, NoAccumulate, Identity<double>, R);
  }

  Vector<double> sink_edges(num_nodes);
  extract(sink_edges, NoMask, NoAccumulate, F, AllIndices, sink);         // sink_edges = F(:,sink)
  reduce(flow, NoAccumulate, PlusMonoid<double>, sink_edges);
  return flow;
}
```


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow Algorithm (Alternative)

Let F be the flow matrix
Let A be the adjacency matrix (or capacity)
Let R be the residual matrix
Let M be a mask matrix constituting
edges in path from source to sink.




```
R = A;      // capacity = adjacency
F = 0;

while (M = BFS(R, src, sink) && M != 0){
    gamma = min(M .* R)
    F = F + gamma*M
    R += (-gamma*M)
}
```


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Maxflow in GBTL (alternative)

```
double maxflow_v2(Matrix<double> const &graph, IndexType src, IndexType sink)
{
  IndexType num_nodes(graph.nrows());
  Matrix<double> R(graph);                              // R = graph, initialized to capacity graph
  Matrix<double> F(num_nodes, num_nodes);               // F = 0
  Matrix<bool>   M(num_nodes, num_nodes);  Matrix<double> G(...), GM(...), mF(...);
  double gamma, flow;

  while (maxflow_BFS(R, source, sink, M)) { // iterate while path found to sink in residual
    // Step 3. Find the minimum capacity on the path: gamma = min(M .* R)
    double gamma;
    eWiseMult(G, NoMask, NoAccumulate, Times<double>, M, R);
    reduce(gamma, NoAccumulate, MinMonoid<double>, G);

    // Step 4. Apply minimum capacity as flow across entire path: F = F + gamma(M - M')
    BinaryOp_Bind2nd<double,Times<double>> multiply_gamma(gamma);
    apply(GM, NoMask, NoAccumulate, multiply_gamma, M);                // GM = gamma * M

    eWiseAdd(F, NoMask, NoAccumulate, Plus<double>, F, GM);            // F = F + GM

    // Step 5. Remove flow from capacity: R = graph – F
    apply(R, NoMask, Plus<double>, AdditiveInverse<double>, GM);       // R += -(GM)


    // Step 5b. Remove zero edges: R<R> = R
    apply(R, R, NoAccumulate, Identity<double>, R);
  }

  Vector<double> sink_edges(num_nodes);
  extract(sink_edges, NoMask, NoAccumulate, F, AllIndices, sink);      // sink_edges = F(:,sink)
  reduce(flow, NoAccumulate, PlusMonoid<double>, sink_edges);
  return flow;
}
```


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

BFS for Maxflow in GBTL

```
double maxflow_BFS(Matrix<double> const &graph, IndexType src, IndexType sink, Matrix<bool> &M)
{
  IndexType num_nodes(graph.nrows());
  Vector<IndexType> parents(num_nodes), wavefront(num_nodes);

  wavefront.setElement(src, 1UL);
  parents.setElement(src, src + 1);

  // Step 1. Perform BFS until sink is reached, or not
  while (!parents.hasElement(sink) && wavefront.nvals() > 0) { // iterate until sink reached or BFS ends.
    // Encode vertex id (+1) in wavefront elements
    index_of_1based(wavefront);

    // MinSelect1st to find the minimum parent vertex id: wavefront'<~visited, REPLACE> = wavefront' min.first graph
    vxm(wavefront, complement(parents), NoAccumulate, MinSelect1st<IndexType,double,IndexType>, wavefront, graph, REPLACE);

    // Merge new parents in wavefront with existing parents and mark visited
    apply(parents, NoMask, Plus<IndexType>, Identity<IndexType>, wavefront);          // parents += (IndexType)wavefront
  }

  if (!parents.hasElement(sink)) return false;               // Early exit if sink not reached

  // Step 2. Find path from sink to src, mark it in M
  IndexType current = sink;
  while (current != src) {
    IndexType parent = parents.extractElement(current) - 1;                 // subtract 1 to account for index_of_1based() above
    M.setElement(parent, current, true);
    current = parent
  }

  return true;
}
```


[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.