# LAGraph Algorithms

PDF build on Thursday 11$^{\text{th}}$ February, 2021 at 00:52 UTC

**Abstract**

Theoretical documentation for LAGraph.

# 1 Introduction

The goal of this document is to present a notation for GraphBLAS algorithms and showcase it using important GraphBLAS algorithms.

# 2 The GraphBLAS

**Goal** The goal of GraphBLAS is to create a layer of abstraction between the graph algorithms and the graph analytics framework, separating the concerns of the algorithm developers from those of the framework developers and hardware designers. To achieve this, it builds on the theoretical framework of matrix operations on arbitrary semirings [3], which allows defining graph algorithms in the language of linear algebra [4]. To ensure portability, the GraphBLAS standard defines a C API that can be implemented on a variety of hardware including GPUs.

**Data structures** A graph with $n$ vertices can be stored as a square adjacency matrix $\mathbf{A} \in \mathbb{N}^{n \times n}$, where rows and columns both represent vertices of the graph and element $A(i, j)$ contains the number of edges from vertex $i$ to vertex $j$. If the graph is undirected, the matrix is symmetric.

**Navigation** The fundamental step in GraphBLAS is the multiplication of an adjacency matrix with another matrix or vector over a selected semiring. For example, the operation $\mathbf{HasMember} \ \mathsf{lor.land} \ \mathbf{IsLocatedIn}$ computed over the *"logical or.logical and"* semiring returns a matrix representing the Places where a Forum's members are located in. Meanwhile, when computed over the conventional arithmetic *"plus.times"* semiring, $\mathbf{HasMember} \oplus.\otimes \mathbf{IsLocatedIn}$ also returns the number of such Persons. A traversal from a certain set of vertices can be expressed by using a boolean vector $\mathbf{f}$ (often referred to as the *frontier*, *wavefront*, or *queue*) and setting `true` values for the elements corresponding to source vertices. For example, for Forums $\mathbf{f} \in \mathbb{B}^{|forums|}$, $\mathbf{f} \ \mathsf{lor.land} \ \mathbf{HasMember}$ returns the Persons who belong to any of the forums in $\mathbf{f}$. The BFS navigation step can also be captured using other semirings such as $\mathsf{lor.first}$, where $\mathsf{first}(x, y) = x$; $\mathsf{lor.second}$, where $\mathsf{second}(x, y) = y$; and $\mathsf{any.pair}$, where $\mathsf{any}(x, y)$ returns either $x$ or $y$, and $\mathsf{pair}(x, y) = 1$ [2].

## 2.1 Notation

Table 1 contains the notation of GraphBLAS operations Additionally, we use $\mathbf{D} = \mathrm{diag}(j, n)$ to construct a diagonal matrix $\mathbf{D} \leftarrow \{j, j, [1, 1, \ldots, 1]\}$. The elements of the matrix are $\mathbf{D}(j, j) = 1$ for $j \in j$.

### 2.1.1 Masks

Masks $\mathbf{C}\langle\mathbf{M}\rangle$ and $\mathbf{w}\langle\mathbf{m}\rangle$ are used to selectively write to the result matrix/vector. The complements of the masks can be selected with the negation symbol, denoted with: $\mathbf{C}\langle\neg\mathbf{M}\rangle$ and $\mathbf{w}\langle\neg\mathbf{m}\rangle$, respectively.

Masks with "replace" semantics (annihilating all elements outside the mask) are denoted with

- $\mathbf{C}\langle\mathbf{M}, \mathrm{r}\rangle$
- $\mathbf{C}\langle\neg\mathbf{M}, \mathrm{r}\rangle$
- $\mathbf{w}\langle\mathbf{m}, \mathrm{r}\rangle$
- $\mathbf{w}\langle\neg\mathbf{m}, \mathrm{r}\rangle$

The structure of the mask is denoted with:

| op./method | name | notation |
|---|---|---|
| `mxm` | matrix-matrix multiplication | $\mathbf{C}\langle\mathbf{M}\rangle\odot{=}\mathbf{A} \oplus.\otimes \mathbf{B}$ |
| `vxm` | vector-matrix multiplication | $\mathbf{w}\langle\mathbf{m}\rangle\odot{=}\mathbf{u} \oplus.\otimes \mathbf{A}$ |
| `mxv` | matrix-vector multiplication | $\mathbf{w}\langle\mathbf{m}\rangle\odot{=}\mathbf{A} \oplus.\otimes \mathbf{u}$ |
| `eWiseAdd` | element-wise addition<br>set union of patterns | $\mathbf{C}\langle\mathbf{M}\rangle\odot{=}\mathbf{A} \oplus \mathbf{B}$<br>$\mathbf{w}\langle\mathbf{m}\rangle\odot{=}\mathbf{u} \oplus \mathbf{v}$ |
| `eWiseMult` | element-wise multiplication<br>set intersection of patterns | $\mathbf{C}\langle\mathbf{M}\rangle\odot{=}\mathbf{A} \otimes \mathbf{B}$<br>$\mathbf{w}\langle\mathbf{m}\rangle\odot{=}\mathbf{u} \otimes \mathbf{v}$ |
| `extract` | extract submatrix<br>extract column vector<br>extract row vector<br>extract subvector | $\mathbf{C}\langle\mathbf{M}\rangle\odot{=}\mathbf{A}(i, j)$<br>$\mathbf{w}\langle\mathbf{m}\rangle\odot{=}\mathbf{A}(:, j)$<br>$\mathbf{w}\langle\mathbf{m}\rangle\odot{=}\mathbf{A}(i, :)$<br>$\mathbf{w}\langle\mathbf{m}\rangle\odot{=}\mathbf{u}(i)$ |
| `assign` | assign matrix to submatrix with mask for $\mathbf{C}$<br>assign scalar to submatrix with mask for $\mathbf{C}$<br>assign vector to subvector with mask for $\mathbf{w}$<br>assign scalar to subvector with mask for $\mathbf{w}$ | $\mathbf{C}\langle\mathbf{M}\rangle(i, j)\odot{=}\mathbf{A}$<br>$\mathbf{C}\langle\mathbf{M}\rangle(i, j)\odot{=}s$<br>$\mathbf{w}\langle\mathbf{m}\rangle(i)\odot{=}\mathbf{u}$<br>$\mathbf{w}\langle\mathbf{m}\rangle(i)\odot{=}s$ |
| `subassign (GxB)` | assign matrix to submatrix with submask for $\mathbf{C}(i, j)$<br>assign scalar to submatrix with submask for $\mathbf{C}(i, j)$<br>assign vector to subvector with submask for $\mathbf{w}(i)$<br>assign scalar to subvector with submask for $\mathbf{w}(i)$ | $\mathbf{C}(i, j)\langle\mathbf{M}\rangle\odot{=}\mathbf{A}$<br>$\mathbf{C}(i, j)\langle\mathbf{M}\rangle\odot{=}s$<br>$\mathbf{w}(i)\langle\mathbf{m}\rangle\odot{=}\mathbf{u}$<br>$\mathbf{w}(i)\langle\mathbf{m}\rangle\odot{=}s$ |
| `apply` | apply unary operator | $\mathbf{C}\langle\mathbf{M}\rangle\odot{=}\mathrm{f}(\mathbf{A})$<br>$\mathbf{w}\langle\mathbf{m}\rangle\odot{=}\mathrm{f}(\mathbf{u})$ |
| `select` | apply select operator | $\mathbf{C}\langle\mathbf{M}\rangle\odot{=}\mathbf{A}\langle\mathrm{f}(\mathbf{A}, k)\rangle$<br>$\mathbf{w}\langle\mathbf{m}\rangle\odot{=}\mathbf{u}\langle\mathrm{f}(\mathbf{u}, k)\rangle$ |
| `reduce` | reduce matrix to column vector<br>reduce matrix to scalar<br>reduce vector to scalar | $\mathbf{w}\langle\mathbf{m}\rangle\odot{=}[\oplus_j \mathbf{A}(:, j)]$<br>$s\odot{=}[\oplus_{ij} \mathbf{A}(i, j)]$<br>$s\odot{=}[\oplus_i \mathbf{u}(i)]$ |
| `transpose` | transpose | $\mathbf{C}\langle\mathbf{M}\rangle\odot{=}\mathbf{A}^{\mathsf{T}}$ |
| `kronecker` | Kronecker multiplication | $\mathbf{C}\langle\mathbf{M}\rangle\odot{=}\mathrm{kron}(\mathbf{A}, \mathbf{B})$ |
| `new` | new matrix<br>new vector | let: $\mathbf{A} \in \mathsf{TYPE}_{\mathsf{PRECISION}}^{n \times m}$<br>let: $\mathbf{u} \in \mathsf{TYPE}_{\mathsf{PRECISION}}^{n}$ |
| `build` | matrix from tuples<br>vector from tuples | $\mathbf{C} \hookleftarrow \{i, j, x\}$<br>$\mathbf{w} \hookleftarrow \{i, x\}$ |
| `extractTuples` | extract index/value arrays | $\{i, j, x\} \hookleftarrow \mathbf{A}$<br>$\{i, x\} \hookleftarrow \mathbf{u}$ |
| `dup` | duplicate matrix<br>duplicate vector | $\mathbf{C} \hookleftarrow \mathbf{A}$<br>$\mathbf{w} \hookleftarrow \mathbf{u}$ |
| `extractElement` | extract scalar element | $s = \mathbf{A}(i, j)$<br>$s = \mathbf{u}(i)$ |
| `setElement` | set element | $\mathbf{C}(i, j) = s$<br>$\mathbf{w}(i) = s$ |

Table 1: GraphBLAS operations and methods based on [1]. *Notation:* Matrices and vectors are typeset in bold, starting with uppercase ($\mathbf{A}$) and lowercase ($\mathbf{u}$) letters, respectively. Scalars including indices are lowercase italic ($k$, $i$, $j$) while arrays are lowercase bold italic ($\boldsymbol{x}$, $\boldsymbol{i}$, $\boldsymbol{j}$). $\oplus$ and $\otimes$ are the addition and multiplication operators forming a semiring and default to conventional arithmetic $+$ and $\times$ operators. $\odot$ is the accumulator operator.

- $\mathbf{C}\langle s(\mathbf{M})\rangle$
- $\mathbf{C}\langle \neg s(\mathbf{M})\rangle$
- $\mathbf{w}\langle s(\mathbf{m})\rangle$
- $\mathbf{w}\langle \neg s(\mathbf{m})\rangle$

Combining structure and replace semantics is possible:

- $\mathbf{C}\langle s(\mathbf{M}), \mathrm{r}\rangle$
- $\mathbf{C}\langle \neg s(\mathbf{M}), \mathrm{r}\rangle$
- $\mathbf{w}\langle s(\mathbf{m}), \mathrm{r}\rangle$
- $\mathbf{w}\langle \neg s(\mathbf{m}), \mathrm{r}\rangle$

Initializing scalars, vectors, and matrices (GraphBLAS methods):

- let: $s \in \mathbb{Q}_{64}$
- let: $\mathbf{w} \in \mathbb{Q}_{32}^{n}$
- let: $\mathbf{A} \in \mathbb{N}_{16}^{m \times n}$
- let: $\mathbf{A} \in \mathbb{Z}_{64}^{k \times m}$

# 3 Algorithms

LAGraph [5] implements graph algorithms using the GraphBLAS C API [6].

Here are a few algorithms that could be included in this document:

**Algorithm 1:** BFS / Levels variant.

**Input:** $\mathbf{A}$, $n$, $startVertex$

**Function** *BFS*
- $\mathbf{f}(startVertex) = \mathtt{T}$
- **for** $level = 1$ **to** $n - 1$ **do**
  - $\mathbf{s}\langle \mathbf{f}\rangle = level$
  - $\mathbf{f}\langle \neg \mathbf{s}, \mathrm{r}\rangle = \mathbf{f}\mathbf{A}$

**Algorithm 2:** BFS / Parents variant.

**Input:** $\mathbf{A}$, $n$, $startVertex$

**Function** *ParentsBFS*
- $\mathbf{f}(startVertex) = 0$
- **for** $level = 1$ **to** $n - 1$ **do**
  - $\mathbf{s}\langle \mathbf{f}\rangle = \mathbf{f}$
  - $\mathbf{f}\langle \neg \mathbf{s}, \mathrm{r}\rangle = \mathbf{f} \ \mathsf{any.firstj1} \ \mathbf{A}$

**Algorithm 3:** Direction Optimizing (push/pull) BFS.

**Input:** $\mathbf{A}, \mathbf{A}^{\mathsf{T}}, n, \mathit{startVertex}$

**Function** *DirectionOptimizingBFS*

- $\mathrm{f}(\mathit{startVertex}) = \mathtt{T}$
- **for** $\mathit{level} = 1$ **to** $n - 1$ **do**
  - $\mathrm{s}\langle \mathrm{f} \rangle = \mathit{level}$
  - **if** $\mathit{Push}(\mathbf{A}, \mathrm{f})$ **then**
    - $\mathrm{f}\langle \neg \mathrm{s}, \mathrm{r} \rangle = \mathrm{f}\mathbf{A}$
  - **else**
    - $\mathrm{f}\langle \neg \mathrm{s}, \mathrm{r} \rangle = \mathbf{A}^{\mathsf{T}}\mathrm{f}$

**Algorithm 4:** Multi-source breadth-first search.

**Data:** ...

**Result:** ...

**Function** *MSBFS*

- $\mathbf{Frontier} = \mathsf{diag}(\mathit{sources}, n)$
- **for** $\mathit{level} = 1$ ***to*** $n - 1$ **do**
  - $\mathbf{Seen}\langle \mathbf{Frontier} \rangle = \mathit{level}$
  - $\mathbf{Frontier}\langle \neg \mathbf{Seen}, \mathrm{r} \rangle = \mathbf{Frontier}\ \mathsf{any.pair}\ \mathbf{A}$

**Algorithm 5:** All-pairs shortest distance (on undirected, unweighted graphs) [7].

**Data:** ...

**Result:** ...

**Function** $\mathit{APD}(\mathbf{A}, n, \mathbf{deg})$

- $\mathbf{Z} = \mathbf{A}$
- $\mathbf{Z} \oplus= \mathbf{A} \oplus.\otimes \mathbf{A}$
- $\mathbf{B} = \{\langle \mathbf{Z}, \mathrm{offdiag} \rangle\}$ `// use the pattern as a Boolean matrix`
- **if** $\mathbf{A} == \mathbf{B}$ **then**
  - **return** $\mathbf{A}$
- $\mathbf{T} = \mathit{APD}(\mathbf{B}, n, \mathbf{deg})$
- $\mathbf{X} = \mathbf{T} \oplus.\otimes \mathbf{A}$
- $\mathbf{Tscaled} = \mathbf{T} \oplus.\otimes \mathrm{diag}(\mathbf{deg})$
- $\mathbf{Xfiltered} = \langle \mathbf{X}, \mathbf{X} < \mathbf{Tscaled} \rangle$
- **return** $(2 \otimes \mathbf{T}) \ominus \mathbf{Xfiltered}$

**Function** $\mathit{APD}(\mathbf{A})$

- $\mathbf{deg} = [\oplus_i \mathbf{A}(i, :)]$
- $\mathbf{Distance} = \mathit{APD}(\mathbf{A}, n, \mathbf{deg})$
- $\mathbf{sp} = [\oplus_i \mathbf{Distance}(i, :)]$

**Algorithm 6:** Betweenness centrality.

**Function** *MSBFS*

// The **NumSp** structure holds the number of shortest paths for each node and starting vertex discovered so far.
// Initialized to source vertices.
**NumSp** $\hookleftarrow \{s, [1, 1, \ldots, 1]\}$
// The **Frontier** holds the number of shortest paths for each node and starting vertex discovered so far.
// Initialized to source vertices.
**Frontier**$\langle$**NumSp**$\rangle = \mathbf{A}(s, :)$
$d = 0$
// The **Sigmas** matrices store frontier information for each level of the BFS phase.
// BFS phase (forward sweep)
**do**
  // **Sigmas**$[d](:, s) = d^{\text{th}}$ level frontier from source vertex $s$
  let: **Sigmas**$[d] \in \mathbb{B}^{n \times nsver}$
  **Sigmas**$[d](:, :) =$ **Frontier** // Convert matrix to Boolean
  **NumSp** $=$ **NumSp** $\oplus$ **Frontier** // Accumulate path counts
  **Frontier**$\langle$**NumSp**$, \mathrm{r}\rangle = \mathbf{A}^{\mathsf{T}} \oplus.\otimes$ **Frontier** // Update frontier
**while** $nvals$(**Frontier**) $> 0$
let: **NumSpInv** $\in \mathbb{Q}_{32}^{n \times nsver}$
**NumSpInv** $= 1.0 \oslash$ **NumSp**
let: **BCU** $\in \mathbb{Q}_{32}^{n \times nsver}$
**BCU**$(:) = 1.0$ // Make **BCU** dense, initialize all elements to $1.0$
let: **W** $\in \mathbb{Q}_{32}^{n \times nsver}$
// Tally phase (backward sweep)
**for** $i = d - 1$ **downto** 0 **do**
  **W**$\langle$**Sigmas**$[i], \mathrm{r}\rangle =$ **NumSpInv** $\oslash$ **BCU**
  **W**$\langle$**Sigmas**$[i-1], \mathrm{r}\rangle = \mathbf{A} \oplus.\otimes$ **W** // Add contributions by successors and mask with that BFS level's frontier.
  **BCU** $\oplus=$ **W** $\otimes$ **NumSp**
// Row reduce **BCU** and subtract $nsver$ from every entry to account for 1 extra value per **BCU** row element
**delta** $= [\oplus_j$ **BCU**$(:, j)]$
**delta** $\ominus= nsver$

**Algorithm 7:** PageRank (used in Graphalytics).

**Data:** $\alpha$ constant (damping factor)
**Result:** ...
**Function** *PageRank*
  **pr**$(:) = 1/n$
  **outdegrees** $= [\oplus_j \mathbf{A}(:, j)]$
  **for** $k = 1$ **to** $numIterations$ **do**
    **importance** $=$ **pr** $\oslash$ **outdegrees**
    **importance** $= \mathrm{times}($**importance**$, \alpha)$ // apply the $\mathrm{times}(x, s) = x \cdot s$ operator
    **importance** $=$ **importance** $\oplus.\otimes$ **A**
    **danglingVertexRanks**$\langle\neg$**outdegrees**$\rangle =$ **pr**$(:)$
    $totalDanglingRank = \frac{\alpha}{n} \otimes [\oplus_i$ **danglingVertexRanks**$(i)]$
    **pr** $= \frac{1-\alpha}{n} \oplus totalDanglingRank$
    **pr** $=$ **pr** $\oplus$ **importance**

**Algorithm 8:** Algebraic Bellman-Ford.

**Function** *SSSP*
- $\mathbf{d}(s) = 0$
- **for** $k = 1$ **to** $n - 1$ **do**
  - $\mathbf{d}' = \mathbf{d}$ min.plus $\mathbf{A}$
  - **if** $\mathbf{d}' == \mathbf{d}$ **then break**
  - $\mathbf{d} \hookleftarrow \mathbf{d}'$

**Algorithm 9:** Delta-stepping SSSP.

**Data:**
$\mathbf{A}, \mathbf{A_H}, \mathbf{A_L} \in \mathbb{Q}^{|V|\times|V|}$
$s, i \in \mathbb{N}$
$\Delta \in \mathbb{Q}$
$\mathbf{t}, \mathbf{t_{Req}} \in \mathbb{Q}^{|V|}$
$\mathbf{t_{B_i}}, \mathbf{e} \in \mathbb{N}^{|V|}$

**Function** *DeltaStepping*
- $\mathbf{A_L} = \langle 0 < \mathbf{A} \leq \Delta \rangle$
- $\mathbf{A_H} = \langle \Delta < \mathbf{A} \rangle$
- $\mathbf{t}(:) = \infty$
- $\mathbf{t}(s) = 0$
- **while** *nvals*$(\langle i\Delta \leq \mathbf{t} \rangle) \neq 0$ **do**
  - $s = 0$
  - $\mathbf{t_{B_i}} = \langle i\Delta \leq \mathbf{t} < (i+1)\Delta \rangle$
  - **while** $\mathbf{t_{B_i}} \neq 0$ **do**
    - $\mathbf{t_{Req}} = \mathbf{A_L^T} \oplus.\otimes (\mathbf{t} \otimes \mathbf{t_{B_i}})$
    - $\mathbf{e} = \langle 0 < \mathbf{e} \oplus \mathbf{t_{B_i}} \rangle$
    - $\mathbf{t_{B_i}} = \langle i\Delta \leq \mathbf{t_{Req}} < (i+1)\Delta \rangle \otimes (\mathbf{t_{Req}} \underset{\oplus}{\mathsf{min}} \mathbf{t})$
    - $\mathbf{t_{B_i}} = \langle i\Delta \leq \mathbf{t_{Req}} < (i+1)\Delta \rangle \otimes (\mathbf{t_{Req}} \oplus_{\mathsf{min}} \mathbf{t})$
    - $\mathbf{t_{B_i}} = \langle i\Delta \leq \mathbf{t_{Req}} < (i+1)\Delta \rangle \otimes (\mathbf{t_{Req}} \mathsf{min}_{\oplus} \mathbf{t})$
    - $\mathbf{t} = \mathbf{t} \;\mathsf{min}\; \mathbf{t_{Req}}$
  - $\mathbf{t_{Req}} = \mathbf{A_H^T} \oplus.\otimes (\mathbf{t} \otimes \mathbf{e})$
  - $\mathbf{t} = \mathbf{t} \;\mathsf{min}\; \mathbf{t_{Req}}$
  - $i = i + 1$

**Algorithm 10:** All-pairs shortest path (Floyd–Warshall algorithm).

**Function** *FloydWarshall*
- $\mathbf{D} \hookleftarrow \mathbf{A}$
- **for** $k = 1$ **to** $n$ **do**
  - $\mathbf{D} = \mathbf{D} \;\mathsf{min}[\mathbf{D}(:, k)$ min.plus $(\mathbf{D}(k, :)]$

**Algorithm 11:** FastSV algorithm.

```
Function FastSV
    n = nrows(A)
    gf = f
    dup = gf
    mngf = gf
    {i, x} ↤ f
    repeat
        // Step 1: Stochastic hooking
        mngf = mngf min A
        mngf = mngf second.min gf
        f(x) = f min mngf
        // Step 2: Aggressive hooking
        f = f min mngf
        // Step 3: Shortcutting
        f = f min gf
        // Step 4: Calculate grandparents
        {i, x} ↤ f
        gf = f(x)
        // Step 5: Check termination
        diff = dup ≠ gf
        sum = [⊕_i diff(i)]
        dup = gf
    until sum == 0
```

**Algorithm 12:** Triangle count (Cohen's algorithm).

```
Function TriangleCount
    L = tril(A)
    U = triu(A)
    B = L ⊕.⊗ U
    C = B ⊗ A
    t = [⊕_ij C(i, j)]/2
```

**Algorithm 13:** Triangle count (Sandia).

```
Function TriangleCount
    L = tril(A)
    C⟨L⟩ = L ⊕.⊗ L
    t = [⊕_ij C(i, j)]
```

**Algorithm 14:** Triangle count (FLAME).

```
Function TriangleCountFlame
    for i = 2 to n − 1 do
        A_20 = A(i + 1 : n, 0 : i − 1)
        a_10 = A(0 : i − 1, i)
        a_12 = A(i, i + 1 : n)
        t ⊕= a_10 ⊕.⊗ A_20 ⊕.⊗ a_12
```

**Algorithm 15:** Local clustering coefficient.

```
Function PageRank
    Tri⟨A⟩ = A ⊕.⊗ A                    // compute triangle count matrix
    tri = [⊕_j Tri(:, j)]               // reduce to triangle count vector
    deg = [⊕_j A(:, j)]                 // reduce to vertex degree vector
    wed = perm2(deg)                    // apply perm2(x) = x · (x − 1) to get wedge count vector
    lcc = tri ⊘ wed                     // LCC vector
```

**Algorithm 16:** $k$-truss algorithm.

**Function** *KTruss*
  $\mathbf{C} \leftarrow \mathbf{A}$
  $nonzeros \leftarrow \text{nvals}(C)$
  **for** $i = 1$ **to** $n - 1$ **do**
    $\mathbf{C}\langle\mathbf{C}\rangle = \mathbf{C} \oplus.\mathsf{land}\ \mathbf{C}$
    $\mathbf{C} = \langle\mathbf{C} \geq k - 2\rangle$
    **if** $nonzeros == nvals(C)$ **then**
      **break**
    $nonzeros \leftarrow \text{nvals}(C)$

**Algorithm 17:** Louvain algorithm (WIP).

**Function** *Louvain*
  $\mathbf{G} \oplus= \mathbf{G}^{\mathsf{T}}$
  $\mathbf{k} = [\oplus_j \mathbf{A}(:, j)]$
  $m = \frac{1}{2}[\oplus_i \mathbf{k}(i)]$
  $\mathbf{S} \leftarrow \mathbf{I}$

  $vertices\_changed \leftarrow \text{nvals}(\mathbf{k})$
  **while** $vertices\_changed > 0$ **do**
    **for** $j \in range(|V|)$ **do**
      $\mathbf{v} = \mathbf{G}(j, :)$
      $\mathbf{t_q} = \mathbf{v}\ \mathsf{any.pair}\ \mathbf{S}$
      $\mathbf{sr} = \mathbf{S}(j, :)$
      $\mathbf{S}(j, :) = \text{empty}$

      $\mathbf{q} \leftarrow \mathbf{k}$
      $\mathbf{q}\langle\mathbf{k}\rangle \otimes= -\mathbf{k}(j)/m$
      $\mathbf{q} \oplus= \mathbf{v}$
      $\mathbf{q_1}\langle\mathbf{t_q}\rangle = \mathbf{q} \oplus.\otimes\ \mathbf{S}$

      $\mathbf{t} = (\mathbf{q_1} == [\mathsf{max}_i\ \mathbf{q_1}(i)])$
      **while** $nvals(t) \neq 1$ **do**
        $\mathbf{p} = \text{random}() \otimes \mathbf{t}$
        $\mathbf{t} = (\mathbf{p} == [\mathsf{max}_i\ \mathbf{p}(i)])$
      $\mathbf{S}(j, :) = \mathbf{t}$

      **if** $nvals(\mathbf{sr} \otimes \mathbf{t}) == 0$ **then**
        $vertices\_changed = \text{nvals}(\mathbf{k})$
      $vertices\_changed = vertices\_changed - 1$

**Algorithm 18:** Community detection using label propagation (for undirected graphs).

**Function** *CDLP*
  $\mathbf{L} \leftarrow \text{diag}([0, 1, \ldots, n - 1])$
  **for** $k = 1$ **to** $t$ **do**
    $\mathbf{F} = \mathbf{A}\ \mathsf{any\,.\,second}\ \mathbf{L}$ `// Frequency matrix`
    $\{i, \_, x\} \leftarrow \mathbf{F}$
    $\text{merge\_sort\_pairs}(i, x)$
    $\mathbf{labels}$ = for each row in $i$, select min mode value from $x$

# References

[1] T. A. Davis, "Algorithm 1000: SuiteSparse:GraphBLAS: Graph algorithms in the language of sparse linear algebra," *ACM Trans. Math. Softw.*, 2019. [Online]. Available: https://doi.org/10.1145/3322125

[2] ——. (2020) User guide for SuiteSparse:GraphBLAS, version 3.3.3. https://people.engr.tamu.edu/davis/GraphBLAS.html.

[3] J. Kepner *et al.*, "Mathematical foundations of the GraphBLAS," in *HPEC*. IEEE, 2016. [Online]. Available: https://doi.org/10.1109/HPEC.2016.7761646

[4] J. Kepner and J. R. Gilbert, Eds., *Graph Algorithms in the Language of Linear Algebra*. SIAM, 2011. [Online]. Available: https://doi.org/10.1137/1.9780898719918

[5] T. Mattson *et al.*, "LAGraph: A community effort to collect graph algorithms built on top of the GraphBLAS," in *GrAPL at IPDPS*, 2019. [Online]. Available: https://doi.org/10.1109/IPDPSW.2019.00053

[6] T. G. Mattson *et al.*, "GraphBLAS C API: Ideas for future versions of the specification," in *HPEC*. IEEE, 2017. [Online]. Available: https://doi.org/10.1109/HPEC.2017.8091095

[7] R. Seidel, "On the all-pairs-shortest-path problem in unweighted undirected graphs," *J. Comput. Syst. Sci.*, vol. 51, no. 3, pp. 400–403, 1995. [Online]. Available: https://doi.org/10.1006/jcss.1995.1078