A Universal Low Complexity Compression Algorithm for Sparse Marked Graphs (Algorithms)

Payam Delgosha* and Venkat Anantharam†

September 21, 2022

This document contains the algorithms in [DA]. Please refer to [DA] for more discussion as well as the proof of optimality and complexity analysis.

The table on Page 2 gives a list of the algorithms, the description for each algorithm, as well as their interdependencies.

References

[DA] Payam Delgosha and Venkat Anantharam. A universal low complexity compression algorithm for sparse marked graphs. To appear on arXiv, also available at https://delgosha.web.illinois.edu/preprints/lcgc.pdf.

*Department of Computer Science, University of Illinois Urbana-Champaign, delgosha@illinois.edu

†Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, ananth@berkeley.edu

Algorithm	Description	Uses algorithms
1	encoding a simple marked graph (main compression algorithm)	2, 14, 4, 5, 6, 7, 8, 17, 24
2	preprocessing a simple marked graph to find its equivalent neighbor list representation	
3	compressing an array consisting of nonnegative integers	17
4	encoding star vertices (part of Algorithm 1)	3
5	encoding star edges (part of Algorithm 1)	
6	finding vertex degree profiles, i.e. the variable Deg (part of Algorithm 1)	
7	encoding Vertex Types (part of Algorithm 1)	3
8	finding Partition Graphs (part of Algorithm 1)	
9	decoding a simple marked graph (main decompression algorithm)	10, 11, 12, 13, 22, 27
10	decompressing an array consisting of nonnegative integers	22
11	decoding star edges	
12	decoding vertex degree profiles	10
13	finding degree sequences of partition graphs and relative vertex indexing	
14	extracting edge types for a simple marked graph	15
15	sending a message from a node to one of its neighbors	
16	computing $N_{i,j}(G)$ for a simple unmarked bipartite graph $G \in \mathcal{G}^{(n_l,n_r)}_{\vec{a},\vec{b}}$	18, 19
17	finding $f^{(n_l,n_r)}_{\vec{a},\vec{b}}(G)$ for $G \in \mathcal{G}^{(n_l,n_r)}_{\vec{a},\vec{b}}$	16
18	computing $\prod_{k'=0}^{k-1}(p-k's)$	
19	computing $\prod_{i'=i}^{j} v_{i'}!$ for an array $\vec{v}$ of nonnegative integers	18
20	decoding the adjacency list of a left vertex $1 \leq i \leq n_l$ given $\widetilde{N}_{i,i}$ for a simple unmarked bipartite graph $G \in \mathcal{G}^{(n_l,n_r)}_{\vec{a},\vec{b}}$	18
21	decoding the adjacency list of the left vertices $i \leq v \leq j$ for $1 \leq i \leq j \leq n_l$ given $\widetilde{N}_{i,j}$ for a simple unmarked bipartite graph $G \in \mathcal{G}^{(n_l,n_r)}_{\vec{a},\vec{b}}$	18, 19, 20
22	decoding for a simple unmarked bipartite graph $G \in \mathcal{G}^{(n_l,n_r)}_{\vec{a},\vec{b}}$ given $f^{(n_l,n_r)}_{\vec{a},\vec{b}}(G)$	19, 21
23	computing $N_{i,j}(G)$ for a simple unmarked graph $G \in \mathcal{G}^{(\tilde{n})}_{\vec{a}}$	18
24	finding $f^{(\tilde{n})}_{\vec{a}}(G)$ and $\vec{\tilde{f}}^{(\tilde{n})}_{\vec{a}}(G)$ for $G \in \mathcal{G}^{(n_l,n_r)}_{\vec{a}}$	23
25	decoding the forward adjacency list of a vertex $1 \leq i \leq \tilde{n}$ given $\widetilde{N}_{i,i}$ for a simple unmarked graph $G \in \mathcal{G}^{(\tilde{n})}_{\vec{a}}$	18

26	decoding the forward adjacency list of vertices $i \le v \le j$ given $\widetilde{N}_{i,j}$ for a simple unmarked graph $G \in \mathcal{G}^{(\tilde{n})}_{\vec{a}}$	18, 25
27	decoding for a simple unmarked graph $G \in \mathcal{G}^{(\tilde{n})}_{\vec{a}}$ given $f^{(\tilde{n})}_{\vec{a}}(G)$ and $\vec{\tilde{f}}^{(\tilde{n})}_{\vec{a}}(G)$	19, 26

Algorithm 1. Encoding a simple marked graph

Input:

n: number of vertices

$G^{(n)}$: A simple marked graph $G^{(n)}$ on the vertex set $[n]$, vertex mark set $\Theta = \{1, \dots, |\Theta|\}$ and edge mark set $\Xi = \{1, \dots, |\Xi|\}$ given as follows:

- its vertex mark sequence $\vec{\theta}^{(n)} = (\theta^{(n)}_v : v \in [n])$, where $\theta^{(n)}_v \in \Theta$ is the mark of vertex v in $G^{(n)}$
- $\mathsf{EdgeList} = (\mathsf{EdgeList}_i : 1 \le i \le m^{(n)})$: the list of edges in $G^{(n)}$ where $\mathsf{EdgeList}_i = (v_i, w_i, x_i, x'_i)$ for $1 \le i \le m^{(n)}$, where $m^{(n)}$ denotes the total number of edges in $G^{(n)}$, and for $1 \le i \le m^{(n)}$, the tuple (v_i, w_i, x_i, x'_i) represents an edge between the vertices v_i and w_i with mark x_i towards v_i and mark x'_i towards w_i, i.e. $\xi_{G^{(n)}}(w_i, v_i) = x_i$ and $\xi_{G^{(n)}}(v_i, w_i) = x'_i$

δ: degree threshold hyperparameter, $\delta \ge 1$

h: depth hyperparameter, $h \ge 1$

Output:

Output: A bit sequence in $\{0,1\}^* - \emptyset$ representing $G^{(n)}$ in compressed form.

```
function MarkedGraphEncode(n, G^(n), δ, h)
    Output ← empty bit sequence    ▷ initialize the output with empty bit sequence
    (θ⃗^(n), d⃗^(n), γ⃗^(n), γ̃⃗^(n), x⃗^(n), x⃗'^(n)) ← Preprocess(n, θ⃗^(n), EdgeList)
                    ▷ Algorithm 2, this finds the equivalent neighbor list representation
    (c⃗, TCount, TIsStar, TMark) ← ExtractTypes(n, G^(n), δ, h)    ▷ Algorithm 14
    Output ← Output + E_Δ(1 + TCount)    ▷ use the Elias delta code to represent TCount
    for 1 ≤ i ≤ TCount do
        Output ← Output + TIsStar(i) + TMark(i)    ▷ use 1 + ⌊log_2 |Ξ|⌋ bits to encode TMark(i)
    end for
    EncodeStarVertices    ▷ Algorithm 4
    EncodeStarEdges    ▷ Algorithm 5
    Deg = (Deg_v : 1 ≤ v ≤ n) ← Array of Dictionary(ℕ × ℕ → ℕ)
    FindDeg    ▷ Algorithm 6
    EncodeVertexTypes    ▷ Algorithm 7
    PartitionAdjList ← Dictionary(ℕ × ℕ → Array of Array of integers)
    PartitionDeg ← Dictionary(ℕ × ℕ → Array of integers)
    PartitionIndex = (PartitionIndex_v : 1 ≤ v ≤ n) ← Array of Dictionary(ℕ × ℕ → ℕ)
    FindPartitionGraphs    ▷ Algorithm 8
    k ← number of keys in PartitionAdjList    ▷ number of partitions graphs to be encoded
    Ouptut ← Output + E_Δ(k + 1)
    for (i, i') ∈ PartitionAdjList.Keys do
        if i < i' then
            Output ← Output + i + i'    ▷ use 1 + ⌊log_2 TCount⌋ bits to encode i and i'
            a⃗ ← PartitionDeg(i, i'), b⃗ ← PartitionDeg(i', i)    ▷ left and right degree sequences
```

$f \leftarrow$ BENCODEGRAPH(SIZE($\vec{a}$), SIZE($\vec{b}$), $\vec{a}$, $\vec{b}$, PartitionAdjList(i, i')) ▷ Algorithm 17
    Output ← Output + $\mathsf{E}_\Delta(1 + f)$
  end if
  if $i = i'$ **then**
    Output ← Output + $i + i'$ ▷ use $1 + \lfloor \log_2 \mathsf{TCount} \rfloor$ bits to encode i and i'
    $\vec{a} \leftarrow$ PartitionDeg(i, i) ▷ degree sequence
    $(f, \vec{\tilde{f}}) \leftarrow$ ENCODEGRAPH(SIZE($\vec{a}$), $\vec{a}$, PartitionAdjList(i, i)) ▷ Algorithm 24
    Output ← Output + $\mathsf{E}_\Delta(1 + f)$
    Output ← Output + $\mathsf{E}_\Delta(1 + \text{SIZE}(\vec{\tilde{f}}))$
    for $1 \le j \le \text{SIZE}(\vec{\tilde{f}})$ **do**
      Output ← Output + $\mathsf{E}_\Delta(1 + \tilde{f}_j)$
    end for
  end if
end for
end function

Algorithm 2. Preprocess a simple marked graph to find its equivalent neighbor list representation

---

Input:

n: number of vertices

A simple marked graph $G^{(n)}$ represented by

- its vertex mark sequence $\vec{\theta}^{(n)} = (\theta_v^{(n)} : v \in [n])$, where $\theta_v^{(n)} \in \Theta$ is the mark of vertex v in $G^{(n)}$.
- $\mathsf{EdgeList} = (\mathsf{EdgeList}_i : 1 \le i \le m^{(n)})$: the list of edges in $G^{(n)}$ where $\mathsf{EdgeList}_i = (v_i, w_i, x_i, x'_i)$ for $1 \le i \le m^{(n)}$. Here, $m^{(n)}$ denotes the total number of edges in $G^{(n)}$, and for $1 \le i \le m^{(n)}$, the tuple (v_i, w_i, x_i, x'_i) represents an edge between the vertices v_i and w_i with mark x_i towards v_i and mark x'_i towards w_i, i.e. $\xi_{G^{(n)}}(w_i, v_i) = x_i$ and $\xi_{G^{(n)}}(v_i, w_i) = x'_i$.

Output:

The equivalent representation of $G^{(n)}$ of the form

- $\vec{\theta}^{(n)} = (\theta_v^{(n)} : v \in [n])$ where $\theta_v^{(n)}$ denotes the vertex mark of v.
- $\vec{d}^{(n)} = (d_v^{(n)} : v \in [n])$ such that $d_v^{(n)}$ for $1 \le v \le n$ is the degree of vertex v in $G^{(n)}$.
- $\vec{\gamma}^{(n)}$: Array of Array of integers, such that for $1 \le v \le n$, the neighbors of vertex v in $G^{(n)}$ is stored in an increasing order as $1 \le \gamma_{v,1}^{(n)} < \gamma_{v,2}^{(n)} < \dots < \gamma_{v,d_v^{(n)}}^{(n)} \le n$.
- $\vec{\tilde{\gamma}}^{(n)}$: Array of Array of integers, such that for $1 \le v \le n$ and $1 \le i \le d_v^{(n)}$, $\tilde{\gamma}_{v,i}^{(n)}$ denotes the index of v among the neighbors of $\gamma_{v,i}^{(n)}$, so that $\gamma^{(n)}_{\gamma_{v,i}^{(n)}, \tilde{\gamma}_{v,i}^{(n)}} = v$.
- $\vec{x}^{(n)}$ and $\vec{x'}^{(n)}$: Array of Array of integers, such that for $1 \le v \le n$ and $1 \le i \le d_v^{(n)}$, $x_{v,i}^{(n)}$ and ${x'}_{v,i}^{(n)}$ denote the two edge marks corresponding to the edge connecting v to $\gamma_{v,i}^{(n)}$, so that $x_{v,i}^{(n)} = \xi_{G^{(n)}}(\gamma_{v,i}^{(n)}, v)$ and ${x'}_{v,i}^{(n)} = \xi_{G^{(n)}}(v, \gamma_{v,i}^{(n)})$.

function PREPROCESS($n, \vec{\theta}^{(n)}$, EdgeList)
$\vec{d}^{(n)} \leftarrow$ Array of integers of size n ▷ initialize $\vec{d}^{(n)}$
$\vec{\gamma}^{(n)}, \vec{\tilde{\gamma}}^{n}, \vec{x}^{(n)}, \vec{x'}^{(n)} \leftarrow$ Array of Array of integers of size n ▷ initialize $\vec{\gamma}^{(n)}, \vec{\tilde{\gamma}}^{n}, \vec{x}^{(n)}$, and $\vec{x'}^{(n)}$
for $1 \leq i \leq n$ **do**
$d_i^{(n)} \leftarrow 0$ ▷ initialize degree sequence with zero
end for
for $1 \leq i \leq m^{(n)}$ **do**
if $v_i > w_i$ **then**
SWAP(v_i, w_i), SWAP(x_i, x'_i) ▷ to make sure that for all $1 \leq i \leq m^{(n)}$, we have $v_i < w_i$
end if
end for
EdgeList $\leftarrow$ SORT(EdgeList) ▷ sort EdgeList with respect to the lexicographic order of the pair (v_i, w_i)
for $1 \leq i \leq m^{(n)}$ **do**
append w_i to $\gamma_{v_i}^{(n)}$ ▷ add w_i to the neighbor list of v_i
append x_i to $x_{v_i}^{(n)}$, append x'_i to ${x'}_{v_i}^{(n)}$ ▷ append the mark pair
append $1 + d_{w_i}^{(n)}$ to $\tilde{\gamma}_{v_i}^{(n)}$
▷ v_i is the newly added neighbor of w_i, and its index among the neighbors of w_i should be one plus the number of existing neighbors of w_i, i.e. $1 + d_{w_i}^{(n)}$
append v_i to $\gamma_{w_i}^{(n)}$ ▷ add v_i to the neighbor list of w_i
append x'_i to $x_{w_i}^{(n)}$, append x_i to ${x'}_{w_i}^{(n)}$ ▷ append the mark pair
append $1 + d_{v_i}^{(n)}$ to $\tilde{\gamma}_{w_i}^{(n)}$
▷ w_i is the newly added neighbor of v_i, and its index among the neighbors of v_i should be one plus the number of existing neighbors of v_i, i.e. $1 + d_{v_i}^{(n)}$
$d_{v_i}^{(n)} \leftarrow d_{v_i}^{(n)} + 1, d_{w_i}^{(n)} \leftarrow d_{w_i}^{(n)} + 1$ ▷ add one to the number of existing neighbors
end for
return $(\vec{\theta}^{(n)}, \vec{d}^{(n)}, \vec{\gamma}^{(n)}, \vec{\tilde{\gamma}}^{(n)}, \vec{x}^{(n)}, \vec{x'}^{(n)})$
end function

Algorithm 3. Compressing an array $\vec{y}$ consisting of nonnegative integers

Input:
n: size of the array
$\vec{y} = (y_1, \dots, y_n)$: array of nonnegative integers
Output:
Output: A prefix–free bit sequence in $\{0,1\}^* - \emptyset$ representing $\vec{y}$
function ENCODESEQUENCE($n, \vec{y}$)
Output $\leftarrow$ empty bit sequence
$K \leftarrow 0$
for $1 \leq i \leq n$ **do**
$K \leftarrow \max\{K, y_i\}$
end for
$K \leftarrow K + 1$ ▷ y_i's are in the range $[0, K-1]$
$\vec{\mathsf{a}} \leftarrow$ Array of integers of size n ▷ left degree sequence
$\vec{\mathsf{b}} \leftarrow$ Array of integers of size K ▷ right degree sequence

$\vec{\gamma} \leftarrow$ Array of Array of integers of size n, where $\vec{\gamma}_i, 1 \leq i \leq n$ is of size 1 ▷ the adjacency list
for $1 \leq i \leq n$ **do**
  $\mathsf{a}_i \leftarrow 1$
  $\mathsf{b}_{1+y_i} \leftarrow \mathsf{b}_{1+y_i} + 1$
  $\vec{\gamma}_i \leftarrow$ array of size 1 containing $1 + y_i$
end for
$f \leftarrow \text{BENCODEGRAPH}(n, K, \vec{\mathsf{a}}, \vec{\mathsf{b}}, \vec{\gamma})$ ▷ Algorithm 17
$\mathsf{Output} \leftarrow \mathsf{Output} + \mathsf{E}_{\Delta}(K)$
for $1 \leq j \leq K$ **do**
  $\mathsf{Output} \leftarrow \mathsf{Output} + \mathsf{E}_{\Delta}(1 + \mathsf{b}_j)$
end for
$\mathsf{Output} \leftarrow \mathsf{Output} + \mathsf{E}_{\Delta}(1 + f)$
end function

Algorithm 4. Encoding Star Vertices (part of Algorithm 1)

procedure ENCODESTARVERTICES ▷ line 9 in Algorithm 1
  $\vec{s} = (s_v : 1 \leq v \leq n) \leftarrow$ Array of zero ones, each element initialized with zero
  for $1 \leq v \leq n$ **do**
    for $1 \leq k \leq d_v^{(n)}$ **do**
      $(i, i') \leftarrow c_{v,k}$
      if $\mathsf{TIsStar}(i) = 1$ or $\mathsf{TIsStar}(i') = 1$ **then**
        $s_v \leftarrow 1$
      end if
    end for
  end for
  $\mathsf{Output} \leftarrow \mathsf{Output} + \text{ENCODESEQUENCE}(n, \vec{s})$ ▷ using Algorithm 3
end procedure

Algorithm 5. Encoding Star Edges (part of Algorithm 1)

procedure ENCODESTAREDGES ▷ line 10 in Algorithm 1
  for $1 \leq x \leq |\Xi|$ **do**
    for $1 \leq x' \leq |\Xi|$ **do**
      for $1 \leq v \leq n$ **do**
        if $s_v = 1$ **then**
          for $1 \leq k \leq d_v^{(n)}$ **do**
            $(i, i') \leftarrow c_{v,k}$
            if $\mathsf{TIsStar}(i) = 1$ or $\mathsf{TIsStar}(i') = 1$ **then**
              if $x_{v,k} = x$ and $x'_{v,k} = x'$ and $\gamma_{v,k}^{(n)} > v$ **then**
                $\mathsf{Output} \leftarrow \mathsf{Output} + 1 + \gamma_{v,k}^{(n)}$ ▷ use $1 + \lfloor \log_2 n \rfloor$ bits to represent $\gamma_{v,k}^{(n)}$
              end if
            end if
          end for

```
            Output ← Output + 0
          end if
        end for
      end for
    end for
end procedure
```

Algorithm 6. Finding vertex degree profiles, i.e. the variable Deg (part of Algorithm 1)

```
procedure FindDeg                                             ▷ line 12 in Algorithm 1
    for 1 ≤ v ≤ n do
        for 1 ≤ k ≤ d_v^{(n)} do
            (i, i') ← c_{v,k}
            if TIsStar(i) = 0 and TIsStar(i') = 0 then
                if (i, i') ∈ Deg_v.Keys then
                    Deg_v(i, i') ← Deg_v(i, i') + 1      ▷ increment the corresponding degree value
                else
                    Deg_v.Insert((i, i'), 1)            ▷ this is the first edge observed with this type
                end if
            end if
        end for
    end for
end procedure
```

Algorithm 7. Encoding Vertex Types (part of Algorithm 1)

```
procedure EncodeVertexTypes                                   ▷ line 13 in Algorithm 1
    VertexTypesDictionary ← Dictionary(Array of integers → ℕ)
    k ← 0                                                 ▷ number of distinct vertex types found
    \vec{y} = (y_v : 1 ≤ v ≤ n) ← Array of integers
    \vec{ν} ← Array of integers
    for 1 ≤ v ≤ n do
        \vec{ν} ← ∅                                        ▷ erasing \vec{ν} to get a fresh array
        ν_1 ← θ_v^{(n)}
        for ((i, i'), l) ∈ Deg_v do
            append (i, i', l) at the end of \vec{ν}
        end for
        if \vec{ν} ∉ VertexTypesDictionary.Keys then
            k ← k + 1                                      ▷ a new vertex type is discovered
            VertexTypesDictionary.Insert(\vec{ν}, k)
        end if
        y_v ← VertexTypesDictionary(\vec{ν})
    end for
    Output ← Output + k        ▷ use 1 + ⌊log_2 n⌋ bits to represent the number of key–value pairs
```

for $(\vec{\nu}, i) \in \mathsf{VertexTypesDictionary}$ **do**
    $\mathsf{Output} \leftarrow \mathsf{Output} + \textsc{Size}(\vec{\nu})$ ▷ use $1 + \lfloor \log_2(1 + 3\delta) \rfloor$ bits to encode $\textsc{Size}(\vec{\nu})$
    for $1 \le j \le \textsc{Size}(\vec{\nu})$ **do**
        $\mathsf{Output} \leftarrow \mathsf{Output} + \nu_j$ ▷ use $1 + \lfloor \log_2(|\Xi| \vee \mathsf{TCount} \vee \delta) \rfloor$ bits to encode ν_j
    end for
    $\mathsf{Output} \leftarrow \mathsf{Output} + i$ ▷ use $1 + \lfloor \log_2 n \rfloor$ bits to encode i
end for
$\mathsf{Output} \leftarrow \mathsf{Output} + \textsc{EncodeSequence}(n, \vec{y})$ ▷ using Algorithm 3
end procedure

Algorithm 8. Finding Partition Graphs (part of Algorithm 1)

procedure $\textsc{FindPartitionGraphs}$ ▷ line 17 in Algorithm 1
    for $1 \le v \le n$ **do** ▷ find $\mathsf{PartitionIndex}$ and $\mathsf{PartitionDeg}$
        for $((i, i'), k) \in \mathsf{Deg}_v$ **do**
            if $(i, i') \notin \mathsf{PartitionDeg}.\textsc{Keys}$ **then**
                $\mathsf{PartitionIndex}_v.\textsc{Insert}((i, i'), 1)$
                $\mathsf{PartitionDeg}.\textsc{Insert}((i, i'), (k))$ ▷ $\mathsf{PartitionDeg}(i, i')$ now becomes an array of length 1 containing k
            else
                $\mathsf{PartitionIndex}_v.\textsc{Insert}((i, i'), \textsc{Size}(\mathsf{PartitionDeg}(i, i')) + 1)$
                append k at the end of $\mathsf{PartitionDeg}(i, i')$
            end if
        end for
    end for
    for $(i, i') \in \mathsf{PartitionDeg}.\textsc{Keys}$ **do** ▷ initialize $\mathsf{PartitionDeg}$
        if $i \le i'$ **then**
            Insert key (i, i') in $\mathsf{PartitionAdjList}$ with value being an array of size $\textsc{Size}(\mathsf{PartitionDeg}(i, i'))$, such that each element of this array is an empty array
        end if
    end for
    for $1 \le v \le n$ **do** ▷ update $\mathsf{PartitionIndex}$ and $\mathsf{PartitionAdjList}$
        for $1 \le k \le d_v^{(n)}$ **do**
            $w \leftarrow \gamma_{v,k}^{(n)}$
            $(i, i') \leftarrow c_{v,k}$
            if $\mathsf{TIsStar}(i) = 0$ and $\mathsf{TIsStar}(i') = 0$ **then**
                $p \leftarrow \mathsf{PartitionIndex}_v(i, i')$ ▷ index of v
                $q \leftarrow \mathsf{PartitionIndex}_w(i', i)$ ▷ index of w
                if $i < i'$ **then**
                    append q at the end of $(\mathsf{PartitionAdjList}(i, i'))_p$
                end if
                if $i = i'$ and $q > p$ **then**
                    append q at the end of $(\mathsf{PartitionAdjList}(i, i))_p$
                end if
            end if
        end for

end for
end procedure

Algorithm 9. Decoding a simple marked graph

Input: $\mathsf{Input} = f^{(n)}(G^{(n)})$ for a simple marked graph $G^{(n)}$ on the vertex set $[n]$. Here, $f^{(n)}(G^{(n)})$ refers to the bit sequence generated by our compression procedure discussed in Algorithm 1.
Output: $\widehat{G}^{(n)}$ a reconstruction of $G^{(n)}$ represented in the edge list form, i.e.

- $\vec{\theta}^{(n)}$: sequence of vertex marks in $\widehat{G}^{(n)}$.
- $\mathsf{EdgeListDec}$: list of edges in $\widehat{G}^{(n)}$.

function MarkedGraphDecode($G^{(n)}$)
  $\mathsf{TCount} \leftarrow \mathsf{E}_\Delta^{-1}(\mathsf{Input}) - 1$ ▷ we encode $1 + \mathsf{TCount}$ in Algorithm 1
  $\mathsf{TIsStar} \leftarrow$ Array of bits of size TCount
  $\mathsf{TMark} \leftarrow$ Array of integers of size TCount
  for $1 \leq i \leq \mathsf{TCount}$ **do**
    $\mathsf{TIsStar}(i) \leftarrow$ read 1 bit from Input
    $\mathsf{TMark}(i) \leftarrow$ read $1 + \lfloor \log_2 |\Xi| \rfloor$ bits from Input
  end for
  $\vec{s} \leftarrow$ DecodeSequence(n, Input)
  ▷ decode for star vertices using Algorithm 10
  $\mathsf{EdgeListDec} \leftarrow$ Array of $\mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$
  ▷ $\mathsf{EdgeListDec}$ is the decoded edge list, each index of the form (v, w, x, x'), where $x = \xi_{\widehat{G}^{(n)}}(w, v)$ and $x' = \xi_{\widehat{G}^{(n)}}(v, w)$
  DecodeStarEdges ▷ Algorithm 11
  $\mathsf{Deg} = (\mathsf{Deg}_v : 1 \leq v \leq n) \leftarrow$ Array of Dictionary($\mathbb{N} \times \mathbb{N} \to \mathbb{N}$)
  $\vec{\theta}^{(n)} \leftarrow$ Array of integers
  DecodeVertexDegreeProfiles ▷ Algorithm 12
  $\mathsf{PartitionDeg} \leftarrow$ Dictionary($\mathbb{N} \times \mathbb{N} \to$ Array of integers)
  $\mathsf{OriginalIndex} \leftarrow$ Dictionary($\mathbb{N} \times \mathbb{N} \to$ Array of integers)
  DecodePartitionDegOriginalIndex ▷ Algorithm 13
  $K \leftarrow \mathsf{E}_\Delta^{-1}(\mathsf{Input}) - 1$ ▷ number of partition graphs
  for $1 \leq k \leq K$ **do**
    $i \leftarrow$ read $1 + \lfloor \log_2 \mathsf{TCount} \rfloor$ bits from Input
    $i' \leftarrow$ read $1 + \lfloor \log_2 \mathsf{TCount} \rfloor$ bits from Input
    if $i < i'$ **then**
      $f \leftarrow \mathsf{E}_\Delta^{-1}(\mathsf{Input}) - 1$
      $\mathsf{AdjList} \leftarrow$ BDecodeGraph($f, \mathsf{PartitionDeg}(i, i'), \mathsf{PartitionDeg}(i', i)$)
      ▷ Algorithm 22
    else
      $f \leftarrow \mathsf{E}_\Delta^{-1}(\mathsf{Input}) - 1$
      $L \leftarrow \mathsf{E}_\Delta^{-1}(\mathsf{Input}) - 1$
      $\vec{\tilde{f}} \leftarrow$ Array of integers with size L
      for $1 \leq l \leq L$ **do**
        $\tilde{f}_l \leftarrow \mathsf{E}_\Delta^{-1}(\mathsf{Input}) - 1$
      end for

$\quad\quad\quad$ $\mathsf{AdjList} \leftarrow$ GraphDecode$(f, \vec{\tilde{f}}, \mathsf{PartitionDeg}(i,i))$ $\triangleright$ Algorithm 27
$\quad\quad$ **end if**
$\quad\quad$ $x \leftarrow \mathsf{TMark}(i)$
$\quad\quad$ $x' \leftarrow \mathsf{TMark}(i')$
$\quad\quad$ $A \leftarrow \mathsf{OriginalIndex}(i, i')$
$\quad\quad$ $B \leftarrow \mathsf{OriginalIndex}(i', i)$
$\quad\quad$ **for** $1 \leq v \leq$ Size$(\mathsf{AdjList})$ **do**
$\quad\quad\quad$ $v' \leftarrow A_v$
$\quad\quad\quad$ **for** $1 \leq j \leq$ Size$(\mathsf{AdjList}_v)$ **do**
$\quad\quad\quad\quad$ $w \leftarrow \mathsf{AdjList}_{v,j}$
$\quad\quad\quad\quad$ $w' \leftarrow B_w$
$\quad\quad\quad\quad$ append (v', w', x, x') at the end of $\mathsf{EdgeListDec}$
$\quad\quad\quad$ **end for**
$\quad\quad$ **end for**
$\quad$ **end for**
$\quad$ **return** $(\vec{\theta}^{(n)}, \mathsf{EdgeListDec})$
end function

Algorithm 10. Decompressing an array consisting of nonnegative integers

---

Input:
$\quad$ n: size of the array
$\quad$ Input: sequence of bits which contains the compressed form of an array generated by Algorithm 3
Output:
$\quad$ $\vec{y} = (y_1, \dots, y_n)$: the decoded array consisting of nonnegative integers
function DecodeSequence(n, Input)
$\quad$ $K \leftarrow \mathsf{E}_\Delta^{-1}(\mathsf{Input})$ $\triangleright$ Symbols in $\vec{y}$ are in the range $[0, K-1]$
$\quad$ $\mathsf{a} \leftarrow$ Array of nonnegative integers of size n where all elements are 1
$\quad$ $\mathsf{b} \leftarrow$ Array of nonnegative integers of size K
$\quad$ **for** $1 \leq j \leq K$ **do**
$\quad\quad$ $\mathsf{b}_j \leftarrow \mathsf{E}_\Delta^{-1}(\mathsf{Input})$
$\quad\quad$ $\mathsf{b}_j \leftarrow \mathsf{b}_j - 1$ $\triangleright$ We encode $1 + \mathsf{b}_j$ in line 19 of Algorithm 3
$\quad$ **end for**
$\quad$ $f \leftarrow \mathsf{E}_\Delta^{-1}(\mathsf{Input})$
$\quad$ $f \leftarrow f - 1$
$\quad$ $\vec{\gamma}_{[1:n]} \leftarrow$ BDecodeGraph$(f, \vec{\mathsf{a}}, \vec{\mathsf{b}})$ $\triangleright$ Algorithm 22
$\quad$ **for** $1 \leq i \leq n$ **do**
$\quad\quad$ $y_i \leftarrow \gamma_{i,1} - 1$ $\triangleright$ as in line 14 of Algorithm 3, $\vec{\gamma}_i$ is an array of size 1 containing $1 + y_i$
$\quad$ **end for**
$\quad$ **return** $\vec{y}$
end function

Algorithm 11. Decoding Star Edges

```
procedure DecodeStarEdges                                        ▷ line 11 in Algorithm 9
    for 1 ≤ x ≤ |Ξ| do
        for 1 ≤ x' ≤ |Ξ| do
            for 1 ≤ v ≤ n do
                if s_v = 1 then
                    b ← read 1 bit from Input
                    while b ≠ 0 do
                        w ← read 1 + ⌊log_2 n⌋ bits from Input
                        append (v, w, x, x') at the end of EdgeListDec
                        b ← read 1 bit from Input
                    end while
                end if
            end for
        end for
    end for
end procedure
```

Algorithm 12. Decoding Vertex Degree Profiles

```
procedure DecodeVertexDegreeProfiles                             ▷ line 14 in Algorithm 9
    VertexTypesList ← Array of Array of integers
    ν⃗ ← Array of integers
    K ← read 1 + ⌊log_2 n⌋ bits from Input                ▷ number of distinct vertex types
    resize VertexTypesList to have K elements, each being an empty array
    for 1 ≤ k ≤ K do
        ν⃗ ← ∅
        l ← read 1 + ⌊log_2(1 + 3δ)⌋ bits from Input              ▷ number of elements in ν⃗
        for 1 ≤ j ≤ l do
            read 1 + ⌊log_2(|Ξ| ∨ TCount ∨ δ)⌋ bits from Input and append to ν⃗
        end for
        i ← read 1 + ⌊log_2 n⌋ bits from Input
        VertexTypesList(i) ← ν⃗
    end for
    y⃗ ← DecodeSequence(n, Input)                                      ▷ Algorithm 10
    Deg = (Deg_v : 1 ≤ v ≤ n) ← Array of Dictionary(ℕ × ℕ → ℕ)
    for 1 ≤ v ≤ n do
        ν⃗ ← VertexTypesList(y_v)
        θ_v ← ν_1                                          ▷ decode the vertex mark of v
        for 1 ≤ k ≤ (Size(ν⃗) − 1)/3 do
            i ← ν_{1+(3k−2)}
            i' ← ν_{1+(3k−1)}
            j ← ν_{1+3k}
            Deg_v.Insert((i, i'), j)
        end for
    end for
end procedure
```

Algorithm 13. Finding Degree Sequences of Partition Graphs and Relative Vertex Indexing

procedure DecodePartitionDegOriginalIndex ▷ line 17 in Algorithm 9
  for $1 \le v \le n$ **do**
    for $((i,i'),k) \in \mathsf{Deg}_v$ **do**
      if $(i,i') \notin \mathsf{PartitionDeg}.\textsc{Keys}$ **then**
        $\mathsf{OriginalIndex}.\textsc{Insert}((i,i'),(v))$ ▷ $\mathsf{OriginalIndex}(i,i')$ is now an array with length 1 containing v
        $\mathsf{PartitionDeg}.\textsc{Insert}((i,i'),(k))$ ▷ $\mathsf{PartitionDeg}(i,i')$ now becomes an array of length 1 containing k
      else
        append v at the end of $\mathsf{OriginalIndex}(i,i')$
        append k at the end of $\mathsf{PartitionDeg}(i,i')$
      end if
    end for
  end for
end procedure

Algorithm 14. Extracting edge types for a simple marked graph

Input:

n: number of vertices

$G^{(n)}$: A simple marked graph on the vertex set $[n]$, vertex mark set $\Theta = \{1,\dots,|\Theta|\}$ and edge mark set $\Xi = \{1,\dots,|\Xi|\}$ given in its neighbor list representation. More precisely, for a vertex $1 \le v \le n$, the following are given

- $d_v^{(n)}$: the degree of vertex v
- $\theta_v^{(n)}$: the vertex mark of v
- for $1 \le i \le d_v^{(n)}$, the tuple $(\gamma_{v,i}^{(n)}, x_{v,i}^{(n)}, {x'}_{v,i}^{(n)})$ where $\gamma_{v,1}^{(n)} < \gamma_{v,2}^{(n)} < \dots < \gamma_{v,d_v^{(n)}}^{(n)}$ are the neighbors of vertex v and for $1 \le i \le d_v^{(n)}$, $x_{v,i}^{(n)} = \xi_{G^{(n)}}(\gamma_{v,i}^{(n)}, v)$ and ${x'}_{v,i}^{(n)} = \xi_{G^{(n)}}(v, \gamma_{v,i}^{(n)})$.
- for $1 \le i \le d_v^{(n)}$, $\tilde{\gamma}_{v,i}^{(n)}$ which is the index of v among the neighbors of $\gamma_{v,i}^{(n)}$, i.e. $\gamma^{(n)}_{\gamma_{v,i}^{(n)}, \tilde{\gamma}_{v,i}^{(n)}} = v$.

h: the depth parameter

δ: the degree parameter

Output:

$\vec{c} = (c_{v,i} : v \in [n], i \in [d_v^{(n)}])$: Array of Array of integers, where for a vertex $v \in [n]$ and $1 \le i \le d_v^{(n)}$, $c_{v,i}$ represents $\psi_{h,\delta}^{(n)}(v, \gamma_{v,i}^{(n)})$, the type of the edge between vertex v and its ith neighbor. The object $c_{v,i}$ is a pair of integers where the first component corresponds to $\tilde{t}_{h,\delta}^{(n)}(v, \gamma_{v,i}^{(n)})$ and the second corresponds to $\tilde{t}_{h,\delta}^{(n)}(\gamma_{v,i}^{(n)}, v)$. More precisely, we have $c_{v,i} = (J_n(\tilde{t}_{h,\delta}^{(n)}(v, \gamma_{v,i}^{(n)})), J_n(\tilde{t}_{h,\delta}^{(n)}(\gamma_{v,i}^{(n)}, v)))$ for all $v \in [n]$ and $i \in [d_v^{(n)}]$.

TCount: the number of explored messages at step $h-1$.

TIsStar: an Array of bits with size TCount, where for $1 \le i \le \mathsf{TCount}$, $\mathsf{TIsStar}(i)$ is 1 if the member of $\bar{\mathcal{F}}^{(\delta,h)}$ corresponding to integer i, i.e. $J_n^{-1}(i)$, is of the form $\star_x$, and $\mathsf{TIsStar}(i) = 0$ otherwise. In other words, $\mathsf{TIsStar}(i) = \mathbb{1}\left[J_n^{-1}(i) \notin \mathcal{F}^{(\delta,h)}\right]$.

TMark: an Array of integers of size TCount, where for $1 \le i \le \mathsf{TCount}$, $\mathsf{TMark}(i)$ is the mark component associated to the member of $\bar{\mathcal{F}}^{(\delta,h)}$ corresponding to integer i, i.e. $J_n^{-1}(i)$. In other words, if $\mathsf{TIsStar}(i) = 1$, with $J_n^{-1}(i) = \star_x$, we have $\mathsf{TMark}(i) = x$; otherwise, if $\mathsf{TIsStar}(i) = 0$, we have $\mathsf{TMark}(i) = (J_n^{-1}(i))[m]$, i.e. the mark component of $J_n^{-1}(i) \in \mathcal{F}^{(\delta,h)}$.

function $\textsc{ExtractTypes}(n, G^{(n)}, \delta, h)$
  $\mathsf{TDictionary} \leftarrow \mathsf{Dictionary}(\mathsf{Array\ of\ integers} \to \mathbb{N})$
  $\mathsf{TMark} \leftarrow$ Array of integers
  $\mathsf{TIsStar} \leftarrow$ Array of bits
  $\mathsf{T} = (\mathsf{T}_{v,i} : v \in [n], i \in [d_v^{(n)}]) \leftarrow$ Array of Array of integers ▷ array of messages
  $\mathsf{TCount} \leftarrow 0$ ▷ Number of elements in TDictionary
  for $1 \le v \le n$ **do** ▷ initialize messages at step 0
    for $1 \le i \le d_v^{(n)}$ **do**
      $\textsc{SendMessage}(v, i, (\theta_v^{(n)}, 0, x_{v,i}^{(n)}))$ ▷ Algorithm 15
    end for
  end for
  $\vec{s} \leftarrow$ Array of $\mathbb{N} \times \mathbb{N}$ with size δ
  for $1 \le k \le h-1$ **do**
    $\mathsf{TCount} \leftarrow 0$
    $\mathsf{TIsStarOld} \leftarrow \mathsf{TIsStar}$ ▷ corresponding to the previous step
    $\tilde{\mathsf{T}} \leftarrow \mathsf{T}$ ▷ messages from the previous step
    $\mathsf{TDictionary}, \mathsf{TIsStar}, \mathsf{TMark} \leftarrow \emptyset$ ▷ erase for the current step
    for $1 \le v \le n$ **do**
      if $d_v^{(n)} > \delta$ **then** ▷ in this case, all the neighbors will receive star messages
        for $1 \le i \le d_v^{(n)}$ **do**
          $\textsc{SendMessage}(v, i, (0, x_{v,i}^{(n)}))$ ▷ Algorithm 15
        end for
      else
        $n_\star \leftarrow 0$ ▷ number of neighbors which have sent a star message in the previous step
        for $1 \le i \le d_v^{(n)}$ **do**
          $s_i \leftarrow (\tilde{\mathsf{T}}_{\gamma_{v,i}^{(n)}, \tilde{\gamma}_{v,i}^{(n)}}, x_{v,i}^{(n)})$
          if $\mathsf{TIsStarOld}(\tilde{\mathsf{T}}_{\gamma_{v,i}^{(n)}, \tilde{\gamma}_{v,i}^{(n)}}) = 1$ **then**
            $n_\star \leftarrow n_\star + 1$
            $i_\star \leftarrow i$ ▷ index of the neighbor who has sent a star message
          end if
        end for
        if $n_\star \ge 2$ **then** ▷ all the neighbors will receive a star message
          for $1 \le i \le d_v^{(n)}$ **do**
            $\textsc{SendMessage}(v, i, (0, x_{v,i}^{(n)}))$ ▷ Algorithm 15
          end for
        else
          $(\pi, \tilde{s}) \leftarrow \textsc{Sort}(s_{[1:d_v^{(n)}]})$ ▷ $\tilde{s}$ is the sorted array such that $\tilde{s}_i = s_{\pi_i}$
          $\vec{t} \leftarrow$ Array of integers with maximum size $3 + 2\delta$
          if $n_\star = 1$ **then**
            $t_1 \leftarrow \theta_v^{(n)}, t_2 \leftarrow d_v^{(n)} - 1$ ▷ preparing $\mathsf{T}_{v,i_\star}$, the message towards $i_\star$

for $1 \le i \le d_v^{(n)}$ **do**
if $\pi_i \neq i_\star$ **then**
append the first and the second component of $\tilde{s}_i$ to $\vec{t}$
$\textsc{SendMessage}(v, i, (0, x_{v,i}^{(n)}))$ ▷ Algorithm 15
end if
end for
append $x_{v,i_\star}^{(n)}$ to $\vec{t}$
$\textsc{SendMessage}(v, i_\star, \vec{t})$ ▷ Algorithm 15
end if
if $n_\star = 0$ **then**
for $1 \le i \le d_v^{(n)}$ **do**
$\vec{t} \leftarrow \mathsf{Array}$ of size 2
$t_1 \leftarrow \theta_v^{(n)}, t_2 \leftarrow d_v^{(n)} - 1$
for $1 \le j \le d_v^{(n)}$ **do**
if $\pi_j \neq i$ **then**
append the first and the second components of $\tilde{s}_j$ to $\vec{t}$
end if
end for
append $x_{v,i}^{(n)}$ to $\vec{t}$
$\textsc{SendMessage}(v, i, \vec{t})$ ▷ Algorithm 15
end for
end if
end if
end if
end for
end for
for $1 \le v \le n$ **do**
for $1 \le i \le d_v^{(n)}$ **do**
if $\mathsf{TIsStar}(\mathsf{T}_{v,i}) = 0$ and $(\mathsf{TIsStar}(\mathsf{T}_{\gamma_{v,i}^{(n)}, \tilde{\gamma}_{v,i}^{(n)}}) = 1$ or $d_v^{(n)} > \delta$ or $d_{\gamma_{v,i}^{(n)}}^{(n)} > \delta)$ **then**
$\textsc{SendMessage}(v, i, (0, x_{v,i}^{(n)}))$ ▷ Algorithm 15
end if
end for
end for
$\vec{c} = (c_{v,i} : v \in [n], i \in [d_v^{(n)}]) \leftarrow \mathsf{Array}$ of $\mathbb{N} \times \mathbb{N}$ ▷ type of edges
for $1 \le v \le n$ **do**
for $1 \le i \le d_v^{(n)}$ **do**
$c_{v,i} \leftarrow (\mathsf{T}_{v,i}, \mathsf{T}_{\gamma_{v,i}^{(n)}, \tilde{\gamma}_{v,i}^{(n)}})$
end for
end for
return $(\vec{c}, \mathsf{TCount}, \mathsf{TIsStar}, \mathsf{TMark})$
end function

Algorithm 15. Sending a message from a node to one of its neighbors

Input:

v: the vertex from which the message is originated
i: the index of the neighbor of v to whom the message is being sent, so the message is from v towards $\gamma^{(n)}_{v,i}$
t: the message, which is an Array of integers

Output:
updates $\mathsf{TDictionary}, \mathsf{TMark}, \mathsf{TIsStar}$ and T

```
procedure SendMessage(v, i, t)
    if t ∈ TDictionary.Keys then
        T_{v,i} ← TDictionary(t)
    else
        TDictionary.Insert(t, 1 + TCount)
        T_{v,i} ← 1 + TCount
        TCount ← 1 + TCount
        append t_{Size(t)} at the end of TMark        ▷ the mark component is the last index in array t
        if t_1 = 0 then                                ▷ t is a star message iff its first component is zero
            append 1 at the end of TIsStar
        else
            append 0 at the end of TIsStar
        end if
    end if
end procedure
```

Algorithm 16. Computing $N_{i,j}(G)$ for a simple unmarked bipartite graph $G \in \mathcal{G}^{(n_l,n_r)}_{\vec{a},\vec{b}}$

Input:
n_l, n_r: the number of left and right nodes, respectively
$\vec{a}$: left degree sequence.
$\vec{\mathsf{b}} = (\mathsf{b}_v : v \in [n_r])$: $\mathsf{b}_v = b^G_v(i)$ for $v \in [n_r]$
i, j: endpoints of the interval, such that $1 \le i \le j \le n_l$
$\vec{\gamma}^G_{[i:j]} = \vec{\gamma}^G_i, \dots, \vec{\gamma}^G_j$: adjacency list of vertices $i \le v \le j$, where $\vec{\gamma}^G_v = (\gamma^G_{v,1}, \dots, \gamma^G_{v,a_v})$ such that $1 \le \gamma^G_{v,1} < \gamma^G_{v,2} < \dots < \gamma^G_{v,a_v} \le n_r$ are the right nodes adjacent to the left node v in G
U: Fenwick tree, where for $v \in [n_r]$, $\mathsf{U}.\textsc{Sum}(v) = U^G_v(i) = \sum_{k=v}^{n_r} b^G_k(i)$

Output:
$N_{i,j} = N_{i,j}(G)$
$\vec{\mathsf{b}} = (\mathsf{b}_v : v \in [n_r])$: vector $\vec{\mathsf{b}}$ updated, so that $\mathsf{b}_v = b^G_v(j+1)$ for $v \in [n_r]$
U: Fenwick tree U updated, so that for $v \in [n_r]$, we have $\mathsf{U}.\textsc{Sum}(v) = U^G_v(j+1) = \sum_{k=v}^{n_r} b^G_k(j+1)$
$l_{i,j} = l^G_{i,j}$

```
function BComputeN(n_l, n_r, a⃗, b⃗, i, j, γ⃗^G_{[i:j]}, U)        ▷ B stands for Bipartite
    if i = j then
        z_i ← 0, l_i ← 1
        for 1 ≤ k ≤ a_i do
            c ← a_i − k + 1
            y ← ComputeProduct(U.Sum(1 + γ^G_{i,k}), c, 1) ÷ ComputeProduct(c, c, 1)
                                                                   ▷ Algorithm 18
            z_i ← z_i + l_i × y
            l_i ← l_i × b_{γ^G_{i,k}}
```

$\mathsf{U}.\textsc{Add}(\gamma^G_{i,k}, -1)$
$\mathsf{b}_{\gamma^G_{i,k}} \leftarrow \mathsf{b}_{\gamma^G_{i,k}} - 1$
end for
$N_{i,i} \leftarrow z_i$
return $(N_{i,i}, \vec{\mathsf{b}}, \mathsf{U}, l_i)$
else
$k \leftarrow (i+j) \div 2$ ▷ $k = \lfloor (i+j)/2 \rfloor$
$(N_{i,k}, \vec{\mathsf{b}}, \mathsf{U}, l_{i,k}) \leftarrow \text{BC}\textsc{ompute}\text{N}(n_l, n_r, \vec{a}, \vec{\mathsf{b}}, i, k, \vec{\gamma}^G_{[i:k]}, \mathsf{U})$ ▷ Algorithm 16 (recursive)
$S_{k+1} \leftarrow \mathsf{U}.\textsc{Sum}(1)$
$(N_{k+1,j}, \vec{\mathsf{b}}, \mathsf{U}, l_{k+1,j}) \leftarrow \text{BC}\textsc{ompute}\text{N}(n_l, n_r, \vec{a}, \vec{\mathsf{b}}, k+1, j, \vec{\gamma}^G_{[k+1:j]}, \mathsf{U})$ ▷ Algorithm 16 (recursive)
$S_{j+1} \leftarrow \mathsf{U}.\textsc{Sum}(1)$
$y \leftarrow \textsc{ProdFactorial}(\vec{a}, k+1, j)$ ▷ $y = \prod_{v=k+1}^{j} a_v!$, Algorithm 19
$r_{k+1,j} \leftarrow \textsc{ComputeProduct}(S_{k+1}, S_{k+1} - S_{j+1}, 1) \div y$ ▷ Algorithm 18
$N_{i,j} \leftarrow N_{i,k} \times r_{k+1,j} + l_{i,k} \times N_{k+1,j}$
$l_{i,j} \leftarrow l_{i,k} \times l_{k+1,j}$
return $(N_{i,j}, \vec{\mathsf{b}}, \mathsf{U}, l_{i,j})$
end if
end function

Algorithm 17. Finding $f^{(n_l,n_r)}_{\vec{a},\vec{b}}(G)$ for $G \in \mathcal{G}^{(n_l,n_r)}_{\vec{a},\vec{b}}$

Input:

n_l, n_r: the number of left and right vertices, respectively
$\vec{a} = (a_i : i \in [n_l])$: left degree vector
$\vec{b} = (b_i : i \in [n_r])$: right degree vector
$(\vec{\gamma}^G_v : v \in [n_l])$: adjacency list of left nodes, where $\vec{\gamma}^G_v = (\gamma^G_{v,1}, \dots, \gamma^G_{v,a_v})$ for $v \in [n_l]$, such that $1 \le \gamma^G_{v,1} < \dots < \gamma^G_{v,a_v} \le n_r$ are the right nodes adjacent to the left node v

Output:

$f^{(n_l,n_r)}_{\vec{a},\vec{b}}(G)$: an integer representing G in the compressed form

function $\text{BE}\textsc{ncode}\text{G}\textsc{raph}(n_l, n_r, \vec{a}, \vec{b}, (\vec{\gamma}^G_v : v \in [n_l]))$
$\mathsf{U} \leftarrow$ Fenwick tree initialized with array $\vec{b}$ ▷ $\mathsf{U}.\textsc{Sum}(v) = \sum_{k=v}^{n_r} b_k = U^G_v(1)$ for $v \in [n_r]$
$\vec{\mathsf{b}} \leftarrow \vec{b}$ ▷ $\mathsf{b}_v = b^G_v(1) = b_v$ for $v \in [n_r]$
$(N_{1,n_l}, \vec{\mathsf{b}}, \mathsf{U}, l_{1,n_l}) \leftarrow \text{BC}\textsc{ompute}\text{N}(n_l, n_r, \vec{a}, \vec{\mathsf{b}}, 1, n_l, \vec{\gamma}^G_{[1:n_l]}, \mathsf{U})$ ▷ Algorithm 16 above
$f \leftarrow N \div l_{1,n_l}$ ▷ $l_{1,n_l} = l^G_{1,n_l} = \prod_{j=1}^{n_r} b_j!$
if $f \times l_{1,n_l} < N$ **then** ▷ this means $N \mod l_{1,n_l} \neq 0$
$f \leftarrow f + 1$ ▷ so that $f = f^{(n_l,n_r)}_{\vec{a},\vec{b}}(G) = \lceil N / l_{1,n_l} \rceil$
end if
return f
end function

Algorithm 18. Computing $\prod_{k'=0}^{k-1} (p - k's)^+$

Input:

Integer $p \geq 0$: the first term in the product
Integer $k \geq 0$: the number of terms in the product
Integer $s \geq 0$: the difference between successive terms

Output:

If $k = 0$, return 1. If $k > 0$ and $p - (k-1)s \leq 0$, return 0. If $k > 0$ and $p - (k-1)s > 0$, returns $\prod_{i=0}^{k-1}(p - is)$.

function ComputeProduct(p, k, s)
  if $k = 0$ **then** ▷ product over empty set is 1
    return 1
  end if
  if $p - (k-1)s \leq 0$ **then**
    return 0
  end if
  if $k = 1$ **then** ▷ there is only one term
    return p
  end if
  $k' \leftarrow k \div 2$ ▷ we compute the product by dividing the terms into two halves
  $L \leftarrow$ ComputeProduct(p, k', s) ▷ product of the first half
  $R \leftarrow$ ComputeProduct($p - k's, k - k', s$) ▷ product of the second half
  return $L \times R$ ▷ aggregate the two pieces to get the final result
end function

Algorithm 19. Computing $\prod_{i'=i}^{j} v_{i'}!$ for an array $\vec{v}$ of nonnegative integers

Input:

$\vec{v}$: array of nonnegative integers
i, j: endpoints of the interval for which the product is being computed

Output:

$\prod_{i'=i}^{j} v_{i'}!$

function ProdFactorial($\vec{v}, i, j$)
  if $i = j$ **then**
    return ComputeProduct($v_i, v_i, 1$) ▷ return $v_i!$ using Algorithm 18 above
  else
    $m \leftarrow (i + j) \div 2$ ▷ split the interval into two halves
    $L \leftarrow$ ProdFactorial($\vec{v}, i, m$)
    $R \leftarrow$ ProdFactorial($\vec{v}, m + 1, j$)
    return $L \times R$
  end if
end function

Algorithm 20. Decoding the adjacency list of a left vertex $1 \leq i \leq n_l$ given $\widetilde{N}_{i,i}$ for a simple unmarked bipartite graph $G \in \mathcal{G}_{\vec{a},\vec{b}}^{(n_l,n_r)}$.

Input:

n_l, n_r: the number of left and right nodes, respectively
i: the index of the left node to be decoded, $1 \le i \le n_l$
$\vec{a}$: left degree sequence
$\vec{\mathsf{b}} = (\mathsf{b}_v : v \in [n_r])$: Array of integers where $\mathsf{b}_v = b_v^G(i)$ for $1 \le v \le n_r$
$\widetilde{N}_{i,i}$: integer satisfying $N_{i,i}(G) \le \widetilde{N}_{i,i} < N_{i,i}(G) + l_i^G$
U: Fenwick tree, where for $1 \le v \le n_r$, $\mathsf{U}.\textsc{Sum}(v) = U_v^G(i) = \sum_{k=v}^{n_r} b_k^G(i)$

Output:

$N_{i,i} = N_{i,i}(G)$
$\vec{\gamma}_i$: the decoded adjacency list of vertex i, so that $\vec{\gamma}_i = \vec{\gamma}_i^G = (\gamma_{i,k}^G : 1 \le k \le a_i)$
$\vec{\mathsf{b}} = (\mathsf{b}_v : 1 \le v \le n_r)$: array $\vec{\mathsf{b}}$ updated so that $\mathsf{b}_v = b_v^G(i+1)$ for $1 \le v \le n_r$
U: Fenwick tree U updated, so that for $1 \le v \le n_r$, we have $\mathsf{U}.\textsc{Sum}(v) = U_v^G(i+1) = \sum_{k=v}^{n_r} b_k^G(i+1)$
l_i: integer such that $l_i = l_i^G$

function $\textsc{BDecodeNode}(n_l, n_r, i, \vec{a}, \vec{\mathsf{b}}, \widetilde{N}_{i,i}, \mathsf{U})$
  $\widetilde{z} \leftarrow \widetilde{N}_{i,i}$
  $z_i \leftarrow 0, l_i \leftarrow 1$
  for $1 \le k \le a_i$ **do**
    $q \leftarrow a_i - k + 1$
    $L \leftarrow 1, R \leftarrow n_r$ ▷ L and R are the endpoints of the binary search interval
    if $k > 1$ **then**
      $L \leftarrow 1 + \gamma_{i,k-1}$ ▷ If $k > 1$, $1 + \gamma_{i,k-1} \le \gamma_{i,k}$, so limit the search
    end if
    while $R > L$ **do** ▷ binary search on the interval $[f, g]$ to find $\gamma_{i,k}^G$
      $v \leftarrow (L + R) \div 2$ ▷ $v = \lfloor (L+R)/2 \rfloor$ is the midpoint
      $y \leftarrow \textsc{ComputeProduct}(\mathsf{U}.\textsc{Sum}(1+v), q, 1) \div \textsc{ComputeProduct}(q, q, 1)$
      ▷ $y = \binom{U_{1+v}^G(i)}{a_i - k + 1}$, Algorithm 18
      if $y \le \widetilde{z}$ **then**
        $R \leftarrow v$ ▷ switch to interval $[L, v]$
      else
        $L \leftarrow v + 1$ ▷ switch to interval $[v+1, R]$
      end if
    end while
    $\gamma_{i,k} \leftarrow L$
    $y \leftarrow \textsc{ComputeProduct}(\mathsf{U}.\textsc{Sum}(1 + \gamma_{i,k}), q, 1) \div \textsc{ComputeProduct}(q, q, 1)$
    ▷ $y = \binom{U_{1+\gamma_{i,k}^G}^G(i)}{a_i - k + 1}$, Algorithm 18
    $\widetilde{z} \leftarrow (\widetilde{z} - y) \div \mathsf{b}_{\gamma_{i,k}}$
    $z_i \leftarrow z_i + l_i \times y$ ▷ here, $l_i = \prod_{k'=1}^{k-1} b_{\gamma_{i,k'}^G}^G(i)$
    $l_i \leftarrow l_i \times \mathsf{b}_{\gamma_{i,k}}$ ▷ l_i becomes $\prod_{k'=1}^{k} b_{\gamma_{i,k'}^G}^G(i)$
    $\mathsf{U}.\textsc{Add}(\gamma_{i,k}, -1)$
    $\mathsf{b}_{\gamma_{i,k}} \leftarrow \mathsf{b}_{\gamma_{i,k}} - 1$
  end for
  $N_{i,i} \leftarrow z_i$
  return $(N_{i,i}, \vec{\gamma}_i, \mathsf{b}, \mathsf{U}, l_i)$
end function

Algorithm 21. Decoding the adjacency list of the left vertices $i \le v \le j$ for $1 \le i \le j \le n_l$ given $\widetilde{N}_{i,j}$ for a simple unmarked bipartite graph $G \in \mathcal{G}^{(n_l,n_r)}_{\vec{a},\vec{b}}$.

Input:

n_l, n_r: the number of left and right nodes, respectively
$1 \le i \le j \le n_l$: the endpoints of the interval to be decoded
$\vec{a}$: left degree sequence
$\vec{\mathsf{b}} = (\mathsf{b}_v : 1 \le v \le n_r)$ where $\mathsf{b}_v = b^G_v(i)$ for $1 \le v \le n_r$
$\widetilde{N}_{i,j}$: integer satisfying $N_{i,j}(G) \le \widetilde{N}_{i,j} < N_{i,j}(G) + l^G_{i,j}$
U: Fenwick tree, where for $1 \le v \le n_r$, $\mathsf{U}.\textsc{Sum}(v) = U^G_v(i) = \sum_{k'=v}^{n_r} b^G_{k'}(i)$
W: Fenwick tree, where for $1 \le v \le n_l$, we have $\mathsf{W}.\textsc{Sum}(v) = \sum_{k'=v}^{n_l} a_{k'}$.

Output:

$N_{i,j} = N_{i,j}(G)$
$\vec{\gamma}_{[i:j]}$: the decoded adjacency list of the vertices $i \le v \le j$, so that $\vec{\gamma}_v = \vec{\gamma}^G_v$ for $i \le v \le j$
$\vec{\mathsf{b}} = (\mathsf{b}_v : 1 \le v \le n_r)$: array $\vec{\mathsf{b}}$ updated so that $\mathsf{b}_v = b^G_v(j+1)$ for $1 \le v \le n_r$
U: Fenwick tree U updated, so that for $1 \le v \le n_r$, we have $\mathsf{U}.\textsc{Sum}(v) = U^G_v(j+1) = \sum_{k'=v}^{n_r} b^G_{k'}(j+1)$
$l_{i,j} = l^G_{i,j}$

```
function BDecodeInterval(n_l, n_r, i, j, a⃗, b⃗, Ñ_{i,j}, U, W)
    if i = j then
        return BDecodeNode(n_l, n_r, i, a⃗, b⃗, Ñ_{i,i}, U)                ▷ Algorithm 20
    else
        k ← (i + j) ÷ 2
        S_{k+1} ← W.Sum(k + 1)
        S_{j+1} ← W.Sum(j + 1)
        r_{k+1,j} ← ComputeProduct(S_{k+1}, S_{k+1} − S_{j+1}, 1)/ProdFactorial(a⃗, k + 1, j)
                                                                  ▷ Algorithms 18 and 19
        Ñ_{i,k} ← Ñ_{i,j} ÷ r_{k+1,j}
        (N_{i,k}, γ⃗_{[i,k]}, b⃗, U, l_{i,k}) ← BDecodeInterval(n_l, n_r, i, k, a⃗, b⃗, Ñ_{i,k}, U, W)
                                                                  ▷ Algorithm 21 (recursive)
        Ñ_{k+1,j} ← (Ñ_{i,j} − N_{i,k} × r_{k+1,j}) ÷ l_{i,k}
        (N_{k+1,j}, γ⃗_{[k+1,j]}, b⃗, U, l_{k+1,j}) ← BDecodeInterval(n_l, n_r, k + 1, j, a⃗, b⃗, Ñ_{k+1,j}, U, W)
                                                                  ▷ Algorithm 21 (recursive)
        N_{i,j} ← N_{i,k} × r_{k+1,j} + l_{i,k} × N_{k+1,j}
        l_{i,j} ← l_{i,k} × l_{k+1,j}
        return (N_{i,j}, γ⃗_{[i:j]}, b⃗, U, l_{i,j})
    end if
end function
```

Algorithm 22. Decoding for a simple unmarked bipartite graph $G \in \mathcal{G}^{(n_l,n_r)}_{\vec{a},\vec{b}}$ given $f^{(n_l,n_r)}_{\vec{a},\vec{b}}(G)$

Input:

f: integer, which is $f^{(n_l,n_r)}_{\vec{a},\vec{b}}(G)$ for the graph G that was given to the encoder during the compression phase
$\vec{a}$: array of left degrees
$\vec{b}$: array of right degrees

Output:
$\vec{\gamma}_{[1:n_l]}$: adjacency list of left nodes such that $\vec{\gamma}_v = \vec{\gamma}^G_v = (\gamma^G_{v,1} < \cdots < \gamma^G_{v,a_v})$ for $1 \le v \le n_l$

```
function BDecodeGraph(f, a⃗, b⃗)
    n_l ← Size(a⃗)
    n_r ← Size(b⃗)
    c ← ProdFactorial(b⃗, 1, n_r)                         ▷ c = ∏_{i=1}^{n_r} b_i! using Algorithm 19
    Ñ_{1,n_l} ← f × c
    U ← Fenwick tree initialized with array b⃗
    W ← Fenwick tree initialized with array a⃗
    b⃗ ← b⃗
    (N_{1,n_l}, γ⃗_{[1:n_l]}, b⃗, U, l_{1,n_l}) ← BDecodeInterval(n_l, n_r, 1, n_l, a⃗, b⃗, Ñ_{1,n_l}, U, W)
                                                          ▷ Algorithm 21
    return γ⃗_{[1:n_l]}
end function
```

Algorithm 23. Computing $N_{i,j}(G)$ for a simple unmarked graph $G \in \mathcal{G}^{(\tilde{n})}_{\vec{a}}$

Input:
$\tilde{n}$: number of nodes
i, j: endpoints of the interval, such that $1 \le i \le j \le \tilde{n}$
$\vec{\mathsf{a}} = (\mathsf{a}_v : 1 \le v \le \tilde{n})$ where $\mathsf{a}_v = a^G_v(i)$ for $i \le v \le \tilde{n}$
$\vec{\gamma}^G_i, \dots, \vec{\gamma}^G_j$: forward adjacency list of vertices $i \le v \le j$, where $\vec{\gamma}^G_v = (\vec{\gamma}^G_{v,1}, \dots, \vec{\gamma}^G_{v,\hat{a}^G_v})$ such that $v < \gamma^G_{v,1} < \cdots < \gamma^G_{v,\hat{a}^G_v} \le \tilde{n}$ are the neighbors of v in G with index greater than v
U: Fenwick tree, where for $i \le v \le \tilde{n}$, $\mathsf{U}.\textsc{Sum}(v) = U^G_v(i) = \sum_{k=v}^{\tilde{n}} a^G_k(i)$
I: an integer specifying the interval $[i, j]$
$\vec{\tilde{f}} = (\tilde{f}_p : 0 \le p \le \lfloor 16\tilde{n}/\log^2 \tilde{n} \rfloor)$: array of integers where if $j - i + 1 > \lfloor \log_2 \tilde{n} \rfloor^2$, $\tilde{f}_I = S^G_{k+1}$ with $k = \lfloor (i+j)/2 \rfloor$ and I being the index corresponding to the interval $[i, j]$ as above

Output:
$N_{i,j} = N_{i,j}(G)$
$\vec{\mathsf{a}} = (\mathsf{a}_v : 1 \le v \le \tilde{n})$: vector $\vec{\mathsf{a}}$ updated, so that $\mathsf{a}_v = a^G_v(j+1)$ for $j + 1 \le v \le \tilde{n}$
U: Fenwick tree U updated, so that for $j + 1 \le v \le \tilde{n}$, we have $\mathsf{U}.\textsc{Sum}(v) = U^G_v(j+1) = \sum_{k=v}^{\tilde{n}} a^G_k(j+1)$
$l_{i,j} = l^G_{i,j}$
$\vec{\tilde{f}}$: array $\vec{\tilde{f}}$ updated, so that if $j - i + 1 > \lfloor \log_2 \tilde{n} \rfloor^2$, $\tilde{f}_I = S^G_{k+1}$ with $k = \lfloor (i+j)/2 \rfloor$ and I being the index corresponding to the interval $[i, j]$ as above

```
function ComputeN(ñ, a⃗, i, j, γ⃗^G_{[i:j]}, U, I, f̃⃗)
    if i = j then
        z_i ← 0, l_i ← 1
        for 1 ≤ k ≤ a_i do
            y ← ComputeProduct(U.Sum(1 + γ^G_{i,k}), a_i − k + 1, 1)        ▷ Algorithm 18
```

$z_i \leftarrow z_i + l_i \times y$
$c \leftarrow (\mathsf{a}_i - k + 1) \times \mathsf{a}_{\gamma^G_{i,k}}$
$l_i \leftarrow l_i \times c$
$\mathsf{a}_{\gamma^G_{i,k}} \leftarrow \mathsf{a}_{\gamma^G_{i,k}} - 1$
$\mathsf{U}.\textsc{Add}(\gamma^G_{i,k}, -1)$
end for
$N_{i,i} \leftarrow z_i$
return $(N_{i,i}, \vec{\mathsf{a}}, \mathsf{U}, l_i, \vec{\tilde{f}})$
else
$k \leftarrow (i + j) \div 2$
$(N_{i,k}, \vec{\mathsf{a}}, \mathsf{U}, l_{i,k}, \vec{\tilde{f}}) \leftarrow \textsc{ComputeN}(\tilde{n}, \vec{\mathsf{a}}, i, k, \vec{\gamma}^G_{[i:k]}, \mathsf{U}, 2I, \vec{\tilde{f}})$
$S_{k+1} \leftarrow \mathsf{U}.\textsc{Sum}(k+1)$
if $j - i + 1 > \lfloor \log_2 \tilde{n} \rfloor^2$ **then**
$\tilde{f}_I \leftarrow S_{k+1}$
end if
$(N_{k+1,j}, \vec{\mathsf{a}}, \mathsf{U}, l_{k+1,j}, \vec{\tilde{f}}) \leftarrow \textsc{ComputeN}(\tilde{n}, \vec{\mathsf{a}}, k+1, j, \vec{\gamma}^G_{[k+1:j]}, \mathsf{U}, 2I+1, \vec{\tilde{f}})$
$S_{j+1} \leftarrow \mathsf{U}.\textsc{Sum}(j+1)$
$r_{k+1,j} \leftarrow \textsc{ComputeProduct}(S_{k+1} - 1, (S_{k+1} - S_{j+1})/2, 2)$ ▷ Algorithm 18
$N_{i,j} \leftarrow N_{i,k} \times r_{k+1,j} + l_{i,k} \times N_{k+1,j}$
$l_{i,j} \leftarrow l_{i,k} \times l_{k+1,j}$
return $(N_{i,j}, \vec{\mathsf{a}}, \mathsf{U}, l_{i,j}, \vec{\tilde{f}})$
end if
end function

Algorithm 24. Finding $f^{(\tilde{n})}_{\vec{a}}(G)$ and $\vec{\tilde{f}}^{(\tilde{n})}_{\vec{a}}(G)$ for $G \in \mathcal{G}^{(n_l,n_r)}_{\vec{a}}$

Input:
$\tilde{n}$: number of nodes
$\vec{a} = (a_v : 1 \leq v \leq \tilde{n})$: degree vector
$\vec{\gamma}^G = (\vec{\gamma}^G_v : 1 \leq v \leq \tilde{n})$: forward adjacency list where $\vec{\gamma}^G_v = (\gamma^G_{v,1} < \cdots < \gamma^G_{v,\hat{a}^G_v})$ is the forward adjacency list of node v

Output:
$f = f^{(\tilde{n})}_{\vec{a}}(G)$
$\vec{\tilde{f}} = (\tilde{f}_i : 1 \leq i \leq \lfloor 16\tilde{n}/\log^2 \tilde{n} \rfloor)$: Array of integers such that $\tilde{f}_i = \tilde{f}^{(n)}_{\vec{a},i}(G)$

function $\textsc{EncodeGraph}(\tilde{n}, \vec{a}, \vec{\gamma}^G)$
$\mathsf{U} \leftarrow$ Fenwick tree initialized with array $\vec{a}$
$\vec{\tilde{f}} \leftarrow$ Array of integers with length $\lfloor 16\tilde{n}/\log^2 \tilde{n} \rfloor$
$\vec{\mathsf{a}} \leftarrow \vec{a}$ ▷ $a^G_i(1) = a_i$ for $1 \leq i \leq \tilde{n}$
$(N_{1,\tilde{n}}, \vec{\mathsf{a}}, \mathsf{U}, l_{1,\tilde{n}}, \vec{\tilde{f}}) \leftarrow \textsc{ComputeN}(\tilde{n}, \vec{a}, 1, \tilde{n}, \vec{\gamma}^G_{[1:\tilde{n}]}, \mathsf{U}, 1, \vec{\tilde{f}})$ ▷ Algorithm 23 above
$f \leftarrow N \div l_{1,\tilde{n}}$
if $f \times l_{1,\tilde{n}} < N$ **then** ▷ this means $N \mod l_{1,\tilde{n}} \neq 0$
$f \leftarrow f + 1$ ▷ so that $f = \lceil N/l_{1,\tilde{n}} \rceil$
end if

return $(f, \vec{\tilde{f}})$
end function

Algorithm 25. Decoding the forward adjacency list of a vertex $1 \le i \le \tilde{n}$ given $\widetilde{N}_{i,i}$ for a simple unmarked graph $G \in \mathcal{G}_{\vec{a}}^{(\tilde{n})}$

Input:

$\tilde{n}$: number of nodes in the graph
$1 \le i \le \tilde{n}$: the index of the node to be decoded
$\widetilde{N}_{i,i}$: integer satisfying $N_{i,i}(G) \le \widetilde{N}_{i,i} < N_{i,i}(G) + l_i^G$
$\vec{\mathsf{a}} = (\mathsf{a}_v : 1 \le v \le \tilde{n})$, where $\mathsf{a}_v = a_v^G(i)$ for $i \le v \le \tilde{n}$
U: Fenwick tree, where for $i \le v \le \tilde{n}$, $\mathsf{U}.\textsc{Sum}(v) = U_v^G(i) = \sum_{k=v}^{\tilde{n}} a_k^G(i)$

Output:

$N_{i,i} = N_{i,i}(G)$.
$\vec{\gamma}_i = (\gamma_{i,k} : 1 \le k \le \hat{a}_i^G)$: forward adjacency list of the graph for vertex i, such that $\gamma_{i,k} = \gamma_{i,k}^G$ for $1 \le k \le \hat{a}_i^G$
$\vec{\mathsf{a}} = (\mathsf{a}_v : 1 \le v \le \tilde{n})$: vector $\vec{\mathsf{a}}$ updated, so that $\mathsf{a}_v = a_v^G(i+1)$ for $i < v \le \tilde{n}$.
U: Fenwick tree U updated, so that for $i+1 \le v \le \tilde{n}$, we have $\mathsf{U}.\textsc{Sum}(v) = \sum_{k=v}^{\tilde{n}} a_k^G(i+1)$
$l_i = l_i^G$.

function $\textsc{DecodeNode}(\tilde{n}, i, \widetilde{N}_{i,i}, \vec{\mathsf{a}}, \mathsf{U})$
  $\widetilde{z} \leftarrow \widetilde{N}_{i,i}$
  $l_i \leftarrow 1$
  $z_i \leftarrow 0$
  for $1 \le k \le \mathsf{a}_i$ **do**
    if $k = 1$ **then**
      $L \leftarrow i + 1$ ▷ we do a binary search on the interval $[L, R]$
    else
      $L \leftarrow \gamma_{i,k-1} + 1$ ▷ since $\gamma_{i,k-1}^G + 1 \le \gamma_{i,k}^G$
    end if
    $R \leftarrow \tilde{n}$ ▷ since $\gamma_{i,k}^G \le \tilde{n}$
    while $R > L$ **do** ▷ binary search to find $\gamma_{i,k}$
      $v \leftarrow (L + R) \div 2$
      if $\textsc{ComputeProduct}(\mathsf{U}.\textsc{Sum}(1+v), \mathsf{a}_i - k + 1, 1) \le \widetilde{z}$ **then** ▷ Algorithm 18
        $R \leftarrow v$ ▷ switch to $[L, v]$
      else
        $L \leftarrow v + 1$ ▷ switch to $[v+1, R]$
      end if
    end while ▷ when the loop is over, we have $L = R = \gamma_{i,k}^G$
    $\gamma_{i,k} \leftarrow L$
    $y \leftarrow \textsc{ComputeProduct}(\mathsf{U}.\textsc{Sum}(1+\gamma_{i,k}), \mathsf{a}_i - k + 1, 1)$ ▷ $y = (U_{1+\gamma_{i,k}^G}^G(i))_{\hat{a}_i^G - k + 1}$ ▷ Algorithm 18
    $z_i \leftarrow z_i + l_i \times y$
    $c \leftarrow (\mathsf{a}_i - k + 1) \times \mathsf{a}_{\gamma_{i,k}}$ ▷ $c = (\hat{a}_i^G - k + 1) a_{\gamma_{i,k}^G}^G(i)$
    $l_i \leftarrow l_i \times c$ ▷ updating l_i

$\mathsf{a}_{\gamma_{i,k}} \leftarrow \mathsf{a}_{\gamma_{i,k}} - 1$ ▷ updating $\vec{\mathsf{a}}$
$\mathsf{U}.\textsc{Add}(\gamma_{i,k}, -1)$ ▷ updating the Fenwick tree
$\widetilde{z} \leftarrow \widetilde{z} - y$ ▷ subtracting the contribution of $\gamma_{i,k}$
$\widetilde{z} \leftarrow \widetilde{z} \div c$ ▷ $\widetilde{z}$ is updated so that it becomes $\widetilde{z}_{i,k+1}$
end for
$N_{i,i} \leftarrow z_i$
return $(N_{i,i}, \vec{\gamma}_i, \vec{\mathsf{a}}, \mathsf{U}, l_i)$
end function

Algorithm 26. Decoding the forward adjacency list of vertices $i \leq v \leq j$ given $\widetilde{N}_{i,j}$ for a simple unmarked graph $G \in \mathcal{G}_{\vec{a}}^{(\tilde{n})}$

Input:
$\tilde{n}$: number of nodes in the graph
$1 \leq i \leq j \leq \tilde{n}$: the interval to be decoded
$\widetilde{N}_{i,j}$: integer satisfying $N_{i,j}(G) \leq \widetilde{N}_{i,j} < N_{i,j}(G) + l_{i,j}^G$
$\vec{\mathsf{a}} = (\mathsf{a}_v : 1 \leq v \leq \tilde{n})$ where $\mathsf{a}_v = a_v^G(i)$ for $i \leq v \leq \tilde{n}$
U: Fenwick tree, where for $i \leq v \leq \tilde{n}$, $\mathsf{U}.\textsc{Sum}(v) = U_v^G(i) = \sum_{k=v}^{\tilde{n}} a_k^G(i)$
I: an integer specifying the interval $[i,j]$
S_{j+1}: which is $S_{j+1}^G = \sum_{k=j+1}^{\tilde{n}} a_k^G(j+1)$
$\vec{\tilde{f}} = (\tilde{f}_p : 0 \leq p \leq \lfloor 16\tilde{n}/\log^2 \tilde{n} \rfloor)$: array of integers where if $j - i + 1 > \lfloor \log_2 \tilde{n} \rfloor^2$, $\tilde{f}_I = S_{k+1}^G$ with $k = \lfloor (i+j)/2 \rfloor$ and I being the index corresponding to the interval $[i,j]$ as above

Output:
$N_{i,j} = N_{i,j}(G)$.
$\vec{\gamma}_i, \dots, \vec{\gamma}_j$: forward adjacency list of vertices in the interval $[i,j]$, such that $\vec{\gamma}_v = (\gamma_{v,k} : 1 \leq k \leq \hat{a}_v^G)$ where $\gamma_{v,k} = \gamma_{v,k}^G$ for $i \leq v \leq j$ and $1 \leq k \leq \hat{a}_v^G$
$\vec{\mathsf{a}} = (\mathsf{a}_v : 1 \leq v \leq \tilde{n})$: array $\vec{\mathsf{a}}$ updated, so that $\mathsf{a}_v = a_v^G(j+1)$ for $j+1 \leq v \leq \tilde{n}$.
U: Fenwick tree U updated, so that for $j+1 \leq v \leq \tilde{n}$, we have $\mathsf{U}.\textsc{Sum}(v) = U_v^G(j+1) = \sum_{k=v}^{\tilde{n}} a_k^G(j+1)$.
$l_{i,j} = l_{i,j}^G$.

function $\textsc{DecodeInterval}(\tilde{n}, i, j, \widetilde{N}_{i,j}, \vec{\mathsf{a}}, \mathsf{U}, I, S_{j+1}, \vec{\tilde{f}})$
if $i = j$ **then**
$(N_{i,i}, \vec{\gamma}_i, \vec{\mathsf{a}}, \mathsf{U}, l_i) \leftarrow \textsc{DecodeNode}(\tilde{n}, i, \widetilde{N}_{i,i}, \vec{\mathsf{a}}, U)$ ▷ Algorithm 25
return $(N_{i,i}, \vec{\gamma}_i, \vec{\mathsf{a}}, \mathsf{U}, l_i)$
end if
if $j - i + 1 > \lfloor \log_2 \tilde{n} \rfloor^2$ **then** ▷ specifying the midpoint k
$k \leftarrow (i+j) \div 2$
$S_{k+1} \leftarrow \tilde{f}_I$
else
$k \leftarrow i$
$S_{k+1} \leftarrow \mathsf{U}.\textsc{Sum}(i) - 2\mathsf{a}_i$
end if
$r_{k+1,j} \leftarrow \textsc{ComputeProduct}(S_{k+1} - 1, (S_{k+1} - S_{j+1})/2, 2)$ ▷ Algorithm 18
▷ finding $\widetilde{N}_{i,k}$ for the left interval $[i,k]$ and decoding $[i,k]$:

$\widetilde{N}_{i,k} \leftarrow \widetilde{N}_{i,j} \div r_{k+1,j}$

$(N_{i,k}, \vec{\gamma}_{[i:k]}, \vec{\mathsf{a}}, \mathsf{U}, l_{i,k}) \leftarrow$ DecodeInterval$(\tilde{n}, i, k, \widetilde{N}_{i,k}, \vec{\mathsf{a}}, \mathsf{U}, 2I, S_{k+1}, \vec{\tilde{f}})$

$\triangleright$ finding $\widetilde{N}_{k+1,j}$ for the right interval $[k+1, j]$ and decoding $[k+1, j]$:

$\widetilde{N}_{k+1,j} \leftarrow (\widetilde{N}_{i,j} - N_{i,k} \times r_{k+1,j}) \div l_{i,k}$

$(N_{k+1,j}, \vec{\gamma}_{[k+1:j]}, \vec{\mathsf{a}}, \mathsf{U}, l_{k+1,j}) \leftarrow$ DecodeInterval$(\tilde{n}, k+1, j, \widetilde{N}_{k+1,j}, \vec{\mathsf{a}}, \mathsf{U}, 2I+1, S_{j+1}, \vec{\tilde{f}})$

$N_{i,j} \leftarrow N_{i,k} \times r_{k+1,j} + l_{i,k} \times N_{k+1,j}$

$l_{i,j} \leftarrow l_{i,k} \times l_{k+1,j}$

return $(N_{i,j}, \vec{\gamma}_{[i:j]}, \vec{\mathsf{a}}, \mathsf{U}, l_{i,j})$

end function

Algorithm 27. Decoding for a simple unmarked graph $G \in \mathcal{G}_{\vec{a}}^{(\tilde{n})}$ given $f_{\vec{a}}^{(\tilde{n})}(G)$ and $\vec{\tilde{f}}_{\vec{a}}^{(\tilde{n})}(G)$

Input:

f: integer, which is $f_{\vec{a}}^{(\tilde{n})}(G)$ for the target graph $G \in \mathcal{G}_{\vec{a}}^{(\tilde{n})}$ which was given to encoder during the compression phase

$\vec{\tilde{f}} = (\tilde{f}_i : 1 \leq i \leq \lfloor 16\tilde{n}/\log^2 \tilde{n} \rfloor)$: Array of integers, where $\tilde{f}_i = \tilde{f}_{\vec{a},i}^{(\tilde{n})}(G)$ for $1 \leq i \leq \lfloor 16\tilde{n}/\log^2 \tilde{n} \rfloor$

$\vec{a}$: array of vertex degrees

Output:

$\vec{\gamma}_{[1:\tilde{n}]}$: the decoded forward adjacency list such that $\vec{\gamma}_v = \vec{\gamma}_v^G = (v < \gamma_{v,1}^G < \cdots < \gamma_{v,\hat{a}_v^G}^G)$ for $1 \leq v \leq \tilde{n}$

function GraphDecode$(f, \vec{\tilde{f}}, \vec{a})$

$\tilde{n} \leftarrow$ Size$(\vec{a})$

$c \leftarrow$ ProdFactorial$(\vec{a}, 1, \tilde{n})$ $\triangleright$ Algorithm 19

$\widetilde{N}_{1,\tilde{n}} \leftarrow f \times c$

$\mathsf{U} \leftarrow$ Fenwick tree initialized with array $\vec{a}$

$\vec{\mathsf{a}} \leftarrow \vec{a}$ $\triangleright$ $a_v^G(1) = a_v$ for $1 \leq v \leq \tilde{n}$

$(N_{1,\tilde{n}}, \vec{\gamma}_{[1:\tilde{n}]}, \vec{\mathsf{a}}, \mathsf{U}, l_{1,\tilde{n}}) \leftarrow$ DecodeInterval$(\tilde{n}, 1, \tilde{n}, \widetilde{N}_{1,\tilde{n}}, \vec{\mathsf{a}}, \mathsf{U}, 0, \vec{\tilde{f}})$ $\triangleright$ Algorithm 26

return $\vec{\gamma}_{[1:\tilde{n}]}$

end function
