Principles of Distributed Computing Summary

Ulla Aeschbacher

4.8.18

Contents

List of Theorems

List of Definitions

List of Algorithms

Chapter 1

Definitions

1.1 General Graph Stuff

Definition 1 BFS_v**:** Performing a breadth first search at node v produces spanning tree BFS_v. This takes time $\mathcal{O}(\Delta)$ using small messages.

Definition 2 Blue Edge: Cheapest edge that connects two subtrees. Formally: Let T be a spanning tree of the weighted graph G and $T' \subseteq T$ a subgraph of T. The minimum weight outgoing edge $b(T')$ is the so-called blue edge of T'.

Theorem 1: For a given weighted graph G, let T denote the MST and T' be a fragment of T. Then the blue edge of T' is also part of T.

Definition 3 Chromatic Number: Given an undirected graph $G = (V, E)$, the chromatic number $\chi(G)$ is the minimum number of colors to solve the vertex coloring problem.

Theorem 2: $\chi(\text{Tree}) \leq 2$

Definition 4 Clean: A graph is clean if the nodes do not know the topology of the graph.

Definition 5 Coordinator Model: We have n players numbered 1 to n, as well as an arbitrary n-node graph G. The i^{th} player knows the edges incident on the i^{th} node.

Definition 6 Degree: The number of neighbors of a vertex v, denoted by $\delta(v)$, is called the degree of v. The maximum degree in a graph G defines the graph degree $\Delta(G) = \Delta$.

Definition 7 Diameter: The diameter of a graph is the maximum distance between two arbitrary nodes in a graph.

Theorem 3 Lower Bound on Computing the Diameter: Any distributed algorithm A that computes the diameter of a graph needs $\Omega\left(\frac{n}{\log n}\right)$ time.

Definition 8 Distance: The distance between two nodes u and v in an undirected graph G is the number of hops of a minimum path between u and v.

Definition 9 Graph-Familiy $\mathcal{G}$**:** We assume that $(n-2)$ can be divided by 8. We define four sets of nodes, each consisting of $q = q(n) = \frac{n-2}{4}$ nodes:

- $L_0 = \{l_i | i \in [q]\}$
- $L_1 = \{l'_i | i \in [q]\}$
- $R_0 = \{r_i | i \in [q]\}$
- $R_1 = \{r'_i | i \in [q]\}$

We define $G' = (V', E')$ as

$$V' = L_0 \cup L_1 \cup R_0 \cup R_1 \cup \{c_L, c_R\}$$

$$\begin{aligned} E' = & \bigcup_{v \in L_0 \cup L_1} \{(v, c_L)\} \\ \cup & \bigcup_{v \in R_0 \cup R_1} \{(v, c_R)\} \\ \cup & \bigcup_{i \in [q]} \{(l_i, r_i), (l'_i, r'_i)\} \cup \{(c_L, c_R)\} \\ \cup & \bigcup_{S \in \{L_0, L_1, R_0, R_1\}} \bigcup_{u \neq v \in S} \{(u, v)\} \end{aligned}$$

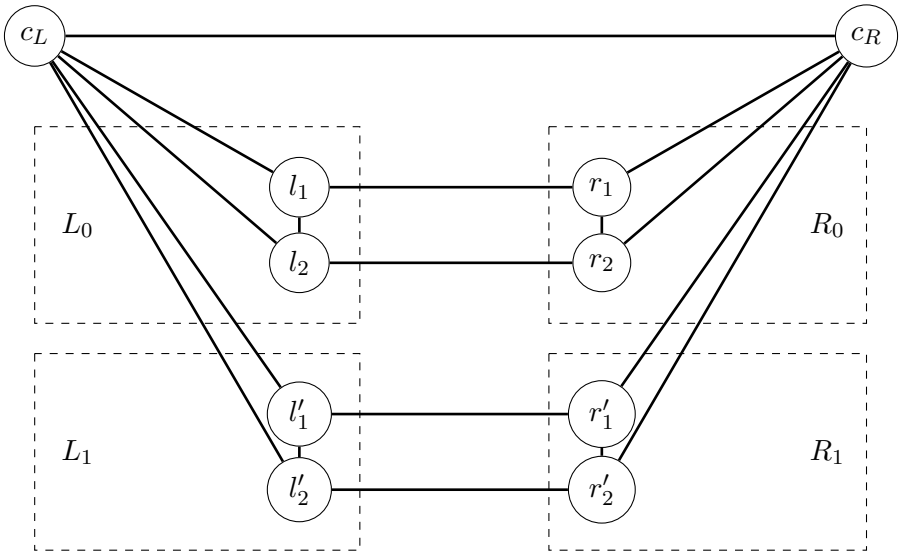

Figure 1.1: Graph $G' \in \mathcal{G}$ with $n = 10$

Family $\mathcal{G}$ contains any graph G that is derived from G' by adding any combination of edges of the for, (l_i, l'_j) or (r_i, r'_j).

Definition 10 Graph Sketch: A compressed representation of the graph.

Definition 11 Labeling Scheme: A labeling scheme consists of an encoder e and a decoder d. The encoder e assigns to each node v a label $e(v)$. The decoder d receives the labels of the nodes in question and returns an answer to some query. The largest size (in bits) of a label assigned to a node is called the label size of the labeling scheme.

Theorem 4 Upper Bound for Adjacency in Trees: It is possible to assign labels of size $2 \log n$ bits to nodes in a tree, so that for every pair u, v of nodes it is easy to tell whether they are adjacent or not, just by looking at their labels.

Theorem 5 Lower Bound for Adjacency in General Graphs: Any labeling scheme for adjacency in general graphs has a label size of at least $\Omega(n)$ bits.

Definition 12 Maximal Independent Set (MIS): Given a graph $G = (V, E)$, a set of vertices $S \subseteq V$ is called a MIS, if it satisfies the following properties:

- The set S is an independent set meaning that no two vertices $u, v \in S$ are adjacent.
- The set S is maximal meaning that for each node $v \notin S$ there exists a neighbor u of v such that $u \in S$.

Definition 13 Maximum Matching: A matching is a set of edges $M \subseteq E$ such that no two of the edges on M share an end-point. A matching is maximal if we cannot add any edge to M without violating the property that we have a matching.

Theorem 6: In any graph, any maximal matching has size at least $\frac{1}{2}$ of the maximum matching.

Definition 14 Minimal Spanning Tree (MST): Given a weighted graph $G = (V, E, \omega)$, the MST of G is a spanning tree T minimizing $\omega(T)$, where $\omega(G') = \sum_{e \in G'} \omega_e$ for any subgraph $G' \subseteq G$

Definition 15 Outgoing Edge: Let T be a spanning tree of the weighted graph G and $T' \subseteq T$ a subgraph of T. Edge $e = (u, v)$ is an outgoing edge of T' if $u \in T'$ and $v \notin T'$ or vice versa.

Definition 16 Radius: The radius of a node u is the maximum distance between u and any other node in the graph. The radius of a graph is the minimum radius of any node in the graph.

Definition 17 Shortest Path Cover: The node set S_i is a shortest path cover if S_i contains a node on every shortest path of length between 2^{i-1} and 2^i.

1.2 Algorithms and Complexity

Definition 18 Synchronous Distributed Algorithm: In a synchronous distributed algorithm, nodes operate in synchronous rounds. In each round, each node executes the following steps:

1. Send messages to neighbors in graph (of reasonable size).
2. Receive messages (that were sent by neighbors in step 1 of the same round)
3. Do some local computation (of reasonable complexity)

Definition 19 Asynchronous Distributed Algorithm: In the asynchronous model, algorithms are event driven. Nodes cannot access a global clock. A message sent from one node to another will arrive in finite but unbounded time.

Definition 20 Synchronous Time Complexity: For synchronous algorithms the time complexity is the number of rounds until the algorithm terminates.

Definition 21 Asynchronous Time Complexity: For asynchronous algorithms the time complexity is the number of time units from the start of the execution to its compeltion in the worst case, assuming that each message has a delay of at most one time unit.

Definition 22 Message Complexity: The message complexity of an algorithm is determined by the total number of messages exchanges.

1.3 Vertex Coloring

Definition 23 Vertex Coloring: Given an undirected graph $G = (V, E)$, assign a color c_v to each vertex $v \in V$ such that the following holds: $e = (v, w) \in E \Rightarrow c_v \neq c_w$.

Definition 24 $\log^*$**:**

$$\forall x \leq 2 : \log^* x := 1 \qquad \forall x > 2 : \log^* c := 1 + \log *(\log x)$$

This is a very slow growing function. $\log^*(10^{80}) = 5$

Definition 25 Cover-free Family: Given a ground set $\{1, 2, \ldots, k'\}$, a family of sets $S_1, S_2, \ldots, S_k \subseteq \{1, 2, \ldots k'\}$ is called a Δ-cover free family if and only if for each set of indices $i_0, i_1, \ldots, i_\Delta \in \{1, 2, \ldots, k\}$, we have

$$S_{i_0} \setminus \left(\bigcup_{j=1}^{\Delta} S_{i_j} \right) \neq \emptyset$$

That is, if no set in the family is a subset of the union of Δ other sets.

Definition 26 Sperner Family: A Sperner family is simply a 1-cover free family.

Theorem 7: For any k and Δ, there exists a Δ-cover free family of size k, $S_1, S_2 \ldots, S_k \subseteq \{1, 2, \ldots, k'\}$, on a ground set of size $k' = \mathcal{O}(\Delta^2 \log k)$.

Definition 27 k**-ary** q**-coloring:** We say B is a k-ary q-coloring if for any set of ientifiers $1 \leq a_1 < \ldots < a_{k+1} \leq n$, we have the foloowing properties:

- $B(a_1, \ldots, a_k) \in \{1, \ldots, q\}$
- $B(a_1, \ldots, a_k) \neq B(a_2, \ldots, a_{k+1})$

Theorem 8 Lower Bound on Coloring Rooted Trees: Any deterministic algorithm for 3-coloring n-node directed paths needs at least $\frac{\log^* n}{2} - 2$ rounds.

Theorem 9 Lower Bound on Coloring Unrooted Trees: Any deterministic distributed algorithm A that colors n-node trees with maximum degree Δ using less than $o(\frac{\Delta}{\log \Delta})$ colors has round complexity at least $\Omega(\log_\Delta n)$.

Theorem 10 Upper Bound on Coloring Unrooted Trees: There is a deterministic distributed algorithm that computes a 3-coloring of any n-node tree in $\mathcal{O}(\log n)$ rounds.

1.4 Distributed Sorting

Definition 28 Sorting: We choose a graph with n nodes $v_1, \ldots, v_n$. Initially each node stores a value. After applying a sorting algorithm, node v_k stores the k^{th} smallest value.

Definition 29 0-1 Sorting Lemma: If an oblivious comparison.exchange algorithm sorts all inputs of 0's and 1's, then it sorts arbitrary inputs.

Definition 30 Node Contention: In each step of a synchronous algorithm, each node can only send and receive $\mathcal{O}(1)$ messages containing $\mathcal{O}(1)$ values, no matter how many neighbors the node has.

Definition 31 Comparator: A comparator is a device with two inputs x, y and two outputs x', y' such that $x' = \min(x, y)$ and $y' = \max(x, y)$.

Definition 32 Comparison Network: A comparison network consists of wires that connect comparators. Some wires are not connected to comparator outputs (input wires) and some are not connected to comparator outputs (output wires).

Definition 33 Sorting Network: A sorting network with width n has n input wires and n output wires. A sorting network routes n values given on the input wires through the wires and comparators of the network such that the values are sorted on the output wires.

Definition 34 Depth: The depth of an input wire is 0. The depth of a comparator is the maximum depth of its input wires plus one. The depth of an output wire of a comparator is the depth of the comparator. The depth of a comparison network is the maximum depth of an output wire.

Definition 35 Bitonic Sequence: A bitonic sequence is a sequence of numbers that first monotonically increases and then monotonically decreases or vice versa.

Definition 36 Distributed Counting: A distributed counter is a variable that is common to all processors in a system and that supports an atomic test-and-increment operation. The operation delivers the system's counter value to athe requesting processor and increments it.

1.5 Network Decomposition

Definition 37 Weak Diameter Network Decomposition: Given a graph $G = (V, E)$, a $(\mathcal{C}, \mathcal{D})$ weak diameter network decomposition of G is a partition of G into vertex-disjoint graphs $G_1, \ldots, G_{\mathcal{C}}$, such that for each $i \in \{1, \ldots, \mathcal{C}\}$, we have the following property: the graph G_i is made of a number of vertex-disjoint and mutually non-adjacent clusters $X_1, \ldots, X_l$, where each two vertices $v, u \in X_j$ have distance at most $\mathcal{D}$ in graph G. We note that we do not bound the number l. We refer to each subgraph G_i as one block of this network decomposition.

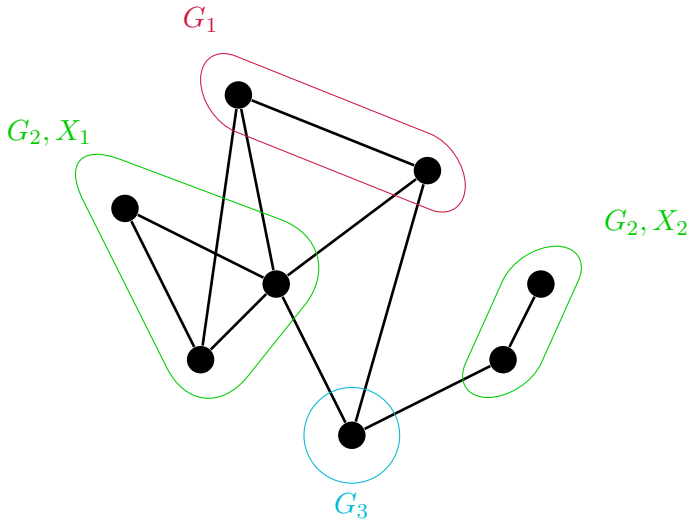

Figure 1.2: $(3, 1)$ Weak Diameter Network Decomposition

Definition 38 Strong Network Decomposition: Given a graph $G = (V, E)$, a $(\mathcal{C}, \mathcal{D})$ strong diameter network decomposition of G is a partition of G into vertex-disjoint graphs $G_1, \ldots, G_{\mathcal{C}}$ such that for each $i \in \{1, \ldots, \mathcal{C}\}$, we have the following property: each connected component of G_i has diameter at most $\mathcal{D}$.

1.6 Wireless Protocols

Definition 39 Initialization: At the end of the initialization, the n nodes should have the IDs $\{1, \ldots, n\}$.

Definition 40 Non-Uniform Network: The nodes know things about the network, e.g. how many nodes there are in total.

Definition 41 Uniform Network: The nodes know nothing about the network.

Definition 42 Collision Detection (CD): Two or more nodes transmitting concurrently is called interference. In a system with collision detection, a receiver can distinguish interference from nobody transmitting. In a system without collision detection, a receiver cannot distinguish the two cases.

Theorem 11 Lower Bound on Leader Election: Any uniform protocol that elects a leader with probability of at least $1 - \frac{1}{2^t}$ must run for at least t time slots.

Theorem 12 Uniform Asynchronous Wakeup with CD: If nodes wake up in an arbitrary (worst-case) way, any algorithm may take $\Omega(\frac{n}{\log n})$ time slots until a single node can successfully transmit.

Chapter 2

Math Stuff

Theorem 13:

$$\alpha > 1: \quad 1 + \frac{\log(\alpha - 1)}{2} \leq \log \alpha$$

Theorem 14 Chernoff Bound: Suppose $X_1, \ldots, X_\eta$ are independent random variables taking values in $[0, 1]$. Let $X = \sum_{i=1}^{l} X_i$ denote their sum and let $\mu = \mathbb{E}[X]$ denote the sum's expected value. For any $0 \leq \delta \leq 1$ it holds

$$Pr[X < (1 - \delta)E[X]] \leq e^{-\frac{\delta^2}{2}E[X]}$$

and for $\delta > 0$

$$Pr[X \geq (1 + \delta)E[X]] \leq e^{-\frac{\min\{\delta, \delta^2\}}{3}E[X]}$$

Definition 43 With High Probability: Some probabilistic event is said to occur with high probability if it happens with a probability $p \geq 1 - \frac{1}{n^c}$, where c is a constant.

Theorem 15 Booles Inequality: For a countable set of events $E_1, E_2, E_3, \ldots$ we have

$$Pr\left[\bigcup_i E_i\right] \leq \sum_i Pr[E_i]$$

Theorem 16 Markovs Inequality: If X is any random variable and $a > 0$ then

$$Pr\left[|X| \geq a\right] \leq \frac{E[X]}{a}$$

Theorem 17: For all $n \in \mathbb{N}$ and $|t| \leq n$ we have

$$e^t \left(1 - \frac{t^2}{n}\right) \leq \left(1 + \frac{t}{n}\right)^n \leq e^t$$

Note that:

$$\lim_{n \to \infty} \left(1 + \frac{t}{n}\right)^n = e^t$$

Theorem 18: For all p, q such that $0 < p < 1$ and $k \geq 1$ we have

$$1 - p \leq \left(1 - \frac{p}{k}\right)^k$$

Chapter 3

Algorithms

3.1 Vertex Coloring

Goal: Color the nodes of a graph with as few different colors as possible.

Algorithm 1 Greedy Sequential

```
1: while there is an uncolored vertex v do
2:     Color v with the minimal color that does not conflict with already colored neighbors
3: end while
```

Theorem 19 Algorithm 1: Terminates in n steps. Uses at most $\Delta + 1$ colors.

Algorithm 2 Reduce

```
1: Assume that initially all nodes have IDs
2: for each node v do
3:     Send ID to all neighbors
4:     Receive IDs of all neighbors
5:     while node v has an uncolored neighbor with higher ID do
6:         Send "undecided" to all neighbors
7:         Receive decisions from neighbors
8:     end while
9:     Choose the smallest admissible free color
10:    Send color choice to all neighbors
11: end for
```

Theorem 20 Algorithm 2: Time complexity n. Uses at most $\Delta + 1$ colors.

Algorithm 3 Slow Tree Coloring

Color the root with 0, the root sends 0 to its children
for each node v **do**
  if node v receives a message c_p from parent **then**
    Choose color $c_v = (1 - c_p) \mod 2$
    Send c_v to children
  end if
end for

Theorem 21 Algorithm 3: Time complexity is the height of the tree.

Algorithm 4 6-color

Assume that initially the nodes have IDs (labels) of size $\log n$ bits
The root assigns to itself the label 0
for each other node v **do**
  Send own color c_v to all children
  repeat
    Receive color c_p from parent
    Interpret c_v and c_p as bit-strings
    Let i be the index of the smallest bit where c_v and c_p differ
    The new label is i (as bit-string) followd by the i^{th} bit of c_v
    Send c_v to all children
  until $c_w \in \{0, \ldots, 5\}$ for all nodes w
end for

3.1.0.1 *Examples:*

grandparent:	*0010110000*	$\rightarrow$	*10010*	$\rightarrow$	
parent:	*1010010000*	$\xrightarrow{5,0}$	*01010*	$\xrightarrow{3,1}$	*111*
child:	*0110010000*	$\xrightarrow{8,1}$	*10001*	$\xrightarrow{0,1}$	*001*

grandparent:	*1100101101*	$\rightarrow$	
parent:	*1011101101*	$\xrightarrow{6,1}$	*1101*
child:	*0010101101*	$\xrightarrow{6,0}$	*1100*

Theorem 22 Algorithm 4: Terminates in $\log^*(n + c)$ time.

Algorithm 5 3-color

Assume that initially the nodes have IDs (labels) of size $\log n$ bits
The root assigns to itself the label 0
for each other node v **do**
    Send own color c_v to all children
    repeat
        Receive color c_p from parent
        Interpret c_v and c_p as bit-strings
        Let i be the index of the smallest bit where c_v and c_p differ
        The new label is i (as bit-string) followd by the i^{th} bit of c_v
        Send c_v to all children
    until $c_w \in \{0, \ldots, 5\}$ for all nodes w
end for
for each node v **do**
    for $x = 5, 4, 3$ **do**
        for each node v **do**
            Recolor v with the color of the parent
            Root choses new, different color from $\{0, 1, 2\}$
        end for
        if $c_v = x$ **then**
            Choose smallest admissible new color $c_v \in \{0, 1, 2\}$
        end if
    end for
end for

Theorem 23 Algorithm 5: Terminates in time $\mathcal{O}(\log^* n)$

3.1.0.2 *A fast tree-coloring with only 2 colors is more than exponentially more expensive than coloring with 3 colors.*

3.1.0.3 *A general graph with constant degree Δ can be colored with $\Delta + 1$ colors in $\mathcal{O}(\log^* n)$ time.*

Algorithm 6 Linial

Given a n-coloring of the graph.
while there are more than $\mathcal{O}(\Delta^2 \log \Delta)$ colors **do**
    Given a k-coloring ϕ_{old} of a graph with maximum degree Δ.
    for each node v of old color $\phi_{old}(v) = q, q \in \{1, \ldots, k\}$ **do**
        Use set $S_q \subseteq \{1, \ldots, k'\}$ in the cover free family as its color-set
        Set new color $\phi_{new}(v) = q', q' \in S_q$ such that q' is not in the color-set of any of the neighbors
    end for
end while

Theorem 24 Algorithm 6: Needs $\mathcal{O}(\log^* n)$ rounds to compute a $\mathcal{O}(\Delta^2 \log \Delta)$-coloring.

Algorithm 7 Color Reduction

for each node v **do**
    $c_v = v$
end for
for $v = \Delta + 2$ **to** n **do**
    $c_v = \min(\{1, \ldots, \Delta + 1\} \backslash \{c_u | (u, v) \in E\})$
end for

Algorithm 8 Kuhn-Wattenhofer

for each node v **do** in parallel
    $c_v = v$
end for
while $k > \Delta + 1$ **do**
    Divide colors into bins of size $2(\Delta + 1)$
    Let each bin be denoted as $G_i = (V_i, E_i)$
    for i **do** in parallel
        Color Reduction(G_i)
        $k = k - \Delta + 1$
    end for
end while

Theorem 25 Algorithm 8: Needs $\mathcal{O}(\Delta \lceil \log(\frac{k}{\Delta+1}) \rceil)$ rounds to compute a $(\Delta + 1)$-coloring.

Theorem 26: By first performing Algorithm 6 and then Algorithm 8 we can achieve a $(\Delta + 1)$-coloring in $\mathcal{O}(\Delta \log \Delta + \log^* n)$ rounds.

Algorithm 9 Luby MIS

while set is not maximal **do**
    for each node v **do**
        v picks a random number $r_v \in [0, 1]$ and sends it to its neighbors.
    end for
    for each node v **do**
        if $r_v > r_u$ for all neighbors u of v **then**
            v joins MIS set S
            v informs its neighbors
            v and all its neighbors are removed from the graph
        end if
    end for
end while

Theorem 27 Algorithm 9: Computes a MIS in $\mathcal{O}(\log n)$ rounds with high probability.

Theorem 28: Given a distributed algorithm $\mathcal{A}$ that computes a MIS of any n-node graph in $T(n)$ rounds, there is a distributed algorithm $\mathcal{B}$ that computes a $(\Delta+1)$-coloring of any n-node graph with maximum degree Δ in $T(n(\Delta+1))$ rounds. Short outline: Compute MIS S_i, color with color i, remove S_i from graph, repeat.

Algorithm 10 Coloring Unrooted Trees

Step 1, takes $\mathcal{O}(\log n)$ iterations

$T_1 = T$
$L_1 = \{v \in T_1 | degree(v) \leq 2\}$
while layer L_{i+1} still get nodes **do**
    $T_{i+1} = T_i \backslash L_i$
    $L_{i+1} = \{v \in T_{i+1} | degree(v) \leq 2\}$
    $i = i + 1$
end while
$T = T[\bigcup_{j=1}^{l} L_j]$

Step 2, takes $\mathcal{O}(\log^* n)$ rounds

for each $T[L_i$**do**$]$
    3-color $T[L_i]$ with Algorithm 5 to get schedule colors
end for

Step 3, takes $l \cdot 3 = \mathcal{O}(\log n)$ rounds

for $i = l$ **until** $i = 1$ **do**
    for $q \in \{1, 2, 3\}$ **do**
        Have final coloring of $T[\bigcup_{j=i+1}^{l} L_j]$
        Pick a final color in $\{1, 2, 3\}$ for all the vertices in L_i with schedule color q.
    end for
end for

3.2 Edge Coloring

Goal: Color the edges of a graph with as few different colors as possible.

Algorithm 11 Edge-Coloring

Part 1

Orient the graph G, so that each edge goes from lower ID to higher ID
for each node v **do**
    Number outgoing edges of v
end for
Define F_i as vertices and edges which are numbered i^{th} by their starting point.
F_i is an oriented pseudo-forest (each component has at most one circle)

Part 2

for each F_i **do** in parallel
    Compute a 3-vertex-coloring with Algorithm 5.
    These are the schedule-colors of F_i.
end for

Part 3

for $k \in \{1, 2, 3\}$ **do**
    Let E_k^i be the set of F_i-edges whose parent endpoint is colored with color k.
    These edges form vertex-disjoint stars.
    for each star centered at node v with nodes $u_1, \ldots, u_l$ **do** in parallel
        v learns colors of edges adjacent to $u_1, \ldots, u_l$
        v computes edge-colors for edges $(v, u_1), \ldots, (v, u_l)$. There will always be a color available from colors $\{1, 2\Delta - 1\}$.
    end for
end for

Theorem 29 Algorithm 11: Needs $\mathcal{O}(\Delta + \log^* n)$ rounds to compute a $(2\Delta - 1)$-edge-coloring.

3.3 Tree Construction Algorithms

Goal: Construct trees from graphs.

Algorithm 12 Flooding

The source (root) sends the message to all neighbors
for each node v upon receiving the message the first time **do**
    Forward the message to all other neighbors
end for
Upon later receiving the message again, a node can discard the message

Theorem 30 Algorithm 12: Time complexity $radius(root)$. Message complexity m where $m = |E|$ is the number of edges (if the nodes do not know the topology) or $n - 1$ (if the nodes know the topology).

Algorithm 13 Echo

1: **for each** leaf v **do**
2: Send a message to its parent
3: **end for**
4: **for each** not-leaf u upon receiving a message from a child **do**
5: Send message to its parent
6: **end for**

Theorem 31 Algorithm 13: Time complexity is determined by the depth of the spanning tree. Message complexity $n-1$. Together with flooding:

flooding/echo	synchronous	asynchronous
time complexity	$2 \cdot radius(root)$	n
message complexity	$4m + n \leq 5m$	$c \cdot m$

Algorithm 14 Dijkstra BFS

1: Phase $p = 1$, treee T which is the root plus all direct neighbors of the root
2: **repeat**
3: Root starts phase p by broadcasting "start p" within T
4: **for each** leaf node u upon receiving "start p" **do**
5: Send a "join $p+1$" message to all neighbors it has not communicated with yet.
6: Collect all answers of neighbors then start echo back to the root
7: **end for**
8: **for each** node v upon receiving "join $p+1$" **do**
9: **if** v not in T **then**
10: Reply with "ack" and become new leaf of tree T at level $p+1$
11: **else**
12: Reply with "nack"
13: **end if**
14: **end for**
15: When echo process terminates at root, root sets new phase $p = p+1$
16: **until** there was no new node detected

Theorem 32 Algorithm 14: Time complexity $\mathcal{O}(D^2)$. Message complexity $\mathcal{O}(m+n \cdot D)$ where D is the diameter of the graph.

Algorithm 15 Bellman-Ford BFS

1: u stores d_u = distance from u to the root. Initially $d_{root} = 0$ and $d_u = \infty$ for all other nodes u.
2: root starts by sending "1" to all neighbors
3: **if** node u receives message "y" with $y < d_u$ from neighbor v **then**
4: u sets $d_u := y$
5: u sends "$y+1$" to all neighbors except v
6: **end if**

Theorem 33 **Algorithm 15:** Time complexity $\mathcal{O}(D)$. Message complexity $\mathcal{O}(n \cdot m)$, where D is the diameter of the graph.

3.3.0.1 *Algorithm 14 has better message complexity and Algorithm 15 has better time complexity. The current best algorithm has time complexity $\mathcal{O}(D \cdot \log^3 n)$ and message complexity $\mathcal{O}(m + n \log^3 n)$.*

Algorithm 16 Gallager-Humblet-Spira (GHS)

1: Each node is root of its own fragment.
2: **repeat**
3: All nodes learn fragment IDs of their neighbors
4: Root of each fragment uses flooding/echo in its fragment to determine the blue edge $b = (u, v)$ of the fragment.
5: Root sends a message to node u. While forwarding the message from root to u, all parent-child relations are inverted.
6: u sends merge request over the blue edge $b = (u, v)$
7: **if** v also sent a merge request over the same blue edge $b = (u, v)$ **then**
8: u or v (with smaller ID) is new fragment root
9: b is directed accordingly
10: **else**
11: v is new parent of u
12: **end if**
13: newly elected root node u or v informs all nodes in its fragment about its identity using flooding/echo
14: **until** all nodes are in the same fragment

Theorem 34 **Algorithm 16:** Time complexity $\mathcal{O}(n \log n)$. Message complexity $\mathcal{O}(m \log n)$.

3.4 Shared Objects on Trees

Goal: Manage access to a common object in a tree.

Algorithm 17 Shared Object: Centralized Solution

Initialization: Shared objects stored at root node r of a spanning tree of the network graph. All nodes know their parent.

Accessing Object by node v
1: v sends request up the tree
2: Request atomically processed by root r
3: Result sent down the tree to node v

3.4.0.1 *Algorithm 17 suffers whem a single node accesses the shared object repeatedly.*

3.4.0.2 *Algorithm 18 suffers from the triangular routing problem: If two close-by nodes access the object in turns, all the traffic is routed through the potentially far away home-base.*

Algorithm 18 Shared Object: Home-Based Solution

Initialization: An object has a home base node that is known to every node. All requests are touted through the home base.

Accessing Object by node v

1: v acquires a lock at the home base, receives object

Algorithm 19 Shared Object: Arrow

Initialization: We are given a rooted spanning tree. Each node has a pointer to its parent, the root r is its own parent. The object is initially stored at r. For all nodes $v : v.successor := null, v.wait := false$.

Start Find Request at Node u

1: **atomically do**
2: u sends "find by u message to parent node
3: $u.parent := u$
4: $u.wait := true$
5: **end do**

Upon w receiving "Find by u" Message from Node v

6: **atomically do**
7: **if** $w.parent \neq w$ **then**
8: w sends "find by u" message to parent
9: $w.parent := v$
10: **else**
11: $w.parent := v$
12: **if** not $w.wait$ **then**
13: Send variable to u
14: **else**
15: $w.successor := u$
16: **end if**
17: **end if**
18: **end do**

Upon w Receiving Shared Object

19: Perform operation on shared object
20: **atomically do**
21: $w.wait := false$
22: **if** $w.successor \neq null$ **then**
23: Send variable to $w.successor$
24: $w.successor := null$
25: **end if**
26: **end do**

Theorem 35 Algorithm 19: For one "find" operation in a concurrent (meaning there can be many find requests at the same time) setting.

- Asynchronous: Time complexity D. Message complexity D where D is the diameter of the spanning tree.
- Synchronous setting: Message complexity $\mathcal{O}(\log |S| \cdot m^*)$ where S is the set of nodes initiating a "find" operation and m^* the message complexity of an optimal algorithm on the tree.

3.5 Shared Objects on Cliques

Goal: Manage access to a common object in a clique.

Algorithm 20 Shared Object: Pointer Forwarding

Initialization: Object is stored at root r of a precomputed spanning tree T.

Accessing object by node v

1: Follow parent pointers to current root r of T
2: Send object from r to u
3: $r.parent := u, \quad u.parent := u$

Theorem 36 Algorithm 20: In the worst case (always first node of linked list that acquires object): Time complexity n. Message complexity n. If not FIFO, can even be unbounded.

Algorithm 21 Shared Object: Ivy

Initialization: Object is stored at root r of a precomputed spanning tree T.

Start Find Request at Node u

1: u send "find by u" message to parent node
2: $u.parent := u$

Upon v receiving "Find by u" Message

3: **if** $v.parent = v$ **then**
4: Send object to u
5: **else**
6: Send "find by u" message to $v.parent$
7: **end if**$v.parent := u$

Theorem 37 Algorithm 21: For one "find" operation and if initial tree is a star, time complexity is $\log n$, where n is the number of processors.

3.6 Distributed Sorting

Goal: Have the k^{th} node store the k^{th}-smallest value.

Algorithm 22 Odd/Even Sort

1: Given an array of n nodes $(v_1, \ldots, v_n)$, each storing a value
2: **repeat**
3: Compare and exchange the values at nodes i and $i+1$, i odd
4: Compare and exchange the values at nodes i and $i+1$, i even
5: **until** done

Theorem 38 Algorithm 22: Sorts correctly in n steps.

Algorithm 23 Shearsort

1: We are given a mesh with m rows and m columns, m even, $n = m^2$.
2: **repeat** alternating odd/even phases
3: **if** is odd phase **then**
4: **for each** row **do**
5: **if** row is odd **then**
6: Sort row such that small values move to the left
7: **else**
8: Sort row such that small values move to the right
9: **end if**
10: **end for**
11: **else**
12: Sort columns such that small values move up
13: **end if**
14: **until** done

Theorem 39 Algorithm 23: Sorts n values in $2 \cdot m \cdot (\log n + 1) = \sqrt{n}(\log n + 1)$ time in snake-like order.

Algorithm 24 Half Cleaner

1: Comparison network of depth 1
2: Compare wire i with wire $i + \frac{n}{2}$ for $i = 1, \ldots, \frac{n}{2}$.

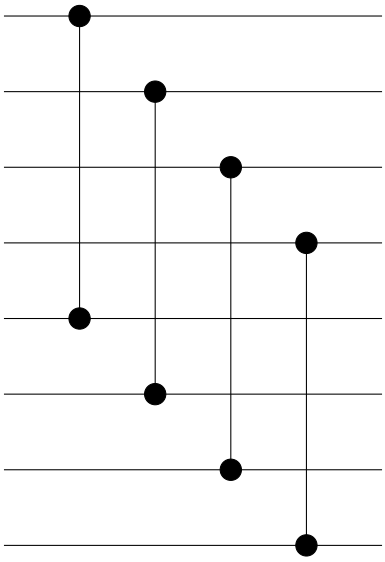

Figure 3.1: Half-Cleaner (HC)

Theorem 40 **Algorithm 24:** Fed a bitonic sequence, it cleans either the upper or the lower half of the n wires. The other half is bitonic.

Algorithm 25 Merger

1: A merger is a comparison network of depth 1.
2: Compare wire i with wire $n - i + 1$ for $i = 1, \ldots, \frac{n}{2}$

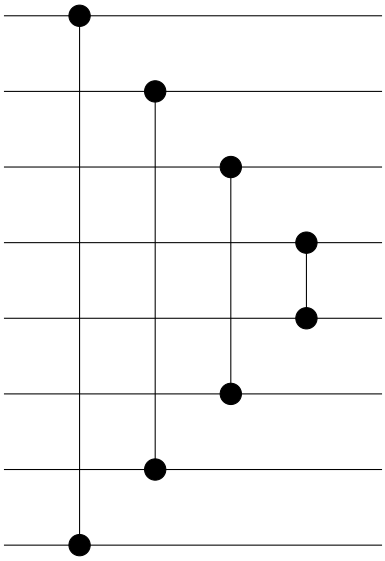

Figure 3.2: Merger

Theorem 41 **Algorithm 25:** Fed two sorted sequences of width $\frac{n}{2}$, it gives two bitonic sequences of width $\frac{n}{2}$.

Algorithm 26 Bitonic Sequence Sorter

1: Consists of a half-cleaner of width n and then two bitonic sequence sorters of width $\frac{n}{2}$ each.
2: A bitonic sequence sorter of width 1 is empty.

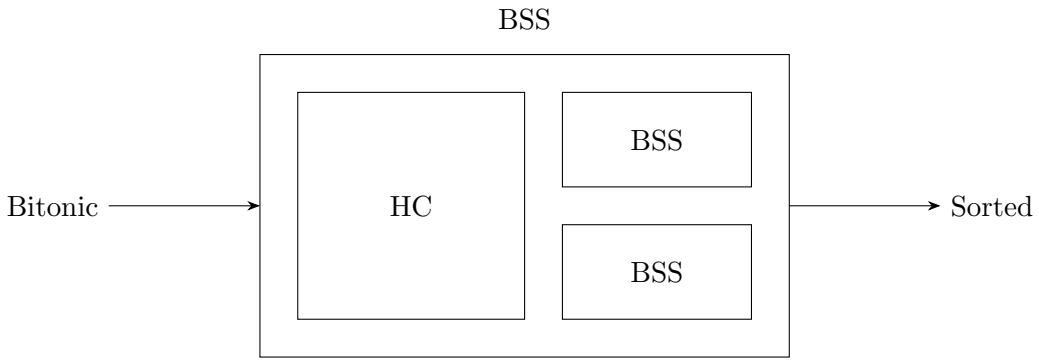

Figure 3.3: Bitonic Sequence Sorter (BSS)

Theorem 42 **Algorithm 26:** Sorts bitonic sequences in depth $\log n$.

Algorithm 27 Merging Network

1: Consists of a merger of width n followed by two bitonic sequence sorters fo width $\frac{n}{2}$ each.

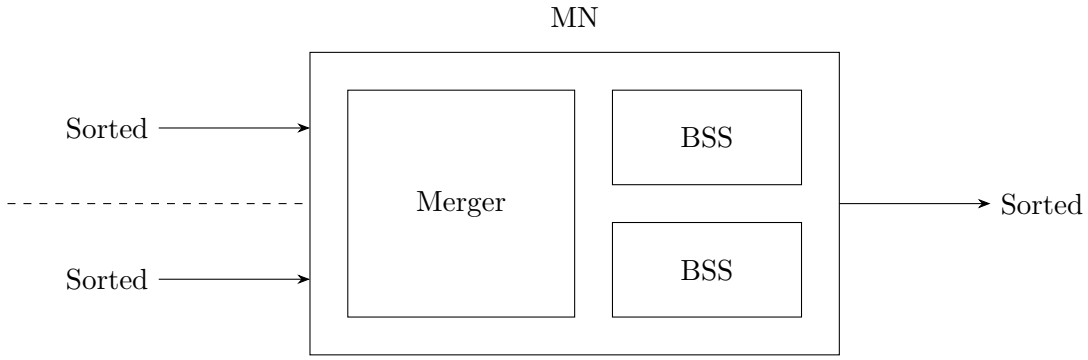

Figure 3.4: Merging Network (MN)

Theorem 43 **Algorithm 27:** Merges two sorted input sequences of length $\frac{n}{2}$ into one sorted sequence of length n.

Algorithm 28 Batcher's Sorting Network

1: Consists of two batcher sorting networks of width $\frac{n}{2}$ each followed by a merging network of width n.
2: A batcher sorting network of width 1 is empty.

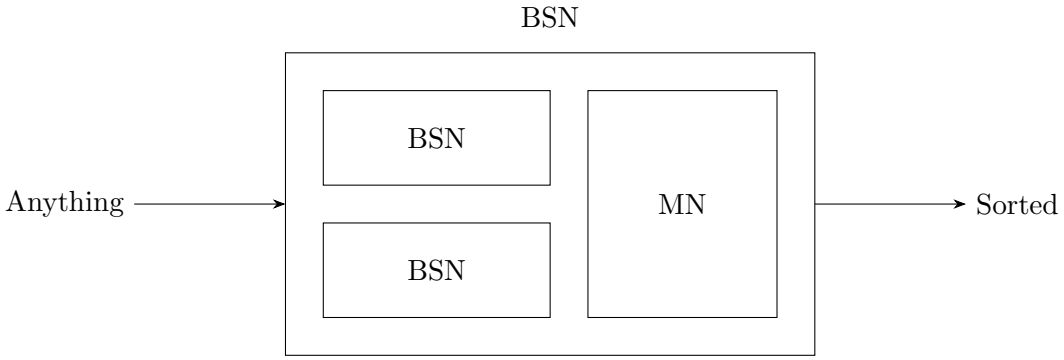

Figure 3.5: Batcher Sorting Network (BSN)

Theorem 44 **Algorithm 28:** Sorts an arbitrary sequence of length n in depth $\mathcal{O}(\log^2 n)$.

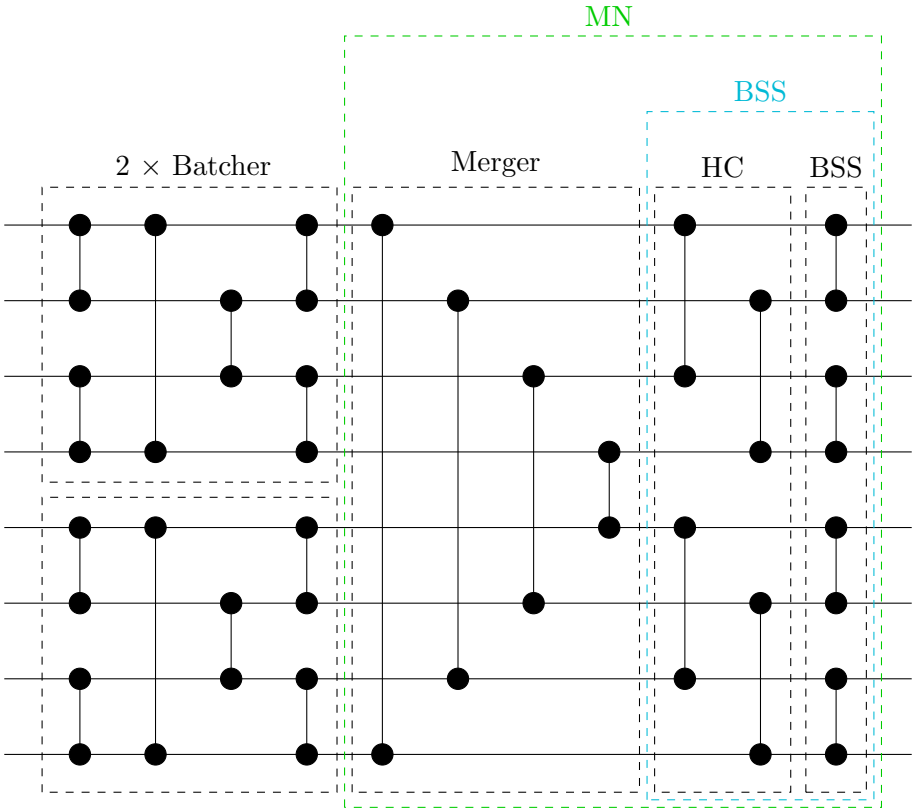

Figure 3.6: Batcher for width 8

3.7 Centralized Maximum Matching

Goal: Use a few local computations to approximate the size of the maximum matching.

Theorem 45 **Algorithm 30:** The expected query complexity of the algorithm for an arbitrary edge e is at most $2^{\mathcal{O}(\Delta)}$.

Algorithm 29 Random Greedy Maximal Matching Algorithm

1: **for each** edge e **do**
2: Pick random number $r_e \in [0, 1]$
3: **end for**
4: **for** edge e with lowest r_e **until** edge e with highest r_e **do**
5: Add e to the matching M if no neighbor e' of e with lower $r_{e'}$ is already in M.
6: **end for**

Algorithm 30 Centralized Maximal Matching Algorithm

1: We want to find out if e is in the matching M or not.
2: Determine random value r_e and all r'_e for all neighbors e' of e.
3: **for each** e' with $r'_e < r_e$ **do**
4: Recursively find out if they are in the matching M
5: **end for**
6: **if** none of the edges e' with $r'_e < r_e$ is in the matching M **then**
7: e is in the matching M
8: **end if**

Theorem 46 Approximating the Size of the Maximal Matching: Pick a set S of k random chosen nodes. The fraction of theses nodes that are matched in M is an unbiased estimator of the fraction of vertices that are matched in M. Thus:

$$|M| \approx \frac{n}{2\,|S|} \sum_{s \in S} 1_{(\text{vertex } s \text{ matched in } M)}$$

For any certainty parameter $\delta \in [0, 0.25]$ and any precision parameter $\epsilon > 0$, suppose we choose a set S of $k = \frac{20 \cdot \Delta \cdot \log 1/\delta}{\epsilon^2}$ at random. Then this function provides a $(1 + \epsilon)$ approximation of the size of the maximal matching with probability at least $1 - \delta$.
The overall expected query complexity for checking a set S of nodes to see whether they are matched in M or not is at most $|S| \cdot \Delta \cdot 2^{\mathcal{O}(\Delta)} = |S| \cdot 2^{\mathcal{O}(\Delta)}$

3.8 Network Decomposition

Goal: Compute a network decomposition with which we can solve a wide range of problems.

Theorem 47 Algorithm 31: Computes a $(\mathcal{C}, \mathcal{D})$ weak diameter network decomposition of any n-node graph G, for $\mathcal{C} = \mathcal{O}(\log n)$ and $\mathcal{D} = \mathcal{O}(\log n)$, in $\mathcal{O}(\log^2 n)$ rounds with high probability.

Theorem 48: Provided a $(\mathcal{C}, \mathcal{D})$ weak diameter network decomposition of a graph G, we can compute a $\Delta + 1$ coloring of G in $\mathcal{O}(\mathcal{C}\mathcal{D})$ rounds.

Algorithm 31 Weak Network Decomposition

for $i = 1$ **until** $\mathcal{C}$ **do**
  for each node u **do**
    Pick random radius r_u with $Pr[r_u = y] = \epsilon(1-\epsilon)^{y-1}$ for $\epsilon \in (0,1)$
    The ball of node u are the vertices within distance r_u of u.
  end for
  for each node v **do**
    Let $Center(v) = u'$ be the smallest-identifier node whose ball contains v.
  end for
  Define G_i by letting all nodes with the same center define one cluster
  Discard nodes who are at the boundary fo their cluster
end for

3.9 Wireless Protocols

Goal: Do initialization and leader election in a wireless network where there is interference if two or more nodes transmit at the same time.

Algorithm 32 Slotted ALOHA

for each node v **do**
  repeat
    Transmit with probability $\frac{1}{n}$
  until One node has transmitted alone
end for
This node is now the leader.

Theorem 49 Algorithm 32: Allows a node to transmit alone and thus become the leader after expected time e.

Algorithm 33 Non-Uniform Initialization

repeat
  Elect a leader v using Algorithm 32.
  v gets the next free number and leaves the process.
until No nodes are left

Theorem 50 Algorithm 33: Initializes n nodes in $\mathcal{O}(e \cdot n)$ time slots.

Algorithm 34 Uniform Initialization with CD

```
1: nextID = 0
2: for each node v do
3:     myBitstrings = ""
4:     bitstringsToSplit = [""]
5:     while bitstringsToSplit is not empty do
6:         b = bitstringsToSplit.pop()
7:         repeat
8:             if b = myBitstring then
9:                 Choose r uniformly at random from {0, 1}
10:                For the next two timeslots, transmit in slot r, listen in the other
11:            else
12:                For the next two timeslots, listen on both
13:            end if
14:        until There was at least 1 transmission on both slots
15:        if b = myBitstring then
16:            myBitstring = myBitstring + r
17:        end if
18:        for r ∈ {0, 1} do
19:            if some node u transmitted alone in slot r then
20:                Node u gets ID nextId and becomes passive
21:                nextId = nextId + 1
22:            else
23:                bitstringsToSplit.push(b + r)
24:            end if
25:        end for
26:    end while
27: end for
```

Theorem 51 **Algorithm 34:** Initializes n nodes in $\mathcal{O}(n)$ time slots.

Algorithm 35 Uniform Initialization without CD

1: Let node l be the leader and S the set of nodes which want to transmit.
2: Split every time slot from Algorithm 34 into two time slots.
3: *First timeslot:* nodes in set S transmits
4: *Second timeslot:* nodes in set $S \cup \{l\}$ transmit
5: This gives the nodes sufficient information to distinguish the different cases. See Table 3.1 for the details.
6: Thus Algorithm 34 works also without CD

	nodes in S transmit	nodes in $S \cup \{l\}$ transmit
$\lvert S\rvert = 0$	$\times$	✓
$\lvert S\rvert = 1, S = \{l\}$	✓	✓
$\lvert S\rvert = 1, S \neq \{l\}$	✓	$\times$
$\lvert S\rvert \geq 2$	$\times$	$\times$

Table 3.1: Distinguishing between noise and silence: ✓stands for a successful transmission, $\times$ for noise/silence

Algorithm 36 Uniform Leader Election

```
1: for each node v do
2:     for k = 1, 2, 3, ... do
3:         for i = 1 until c · k do
4:             Transmit with probability p = 1/2^k
5:             if N thenode v was the only node which transmitted
6:                 v becomes the leader
7:                 break
8:             end if
9:         end for
10:    end for
11: end for
```

Theorem 52 Algorithm 36: Elects a leader with high probability in $\mathcal{O}(\log^2 n)$ time slots if n is not known.

Algorithm 37 Uniform Leader Election with CD

```
1: for each node v do
2:     repeat
3:         Transmit with probability 1/2
4:         if A thent least one node transmitted
5:             All nodes that did not transmit quit the protocol
6:         end if
7:     until One node transmits alone
8: end for
```

Theorem 53 Algorithm 37: Elects a leader with high probability in $\mathcal{O}(\log n)$ time slots if we have collision detection.

Algorithm 38 Fast Uniform Leader Election with CD

Phase 1
$i = 1$
repeat
  $i = 2i$
  Transmit with probability $\frac{1}{2^i}$
until No node transmitted

Phase 2
$l = \frac{i}{2}$
$u = i$
while $l + 1 < u$ **do**
  $j = \lceil \frac{l+u}{2} \rceil$
  Transmit with probability $\frac{1}{2^j}$
  if No node transmitted **then**
    $u = j$
  else
    $l = j$
  end if
end while

Phase 3
$k = u$
repeat
  Transmit with probability $\frac{1}{2^k}$
  if No node transmitted **then**
    $k = k - 1$
  else
    $k = k + 1$
  end if
until Exactly one node transmitted

Theorem 54 Algorithm 38: With probability at least $1 - \frac{\log\log n}{\log n}$ we find a leader in time $\mathcal{O}(\log\log n)$.

3.10 Computing the Diameter

Goal: Compute the diameter Δ of the network, so that we can use flooding/echo to solve everything in time $\mathcal{O}(\Delta)$.

Algorithm 39 Compute All Pairs Shortes Path (APSP)

Assume we hace leader node l
Compute BFS_l of leader l
Send a pebble P to traverse BFS_l in a depth-first-search way
while P traverses BFS_l **do**
    if P visits a new node v **then**
        Immediately start BFS_v from node v
        Pebble P waits one time slot
    end if
end while

Theorem 55 Algorithm 39: Computes APSP in time $\mathcal{O}(n)$.

3.11 Minimal Spanning Tree

Goal: Compute minimum spanning tree (MST) in a model, where the maximum number of bits that a computer can send is $\mathcal{O}(\log n)$.

Algorithm 40 Finding the Minimum Weight Outgoing Edge (MWOE)

for each Component S_i **do**
    if $|S_i| \leq \sqrt{n}$ **then**
        for each node v **do**
            Compute smallest weight outgoing edge with weight $c(v)$
        end for
        Perform Convergecast on the BFS tree of S_i
        Leader s_i now knows the overall MWOE
        s_i broadcasts the MWOE and the random bit $t(S_i)$ to all nodes of S_i
    else
        There are at most $\sqrt{n}$ of these components
        We can handle these components by performing their communications on the BFS of the whole graph G simultaneously.
    end if
end for

Algorithm 41 Boruvska's MST

Start with each node being a separate component of the forest.
repeat
  for each component S_i with leader node s_i **do**
    Compute random $t(S_i) \in \{0, 1\}$
    Find the MWOE of the component with Algorithm 40
    if $t(S_i) == 1$ **then**
      Suggest to merge with the component on the other end of the MWOE
    else
      Accept incoming suggested merge-edges from component S_j
      s_i becomes leader of the new merged part
    end if
    if $t(S_i) == 1$ **then**
      Learn ID of new leader
      The endpoint e_i that was merged on knows the ID
      if $|S_i| \leq \sqrt{n}$ **then**
        Do it directly inside the component in $\mathcal{O}(\sqrt{n})$ rounds
      else
        Broadcast it to all nodes of the graph in $\mathcal{O}(D + \sqrt{n})$ rounds
      end if
    end if
    Learn size of new component
    if $t(S_i) == 0$ **then**
      The endpoints that were merged on know the size of both components
      if $|S_i| \leq \sqrt{n}$ **then**
        Compute new component size by performing converge-case
      else
        Compute new component size by doing it through the global BFS tree
      end if
      Deliver information to all components of S_i
      Each merge-endpoint deliver the information through its own component
    end if
  end for
until The number of components is 1.

3.12 Graph Connectivity on Graph Sketch

Goal: Compute the number of connected components of a graph by having each node send a local computation to a coordinator.

Algorithm 42 Graph Connectivity

Have coordinator and n nodes
for each node v **do**
    Send message of size $\mathcal{O}(\log^4 n)$, containing $\mathcal{O}(\log^2 n)$ many sketches of size $\mathcal{O}(\log^2 n)$ bits each.
    Each sketch has $\mathcal{O}(\log n)$ parts, where for the i^{th} part an $\mathcal{O}(\log n)$ bit string is generated.
    A random subset of the edges incident to the node is sampled, choosing each edge with probability 2^{-i}, then the XOR of the random edge IDs over all sampled edges is stored.
end for
repeat
    The coordinator identifies an outgoing edge for every component.
    The coordinator runs Boruvska's Algorithm 41 to merge the components
until All components are merged.

3.13 Labeling Schemes

Goal: Store information in the labels of the nodes so we can easily compute a function of two nodes by only looking at the labels.

Algorithm 43 Naïve-Distance-Labeling(T)

Let l be the label of the root r in T
Let $T_1, \ldots, T_\delta$ be the sub-trees rooted at each of the δ children of r
for $i = 1, \ldots, \delta$ **do**
    The root of T_i gets the label obtained by appending i to l
    Algorithm 43(T_i)
end for

Theorem 56 Algorithm 43: A label of a node v corresponds to a path from r to v in T and the nodes on the path are labeled $(l_1), (l_1, l_2), (l_1, l_2, l_3)$ and os on. The distance between u and v in T is obtained by reconstructing the paths from $e(u)$ and $e(v)$. This takes $\mathcal{O}(n \log n)$.

Algorithm 44 Heavy-Light-Decomposition(T)

Node r is the root of T
Let $T_1, \ldots, T_\delta$ be the sub-trees rooted at each of the δ children of r
Let T_{max} be the largest sub-tree in terms of numbers of nodes
Mark the edge (r, T_{max}) as heavy
Mark all edges to other children as light
Assign the names $1, \ldots, \delta - 1$ to the light edges of r
for $i = 1, \ldots, \delta$ **do**
    Heavy-Light-Decomposition(T_i)
end for

Theorem 57 **Algorithm 44:** We label a node in the tree by recording how to get from the root to the node. We record he number of heavy paths a node takes and also the light nodes that it takes in between. For instance, if node u can be reached by first taking 2 heavy edges, the the 7^{th} light edge, the 3 heavy edges , then the light edges 1 and 4, the label assigned to u would be $(2, 7, 3, 1, 4)$. This takes $\mathcal{O}(\log^2 n)$.

Algorithm 45 Hub-Labeling

1: **for** $i = 1, \ldots, \log D$ **do**
2: Compute the shortes path cover S_i
3: **end for**
4: **for each** $v \in V$ **do**
5: Let $F_i(v)$ be the set $S_i \cap B(v, 2^i)$, where $B(v, 2^i)$ are the nodes within the ball of radius 2^i around v
6: Let $F(v)$ be the set $F_1(v), F_2(v), \ldots$
7: The label of v consists of the nodes in $F(v)$, with their distance to v
8: **end for**

Theorem 58 **Algorithm 45:** The decoder can scan through both labels in parallel in time $\mathcal{O}(h \log n \log \Delta)$, where h is the so-called highway dimension of G, defined as $h = \max_{i,v} F_i(v)$. h is conjured to be small for road networks where this algorithm is used. In practice, this algorithm is still too slow.