#1. Input $G = (V, E, w)$

CC = connected components

S = memory available per machine $= n^{1+c}$

1. $T = \phi$

2.1. $M = \lceil \frac{m}{S} \rceil$

2.2. Label each $e \in E$ as "active"

3. While $M > 1$ do

4. for ID $i \in \{1, \dots, M\}$ independently do

5. Compute a spanning forest F_i on active edges that machine i holds

6. All edges in i that are not in F_i are marked as 'inactive

7. For each connected components in F_i, find the minimum edge based on weights, $e = (u, v)$ that connects CC to other CC

8. Mark the chooseen edge as selected

9. Send all "selected" edges on machine i to machine with ID $= \lceil \frac{i}{n^c} \rceil$.

10. $M = \lceil \frac{M}{n^c} \rceil$

11. Output T as CC of the "selected" edge-set on machine with ID=1

<u>Round Complexity</u>:

Each iteration of while-loop can be executed in $O(1)$ rounds. Lines 5 and 6 can be executed locally on each machine in a single round. Line 9 is executed in same round as prior lines, each machine creates a message that it wants to send to other machine and sends those at once.

Since M is initialized to $O(m/S) \in O(n^2/S)$ and is getting reduced by a factor of $O(n^c)$, we have that within $O(1/c)$ iterations the value of M will become 1 or less. Thus, the algorithm can be executed in

$O(1/c)$ MPC rounds

Correctness

In each iteration, the algorithm adds the minimum-weight edge that connects different CC to the MST. As the algorithm runs number of CC reduce, and MST expands. After $O(1/c)$ rounds, there is only 1 CC left which shows the MST. The algorithm is correct as it always picks the minimum-weight edge to connects different CC.

#2. Theorem 7. Algorithm 2 runs in $O(\log n)$ rounds whp.

Let X be random variable representing # of verticies removed in one round. Let X_u be a variable for vertex u being removed in a round. Then,

$$X = \sum_u X_u$$

$$\mathbb{E}[X] = \mathbb{E}[\sum_u X_u]$$

$$= \sum_u \mathbb{E}[X_u]$$

Random # are drawn uniformly and independently, possibility of vertex u being smallest among its neighbors is $\frac{1}{n^4}$.

$$\mathbb{E}[X_u] = P[X_u = 1] = \frac{1}{n^4}$$

so

$$\mathbb{E}[X] = \sum_u \mathbb{E}[X_u]$$
$$= \sum_u \frac{1}{n^4}$$
$$= \frac{1}{n^4} \sum_u 1$$
$$= \frac{1}{n^4} n$$
$$= \frac{1}{n^3}$$
$$= n^{-3}$$

Since expected # of verticies removed in one round is proportional to n^{-3}, a constant fraction of verticies can be expected in each round.

Let T be # of rounds the algorithm takes. Probability of graph not being empty after round t is:-

$$P(T>t) \leq \left(1-\frac{1}{n^3}\right)^t$$

$$P(T>t) \leq e^{(-n^{-3}).t}$$

Let suppose $t = c^* \log_3 n$ for some constant c.

$$P(T>c*\log_3 n) \leq e^{-n^{-3}* c* \log_3 n}$$

$$= e^{-c\frac{\log_3 n}{n^3}}$$

When n is large, $\frac{\log_3 n}{n^3}$ is very small, so $P(T>c*\log_3 n)$ approaches 0, saying the algorithm runs in $O(\log_3 n) \simeq O(\log n)$ rounds whp.

#3 Input $G=(V,E)$

1. $U = \phi$
2. For each $v \in V(G)$
3. Assign a color from {red, blue, green} uniformly at random and indepenedtly
4. For each $v \in V(G)$
5. Check the color of its neighbor using provided MPC routine
6. if v has same color as its neighbor, add v to set U.
7. while $U \neq \phi$, repeat 4, 5, 6.

Correctness:-

The algorithm colors the verticies w/ random 3 colors untill no 2 adjacent verticies have same color. This algorithm does not stop untill all " " have different colors, it is for sure giving us the intended result.

~~Round Complexity:~~

In each round, the algorithm colors the verticies in set U and checks against the colors of its adjacent verticies. Probably of having a vertex having the same color as its neighbor is $1/3$.

Let T be # of rounds the algorithm takes.

$E[U]$ after 1^{st} round is $n/3$, due to probability a vertex and its neighbor having same color is $1/3$. Each subsequent round we expect the $|U|$ to drop by $1/3$. Thus

$$|U|_i = \frac{1}{3}|U|_{i-1}$$

Let use $k = c * \log n$ for some constant c.

$$P(T > c * \log n) \leq \left(\frac{2}{3}\right)^{c * \log n}$$
$$= n^{-2c}$$

When c is large, then n^{-2c} approaches 0, meaning the algorithm runs is $O(\log n)$ whp.

#4. Algorithm B:

Input $= G(V,E)$

1. $\hat{M} = \phi$

2. For $i = 1 \ldots \log\left(\frac{1}{\epsilon}\right)$ do

3. Let A be set of edges of G such that each edge is sampled w/ probability $\frac{\epsilon}{2(i+1)}$

4. Let $\tilde{A}$ be subset of edges of A which do not share endpoints with other edge in A

5. Run algorithm A on a subgraph $G(\tilde{A})$ to compute r-approximate maximum matching

6. $\hat{M} = \hat{M} \cup \tilde{A}$

7. Remove from G all the verticies with endpoint in $\tilde{A}$

8. return $\hat{M}$

$(2+\varepsilon)$-approximate maximum matching

Let $M_{optimal}$ be optimal maximum matching in $G=(V,E)$ and

let $|M_{optimal}| = M$

For each iteration i, let $E_{\tilde{A}_i}$ the edges in $\tilde{A}$ computed by algorithm A on subgraph $G(\tilde{A})$.

Algorithm A computes an r-approximate maximum matching (given), we know $|E_{\tilde{A}_i}| >= \frac{|\tilde{A}_i|}{r}$

Also, edges in $E_{\tilde{A}_i}$ are non-adjacent (from line 1) and total # of removed edges at end of iteration i is at most $2|E_{\tilde{A}_i}|$.

Let e be an edge s.t. $e \in M_{optimal}$

$Pr(e \text{ is included in } E_{\tilde{A}_i} \text{ iteration } i) = p = \frac{\varepsilon}{2(i+1)}$

$Pr(e \text{ is not " " " in iteration } i) = 1-p$

$$
\begin{aligned}
Pr(e \text{ is not " " " in all iterations}) &= (1-p)^{\log(\frac{1}{\varepsilon})} \\
&\le e^{-p*\log(\frac{1}{\varepsilon})} \\
&= \left(\frac{1}{\varepsilon}\right)^{1-p} \\
&\le \varepsilon
\end{aligned}
$$

So, $Pr(\text{edge } e \in M_{optimal} \text{ is included at least in one } E_{\tilde{A}_i}) \ge 1-\varepsilon$

Since, verticies are removed from G in each iteration, total # of verticies removed at end $\le 2 * \sum |E_{\tilde{A}_i}|$.

$$
\begin{aligned}
(1-\varepsilon) * M \le 2 * \sum |E_{\tilde{A}_i}| &\le 2 * \sum \left(\frac{|\tilde{A}_i|}{r}\right) \\
&= \frac{2}{r} \sum |\tilde{A}_i|
\end{aligned}
$$

$$
\begin{aligned}
|\hat{M}| &= \sum |E_{\tilde{A}_i}| \\
&\ge r(1-\varepsilon)\frac{M}{2}
\end{aligned}
$$

$$\frac{|\hat{M}|}{M} \geq \frac{r}{2}(1-\varepsilon)$$

Let $r = 2+\varepsilon$

$$\frac{|\hat{M}|}{M} \geq \frac{(2+\varepsilon)(1-\varepsilon)}{2}$$

$$\geq \frac{(2+\varepsilon)}{2}$$

Therefore, algorithm B outputs atleast a $(2+\varepsilon)$ approximate maximum matching