The problem is to find all connected components of a given graph $G(V, E)$. $|V| = n$ and $|E| = m$

Input: A list of all edges in the graph. The edges are ordered so the list is of length $2m$. (both (u, v) and (v, u) are included in the list.

Output: An array $D[N]$, such that $D[i] = D[j]$ iff nodes i and j are in the same component.

Processors: We use $2m + n$ processors on the PRAM - one processor per ordered edge and one per vertex.

1 Operations on Pseudo-trees

A **pseudo-tree** is a tree with the root being its own parent, i.e. there is a self-loop at the root of the tree.

Operations performed on the pseudo-trees:

Ordered Grafting. This is an operation which will be done in parallel by all edge-processors. The processor for the edge (u,v) will perform the following operation:

if $(D(u) = D(D(u)))$ and $(D(v) < D(U))$
then $D(D(u)) = D(v)$.

That is, if u is a root or child of root, then graft u's tree to the parent of v. Note that the restriction $(D(v) < D(u))$ ensures that we don't end up with cycles after a grafting operation - an edge can graft only one way. More precisely, suppose that the D array is ordered (i.e. $D(u) < u$) before an Ordered Grafting operation. Then the D array is ordered after the Ordered Grafting also, implying that there cannot be a cycle.

Thus, given a forest of pseudo-trees, this operation will maintain the forest.

Unordered Grafting: This is similar to Ordered Grafting, but without the restriction of $(D(v) < D(u))$ while grafting u's tree to v's tree. More precisely, an Unordered Grafting operation is performed by all edge-processors in parallel with the processor for the edge (u, v) performing the following operation:

if $(D(u) = D(D(u)))$
then $D(D(u)) = D(v)$.

That is, if u is a root or child of root, then graft u's tree to the parent of v. Note that without the restriction $(D(v) < D(u))$ we are not immediately ensured of a cycle free D-array.

Pointer Doubling. This is an operation peformed by all vertex-processors in parallel. The processor for the vertex u will perform the following operation:

$$D(u) = D(D(u))$$

This operation corresponds to shrinking the pseudo-trees by approximately halving the height of the trees. Note that this operation also does not introduce any cycles, hence will maintain a forest of pseudotrees.

2 Algorithm 1: Time $O(nlogn)$

For i=1 to n
  D(i) = i
Repeat (as long as required)
  Ordered Grafting step
  Repeat Pointer Doubling (as many times as required to reduce
  all pseudo-trees to stars)

Correctness and time complexity: We begin with a forest of Pseudo-stars (Each vertex points to itself). Neither Ordered Grafting nor Pointer Doubling can introudce cycles, as observed above. The Pointer Doubling loop will reduce every tree to a star. So at the beginning of every iteration of the outer loop, D forms a forest of pseudo-stars.

Every iteration involves one Ordered Grafting step. This step will reduce the number of pseudo-trees by at least 1. (Note that there exist cases for which every iteration will reduce the number of trees by exactly one.) So even in the worst case n iterations of the outer loop will suffice to reduce each connected components to a pseudo-star.

Within each iteration, the height of the trees formed by the grafting step can be at most n. So repeating each inner loop $logn$ times suffices.

So the algorithm ends after $nlogn$ steps with each connected component reduced to a star. Two vertices in the same connected component will have the same D values (the root of their star), and two vertices in different connected components will have different D values (the two different roots). Hence the algorithm is correct and finishes in time $O(nlogn)$

Time: Pointer Doubling might be required $log(ht)$, i.e. O(log n) times in every iteration. Also, in every iteration of the outer loop, the number of pseudo-trees may decrease by only 1 in the worst case (see example below). Therefore the total time required is O(n logn).

Since a single processor has to perform DFS to find connected components (time $O(n+m)$), **speedup** $= \frac{m+n}{nlogn}$, and **efficiency** $= O(\frac{1}{nlogn})$.

3 Algorithm 2: time $O(logn)$

for i=1 to n
    $D(i) = i$
Do $klogn$ times
    Ordered Grafting
    Unordered Grafting of stars which have not taken part in
    Unordered Grafting step. (*Call these inactive stars*)
    Pointer Doubling.

Correctness and Time Complexity: We first prove that the following three properties are invariants of the algorithm in the beginning of every iteration and on exit.

- i and $D(i)$ are in the same connected component.
- D forms a pseudo-tree.
- Unless i is a leaf, $D(i) \leq i$.

Proof of Invariance: These properties hold right in the beginning of the algorithm. Suppose they hold at the beginning of an iteration. We will show that they hold evn after that iteration.

Let us see how each of the three operations in an iteration affects these purported invariants:

- *Ordered Grafting:* $D(i)$ is made j only if (i, j) is an edge of the graph. So the first properties holds. Since an edge-processor grafts a pseudo tree from its root onto a non-leaf node of another tree, even with the leaves being out of order our earlier observation reagrding maintainance of a pseudo-forest holds - no cycles are introduced. Thirdly, because of our ordering restriction on grafting, the last property will also hold. Thus all three properties are maintained after Ordered Grafting step.

- *(Unordered) Star Grafting:* Again, the first property is obviously maintained. Further we claim that no cycles can be introduced even after unordered grafting. Since only inactive stars participate in this step, a cycle is possible only if two inactive stars graft onto each other. But this is not possible, since not both would have refrained from grafting in the Ordered Grafting step - the one with smaller numbered root *would* have grafted onto the root of the other. Thus this step also maintains pseudo-forest property. Thirdly, because the grafting is unordered, we lose some ordering ($D(i) < i$). But since only stars graft in this step, we can see that the ordering is lost only for nodes which become parents of leaves after the step. Thus the first two properties continue to hold after this step, and $D(i) < i$ for all nodes except leaves and parents of leaves.

- *Pointer Doubling:* By transitivity, node $D(D(i))$ belongs to the same connected component as node i. So after pointer doubling, $D(i)$ belongs to same component as i. Pointer Doubling is not going to introduce any cycles. Thirdly, after pointer doubling step, all parents of leaves (as well as leaves) become leaves. So property 3 holds again.

Thus, after one iteration, all three properties hold, if they held before that iteration. Since they hold in the beginning of the algorithm, they hold at the beginning of every iteration and at the end of the algorithm.

Time Analysis: All connected components become pseudo-stars after $O(logN)$ steps.

Let $h_i(T)$ be the vertex height of a pseudo-tree T after iteration i.

Let K be some connected component.

Define $h_i(K)$ as the sum of the height of the trees that comprise the connected componenet K.

In every iteration:

- The grafting steps cannot increase $h_i(K)$ for any connected componenet K.

- Pointer Doubling makes

$$h_{i+1}(T) \leq \frac{3}{4} h_i(T)$$

  for every non-star-tree. Stars remain of constant height.

Therefore all connected componenets become stars after time $log_{\frac{4}{3}} N$.

One can use the following simple Lemma (which is stated without proof) to program the end of the algorithm (which may occur before the time mentioned above):

Lemma: If a tree does not do anything in an iteration then it was a complete connected component before that iteration started.

Since a single processor has to perform DFS to find connected components (time $O(n+m)$), **speedup** $= \frac{m+n}{logn}$, and **efficiency** $= O(\frac{1}{logn}$.