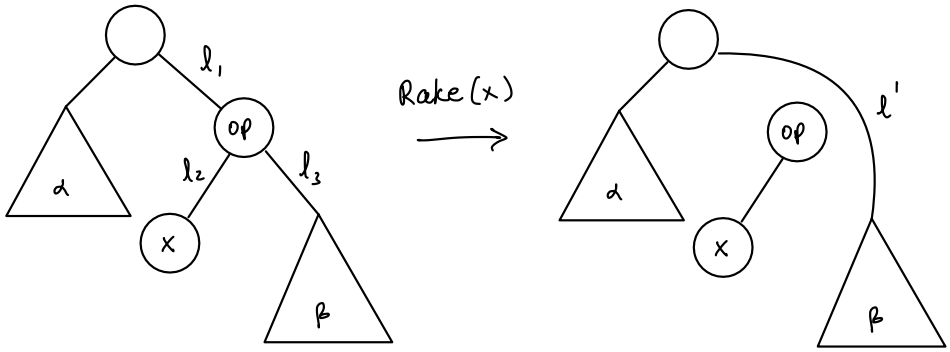
Homework Assignment #3
Due: September 26, 2025 at 11:59 p.m.

1. Suppose we want to use tree contraction to evaluate a binary tree that represents a (monotone) boolean expression. Each internal node has exactly two children. Each leaf has a value that is either TRUE or FALSE. Each internal node contains the operator AND or OR.

We associate a label with each edge. The label is one of the following three functions: T, F or id, where $T(x) = \text{TRUE}, F(x) = \text{FALSE}, id(x) = x$. The label on an edge connecting a child to its parent represents the function that should be applied to the value coming up from the child before it becomes an operand of the parent node's operator. Initially, every edge has the label id.

[6] **(a)** Fill in the following code to construct the label ℓ' of the new edge when the following rake operation is applied.




```
1: if ℓ1 = T then
2: |  ℓ' ← _____
3: else if ℓ1 = F then
4: |  ℓ' ← _____
5: else ▷ ℓ1 = id
6: |  if op = AND then
7: |  |  if ℓ2(x) = TRUE then
8: |  |  |  ℓ' ← _____
9: |  |  else ▷ ℓ2(x) = FALSE
10: |  |  |  ℓ' ← _____
11: |  else ▷ op = OR
12: |  |  if ℓ2(x) = TRUE then
13: |  |  |  ℓ' ← _____
14: |  |  else ▷ ℓ2(x) = FALSE
15: |  |  |  ℓ' ← _____
```

[2] **(b)** Now suppose we also want to allow NAND and NOR operators in the expressions represented by the trees, where x NAND $y = \neg(x$ AND $y)$ and x NOR $y = \neg(x$ OR $y)$. How would this change the set of labels needed for the edges? (You do *not* have to give all the details of how rake would compute new labels.)

[4] **(c)** Now, suppose we want to compute a minimum vertex cover of a binary tree. A *vertex cover* is a set C of tree nodes such that at least one endpoint of each tree edge is in C. A *minimum* vertex cover is a vertex cover that is as small as possible.

The following greedy algorithm works for a single process. Start from the leaves and work your way up. Do not include leaves in C. Once you have decided whether to include the children of an internal node x in C, then put x in C if and only if one of its children is not in C.

Explain how to construct a vertex cover of a binary tree on a PRAM. What is the time and work required for your algorithm?

Hint: use part (b) and the fact that we can run tree contraction in reverse to compute the value of every subexpression. At each step of the reverse tree contraction, we know the value of all nodes present in the tree. Undoing a rake step adds a leaf and its parent back into the tree, and we can easily compute that parent's value.

2. We discussed the following problem in class. We are given two sorted arrays $A[1..n]$ and $B[1..m]$ (where $m \leq n$ and assume all elements in the two arrays are distinct). We must find, for each $B[i]$ find the rank of $B[i]$ in A, i.e., the number of elements of A that are less than $B[i]$. The $O(\log \log n)$ time algorithm using $n+m$ processors $P_1, \ldots, P_{n+m}$ works as follows (see also Algorithm 4.2 on p. 80 of the textbook).

1: divide B into $\sqrt{m}$ segments $B_1, \ldots, B_{\sqrt{m}}$, each of size $\sqrt{m}$
2: **for** $i \in \{1, \ldots, \sqrt{m}\}$ in parallel **do**
3: | find the rank of the *first* element of B_i in A and store this rank in $j[i]$
4: let A_i be the segment $A[j(i)+1..j(i+1)]$
5: **for** $i \in \{1, \ldots, \sqrt{m}\}$ in parallel **do**
6: | find the ranks of all elements of B_i in $A[i]$ recursively; use $|A_i| + |B_i|$ processes to do this
7: **for** $k \in \{1, \ldots, m\}$ in parallel **do**
8: | let i be $\lceil k/\sqrt{m} \rceil$, so that $B[k]$ is in group B_i
9: | compute the rank of $B[k]$ in A as the rank of B_i's smallest element in A (computed in
10: | step 3 and stored in $j[i]$) plus the rank of $B[k]$ in A_i (computed in step 6)

We saw how to perform each step (aside from the recursive calls) in constant time. Since the depth of recursion is $O(\log \log n)$, the total running time of the algorithm is $O(\log \log n)$.

When we discussed this algorithm, we glossed over how to allocate the $n+m$ processes to the recursive calls in step 6. It is actually a bit tricky to do this allocation in $O(1)$ time.

One way to allocate processors to subproblems is to use processors $P_{(i-1)\sqrt{m}+j(i)+1}, ..., P_{i\sqrt{m}+j(i+1)}$ for the ith subproblem involving A_i and B_i. Note that this allocates exactly $|A_i| + |B_i|$ processors to the subproblem, as required. This question asks you to explain how each processor can figure out which subproblem it is assigned to *and* its index among all processors working on that subproblem. Your answer to each of the following parts must take $O(1)$ time on a CREW PRAM.

[2] **(a)** Suppose we divide the $n+m$ processors into groups $G_1, \ldots, G_r$ of size $\sqrt{m}$ (where $r = \frac{n+m}{\sqrt{m}}$). Thus, G_ℓ consists of processes $P_{(\ell-1)\sqrt{m}+1}, ..., P_{\ell\sqrt{m}}$. Explain how the kth process of group G_ℓ can determine whether the *first* process in group G_ℓ is assigned to subproblem k.

[2] **(b)** Explain how *every* process in group G_ℓ can learn which subproblem the *first* process in group G_ℓ is assigned to.

[1] **(c)** How many different subproblems can the processes in group G_ℓ be assigned to?

[2] **(d)** Explain how every process in group G_ℓ can figure out which subproblem it is assigned to.

[2] **(e)** Explain how every process can compute its index among all processors assigned to its subproblem.