Algorithm 1: Baseline

Input: $\mathcal{S}$: A set of trees; τ: tree edit distance threshold;
Output: $\mathcal{A} = \{\langle T_a, T_b \rangle \mid T_a \in S, T_b \in S, \mathtt{ted}(T_a, T_b) \leq \tau\}$

begin
  $R = \phi$;
  foreach $T \in S$ **do**
    `// string get by post order traverse of` T
    $t =$ Postorder(T);
    add $\langle T, t \rangle$ into R;
  `// StringSimJoin returns all the pairs with string edit distance no larger than` τ
  $\mathcal{C} =$ StringSimJoin(R, τ);
  foreach $\langle T_a, T_b \rangle \in \mathcal{C}$ **do**
    if $\mathit{ted}(T_a, T_b) \leq \tau$ **then** add $\langle T_a, T_b \rangle$ into $\mathcal{A}$;
  return $\mathcal{A}$;

Algorithm 2: TreeJoin

Input: $\mathcal{S}$: A set of trees; τ: tree edit distance threshold;
Output: $\mathcal{A} = \{\langle T_a, T_b \rangle \mid T_a \in S, T_b \in S, \mathtt{ted}(T_a, T_b) \leq \tau\}$

begin
  $\mathcal{I} = \phi$; // an inverted index
  foreach $T \in S$ **do**
    `// Here we need some dynamic programming to select` $\tau + 1$ `subtrees of` T `based on` $\mathcal{I}$
    `// We may propose some pruning techniques here`
    `// All the trees passed the pruning techniques should be added into the candidate set` $\mathcal{C}$
    foreach $T' \in \mathcal{C}$ **do**
      if $\mathit{ted}(T, T') \leq \tau$ **then** add $\langle T, T' \rangle$ into $\mathcal{A}$;
    `// Indexing`
    foreach *node* $n \in T$ **do**
      `//` $T(n)$ `is the subtree of` T `rooted at` n`. EulerTour: http://en.wikipedia.org/wiki/Euler_tour_technique`
      $t =$ EulerTour($T(n)$);
      add T into $\mathcal{I}[t]$;
  return $\mathcal{A}$;