Bioinformatics Assignment

AMC: Z0132271

1 Question 1

1.1 Part A

2 Question 2

2.1 Part A

The algorithm *BUILD* presented by Aho et al. focuses on the algorithmic goal of reconstructing a tree T that satisfies an input set of constraints C. The lowest common ancestor of two nodes x and y, denoted $a = LCA(x, y)$, is a node a such that no proper descendant of a (i.e. no node b with a as an ancestor and b is not a) which is an ancestor of both x and y. This concept is used to formulate constraints for the structure of a tree; the constraint $(i, j) < (k, l)$ over the set of leaves $\{i, j, k, l\}$ specifies that the node $LCA(i, j)$ is a proper descendant of $LCA(k, l)$.

The proposed algorithm implements a recursive process, upon input of a set of nodes S and a set of constraints C (of the aforementioned structure) over these nodes. The base of the recursion is where S only contains a single node; in this case the algorithm outputs the singleton tree T purely consisting of this node as the root. Otherwise, we compute a partition upon the nodes of S with respect to the constraints C. A partition π_C is the subdivision of a set of nodes into subsets $S_1, S_2, \ldots, S_r$ such that the descendants of each child m of the root node of T constitutes the set S_m. To satisfy C, for each constraint $(i, j) < (k, l) \in C$ the corresponding partition $\pi_C = S_1, \ldots, S_r$ must satisfy two conditions. Firstly, both i and j must reside within the same set. Secondly, if k and l lie within the same set, this must imply all nodes i, j, k and l lie within the same set. All sets in the partition must satisfy these conditions; namely, no two nodes may reside within the same set unless specified by either of the prior rules.

For a tree T to exist satisfying C, we must be able to find a satisfactory partition $\pi_C = S_1, \ldots, S_r$ where $r \geq 2$—as the existence of at least two subsets to recurse into is a necessary condition when constructing a tree where each non-leaf node has at least two children. Thus, after computing the partition we must check it contains a minimum of two sets; if not, we output a null tree.

Once a sufficient partition π_C has been constructed, for each constituent set $S_m \in \pi_C$ we generate a corresponding subset of constraints $C_m \subseteq C$ such that C_m contains constraints only involving the nodes of S_m. We then recurse, implementing the algorithm upon the inputs S_m and C_m for all $1 \leq m \leq r$. If a tree output by any of these lower recursions is null, the null tree is output at the current level of recursion. Otherwise, on construction of the collection of non-null trees $T_1, \ldots, T_r$, we output the composite tree T consisting of a new node as the root with r children, where each child node $1 \leq m \leq r$ is the root of the tree T_m (with the corresponding full tree expanded below it).

Thus, on conclusion of the highest level of recursion one of two outputs is observed: the tree T with leaves S and a structure satisfying the constraints C, or the null tree, indicating no tree exists satisfying the given constraints.

2.2 Part B

Algorithm 1 Compute $\pi_C = S_1, S_2, \ldots, S_r$

Input: constraint set C upon node set S
Output: partition π_C

initialise collection of sets $\pi_C \leftarrow \emptyset$
if C is empty **then**
    return $\pi_C =$ the collection of singleton sets each containing one node from S
else
    for all constraints $(i, j) < (k, l) \in C$ **do**
        if i is in some set S_m **and** j is not in any set **then**
            allocate j to S_m
            add S_m to the collection of sets π_C
        else if j is in some set S_m **and** i is not in any set **then**
            allocate i to S_m
            add S_m to π_C
        else if i is in set S_m **and** j is in a different set S_n **then**
            merge S_m and S_n to form a new set $S_l = S_m \cup S_n$ within π_C
        else if neither i or j is in any set **then**
            create a new set $S_m = \{i, j\}$
            add S_m to π_C
        end if
    end for
    for all nodes $n \in S$ which are not in any set **do**
        create a new singleton set $S_m = \{n\}$
        add S_m to π_C
    end for
    for all constraints $(i, j) < (k, l) \in C$ **do**
        if k and l are both in set S_m **and** i and j are in a different set S_n **then**
            merge S_m and S_n to form a new set $S_l = S_m \cup S_n$ within π_C
        end if
    end for
    return $\pi_C =$ the collection of all remaining sets
end if

2.3 Part C

Firstly, we shall label each constraint such that they can subsequently be referred to by index only:

1. $(e, f) < (k, d)$
2. $(c, h) < (a, n)$
3. $(j, n) < (j, l)$
4. $(c, a) < (f, h)$
5. $(j, l) < (e, n)$
6. $(n, l) < (a, f)$
7. $(d, i) < (k, n)$
8. $(d, i) < (g, i)$
9. $(c, l) < (g, k)$
10. $(g, b) < (g, i)$
11. $(g, i) < (d, m)$
12. $(c, h) < (c, a)$

Figure 1: Step 1

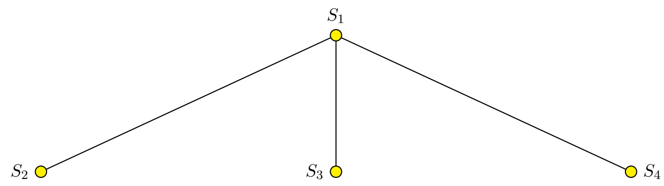


Figure 2: Step 2

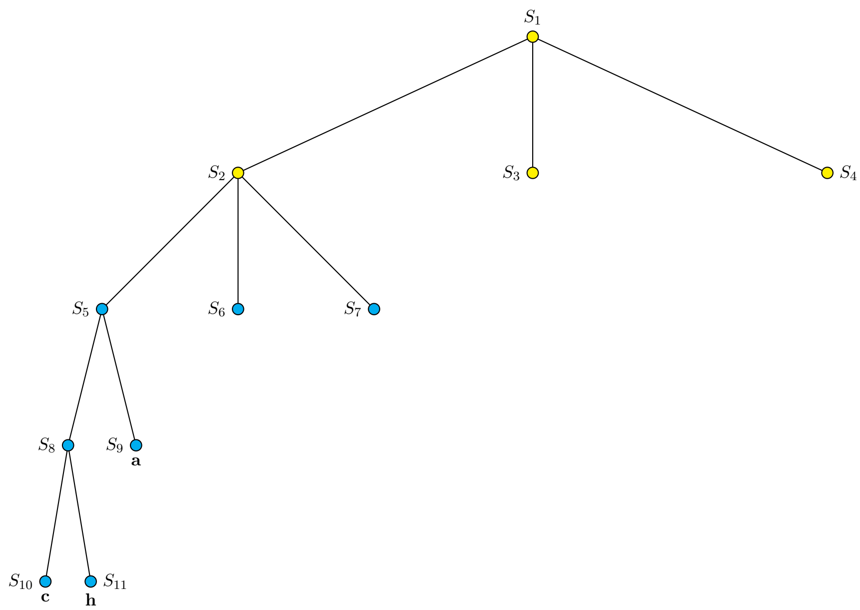


Figure 3: Step 3

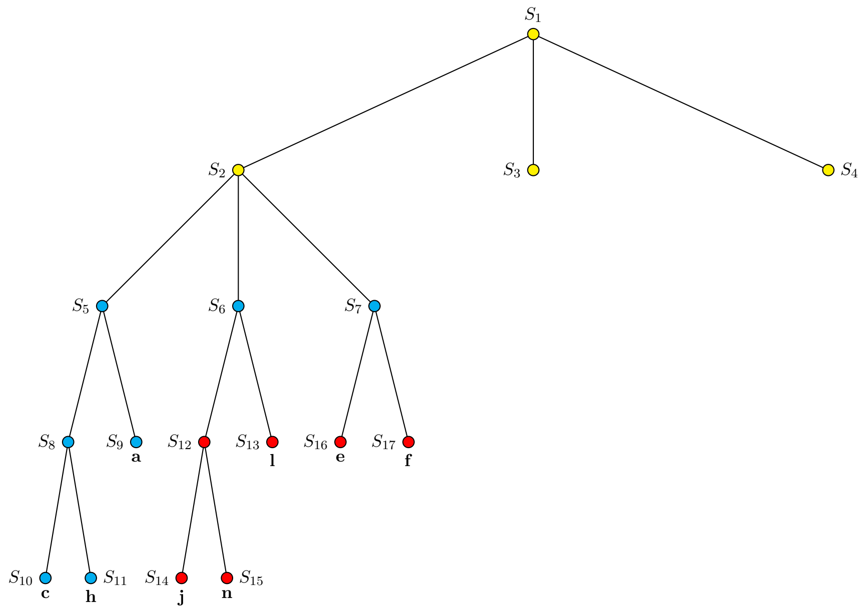
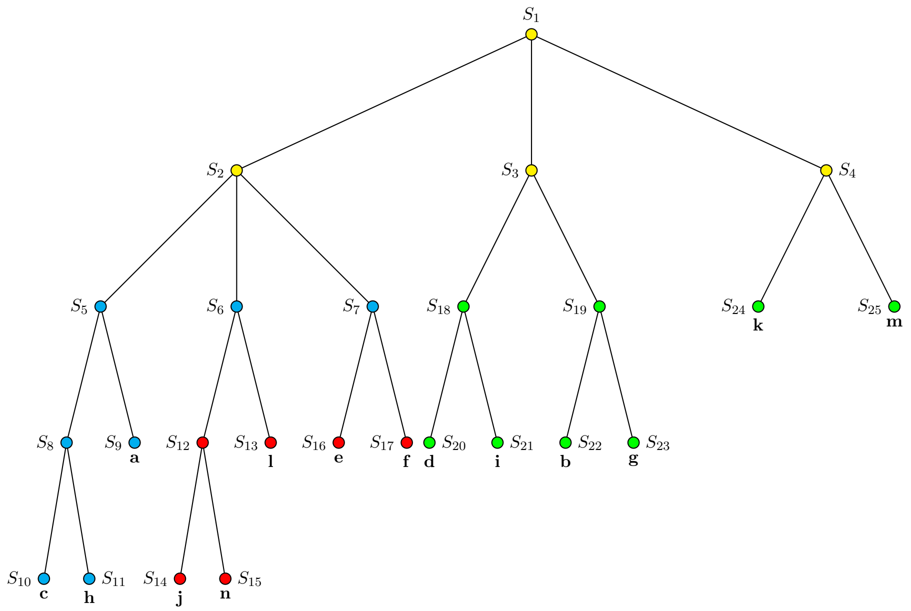


Figure 4: Step 4—full tree



13. $(e, f) < (h, l)$ 14. $(j, l) < (j, a)$ 15. $(k, m) < (e, i)$ 16. $(j, n) < (j, f)$

The initial inputs into the top layer of recursion are the set of nodes $S = \{a, b, \ldots, n\}$ and the set of constraints $C = \{1, 2, \ldots, 16\}$ (as indexed above). We will refer to these as S_1 and C_1.

The first partition we compute is π_{C_1}. Following the fist condition outlined in 2.1 (and detailed in lines 5–18 in 2.2), we generate the initial partition sets $\pi_{C_1} = \{e, f\}, \{c, h, a, j, n, l\}, \{d, i, g, b\}, \{k,$ The second condition (algorithm lines 23–26) means we must compute mergers, resulting in the partition $\pi_{C_1} = \{a, c, e, f, h, j, l, n\}, \{b, d, g, i\}, \{k, m\}$. We will denote these subsets S_2, S_3, and S_4. This process is shown through the yellow nodes in Figure 1.

Now we recurse into the subset $S_2 = \{a, c, e, f, h, j, l, n\}$ with the corresponding constraints subset $C_2 = \{2, 3, 4, 5, 6, 12, 13, 14, 16\}$ through applying $BUILD(S_2, C_2)$. This recursive step produces the partition $\pi_{C_2} = \{a, c, h\}, \{j, l, n\}, \{e, f\}$. We will denote these as S_5, S_6, and S_7. Next we recurse further into $S_5 = \{a, c, h\}$ with $C_5 = \{12\}$. This generates the partition $\pi_{C_5} = \{c, h\}, \{a\} = S_8, S_9$. When recursing into S_8 we find $C_8 = \emptyset$ and so output the partition $\pi_{C_8} = \{c\}, \{h\} = S_{10}, S_{11}$, thus meaning the nodes c and h are the leftmost deepest leaves. We can also see that S_9 is a singleton set, and is thus the leaf node a at the above level of recursion. This is visualised through the blue nodes in Figure 2.

As we go up the levels of recursion, we next need to compute $BUILD(S_6, C_6)$ where $S_6 = \{j, l, n\}$ and $C_6 = \{6\}$ to generate the partition $\pi_{C_6} = \{j, n\}, \{l\} = S_{12}, S_{13}$. We first recurse into $BUILD(S_{12}, C_{12})$, where $C_{12} = \emptyset$, resulting in two singleton sets S_{14} and S_{15} indicating j and n are leaves at this level. On observation of the singleton set S_{13} at the level above we find that l is a leaf here. The next lowest recursion, $BUILD(S_7 = \{e, f\}, C_7 = \emptyset)$, gives the singleton sets $S_{16} = \{e\}$ and $S_{17} = \{f\}$, indicating e and f are leaves at this level. Figure 3 demonstrates this process through the red nodes.

Next we recurse all the way back to the top level to explore $BUILD(S_3 = \{b, d, g, i\}, C_3 = \{8, 10\})$, generating the partition $\pi_{C_3} = \{d, i\}, \{b, g\} = S_{18}, S_{19}$. Recursing into $BUILD(S_{18}, C_{18} = \emptyset)$ gives the partition $\pi_{C_{18}} = \{d\}, \{i\} = S_{20}, S_{21}$ and thus leaves d and i at this level. Similarly, $BUILD(S_{19}, C_{19} = \emptyset)$ gives $\pi_{C_{19}} = \{b\}, \{g\} = S_{22}, S_{23}$ and leaves b and g. Once again, we return to the top level of recursion and compute $BUILD(S_4 = \{k, m\}, C_4 = \emptyset)$, giving the partition $\pi_{C_{C_4}} = \{k\}, \{m\} = S_{24}, S_{25}$, which when we recurse into give the final two leaves k and m. This concluding part of the algorithm, and the consequentially generated full tree, is shown through in Figure 4 (the green nodes indicating the additions from this part).

2.4 Part D

The proposed algorithm *Reverse-BUILD* in Algorithm 2 implements an iterative method for building a set of constrains C from a given tree T through which we could apply the $BUILD$ algorithm and construct a tree isomorphic to T. It relies upon splitting T into distinct layers, where each layer contains all nodes at the sample depth, and considering each of these layers L_i through a bottom-up approach. We start at the deepest layer and assign each of the leaves its own singleton set; sets on sibling leaves are then merged and assigned to label the corresponding parent node p.

We then consider the next layer, where similarly any leaves are assigned to singleton sets. Next, sibling nodes in layer L_i with the same parent p in layer L_{i+1} are grouped. Each of these groups corresponds to a partition; namely, for the parent node p in layer L_{i+1} with N children in layer L_i, a partition is constructed of the sets associated with each of these children: $\pi_p = S_1, S_2, \ldots, S_N$. For each of the constituent sets $S_m \in \pi_p$, we then

Algorithm 2 Reverse-BUILD

Input: tree T with labelled leaves S

Output: a corresponding the set of constraints C

1: initialise $C \leftarrow \emptyset$
2: **for all** non-root layers L_i of T (from deepest leaves to the layer before the root) **do**
3:     assign each leaf l at layer L_i the singleton set containing its value $S_l = \{v_l\}$
4:     **for all** non-leaf nodes p in layer L_{i+1} (above L_i) **do**
5:         form the partition $\pi_p = S_1, S_2, \ldots, S_N$ consisting of the sets assigned to each of the N children of p
6:         **for all** sets $S_m \in \pi_p$ containing more than one element **do**
7:             **for all** other sets $S_n \in \pi_p \setminus S_m$ **do**
8:                 **if** no constraint $(a, b) < (c, d) \in C$ exists where $a, b \in S_m$ and either $c \in S_n$ or $d \in S_n$ **then**
9:                     sample two distinct leaves from the first set $a, b \in S_m$
10:                     set c equal to either of these leaves a or b
11:                     sample another leaf from the second set set $d \in S_n$
12:                     add a new constraint over these values: $C \leftarrow (a, b) < (c, d)$
13:                 **end if**
14:             **end for**
15:             merge the sets of π_p and assign this to the corresponding parent node p
16:         **end for**
17:     **end for**
18: **end for**
19: **return** the set of constraints C

cycle through all other sets S_n in the same partition and ensure there is a constraint which links the elements of S_m and S_n in line with the satisfaction of the two rules outlined in Section 2.1.

The construction of this constraint (in absence of one existing already) is implemented through the lines 8–13 in Algorithm 2. We need to ensure this is the case between every set (i.e. the rules are followed exactly) in the partition such that the correct dependencies between nodes are exhibited in the full constraint set. If this is followed, it maintains that when the $BUILD$ algorithm is applied to the full set we can once again generate the same partitions (when we reach this level of the tree).