Bioinformatics Assignment - LLLL76

QUESTION ONE

A

Given a set of leaf nodes, S, and a set of constraints on these nodes, C, build a tree such that all the constraints are true. A single constraint is in the form $(x, y) < (a, b)$ where (a, b) is a single node that is an ancestor of both a and b and has no descendants that are also an ancestor of both a and b. In the example constraint (x,y) is a descendent of (a,b), this forms a constraint. The build algorithm recursively splits the set of leaf nodes into sub-sets so as to create sub-trees in the tree that is created by the algorithms execution. The algorithm starts with a recursion end check, if S has length one then just return that node itself, a single node cannot be a tree. At the first execution of the function BUILD, compute π_c, this is the splitting of the input set given the constraints using three rules. Rule one is for each constraint in the form $(x, y) < (a, b)$, x and y are in the same block. Rule two is for each constraint in the form $(x, y) < (a, b)$, if a and b are in the same block then x,y,a,b are all in the same block. Rule three is that no two leaves are in the same block unless it is shown by rules one and two. Once these rules have been applied a number of subsets exist, 1 to r. If r is equal to one then a null tree is returned because the number of subsets is zero. For each of these numbers calculate C_m or the set of constants that apply to that subset and T_m or the tree that is created by that subset, this is done by recursively calling the build algorithm on the set of nodes that are in this subset and the set of constraints that apply to this subset. Once this has been done for all children of the root node then a tree exists, a new root node is created such that each of its children is the root node of the trees T_x where x is each of the numbers 1 to r corresponding to subsets of the set of leaf nodes. This tree is then returned.

B

```
π_c(SetOfNodes):
    set = ()
    For child in Children(root):
        set.append(GetLeaves(child, SetOfNodes))
    Return(set)

GetLeaves(node,SetOfNodes):
    If Children(node) = Null:
        Return(node)
    Else:
        set = ()
        For child in Children(node):
            set.append(GetLeaves(child, SetOfNodes))
        Return(set)
```

C

$$T(n, m) <= K\ T(n_i, m_i) + L\ f(n_i, m_i)$$

where

$$K <= 2n - 1$$

$$L <= n - 1$$

and

n_i is the nodes in one of the subsets of n that are the result of running the partition step. m_i is one of the subsets of m that are the result of running the partition step.
Therefore the time complexity is given as:

$$T(n,m) <= O(2n-1) + O(n-1)f(n,m)$$
$$T(n,m) <= O(n) + O(n)f(n,m)$$
$$T(n,m) <= O(n)f(n,m)$$

D

$BUILD($
$(a,b,c,d,e,f,g,h,i,j,k,l,m,n),$
$((e,f) < (k,d)(c,l) < (g,k)(c,h) < (a,n)(g,b) < (g,i)(j,n) < (j,l)(g,i) < (d,m)(c,a) < (f,h)(c,h) < (c,a)(j,l) < (e,n)(e,f) < (h,l)(n,l) < (a,f)(j,l) < (j,a)(d,i) < (k,n)(k,m) < (e,i)(d,i) < (g,i)(j,n) < (j,f))$
$):$

$\pi_c = (e,f,a,c,h,j,l,n),(d,i,b,g),(k,m)$

$For\ m := 1\ to\ 3\ do$

$C_1 = ((c,h) < (a,n),(j,n) < (j,l),(c,a) < (f,h),(j,l) < (e,n),(n,l) < (a,f),(c,h) < (c,a),(e,f) < (h,l),(j,l) < (j,a),(j,n) < (j,f))$
$T_1 = BUILD($
$(e,f,a,c,h,j,l,n),$
$((c,h) < (a,n),(j,n) < (j,l),(c,a) < (f,h),(j,l) < (e,n),(n,l) < (a,f),(c,h) < (c,a),(e,f) < (h,l),(j,l) < (j,a),(j,n) < (j,f))$
$)$
$\pi_c = (c,h,a),(j,n,l),(e,f)$
$For\ m := 1\ to\ 3\ do$
$C_1 = ()$
$T_1 = BUILD((c,h,a),())$
$\pi_c = (c),(h),(a)$
$For\ m := 1\ to\ 3\ do$
$C_1 = ()$
$T_1 = BUILD((c),())$
$Return(c)$
$C_2 = ()$
$T_2 = BUILD((h),())$
$Return(h)$
$C_3 = ()$
$T_3 = BUILD((a),())$
$Return(a)$
$C_2 = ()$
$T_2 = BUILD((j,n,l),())$
$\pi_c = (j),(n),(l)$
$For\ m := 1\ to\ 3\ do$
$C_1 = ()$
$T_1 = BUILD((j),())$
$Return(j)$
$C_2 = ()$
$T_2 = BUILD((n),())$

$Return(n)$

$C_3 = ()$

$T_3 = BUILD((l), ())$

$Return(l)$

$C_3 = ()$

$T_3 = BUILD((e, f), ())$

$\pi_c = (e), (f)$

$For\ m := 1\ to\ 2\ do$

$C_1 = ()$

$T_1 = BUILD((e), ())$

$Return(e)$

$C_2 = ()$

$T_2 = BUILD((f), ())$

$Return(f)$

$C_2 = ((d, i) < (g, i), (g, b) < (g, i))$

$T_2 = BUILD($

$(d, i, b, g),$

$((d, i) < (g, i), (g, b) < (g, i))$

$)$

$\pi_c = (d, i), (g, b)$

$For\ m := 1\ to\ 2\ do$

$C_1 = ()$

$T_1 = BUILD((d, i), ())$

$\pi_c = (d), (i)$

$For\ m := 1\ to\ 2\ do$

$C_1 = ()$

$T_1 = BUILD((d), ())$

$Return(d)$

$C_2 = ()$

$T_2 = BUILD((i), ())$

$Return(i)$

$C_2 = ()$

$T_2 = BUILD((g, b), ())$

$\pi_c = (g), (b)$

$For\ m := 1\ to\ 2\ do$

$C_1 = ()$

$T_1 = BUILD((g), ())$

$Return(g)$

$C_2 = ()$

$T_2 = BUILD((b), ())$

$Return(b)$

$C_3 = (k, m)$

$T_3 = BUILD($

$(k, m),$

$()$

$)$

$\pi_c = (k), (m)$

$For\ m := 1\ to\ 2\ do$

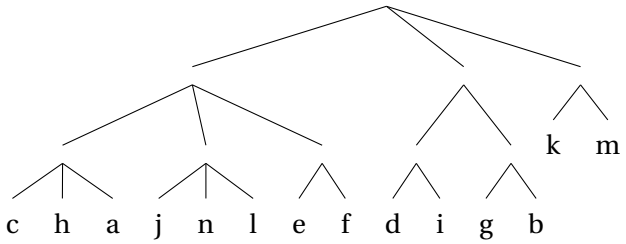
$C_1 = ()$

$T_1 = BUILD((k),())$
$\quad Return(k)$
$C_1 = ()$
$T_1 = BUILD((m),())$
$\quad Return(m)$

$Create\ Tree\ T\ with\ new\ root\ and\ the\ roots\ of\ T_m\ as\ children\ where\ 1 <= m <= r$

$Return(T)$

Where T =



E

$REVERSE - BUILD(T)$:
$\quad For\ each\ internal\ node\ x$:
$\quad\quad Create\ a\ set\ S_x$
$\quad\quad For\ range\ (1 : NumberOfChildren(x) - 1)$:
$\quad\quad\quad S_x.append((ListOfChildren[0], ListOfChildren[1]))$
$\quad\quad\quad Remove(ListOfChildren[0])$
$\quad\quad End$
$\quad End$
$\quad For\ each\ internal\ node\ x$:
$\quad\quad For\ each\ pair\ in\ set\ S_x$:
$\quad\quad\quad Create\ constraint\ S_{child} < S_{parent[0]}$
$\quad\quad End$
$\quad End$

QUESTION TWO

A

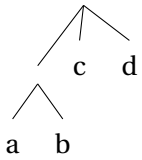
Step five of the algorithm is where nesting and subtrees are preserved. This is done by merging two nodes that appear in all graphs as an identical subtree. The algorithmic steps of this are a test of all edges in the graph made in step three, if the weighting of any edge is equal to the number of tree input to the algorithm, this means that the two nodes that are to be merged have a common ancestor that isn't the root in all trees meaning that it is a subtree and by merging these nodes this subtree cannot be dissected and will be maintained in the final supertree.

B

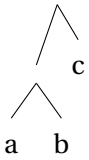
The build algorithm is based upon constraints, these can be represented as graphs and the set of all graphical representations of the constraints for any given input to the Build algorithm can be used as

input to the MinCutSupertree algorithm to achieve the same result as the build algorithm.

There are two cases that a constraint can fall into. If the constraint is in the form $(a,b) < (c,d)$ and $a != b != c != d$ then the tree will look like:



If the constraint is in the form $(a,b) < (c,d)$ and $a == d$ *or* $b == d$ then the tree will look like



These are the only two possible forms that a constraint can take.