ECS 122A Lecture 8 Jasper Lee

Tree DP

Not all DP algs/probs have to follow rectangular table structure.

Have already seen some examples in DtC (Tree DP can involve DtC elements).

Longest monochromatic path

Find longest all-red path

Len[node] ≐ longest all-red path within subtree rooted at node

Len-root[node] ≐ longest all-red path that starts at node and lies within the subtree rooted at node

Runtime: $O(n)$ time

Each node we do $O(\deg(\text{node}))$ work

⇒ sums to $O(\text{tree size})$ work

Recurrence:

$$\text{Len}[\text{node}] = \max\left(\text{Len}[\text{node.left}], \text{Len}[\text{node.right}], \begin{cases} \text{if node is red,} \\ \quad 1 + \text{Len-root}[\text{node.left}] \\ \quad + \text{Len-root}[\text{node.right}] \\ \text{else } 0 \end{cases}\right)$$

$$\text{Len-root}[\text{node}] = \begin{cases} \text{if node is red,} \\ \quad 1 + \max(\text{Len-root}[\text{node.left}], \\ \quad\quad \text{Len-root}[\text{node.right}]) \\ \text{else } 0 \end{cases}$$

Max independent set on binary trees

Setting: Given a binary tree T, choose most # vertices s.t. no edge has both vertices chosen.

What's the "either/or" "cover all cases" observation?

Either use a vertex or not

Have DP tree instead of table.

Can we do the obv thing:

A[node] = max independent set in subtree rooted at node

?

What happens if we try writing a recurrence?

$$A[node] = \max\Big(A[node.left] + A[node.right],$$

(not taking node)

$$1 + A[node.left.left] + A[node.left.right] + A[node.right.left] + A[node.right.right] \Big)$$

Take node, cannot take children, so only consider grandchildren

$O(n)$ time

Each vertex's work = $O(\#\text{children} + \#\text{grandchildren})$ $\Rightarrow$ sums to $O(\text{tree size}) + O(\text{tree size})$

Correctness:

Either take node or not.

Not take node: Any such soln can be interpreted as union of a soln on left subtree & a soln on right subtree.

Conversely, any pair of solns for the left and right subtrees union to a soln w/out using node.

So might as well take max of both subtree solns.

Take node: If we take node, then any such soln cannot take any child of node.

So any soln is of the form node $\cup$ {ind set on subtrees rooted at grandchildren}

Similarly to above, all such form of soln is an independent set for tree rooted at node.

So take max.

Can you generalize this to trees that aren't binary?

Alternative DP alg:

A[node, can-take] (can-take: bool)

≔ If can-take, then max ind set size for subtree rooted at node.

If not can-take, then max ind set size for subtree rooted at node that does not incl node.

A[node, 0] = A[node.left, 1] + A[node.right, 1]

A[node, 1] = max(A[node, 0],

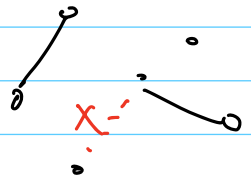
1 + A[node.left, 0] + A[node.right, 0])

Similar correctness analysis as prev alg.

Max matching in binary trees

Def A matching (in a graph) is a subset of edges s.t. each vertex is incident to at most one edge in the matching.

Compute a max size matching in O(n) time for binary trees



Observation: Either take edge or not

DP table?

M[node, can_take] = Max matching size in subtree rooted at node

M[node, 0] = Max matching size in subtree that does not take an edge incident to node.

Recurrence:

$$M[node, 0] = M[node.left, 1] + M[node.right, 1]$$

$$M[node, 1] = \max\big(M[node, 0],\ 1 + M[node.left, 0] + M[node.right, 1],\ 1 + M[node.right, 0] + M[node.left, 1]\big)$$

don't take edge

take edge to left child

take edge to right child

Correctness:

M[node, 0]: every soln is a union of matching in left+right subtrees, and every such union is a soln for the optimization for M[node, 0]. So we take max.

M[node, 1]: either take an edge to a child or don't. If not, best soln is M[node, 0]. If yes, either left or right edge.

If take left edge, then any soln must not take any other edge for left child, so w/ similar reasoning before the best soln taking node → node.left is
1 + M[node.left, 0] + M[node.right, 1].

Same argument if taking node → node.right edge //