Week 3 Recitation

1 This Week's Recap

1. Tree DP
   (a) How to approach
      i. Have states be nodes
      ii. Have base cases be leaves
      iii. Have transitions be relations from parents to children
   (b) Height of all vertices in a tree in $O(n)$ time
      i. The height of a vertex is the length of the longest path from it to another vertex in its subtree
      ii. States: $DP[i]$ = height of vertex i
      iii. Transition:

$$DP[i] = \begin{cases} 0 & \text{if vertex } i \text{ is a leaf} \\ 1 + \max_{j \in \text{children}(i)} DP[j] & \text{otherwise} \end{cases}$$

   (c) Maximum weighted unrelated subset of size k
      i. Given a binary tree with weighted nodes, find the maximum weight of a set of k nodes such that none are ancestors/descendants of each other.
      ii. States: $DP[i][x]$ = the maximum value achieved by selecting x nodes from the subtree rooted at i
      iii. Transition (assume children$(i) = \{j_1, j_2\}$):

$$DP[i][x] = \begin{cases} 0 & x = 0 \\ w_i & x = 1 \text{ and } i \text{ is a leaf} \\ \max\left(w_i, \max_{0 \le x_1, x_2, x_1 + x_2 = x} DP[j_1][x_1] + DP[j_2][x_2]\right) & x = 1 \text{ and } i \text{ is not a leaf} \\ \max_{0 \le x_1, x_2, x_1 + x_2 = x} DP[j_1][x_1] + DP[j_2][x_2] & \text{otherwise} \end{cases}$$

      This runs in $O(nk^2)$ time.
      iv. Final answer is $DP[\text{root}][k]$.
2. Range Tree Prefix/Suffix Queries

(a) How to Approach

   i. Define a tree whose leaves represent individual entries, and internal nodes define ranges over elements within its subtree

   ii. Define an associative operation (eg. sum, max, min) over the values for each node within its subtree

   Note: when the answer extracted may not be unique, it's also possible for the operations to be non-associative. For example, '10 arbitrary elements in the range corresponding to the subtree'.

(b) General operations in the tree

   i. Update: locate the required entry, and update its value, along with that of all its ancestors

   ii. PrefixOp: To compute a value over range $[1,...,k]$, walk upwards from the leaf corresponding to k, combine everything to the left of the path (as well as k if the range is inclusive, as it is here).

(c) Dynamic Prefix Sum Under Entry Modification

   i. Tree stores original array values in leaves, and internal nodes represent sum of leaves within its subtree

   ii. Combine operation: add together sums of left and right children to get value of current node.

(d) Inversion Counting

   i. Tree represents array $V[i]$ = total count of value i as we iterate through the original array, and internal nodes represent sum over subtree.

   ii. Merge operation: Sum

   iii. Intuition: as we iterate through the array at index i, we want to count the number of previous elements that is larger than it. This is a suffix sum on the frequency array of the elements encountered.

(e) Weighted LIS: given length n array $A[1\dots n]$, along with weights $w[1\dots n]$, find increasing subsequence $i_1 < i_2 < \dots < i_k$ ($A[i_1] < A[i_2] < \dots < A[i_k]$) with maximum total weight $\sum_{j=1}^{k} A[i_j]$).

   i. Dynamic program is $DP[i] = w[i] + \max_{j<i, A[j]<A[i]} DP[j]$.

   ii. Tree encodes array $B[1\dots n]$ where $B[x] = \max_{j<i:A[j]=x} DP[j]$. Tree supports entry modify and querying prefix maximum.

   iii. Algorithm: go through A in increasing order of i. For each $A[i]$, query for

$$\max_{x<A[i]} B[x]$$

which is the same as the max of $DP[j]$ for all j encountered so far with $A[j] < A[i]$. Adding $w[i]$ to this gives $DP[i]$, which we then update $B[A[i]]$ with.

iv. Example of the B array at various points of time with the input

$$A = \langle 34, 23, 74, 52, 54, 43, 88, 23, 19 \rangle,$$
$$w = \langle 8, 3, 2, 9, 3, 1, 5, 7, 2 \rangle.$$

First, we relabel the elements of A by their positions in the sorted list of unique elements. This turns A into

$$\widehat{A} = \langle 3, 2, 7, 5, 6, 4, 8, 2, 1 \rangle$$

while preserving how all pairs of i and j compare. Then processing things in order of i, with B as defined above, gives

i	$(\widehat{A}[i], w[i])$	DP[i]	B at end of this step
init		-	$<0,0,0,0,0,0,0,0>$
1	(3,8)	8	$<0,0,8,0,0,0,0,0>$
2	(2, 3)	3	$<0,3,8,0,0,0,0,0>$
3	(7, 2)	10	$<0,3,8,0,0,0,10,0>$
4	(5, 9)	17	$<0,3,8,0,17,0,10,0>$
5	(6, 3)	20	$<0,3,8,0,17,20,10,0>$
6	(4, 1)	9	$<0,3,8,9,17,20,10,0>$
7	(8, 5)	25	$<0,3,8,9,17,20,10,25>$
8	(2, 7)	7	$<0,7,8,9,17,20,10,25>$
9	(1, 2)	2	$<2,7,8,9,17,20,10,25>$

Note that handling the duplicate element in this setup is easy. We just overwrite the previous value if the new value ($\max_{j \in [1 \ldots i-1]} B[j]$+weight) is greater than the current value to preserve the invariant that $B[i]$ is the max weighted LIS of the order ith element in A.

2 One More Problem Using Range Queries

Max weighted non-overlapping intervals: given intervals $[l_1, r_1] \ldots [l_n, r_n]$, with associated weights $w_1 \ldots w_n$, find the maximum weight of a subset of them that have no overlap (endpoints touching also counts as overlap).

Solution sketch: Since only the relative ordering of the $2n$ end points matter, we can sort them and reduce their values to integers in the range $[1, 2n]$.

Also, sort the intervals in order of their right end points, aka. $r_1 \leq r_2 \leq r_3 \leq \ldots \leq r_n$.

Define DP state $DP[i]$ to be the maximum weight of a subset of non-overlapping intervals with $[l_i, r_i]$ as the rightmost one.

Then the DP transition is

$$DP[i] = w[i] + \max_{j<i, r[j]<l[i]} DP[j]$$

and the base case is $Dp[i] \geq w_i$.

To perform the transition efficiently, at each time, maintain an array $B[1 \ldots 2n]$ that stores for each right end point the max DP value of an interval ending at it.

Aka, at each point i, we want to have

$$B[x] = \max_{j<i, x=r[j]} DP[j]$$

Maintain this using a tree that supports prefix max queries. Then as we go through the is, we compute $DP[i]$ using a prefix max query on $B[1 \ldots (l_i - 1)]$ (plus W_i as in the transition), and then updates $B[r_i]$ with $DP[i]$. Both of these operations take $O(\log n)$ time each, which gives $O(n \log n)$ over all i locations.