Recitation #9

Divide & Conquer w/ Dynamic Programming

Review: Goodness Trees each node has value (any integer)

\[ GN(v) = \begin{cases} 
    v.val + 2 \sum_{\text{w child of } v} GN(w) & \text{if } v \neq \text{nil} \\
    0 & \text{if } v = \text{nil} 
\end{cases} \]

For binary tree:

\[ GN(v) = \begin{cases} 
    0 & \text{if } v = \text{nil} \\
    v.val + 2GN(v.left) + 2GN(v.right) & \text{otherwise} 
\end{cases} \]

Given n values in Data[1:n], what tree maximizes GN(T) when want these values in pre/in/post order?

Recall from class: binary tree, inorder data

\[ \text{InBest}(i, j) = \begin{cases} 
    0 & \text{if } i > j \\
    \max_{i \leq k \leq j} 2 \text{InBest}(i, k) + \text{Data}[k] + 2 \text{InBest}(k+1, j) & \text{o.w.} 
\end{cases} \]
Also recall from class: general trees, preorder data

pre-order \implies \text{first value is root!}
& values in all but rightmost subtree
are consecutive

What is the first index (call it $k$) in rightmost subtree?

\implies \text{consider all } k \text{ and choose one that gives max } GN.

$\text{RegenBest}(i,j) = PGB(i,j) = \max \text{ GN of any tree with values Data}[i,j].$

\begin{align*}
= \begin{cases} 
\text{Data}[i] & \text{if } i = j \\
\max_{i < k \leq j} \text{PGB}(i, k-1) + 2\text{PGB}(k, j) & \text{otherwise}
\end{cases}
\end{align*}
Problem: find binary tree maximizing EN with data in preorder

Data[]

Consider smallest index k in right subtree

left subtree contains

Data[2..k-1]

right subtree contains

Data[k+1..n]

PreBinBest(i,j) =

\[
\begin{cases}
0 & \text{if } i > j \\
\max_{1 \leq k \leq j} 2\text{PreBinBest}(i,k-1) + \text{Data}(i) + \text{PreBinBest}(k+1,j) & \text{otherwise}
\end{cases}
\]

How to implement?

- look-up table L[0..n, 0..n] to avoid computing some subproblems
- choice table C[1..n, 1..n] to record index k that maximizes PreBinBest(i,j)
- can reconstruct optimal tree from C

L, C have \(O(n^2)\) entries, computing each one takes time \(O(n)\)

\[
\Rightarrow \text{total: } O(n) \cdot O(n^2) = O(n^3)
\]

Recall from class: best binary search tree given elements \(e_1, e_2, \ldots, e_n\) (in order) accessed \(r_1, r_2, \ldots, r_n\) times.
each time, cost of accessing $e_i$ = # nodes in path from root to $e_i$ (inclusive)

Consider root value (say $e_k$)

left subtree contains $e_1, \ldots, e_{k-1}$

right subtree contains $e_{k+1}, \ldots, e_n$

\[
\text{Cost}(i,j) = \begin{cases} 
0 & i > j \\
\max_{i \leq k \leq j} \text{Cost}(i,k-1) + \sum_{k=i}^{j} r_k + \text{Cost}(k+1,j) & \text{i \leq k}
\end{cases}
\]

the root adds 1 to the cost of each node, every time it is accessed!

Problem: same but all elements $e_1, \ldots, e_n$ must be stored at the leaves.

How to find a value? Use guides in internal nodes as in 2-3 trees.
all leaves are consecutive \( e_1, \ldots, e_n \)

What is largest index in left subtree? Say \( k \).

\[
\text{Cost}(i, j) = \begin{cases} 
0 & \text{if } i > j \\
\min_{1 \leq k \leq j} \text{Cost}(i, k) + \sum_{h=i}^{k} r_h + \text{Cost}(k+1, j) & \text{if } i \leq j 
\end{cases}
\]

How to implement?

- Lookup table \( L[0:n, 0:n] \) to avoid computing same subproblems.
- Choice table \( C[1:n, 1:n] \) to record index \( k \) that maximizes Cost\((i, j)\).
  can reconstruct tree (leaves & internal guides) from \( C \).

\( L, C \) have \( O(n^2) \) entries, computing each takes time \( O(n) \)

\[
\text{total} = O(n) \cdot O(n^2) = O(n^3)
\]