1 577 Discussion 11 October 2019

Questions ?

Have students submitted placeholder files for Q's 1-3 on midterm ?

↳ Encourage all to do so.

Divide and Conquer write-ups

- Describe your algorithm
  - Specification (Can reference the problem statement, but otherwise be explicit)
    ↳ if distinct from the problem statement, add a sentence explaining how to solve the original problem
  - Algorithm
    - High-level pseudocode works best, but prose is OK too.
    - Avoid low-level pseudocode and imprecise prose.
    - Use algorithms from class to do heavy-lifting when possible.
      (If/when you do, tell us it's from class.)
    - Organize your code to make proving correctness as painless as possible.

- Prove correctness
  - Use induction to match a recursive algorithm.

    Base Case ⟷ Base Case ⇝ Estimated 1 sentence
    Ind. Step ⟷ Rec. Case ⇝ 4-6 sentences
    Ind. Hyp. ⟷ "Recursive calls are correct"

  (Understand that you are proving that the implementation computes the specification.)

- Analyze Running Time.
  Recursion tree method:
  - Shape
  - Work per node
  - Add it all up (should get geometric series)

D&C Write-up Example : HW1 #3

(Given a complete binary tree with #s on the leaves, compute the minimum number of inversions possible when swapping at internal nodes of the tree.)

Algo

INPUT : A list of 2^k numbers (the leaves' numbers in order), A

OUTPUT : (i) the minimum # of inversions as in the problem statement

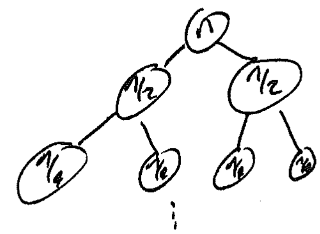
(ii) a sorted copy of A

Solve HW1#3 (A) :

- If $k=0$:
  - return $(0, A)$
- Else :
  - Let $L, R \leftarrow$ left & right halves of A.
  - Let $c_L, L' \leftarrow$ Solve HW1#3 (L).
  - Let $c_R, R' \leftarrow$ Solve HW1#3 (R).
  - Let $c_1 \leftarrow$ COUNT-CROSS (L', R')
  - Let $c_2 \leftarrow$ COUNT-CROSS (R', L')
  - Let $A' \leftarrow$ MERGE (L', R')
  - Return $\left(c_L + c_R + \min(c_1, c_2),\ A'\right)$

Solve the original problem by calling Solve HW1#3 & ignoring (ii).

Running Time Analysis



Depth: $O(\log n)$.

work per node: linear in input size

work per level: $O(n)$

Total work : $O(n \log n)$.

(COUNT-CROSS is from class.)

INPUT : two sorted arrays L, R

OUTPUT : # inversions in concat. LR

(MERGE is from class.)

INPUT : two sorted arrays L, R

OUTPUT : ~~#~~ sorted copy of their concatenation

Correctness : By induction on k.

Base case ($k=0$) : There are no swaps possible, so the answer is 0 inversions and A is already sorted.

Ind. step ($k>0$) : Each ~~inversion in~~ potential inversion in A (pair of positions i, j with $i<j$) either has i, j pointing into L, i, j pointing into R, or ~~i,j pointing~~ i pointing into L and j pointing into R.

We minimize all three independently. The first by recursing on L, the second by recursing on R, and the latter by choosing whether to swap at the root. By the inductive hypothesis, c_L and c_R are the first two counts, and L' and R' are sorted copies of L and R. By correctness of COUNT-CROSS, $\min(c_1, c_2)$ is the minimum number of the third type. By correctness of MERGE, A' is a sorted copy of A.

(3)

Dynamic Programming Writeups

- Describe your algorithm.
  - Specify subproblems ("OPT(i,j) ≐ the smallest ... ")
  - Say how to solve original problem.
  - Give a recurrence ("OPT(i,j) = min{ ... ")
    - Don't forget base cases
  - Give a sentence or two about implementation
    - "Implement recursively with memoization"
    - If giving an iterative solution, explain the order of iteration, and how to save space (if necessary).
    - Use pseudo code as a last resort (but prefer it to unclear prose)

→ Avoid words like "current" or other references to context (except globally fixed quantities). Specs should be standalone.

- Prove correctness.
  - Use induction to match recursive algorithm.

    Base case ⟷ Base case ⇝ Est. 1-2 sentences

    Ind. step ⟷ Recursive case ⇝ Est. 3-4 sentences.

    Ind. Hyp. ⟷ "Recursive calls are correct"

  - The inductive step almost always has this format:

    " The optimal solution is ___, ___, or ___.

    ~~Among the first case,~~

    Among solutions of the first type, the best is ______.

    " " " " second " " " ______.

    ⋮

    The recurrence picks the best among these. "

- Resource ~~Running~~ Time Analysis:

  Time ≤ (# distinct subproblems) · (local work per subproblem)

  ~~Space ≤ (# distinct subproblems) · (space to store an answer) + (local worst-case space required per subproblem)~~

  Space: Depends on recurrence. Only keep cells in table as long as needed.

④ D.P. Write-up Example: HW 3 #3

(Given a seq. of books & shelf-width w, find min height of shelving required to store all the books in order.)

$OPT(i) \doteq$ Minimum height of shelving required to store books $i+1 \cdots n$.

(We want to know $OPT(0)$.)

Base case: $OPT(n) = 0$.

Recurrence: $(i<n)$

$$OPT(i) = \min_{\substack{j:\ i+1 \le j \le n \\ \sum_{i+1 \le k \le j} t_k \le w}} \left(\left(\max_{i+1 \le k \le j} h_k \right) + OPT(j) \right)$$

Implement recursively using memoization. OR Build a table $OPT[0 \cdots n]$ ~~matching~~ s.t. $OPT[i] = OPT(i)$, by starting with $i=n$ and working downward.

~~Evaluate the recurrence in $O(n)$ time by trying j in order of $i+1$, $i+2$, …, keeping track of the maximum height of a book seen so far and the total thickness of books seen so far.~~

Evaluate the recurrence in $O(n)$ time by trying j in order of $i+1, i+2, \cdots$, keeping track of $\max_{i+1 \le k \le j} h_k$ and $\sum_{i+1 \le k \le j} t_k$ along the way. (Each can be updated in $O(1)$ time as $j \to j+1$.)

<u>Correctness</u> By induction on i (from $i=n$ down)

Base case: $i=n$. No books to stack $\Rightarrow$ Zero height is optimal.

Ind. step: $i<n$. The optimal solution places books $i+1 \cdots j$ on the first shelf, for some $i+1 \le j \le n$ with $\sum_{i+1 \le k \le j} t_k \le w$.

For fixed j, the optimal solution putting books $i+1 \cdots j$ on the first shelf has height $\left(\max_{i+1 \le k \le j} h_k \right) + OPT(j)$, by the inductive hypothesis.

So ~~the value~~ since the recurrence selects the minimum of these quantities over all j, it is correct.

<u>Resources</u> #subproblems $= O(n)$. Work/subproblem $= O(n)$. Total time: $O(n^2)$.

Space: $O(n)$ for the table/memoization.