ALGORITHM DESIGN USING DIVIDE & CONQUER METHOD: I

Partha P Chakrabarti

Indian Institute of Technology Kharagpur

Algorithm Design by Recursion Transformation

- Algorithms and Programs
- Pseudo-Code
- Algorithms + Data Structures = Programs
- Initial Solutions + Analysis + Solution Refinement + Data Structures = Final Algorithm
- Use of Recursive Definitions as Initial Solutions
- Recurrence Equations for Proofs and Analysis
- Solution Refinement through Recursion Transformation and Traversal
- Data Structures for saving past computation for future use

1. Initial Solution
   a. Recursive Definition – A set of Solutions
   b. Inductive Proof of Correctness
   c. Analysis Using Recurrence Relations
2. Exploration of Possibilities
   a. Decomposition or Unfolding of the Recursion Tree
   b. Examination of Structures formed
   c. Re-composition Properties
3. Choice of Solution & Complexity Analysis
   a. Balancing the Split, Choosing Paths Divide & Conquer
   b. Identical Sub-problems
4. Data Structures & Complexity Analysis
   a. Remembering Past Computation for Future
   b. Space Complexity
5. Final Algorithm & Complexity Analysis
   a. Traversal of the Recursion Tree
   b. Pruning
6. Implementation
   a. Available Memory, Time, Quality of Solution, etc

Structure of Recursive Definition

$f(X)$

{ 1. Base Condition $B(X)$

If $B(X)$ return $(J(X))$

2. Decomposition $D(X)$

$\langle X_1, X_2, \ldots, X_K \rangle = D(X)$

3. Recursive Call

for each i, $Y_i = f(X_i)$

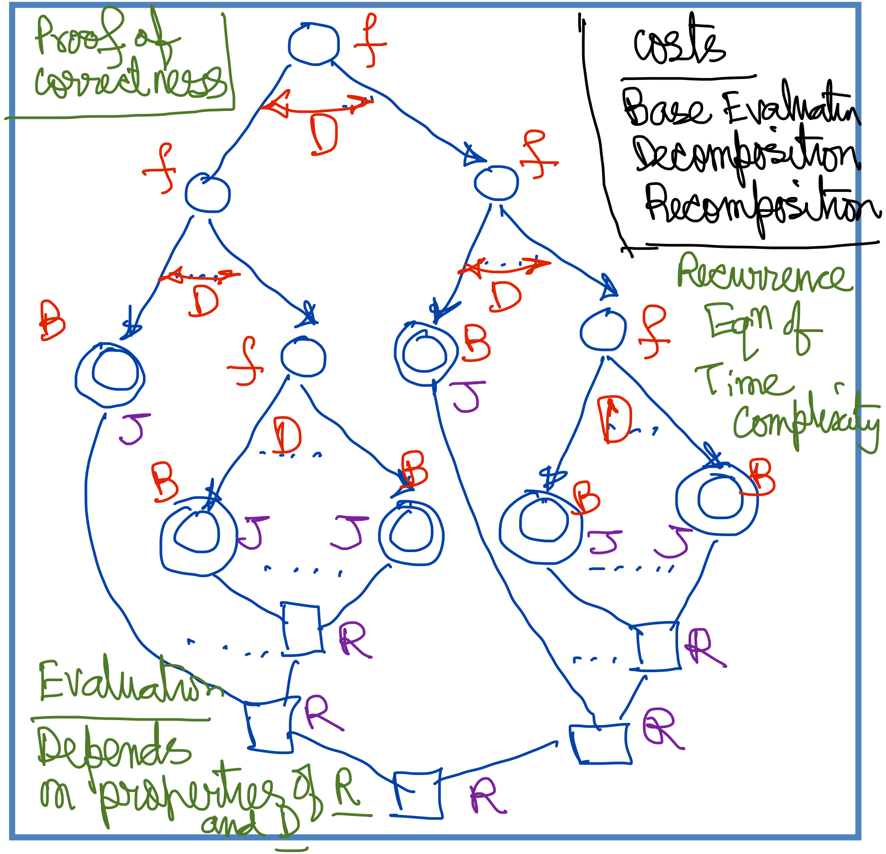
4. Recomposition $R(Y)$

$J = R(Y_1, Y_2, \ldots, Y_K)$

5. Returning the Result

Return (J)

}



Examples

Max (L)

B: $|L| = 1$

D: split of L

R: maximum of two elements

costs:

B: $O(1)$

D: depends of how we store L and where we decide to split $O(1)$

R: $O(1)$

↳ associative & commutative

Fib

B: $n=0$, $n=1$

D: $n-1$, $n-2$

R: $+$

Coins

B: {} NULL and $|L|=0$ or 1

D: Move one element at a time from list T to S

R: minimum

alternative

D': inclusion / exclusion method

Basics of Divide & Conquer Method

1. Mechanism of Decomposition and the CHOICE POINTS

$$T(n) = T(n_1) + T(n_2) + \dots + T(n_k) + \text{decomposition}(n) + \text{recomposition}(k)$$

various alternatives of D and R

properties of D and R

↳ choice points: how to optimally split the problem

2. Evaluation

3. Dynamic Programming

↳ solving identical subproblems once and memoizing the past solutions

4. Data Structuring

Organizing data of past computation for future use

5. Branch & Bound:

Pruning of unnecessary paths in the computation of the recursion tree

Basics of Divide & Conquer Method

CHOICE OF D (Decomposition) and R (Recomposition)

→ Time Complexity

→ optimize this time complexity

Analysis

1. By analysis of the recurrence eqn formed

2. By Analysis of the Recursion Tree and Costs (Computation) incurred during D and R

→ split of D
(max, max-min, max-2^{nd} max)

(coins)

3. what structures are formed which enable us to develop alternative insights for an iterative solution to the recursive defn

Sorting & Searching Problems

Searching

1. Unordered Set or List
2. Ordered List or Sequence
3. A combination
4. Static / Dynamic

Sorting

1. Ascending or Descending Order
2. $\boxed{\text{In-Memory Sorting}}$
3. Secondary storage Sorting

Searching an Unordered List

Search-U (L, x)

$\{$ Let $L = \{s_1, s_2, \ldots, s_n\}$

B

if $(|L| = 0)$ return (false)

if $(|L| = 1)$

if $(s_1 = x)$ return (true)

else return (false)

D

split L into non-empty L_1, L_2

R

OR

if (Search-U (L_1, x))

return (true)

else return (Search-U (L_2, x))

$\}$

1. $T(n) = T(k) + T(n-k) + O(1)$

$= \Theta(n)$

Choice: Any split is equal in complexity

$\hookrightarrow$ choose $\underline{1}, \underline{n-1}$

Solution:

An iterative Loop

for $i = 1$ to n do

$\{$ if $(x = s_i)$ return (true)

$\}$ return (false)

Searching an Unordered List: Finalization

Finalization

1. choice
2. check whether this is optimal
   → unless every element is checked in the WORST CASE we may not find the correct answer by any algo.

VARIANT

1. SearchU(L, S) ✓
   where we have
   $L = \{l_1, l_2, \ldots, l_n\}$
   $S = \{s_1, s_2, \ldots, s_m\}$

Both unordered and we have to return the elements of S that are in L.

$\{L \cap S\}$ → how do we solve this problem

Searching Ordered Lists

search_O (L, x)
$\{$ $L = \{s_1, s_2, \ldots, s_n\}$
$[s_i \leq s_j$ for $j > i]$

B: If $|L| = \cancel{1}\,\emptyset$ return (False, ∞)

D: choose s_i

if $(s_i = x)$ return (true, i)
else
if $(s_i < x)$
$\{$ $L' = \{s_{i+1}, \ldots, s_n\}$
$\langle y, z\rangle$ = search_O(L', x)
if (y) return (true, $z+i$)
else return (false, ∞)
$\}$
else $\{$ $L' = \{s_1, \ldots, s_{i-1}\}$

$\langle y, z\rangle$ = search_O (L', x)
if (y) return ($\langle$true, $z\rangle$)
else return (false, ∞)
$\}$
$\}$

D: choice of i

R: OR $\rightarrow$ compute only in one direction

$$T(n) = \max\{T(i-1), T(n-i+1)\} + O(1)$$

Searching Ordered Lists: Data Structure View

1 2 3 4 5

$\{7, 9, 12, 15, 18\}$ $x = 15$

choice: 9 $\boxed{2}$

<7, 4> $\boxed{3}$

$\{7\}$ $\{12, 15, 18\}$

choice 18

<7, 2>

$\{12, 15\}$ $\{\}$

choice 15 $\boxed{2}$

terminates with true

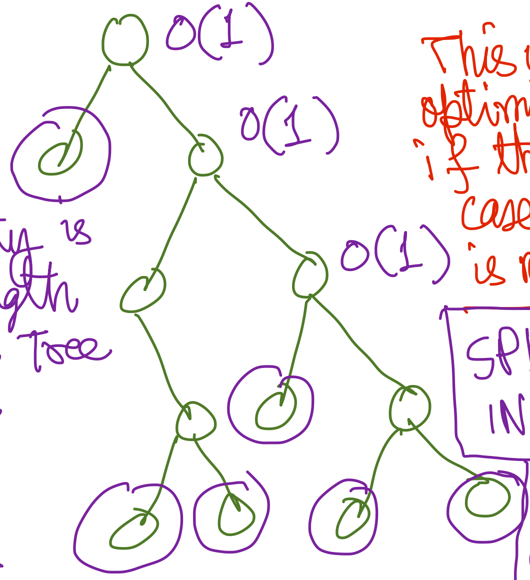
BINARY SEARCH ALGORITHM

Binary Search Trees

Algorithm Complexity

choose $i = \frac{n}{2}$

$T(n) = \max\left(T(i-1), T(n-i+1)\right) + O(1)$



complexity is the length of the Tree in the worst case

This will be optimized if the worst case length is minimized

SPLIT IN MIDDLE

choose $i = n/2$

BALANCE BINARY SEARCH TREES

Problem Variations and Approaches

$$T(n) = T(n/2) + O(1)$$
$$= \underline{O(\log n)}$$

which is the height of the Balanced Recursion Tree in the worst case

Other options

$$T(n) = \max\{T(\tfrac{1}{2}n), T(\tfrac{2}{3}n)\} + O(1)$$
$$= T(\tfrac{2}{3}n) + O(1)$$
$$= O(\log n)$$

$T(n) \to$ into 3 parts and done 2 comparison

$$\boxed{T(n) = T\left(\frac{n}{3}\right) + 2}$$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

VARIATIONS

1. Search_O (L, S) — ordered / Ordered or unordered
2. Dynamic Search ⟶ interleaved queries

(insert into L, find in L) (insert, delete, find)

Overview of Algorithm Design

1. Initial Solution
   a. Recursive Definition – A set of Solutions
   b. Inductive Proof of Correctness
   c. Analysis Using Recurrence Relations
2. Exploration of Possibilities
   a. Decomposition or Unfolding of the Recursion Tree
   b. Examination of Structures formed
   c. Re-composition Properties
3. Choice of Solution & Complexity Analysis
   a. Balancing the Split, Choosing Paths
   b. Identical Sub-problems
4. Data Structures & Complexity Analysis
   a. Remembering Past Computation for Future
   b. Space Complexity
5. Final Algorithm & Complexity Analysis
   a. Traversal of the Recursion Tree
   b. Pruning
6. Implementation
   a. Available Memory, Time, Quality of Solution, etc

1. Core Methods
   a. Divide and Conquer
   b. Greedy Algorithms
   c. Dynamic Programming
   d. Branch-and-Bound
   e. Analysis using Recurrences
   f. Advanced Data Structuring
2. Important Problems to be addressed
   a. Sorting and Searching
   b. Strings and Patterns
   c. Trees and Graphs
   d. Combinatorial Optimization
3. Complexity & Advanced Topics
   a. Time and Space Complexity
   b. Lower Bounds
   c. Polynomial Time, NP-Hard
   d. Parallelizability, Randomization

Thank you

Any Questions?