Introduction to Recursive Formulations for Algorithm Design: IV

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
   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

Coin Selection Problem

Given a set C of n coins having denomination values $\{c_1, c_2, \dots, c_n\}$ and a desired final value of V, find the minimum number of coins to be chosen from C to get an exact value of V from the sum of denominations of the chosen subset

example: $C = \{8, 6, 5, 2, 1\}$, $V = 11$

$S_1 = \{8, 2, 1\}$, $\underline{S_2 = \{6, 5\}}$

↑ minimum

Coins (S, T, x, z, n)

S: set of coins selected till now

T: remaining set of coins from which we can select

x: value of set S

z: remaining value desired to be chosen from T

n: The number of coins selected

$\boxed{\text{coins}(\text{NULL}, C, 0, V, 0)}$

→ Base condition

→ Recursive condition

First Recursive Definition

$\langle P, d \rangle = coins(S, T, x, z, n)$
{ Let $S = \{s_1, s_2, \ldots, s_n\}$
$T = \{t_1, t_2, \ldots, t_m\}$

BASE CONDITIONS

if $(z = 0)$ return $(\langle S, n \rangle)$
if $(z < 0)$ return $(\langle NULL, \infty \rangle)$
if $(T = NULL)$ return $(\langle NULL, \infty \rangle)$

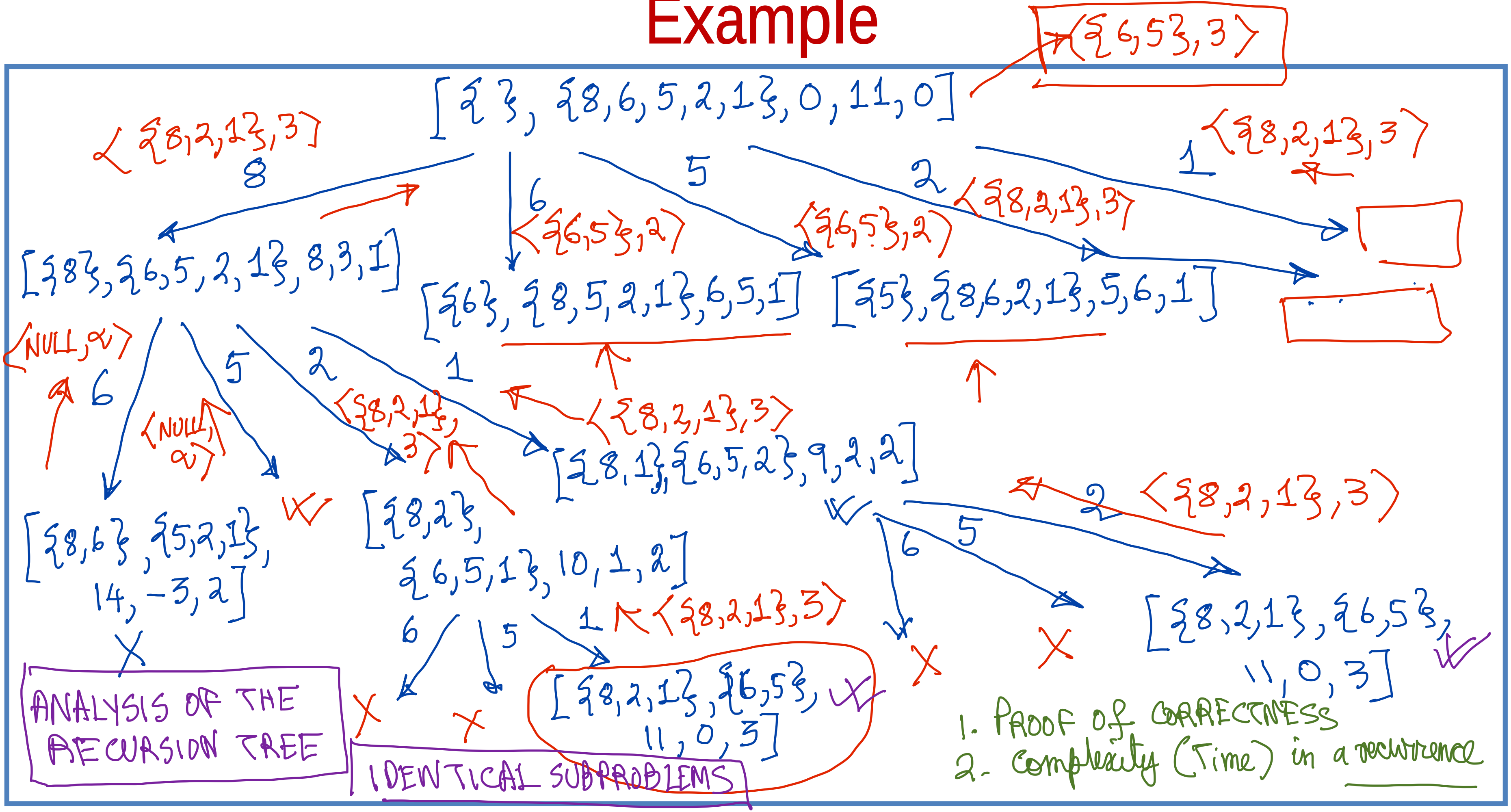
$P_{min} = NULL$
$min = \infty$

RECURSIVE CONDITION

for ($i = 1$ to m) do
{ $W = S + \{t_i\}$
$U = T - \{t_i\}$
$\langle P', d' \rangle = coins(W, U, x + t_i, z - t_i, n+1)$

if $(d' < min)$
{ $min = d'$
$P_{min} = P'$
}
}
return $(\langle P_{min}, min \rangle)$
}

Example



⟨{6,5},3⟩
[{ }, {8,6,5,2,1}, 0, 11, 0]
⟨{8,2,1},3⟩
[{8}, {6,5,2,1}, 8, 3, 1]
[{6}, {8,5,2,1}, 6, 5, 1]
[{5}, {8,6,2,1}, 5, 6, 1]
⟨{6,5},2⟩
⟨NULL, ∞⟩
[{8,6}, {5,2,1}, 14, −3, 2]
[{8,2}, {6,5,1}, 10, 1, 2]
[{8,1}, {6,5,2}, 9, 2, 2]
[{8,2,1}, {6,5}, 11, 0, 3]
ANALYSIS OF THE RECURSION TREE
IDENTICAL SUBPROBLEMS
1. PROOF OF CORRECTNESS
2. Complexity (Time) in a recurrence

Improved Recursive Definition

Instead of

$$U = T - \{t_i\}$$

we do the following

$$U = T - \{t_1, t_2, \ldots, t_i\}$$

$[\{\}, \{8, 6, 5, 2, 1\}, 0, 11, 0]$

$\xrightarrow{8}$ $[\{8\}, \{6, 5, 2, 1\}, 8, 3, 1]$

$\xrightarrow{6}$ $[\{6\}, \{5, 2, 1\}, 6, 5, 1]$

$\xrightarrow{5}$ $[\{5\}, \{2, 1\}, 5, 6, 1]$

Identical subproblems that were generated earlier will not be generated now.

1. PROOF OF CORRECTNESS
2. TIME COMPLEXITY BASED ON Recurrence Relation
3. ANALYSIS OF RECURSION STRUCTURE

Alternative Recursive Definition

$\langle P, d \rangle = \text{coins2}(S, T, x, z, n)$

$\{$ Let $S = \{s_1, s_2, \dots, s_n\}$

$T = \{t_1, t_2, \dots, t_m\}$

<u>BASE CONDITIONS</u>

If $(z = 0)$ return $(\langle S, n \rangle)$

If $(z < 0)$ return $(\langle \text{NULL}, \infty \rangle)$

If $(T = \text{NULL})$ return $(\langle \text{NULL}, \infty \rangle)$

$P_{min} = \text{NULL}$

$min = \infty$

<u>Recursive condition</u>

(Inclusion - Exclusion Principle)

$\langle P_1, d_1 \rangle = \text{coins2}(S + t_1, T - t_1, x + t_1, z - t_1, n + 1)$

$\langle P_2, d_2 \rangle = \text{coins2}(S, T - t_1, x, z, n)$

If $(d_1 \leq d_2)$

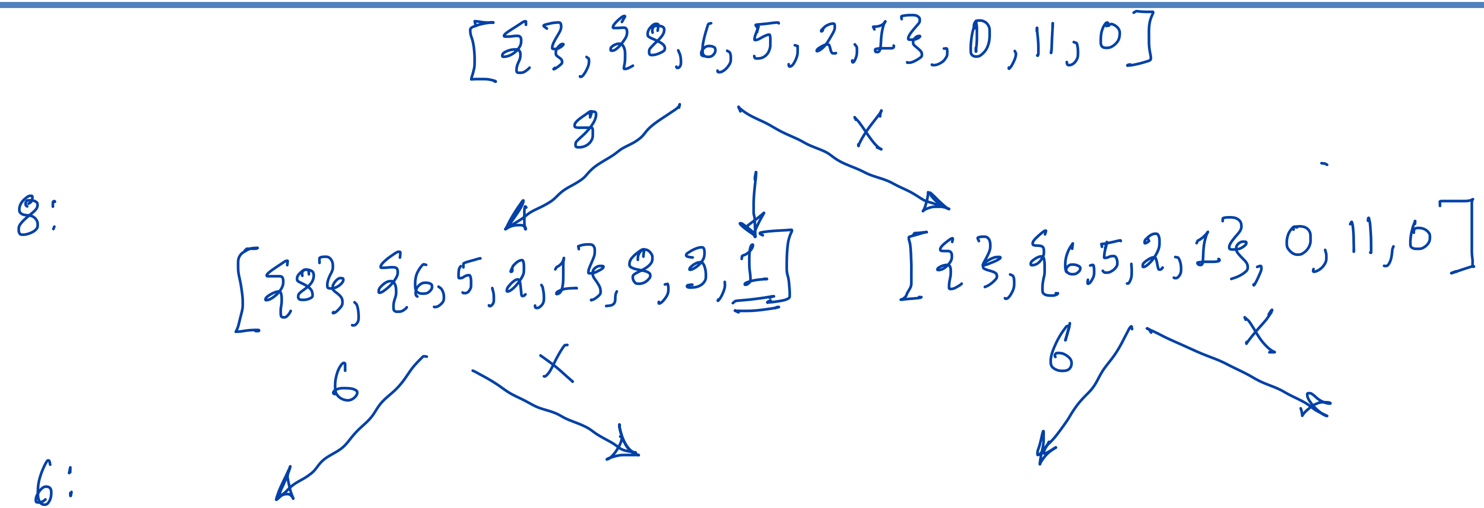
$\{P_{min} = P_1;\ min = d_1\}$

else $\{P_{min} = P_2;\ min = d_2\}$

return $(\langle P_{min}, min \rangle)$

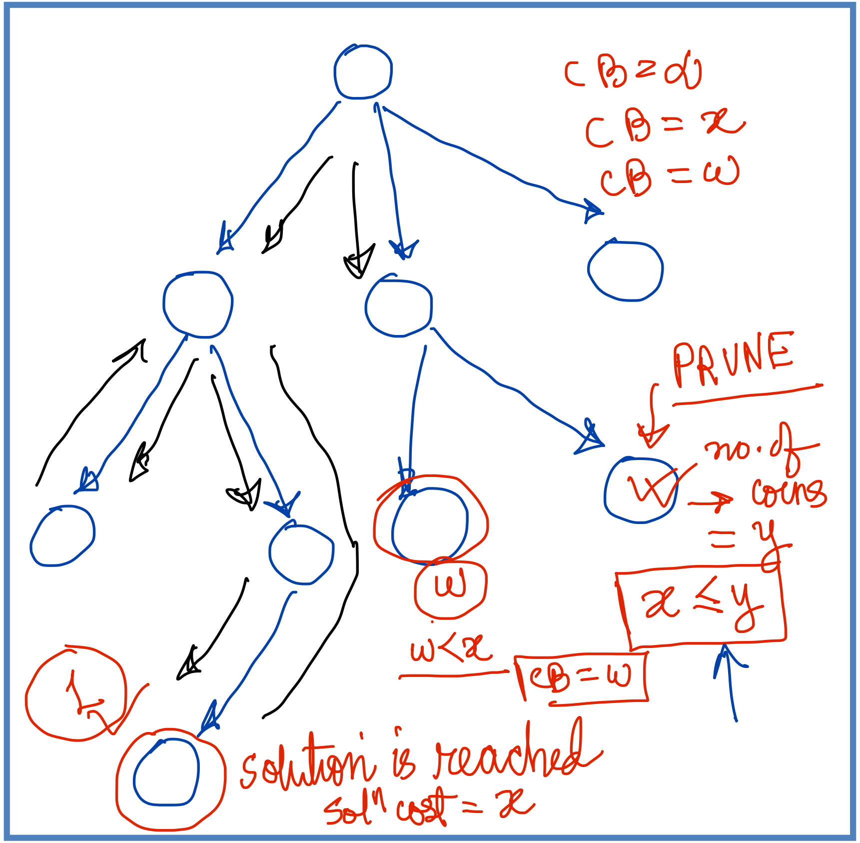
$\}$

Example



1. Inductive Proof
2. Time Recurrence
3. Identical Subproblems

Traversal and Potential Pruning



Maintain a global current best

$CB = \infty$ (initially)

Recursion is evaluated in a depth-first manner

BASE CONDITIONS are Revised for Pruning

if $(z = 0)$ { if $(n < CB)$, $CB = n$ [update the current best]
return($\langle S, n \rangle$)
}

if $(z < 0)$ return($\langle NULL, \infty \rangle$)

if $(T = NULL)$ return($\langle NULL, \infty \rangle$)

$\boxed{\text{if } (n \geq CB) \text{ return}(\langle NULL, \infty \rangle)}$

PRUNING

WORK IT OUT ON OUR EXAMPLE

Finalizing the Algorithm

2 options of Recursive Definitions

PRUNING (

→ IDENTICAL SUBPROBLEMS
ARISE

Special Case

$C = \{16, 8, 4, 2, 1\}$ $\{2^i\}$

If $V = 25$

16 8 X 4 X 2 X 1 X

choice for 16 will be part of optimal solution

$[\{16\}, \{8, 4, 2, 1\}\ 16, 9, 1]$

8 1 X 2 X 1 X

$[\{16, 8\}\ \{4, 2, 1\}, 24, 1, 2]$

4 X 2 X 1 SOLUTION

We can make a SINGLE choice from the various recursive sub-problems.

$\{100, 50, 25, 20, 10, 5, 3, 2, 1\}$

Summary

1. Initial Solution
2. ~~Analyze the Recursion~~
   (a) Balancing the split [D&C]
   (b) Identical Sub-problems (Memoization) [DP]
   (c) Choice (Greedy) from the subproblems upfront [G]
   (d) Traversal or Evaluation of the Recursion allows for pruning or pre-emption based on solutions already found. [BB]

3. Proof of correctness
4. Analysis of Complexity [Recurrence Eqns]
5. Data Structures [Asymptotic Analysis]

Problems we have examined

1. Max, Max-Min, Max1-Max2
2. FIB
3. Coins

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?