ALGORITHM DESIGN USING DYNAMIC PROGRAMMING METHOD: II

Partha P Chakrabarti

Indian Institute of Technology Kharagpur

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   Memoizations
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

1. Recursive formulation
2. Optimal substructure
3. Evaluation
   - Memoization store & Reuse
   - Top-down
   - Iterative
4. Special Data Structures

String Matching Problems

1. Pattern Matching in a Text

S : string of characters
P : string of characters
Find occurrences of P in S
(a) Exact Match
(b) Approximate match

S : a a b a c a a b a b a c a a
P : a b a b a c

S : a b a b a b a c
P : a b a

2. Sequence Alignment Problem

X : INTERACTION
Y : CONTRADICT

NTRACT   CTI   NTRAI

a) All matches of length ≥ k
b) Longest match
↳ Longest Common Subsequence

Protein Alignment

A T C G

X : C G A T A A T T G A G A
Y : G T T C C T A A T A

EDIT DISTANCE PROBLEMS

Exact Pattern Matching in a String

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

$P = \{p_1, p_2, \dots, p_m\}$

```
match (S, P)
{ if (|S| < |P|) return 0;
  if (|P| = 0) return 1;
  x = 0; y = 0
  if (s1 = p1)
     {x = match_exact(S - {s1},
                      P - {p1}))}
  y = (match (S - {s1}, P))
  return (x V y)
}
```

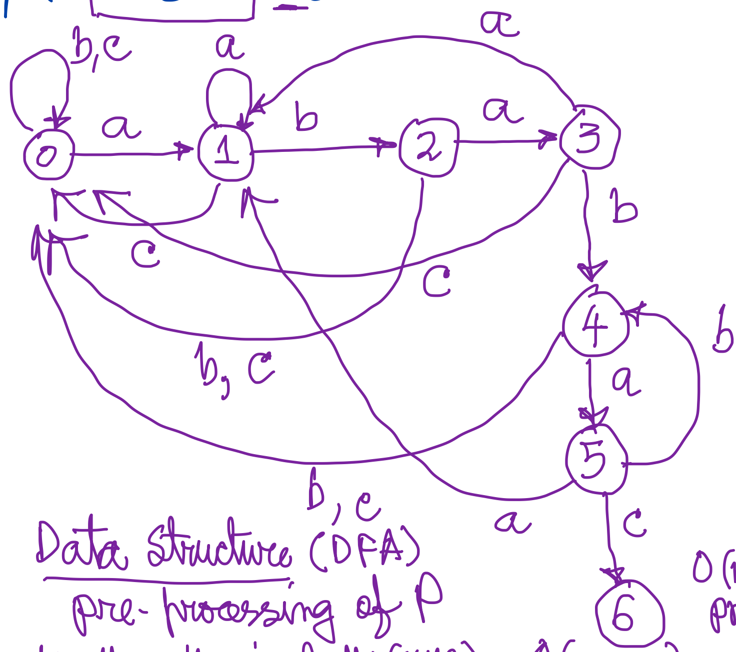
match_exact (S, P) has to be written separately

$O(n.m)$

easily converted into iteration

S: aabacaababacaa

P: ababac → b



Data Structure (DFA)

pre-processing of P

Knuth-Morris-Pratt (KMP) $O(n+m)$

$O(m)$ preprocessing

Longest Common Subsequence (LCS)

X : INTERACTION
Y : CONTRADICT
Z_1 : NTRACT ←
Z_2 : NTRAI
Z_3 : CTI

S1 : C G A T A A T T G A G A
S2 : G T T C C T A A T A
↑

a) Longest Common Subsequence

b) All Common Subsequences of length $\geq k$
or length $\geq (1-\varepsilon)$ LCS

- associative
- mutations / commutativity

More General Formulation
↳ EDIT Distance

LCS: Recursive Formulation

LCS (X, Y)
{ if (X = NULL or Y = NULL)
return (NULL);

Let $X = \{x_1, x_2, \cdots, x_n\}$
$Y = \{y_1, y_2, \cdots, y_m\}$

if $(x_1 = y_1)$
{ $Z_1 = x_1 \,||\, LCS(X - \{x_1\}, Y - \{y_1\})$
return (Z_1)
}
else
{
$Z_2 = LCS(X - \{x_1\}, Y)$; ✓
$Z_3 = LCS(X, Y - \{y_1\})$; ✓
$Z = \max(Z_2, Z_3)$
return (Z)
}
}

X: YET
Y: YES
↳ ET, ES

INTERACTION
CONTRDICT

Z_2: X = NTERACTION
Y = CONTRADICT

Z_3: X = INTERACTION
Y = ONTRADICT

LCS: Dynamic Programming

$$LCS(X_i, Y_j) = 0 \quad \text{if } i=0 \text{ or } j=0$$

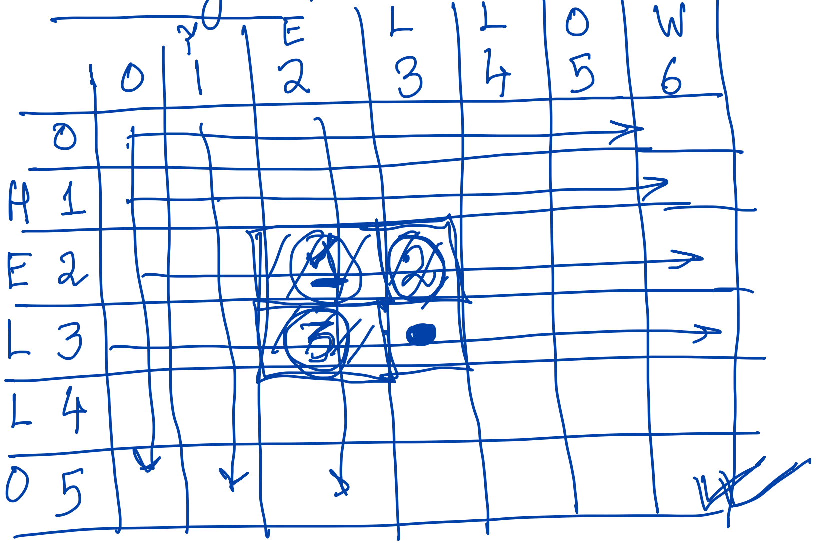
$$= LCS(X_{i-1}, Y_{j-1}) + 1 \quad \text{if } x_i = y_j \ \& \ i>0, j>0$$

$$= \max\{ LCS(X_{i-1}, Y_j), \ LCS(X_i, Y_{j-1}) \}$$

X = HELLO
Y = YELLOW

ELLO

memoize $L[i,j]$



$LCS(X_i, Y_j)$

Top-down, Bottom-up

LCS: Dynamic Programming Implementation

L[i,j]

```
LCS_iter( )
{ for (i = 1 to n)   L[i,0]=0
  for (j = 1 to m)   L[0,j]=0

  for (i = 1 to n)
    for (j = 1 to m)
    { if (X[i] = Y[j])
         L[i,j] = L[i-1,j-1]+1
      else
         L[i,j] = max{ L[i-1,j],
                       L[i,j-1] }
    }
}
```

O(n.m)

Edit Distance

X: HELLO
Y: YELLOW

spelling errors
Letter switches
Typos

General Approximate Match

S1: HELLO
S2: YELLOW

Transforming S1 to S2 using ins, del, subst

→ subst (H, Y)
→ delete (H) or insert (Y)

costs for each of
insert
delete
substitute

match : cost = 0

ex: ins : 1, del : 1, subs : 2

ins(Y), del (H)

	HELLO	(YELLOW)
(ins Y)	YHELLO	(ELLOW)
del (H)	YELLO	(ELLOW)
match	YELLO	(LLOW)
match	YELLO	(LOW)
match	YELLO	(OW)

match (YELLO_) (W)
ins(W)
YELLOW (φ)

cost 3

Edit Distance: Formulation

$d[i,0] = \sum_{k=1}^{i} del_i$

$d[0,j] = \sum_{k=1}^{j} ins_j$

$d[i,j] = d[i-1, j-1]$ if $x_i = y_j$

$$= \min \begin{cases} d[i-1,j] + del_i, \\ d[i, j-1] + ins_j, \\ d[i-1, j-1] + sub_{i,j} \end{cases}$$

costs: : ins = 1, del = 1
subst = 2

	0	Y_1	E_2	L_3	L_4	O_5	W_6
0	0	1	2	3	4	5	6
H 1	1	2	3	4	5	6	7
E 2	2	3	2	3	4	5	6
L_3	3	4	3	2	3	4	5
L_4	4	5	4	3	2	3	4
O_5	5	6	5	4	3	2	3

$O(m.n)$ time

$O(m.n)$ space ← memoized space

Edit Distance using Dynamic Programming

As a practice write down

a) Top-down algorithm with memoization

b) Bottom-up iterative algorithm

Variations

1. More operators like exchanges in rows (commutativity)

   thouhg though

2. Various kinds of approximate matches

3. Multidimensional matching or alignment

→ HASH TABLES

Dynamic Programming

1. Knapsack Problem
2. Matrix, Graph Path
3. Optimal Weighted BST

Thank you

Any Questions?