

Three Components of Dynamic Programming

1. Formulate the answer as a recurrence relation or recursive algorithm.
2. Show that the number of different instances of your recurrence is bounded by a polynomial.
3. Specify an order of evaluation for the recurrence so you always have what you need.

Approximate String Matching

A common task in text editing is string matching - finding all occurrences of a word in a text.

Unfortunately, many words can be misspelled. How can we search for the string closest to the pattern?

What are the most common mistakes in spelling?

Common Spelling Errors

Let P be a pattern string and T a text string over the same alphabet.

Differences may be:

- Corresponding characters may differ: KAT → CAT
- T is missing a character from P : CAAT → CAT
- T has added a character from P : CT → CAT
- Characters may be transposed: CTA → CAT

A 3-Approximate Match

Insertion
↓

$P = \text{unessessarily}$

$T = \text{unnecessarily}$

↑ ↑

Deletion Mutation

Finding such a matching seems like a hard problem because we must figure out where you add *blanks*, but we can solve it with dynamic programming.

un_essessarily

unne_cessarily

Applying Dynamic Programming

1. Formulate the answer as a recurrence relation or recursive algorithm.

$D(i, j)$ = The minimum number of differences between $P_1P_2P_3...P_i$ and $T_1T_2T_3...T_j$

$D(i, j)$ is the minimum of all ways to extend smaller strings...

- (a) If $P_i = T_j$ $D(i, j) = D(i-1, j-1)$ (character match!)
- (b) $D(i, j) = D(i-1, j-1) + 1$ (character differs)
- (c) $D(i, j) = D(i-1, j) + 1$ (extra character in text)
- (d) $D(i, j) = D(i, j-1) + 1$ (extra character in pattern)

Applying Dynamic Programming

2. Show that the number of different instances of your recurrence is bounded by a polynomial.

For strings of length m and n , we may need to examine all combinations of i from 0 to m and j from 0 to n . Thus there are $(m + 1)(n + 1)$ instances.

3. Specify an order of evaluation for the recurrence so you always have what you need.

Take $i = 0$ to m , then j from 0 to n .

Boundary Conditions

What should the value of $D(i, 0)$ be? What does it intuitively mean?

We are comparing the first i values of a pattern to the NULL string.

This means our only option is to do i deletions to reach the NULL string. For $D(0, i)$, we must insert i to generate the text string.

*	U	N	E	S	C	E	S	S	A	R	A	L	Y
U													
N													
E													
C													
E													
S													
S													
A													
R													
A													
L													
Y													

*	U	N	E	S	C	E	S	S	A	R	A	L	Y	
U	0	1	2	3	4	5	6	7	8	9	10	11	12	13
N	1	0	1	2	3	4	5	6	7	8	9	10	11	12
E	2	1	0	1	2	3	4	5	6	7	8	9	10	11
C	3	2	1											
E														
S														
S														
A														
R														
A														
L														
Y														

*	U	N	E	S	C	E	S	S	A	R	A	L	Y	
U	0	1	2	3	4	5	6	7	8	9	10	11	12	13
N	1	0	1	2	3	4	5	6	7	8	9	10	11	12
E	2	1	0	1	2	3	4	5	6	7	8	9	10	11
C	3	2	1	1	2	3	4	5	6	7	8	9	10	11
E	4	3	2	1	2	3	3	4						
S														
S														
A														
R														
A														
L														
Y														

*	U	N	E	S	C	E	S	S	A	R	A	L	Y	
U	0	1	2	3	4	5	6	7	8	9	10	11	12	13
N	1	0	1	2	3	4	5	6	7	8	9	10	11	12
E	2	1	0	1	2	3	4	5	6	7	8	9	10	11
C	3	2	1	1	2	3	4	5	6	7	8	9	10	11
E	4	3	2	1	2	3	4	5	6	7	8	9	10	11
S	5	4	3	3	2	2	3	4	5	6	7	8	9	10
S	6	5	4	3	3	2	3	4	5	6	7	8	9	10
A	7	6	5	4	3	4	3	2	3	4	5	6	7	8
R	8	7	6	5	4	4	3	2	3	4	5	6	7	8
A	9	8	7	6	5	5	4	3	2	3	4	5	6	7
L	10	9	8	7	6	5	6	5	4	3	2	3	4	5
Y	11	10	9	8	7	6	6	5	4	3	3	4	5	6
Y	12	11	10	9	8	7	7	6	5	4	4	3	4	5
Y	13	12	11	10	9	8	8	7	6	5	5	4	3	4

*	U	N	E	S	C	E	S	S	A	R	A	L	Y	
U	0	1	2	3	4	5	6	7	8	9	10	11	12	13
N	1	0	1	2	3	4	5	6	7	8	9	10	11	12
E	2	1	0	1	2	3	4	5	6	7	8	9	10	11
C	3	2	1	1	2	3	4	5	6	7	8	9	10	11
E	4	3	2	1	2	3	4	5	6	7	8	9	10	11
S	5	4	3	3	2	2	3	4	5	6	7	8	9	10
S	6	5	4	3	3	3	2	3	4	5	6	7	8	9
A	7	6	5	4	3	4	3	2	3	4	5	6	7	8
R	8	7	6	5	4	4	4	3	2	3	4	5	6	7
A	9	8	7	6	5	5	5	4	3	2	3	4	5	6
L	10	9	8	7	6	5	6	5	4	3	2	3	4	5
Y	11	10	9	8	7	6	6	6	5	4	3	3	4	5
Y	12	11	10	9	8	7	7	7	6	5	4	4	3	4
Y	13	12	11	10	9	8	8	8	7	6	5	5	4	3

What about transpositions?

One of the most common forms of typos is the reversal of two neighboring characters. How can we modify our algorithm to take this into account?

We add on a fourth condition: When calculating $D(i, j)$, if both $T_i = P_{j-1}$ and $T_{i-1} = P_j$ we might have a transposition. $D(i-2, j-2) + 1$ is the fourth value to compare.

Searching for Sub-strings

How can we alter this algorithm to search a longer text for the positions that optimally match a much shorter string?

- What boundary conditions do we use?
- What value do we return?
- How much memory do we use?

Sub-string Example

	*	U	N	E	S	C	E	S	S	A	R	A	L	Y
*	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	1	1	1	1	1	0	1	1	1	1	1	1	1	1
A	2	2	2	2	2	1	1	2	2	1	2	1	2	2
E	3	3	3	2	3	2	1	2	3	2	2	2	2	3
S	4	4	4	3	2	3	2	1	2	3	3	3	3	3
A	5	5	5	4	3	3	3	2	2	2	3	4	4	4
R	6	6	6	5	4	4	4	3	3	3	2	3	4	5

Example

	*	T	H	E	_	E	L	D	E	R	_	M	U	R	M	U	R	E	D	_	A	G	A	I	N
*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1
U	2	2	2	2	2	2	2	2	2	2	2	1	0	1	1	0	1	2	2	2	2	2	2	2	
R	3	3	3	3	3	3	3	3	3	2	3	2	1	0	1	1	0	1	2	3	3	3	3	3	
D	4	4	4	4	4	4	4	3	4	3	3	3	2	1	1	2	1	1	1	2	3	4	4	4	
E	5	5	5	4	5	4	5	4	3	4	4	4	3	2	2	2	1	2	2	3	4	5	5	5	
R	6	6	6	5	5	5	5	4	3	4	5	4	3	3	3	2	2	2	3	3	4	5	6	6	

Problem

Give a $O(n^2)$ algorithm to find the longest monotonically increasing sequence in a sequence of n numbers.

6 2 9 8 3 1 7 4 5

First, assume that the sequence has to be consecutive. Then assume you can take any sub-set.

The Principle of Optimality

To use dynamic programming, the problem must observe the *principle of optimality*, that whatever the initial state is, remaining decisions must be optimal with regard to the state following from the first decision.

Combinatorial problems may have this property but may use too much memory/time to be efficient.

In other words: *The details of our past solutions won't affect our current solution.*

Example: The Traveling Salesman Problem

What recurrence relation will return the optimal solution to the Traveling Salesman Problem?

If $T(i)$ is the optimal tour on the first i points, will this help us in solving larger instances of the problem?

Can't we set $T(i+1)$ to be $T(i)$ with the additional point inserted in the position that will result in the shortest path?

No!



T(4)



T(5)



Shortest Tour

A Correct Algorithm

Let $C(i, j)$ be the edge cost of moving from i to j . Further, Let $T(i; j_1, j_2, \dots, j_k)$ be the optimal tour that goes from city 1 to i in minimal time, passing through each of the k cities exactly once in any order.

The cost of the optimal tour is thus $T(1; 2, 3, \dots, n)$.

$$T(i; j_1, j_2, \dots, j_k) = \min_{1 \leq m \leq k} C(i, j_m) + T(j_m; j_1, j_2, \dots, j_{m-1}, j_{m+1}, \dots, j_k)$$

When *can't* you use Dynamic Programming?

Dynamic programming computes recurrences efficiently by storing partial results. Thus dynamic programming can only be efficient when there are not too many partial results to compute!

Since there are $n!$ permutations of an n -element set, we cannot use dynamic programming to store the best solution for each subpermutation. There are 2^n subsets of an n -element set - we cannot use dynamic programming to store the best solution for each.

So, when *can* you use it?

There are only $n(n-1)/2$ contiguous substrings of a string, each described by a starting and ending point, so we can use it for string problems.

There are only $n(n-1)/2$ possible subtrees of a binary search tree, each described by a maximum and minimum key, so we can use it for optimizing binary search trees.

Summary

Dynamic programming works best on objects that are linearly ordered and cannot be rearranged - characters in a string, files in a filing cabinet, points around the boundary of a polygon, the left-to-right order of leaves in a search tree.

Whenever your objects are ordered in a left-to-right way, you should *smell* dynamic programming!