Dynamic Programming

February 2024

Overview

Fibonacci

0 1 1 2 3 $\boxed{5}$ 8 13 21 34 . . .

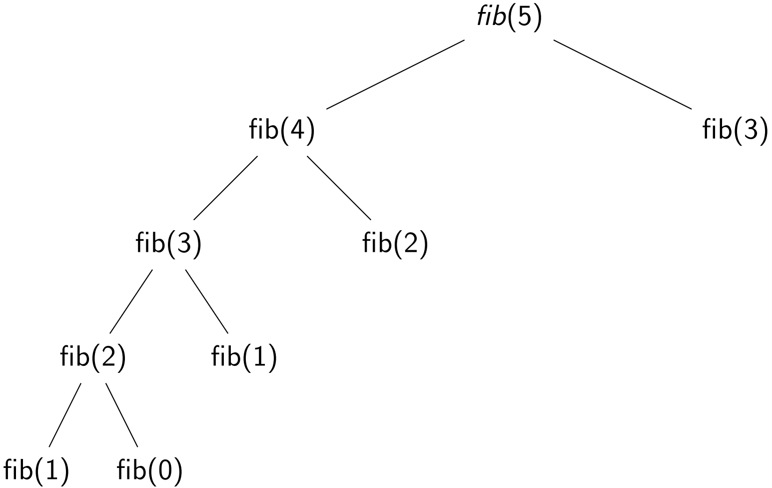
$$\text{fib}(5) = \text{fib}(4) + \text{fib}(3)$$
$$\text{fib}(5) = 3 + 2$$
$$\text{fib}(5) = \boxed{5}$$

Fibonacci

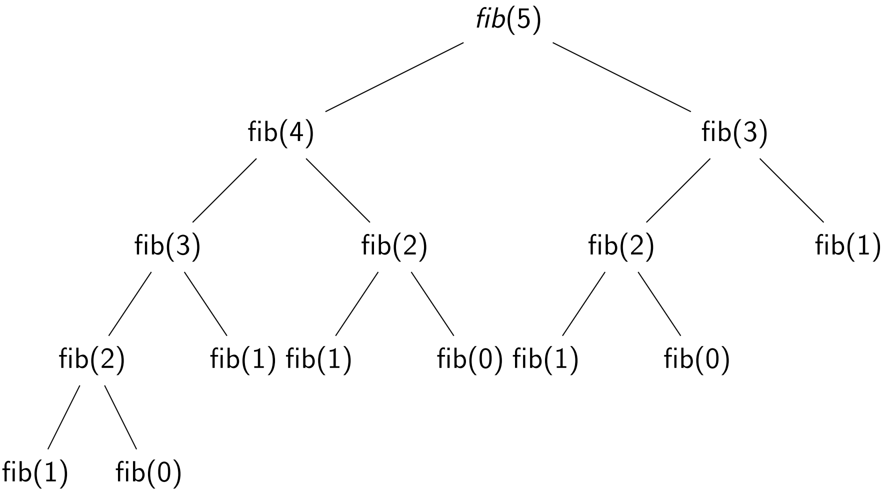
0 1 1 2 3 5 8 13 21 34 ...

```
public int fib(int n) {
   if (n <= 1) {
      return n;
   }
   return fib(n - 1) + fib(n - 2);
}
```

Fibonacci

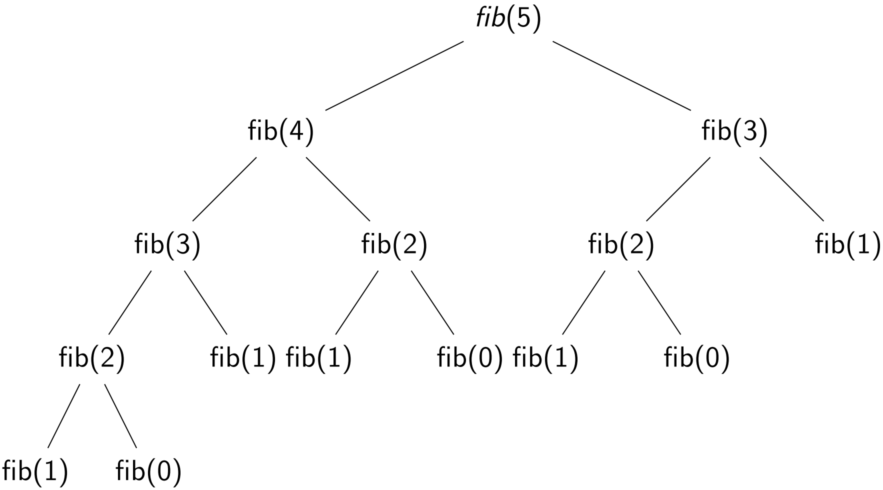


Fibonacci



Complexity?

Fibonacci



Complexity: $O(2^n)$

Can we do better?

Those who cannot remember the past are condemned to repeat it

George Santayana

What is DP?

- Dynamic Programming (shortened as DP) is a technique for solving a problem in terms of combining the solutions of sub-problems.

What is DP?

- Dynamic Programming (shortened as DP) is a technique for solving a problem in terms of combining the solutions of sub-problems.
- Original problem must have overlapping sub-problems.

What is DP?

- Dynamic Programming (shortened as DP) is a technique for solving a problem in terms of combining the solutions of sub-problems.
- Original problem must have overlapping sub-problems.

It can be used on:

- Optimization problems
- Counting problems

What is DP?

- Dynamic Programming (shortened as DP) is a technique for solving a problem in terms of combining the solutions of sub-problems.
- Original problem must have overlapping sub-problems.

It can be used on:

- Optimization problems
- Counting problems

Types:

- Top-down + memoization
- Bottom-up

Top-Down

```java
public int fib(int n, int[] memo) {
    if (n <= 1) {
        return n;
    }

    if (memo[n] != 0) {
        return memo[n];
    }

    memo[n] = fib(n - 1, memo) + fib(n - 2, memo);
    return memo[n];
}
```

Complexity: O(n)

Bottom-Up

```java
public int fib(int n, int[] dp) {
   dp[1] = 1;
   for (int i = 2; i <= n; i++) {
      dp[i] = dp[i - 1] + dp[i - 2];
   }
   return dp[n];
}
```

Complexity: O(n)

Binomial Coefficients

$$\binom{n}{k}$$

Counts the number of ways to choose k things from n possibilities.

How do we compute it?

Binomial Coefficients

$$\binom{n}{k}$$

Counts the number of ways to choose k things from n possibilities.

How do we compute it?

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Binomial Coefficients

$$\binom{n}{k}$$

Counts the number of ways to choose k things from n possibilities.

How do we compute it?

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

E.g.

$$\binom{5}{3} = \frac{5!}{3!(5-3)!} = 10$$

Binomial Coefficients

Another way of solving it is using Pascal's triangle:

$$
\begin{array}{ccccccccccccc}
 & & & & & & 1 & & & & & & \\
 & & & & & 1 & & 1 & & & & & \\
 & & & & 1 & & 2 & & 1 & & & & \\
 & & & 1 & & 3 & & 3 & & 1 & & & \\
 & & 1 & & 4 & & 6 & & 4 & & 1 & & \\
 & 1 & & 5 & & 10 & & 10 & & 5 & & 1 & \\
1 & & 6 & & 15 & & 20 & & 15 & & 6 & & 1
\end{array}
$$

Binomial Coefficients

Another way of solving it is using Pascal's triangle:

0							1						
1						1		1					
2					1		2		1				
3				1		3		3		1			
4			1		4		6		4		1		
5		1		5		10		10		5		1	
6	1		6		15		20		15		6		1
	0		1		2		3		4		5		6

Binomial Coefficients

E.g.

$$\binom{5}{3}$$

0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
5	1	5	10	10	5	1	
6	1	6	15	20	15	6	1
	0	1	2	3	4	5	6

Ketchup

Ketchup is at Rosieâs party, and luckily, he is the first to arrive, so Rosie tells him that he can take all the souvenirs that fit in his bag.
Ketchup puts a label on each souvenir indicating its weight and a numerical value representing how much he would like to have it.
Ketchup has a bag that can hold a total weight of W pounds, so he would want to choose the souvenirs to maximize the total value.

$W = 10$

weight	value
3	2
10	10
2	3
5	20

Ketchup with repetition

Ketchup is at Rosieâs party, and luckily, he is the first to arrive, so Rosie tells him that he can take all the souvenirs that fit in his bag.
Ketchup puts a label on each souvenir indicating its weight and a numerical value representing how much he would like to have it.
Ketchup has a bag that can hold a total weight of W pounds, so he would want to choose the souvenirs to maximize the total value.

$W = 10$

weight	value
3	2
10	10
2	3
5	20

$ANS = 40$

0-1 Knapsack

Given **n** items to pick from, each one with its own value v_i and weight w_i and a maximum weight **W** that the knapsack can hold. If we can either ignore or take a particular item, what will be the maximum value of items that can be carried?

0-1 Knapsack

Given **n** items to pick from, each one with its own value v_i and weight w_i and a maximum weight **W** that the knapsack can hold. If we can either ignore or take a particular item, what will be the maximum value of items that can be carried?

- with repetition: If there is unlimited quantities of each item available
- with no repetition: If we can only pick one of each item

0-1 Knapsack with repetition

$$K(w) = \text{The maximum value of a Knapsack with capacity w}$$

Subproblems?

$$K(w) = \max\{K(w), K(w - w_i) + v_i\}$$

Where: $1 \leq i \leq n$

0-1 Knapsack with repetition

K(w) = The maximum value of a Knapsack with capacity w

Subproblems?

$$K(w) = \max\{K(w), K(w - w_i) + v_i\}$$

**Only if the weight can be fit into the knapsack*

Where: $1 \leq i \leq n$

Ketchup with repetition

K(10) = The maximum value of a Ketchup's bag with capacity 10

Subproblems?

$$K(1) = \max\{K(1),\, K(1 - w_i) + v_i\}$$
$$K(2) = \max\{K(2),\, K(2 - w_i) + v_i\}$$
$$\vdots$$
$$K(10) = \max\{K(10),\, K(10 - w_i) + v_i\}$$

Where: $1 \leq i \leq 4$

Ketchup with repetition

W = 10

WEIGHT	3	10	2	5
VALUE	2	10	3	20

	0	1	2	3	4	5	6	7	8	9	10
K ->	0	0	0	0	0	0	0	0	0	0	0
w = 1	0	0	0	0	0	0	0	0	0	0	0
w = 2	0	0									

Ketchup with repetition

W = 10

WEIGHT	3	10	2	5
VALUE	2	10	3	20

	0	1	2	3	4	5	6	7	8	9	10
K ->	0	0	0	0	0	0	0	0	0	0	0
w = 1	0	0	0	0	0	0	0	0	0	0	0
w = 2	0	0	3	0	0	0	0	0	0	0	0
	⋮										
w = 6	0	0	3	3	6	20	20	0	0	0	0
w = 7	0	0	3	3	6	20	20				

Ketchup with repetition

W = 10

WEIGHT	3	10	2	5
VALUE	2	10	3	20

	0	1	2	3	4	5	6	7	8	9	10
K ->	0	0	0	0	0	0	0	0	0	0	0
w = 1	0	0	0	0	0	0	0	0	0	0	0
w = 2	0	0	3	0	0	0	0	0	0	0	0
⋮											
w = 6	0	0	3	3	6	20	20	0	0	0	0
w = 7	0	0	3	3	6	20	20	23	0	0	0
⋮											
w = 10	0	0	3	3	6	20	20	23	23	26	40

0-1 Knapsack with repetition

```java
public int knapsackRep(int[] value, int[] weight, int
    maxWeight, int N) {
  int[] K = new int[maxWeight + 1];
  for (int w = 1; w <= maxWeight; w++) {
     for (int i = 1; i <= N; i++) {
        if (weight[i] <= w) {
           K[w] = Math.max(K[w], K[w - weight[i]] + value[i]);
        }
     }
  }
  return K[maxWeight];
}
```

Complexity?

0-1 Knapsack with repetition

```java
public int knapsackRep(int[] value, int[] weight, int
    maxWeight, int N) {
  int[] K = new int[maxWeight + 1];
  for (int w = 1; w <= maxWeight; w++) {
     for (int i = 1; i <= N; i++) {
        if (weight[i] <= w) {
           K[w] = Math.max(K[w], K[w - weight[i]] + value[i]);
        }
     }
  }
  return K[maxWeight];
}
```

Complexity: O(WN)

Ketchup with no repetition

Ketchup is at Rosieâs party, and luckily, he is the first to arrive, so Rosie tells him that he can take all the souvenirs that fit in his bag.
Ketchup puts a label on each souvenir indicating its weight and a numerical value representing how much he would like to have it.
Ketchup has a bag that can hold a total weight of W pounds, so he would want to choose the souvenirs to maximize the total value.

$W = 10$

weight	value
3	2
10	10
2	3
5	20

$ANS = 25$

0-1 Knapsack with no repetition

K(w, i) = The maximum value of a Knapsack with capacity w and items i

Subproblems?

- If the weight can be fit into the knapsack:

$$K(w, i) = \max\{K(w, i-1), K(w - w_i, i-1) + v_i\}$$

Where: $1 \leq i \leq n$

0-1 Knapsack with no repetition

K(w, i) = The maximum value of a Knapsack with capacity w and items i

Subproblems?

- If the weight can be fit into the knapsack:

$$K(w, i) = \max\{K(w, i - 1), K(w - w_i, i - 1) + v_i\}$$

- If the weight cannot be fit into the knapsack:

$$K(w, i) = K(w, i - 1)$$

Where: $1 \leq i \leq n$

Ketchup with no repetition

K(10, i) = The maximum value of Ketchup's bag with capacity 10 and souvenirs i

Subproblems?

$$K(1, 1) = \max\{K(1, 1 - 1), K(1 - w_1, 1 - 1) + v_1\}$$
$$K(2, 1) = \max\{K(2, 1 - 1), K(2 - w_1, 1 - 1) + v_1\}$$
$$\vdots$$
$$K(10, 4) = \max\{K(10, 4 - 1), K(10 - w_4, 4 - 1) + v_4\}$$

Where: $1 \leq i \leq 4$

Ketchup with no repetition

W = 10					
	WEIGHT	3	10	2	5
	VALUE	2	10	3	20

	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0
7	0	0	0	0	0
8	0	0	0	0	0
9	0	0	0	0	0
10	0	0	0	0	0

Ketchup with no repetition

W = 10					
	WEIGHT	3	10	2	5
	VALUE	2	10	3	20

	0	1	2	3	4
0	0	0	0	0	0
1	0		0	0	0
2	0		0	0	0
3	0		0	0	0
4	0		0	0	0
5	0		0	0	0
6	0		0	0	0
7	0		0	0	0
8	0		0	0	0
9	0		0	0	0
10	0		0	0	0

Ketchup with no repetition

W = 10

WEIGHT	3	10	2	5
VALUE	2	10	3	20

	0	1	2	3	4
0	0	0	0	0	
1	0	0	0	0	
2	0	0	0	3	
3	0	2	2	3	
4	0	2	2	3	
5	0	2	2	5	
6	0	2	2	5	
7	0	2	2	5	
8	0	2	2	5	
9	0	2	2	5	
10	0	2	10	10	

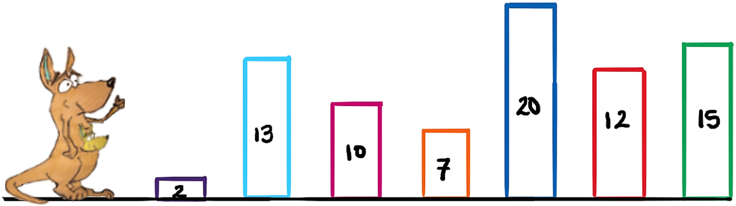
Ketchup with no repetition

W = 10	WEIGHT	3	10	2	5
	VALUE	2	10	3	20

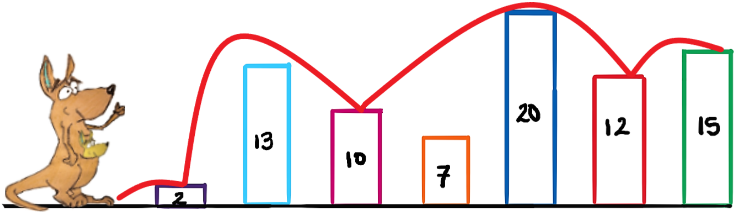
	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	3	3
3	0	2	2	3	3
4	0	2	2	3	3
5	0	2	2	5	20
6	0	2	2	5	20
7	0	2	2	5	23
8	0	2	2	5	23
9	0	2	2	5	23
10	0	2	10	10	25

Kitt the kangaroo

Kitt the kangaroo was wandering around the town and came across blocks of different heights. So she decided to jump on them; however, once she was on a block, she could only jump to a higher height block from where she was. Also, it is important to know that she can jump non-consecutive blocks. Help Kitt calculate the maximum number of jumps she can do.



Kitt the kangaroo was wandering around the town and came across blocks of different heights. So she decided to jump on them; however, once she was on a block, she could only jump to a higher height block from where she was. Also, it is important to know that she can jump non-consecutive blocks. Help Kitt calculate the maximum number of jumps she can do.



LIS:Longest Increasing Subsequence

Given a sequence of **n** elements, the longest increasing subsequence is the maximum-length sequence elements that go from left to right and where each element is larger than the previous one.

$$L(k) = \text{Longest increasing subsequence for an array of k elements.}$$

Subproblems?

$$L(k) = \max\{L(k), L(i) + 1\}$$

Where:

- $1 < i < k$
- $1 < k <= N$

LIS:Longest Increasing Subsequence

$L(k)$ = Longest increasing subsequence for an array of k elements.

Subproblems?

$$L(k) = \max\{L(k), L(i) + 1\}$$

**Only if element$_k$ is greater than element$_i$*

Where:

- $1 < i < k$
- $1 < k <= N$

$$L(k) = \text{The maximum number of jumps Kitt can do.}$$

Subproblems?

$$L(2) = \max\{L(2), L(1) + 1\}$$

$$\vdots$$

$$L(7) = \max\{L(7), L(6) + 1\}$$

Where:

- $1 < i < k$
- $1 < k <= 7$

LCS: Longest Common Subsequence

Given two sequences A and B, we want to find the length of the longest common subsequence C, where C is a subsequence of A and B.

E.g.

Input:

- backpack
- knapsack

LCS: Longest Common Subsequence

Given two sequences A and B, we want to find the length of the longest common subsequence C, where C is a subsequence of A and B.

E.g.

Input:

- b a ck p a c k
- kn a p s a c k

Output: 5

L(i, j) = Longest common subsequence for i and j.

Subproblems?

Where:

- $1 \leq i \leq A$
- $1 \leq j \leq B$

LCS: Longest Common Subsequence

$$L(i, j) = \text{Longest common subsequence for i and j.}$$

Subproblems?

$$L(i, j) = \max\{L(i - 1, j), L(i, j - 1)\}$$

Where:

- $1 \leq i \leq A$
- $1 \leq j \leq B$

LCS: Longest Common Subsequence

$$L(i, j) = \text{Longest common subsequence for i and j.}$$

Subproblems?

$$L(i, j) = \max\{L(i - 1, j), L(i, j - 1)\}$$

- If A_i is equal to B_i :

$$L(i, j) = \max\{L(i, j), L(i - 1, j - 1) + 1\}$$

Where:

- $1 \leq i \leq A$
- $1 \leq j \leq B$

LCS: Longest Common Subsequence

A = backpack
B = knapsack

			k	n	a	p	s	a	c	k
		0	1	2	3	4	5	6	7	8
	0	0	0	0	0	0	0	0	0	0
b	1	0	0	0	0	0	0	0	0	0
a	2	0								
c	3	0	0	0	0	0	0	0	0	0
k	4	0	0	0	0	0	0	0	0	0
p	5	0	0	0	0	0	0	0	0	0
a	6	0	0	0	0	0	0	0	0	0
c	7	0	0	0	0	0	0	0	0	0
k	8	0	0	0	0	0	0	0	0	0

LCS: Longest Common Subsequence

A = backpack
B = knapsack

		0	k 1	n 2	a 3	p 4	s 5	a 6	c 7	k 8
	0	0	0	0	0	0	0	0	0	0
b	1	0	0	0	0	0	0	0	0	0
a	2	0	0	0	1	1	1	1	1	1
c	3	0	0	0	1	1	1	1	2	2
k	4	0	1	1	1	1	1	1	2	3
p	5	0	1	1	1	2	2	2	2	3
a	6	0	1	1	2	2	2	3	3	3
c	7	0	1	1	2	2	2	3	4	4
k	8	0								

LCS: Longest Common Subsequence

A = backpack
B = knapsack

			k	n	a	p	s	a	c	k
		0	1	2	3	4	5	6	7	8
	0	0	0	0	0	0	0	0	0	0
b	1	0	0	0	0	0	0	0	0	0
a	2	0	0	0	1	1	1	1	1	1
c	3	0	0	0	1	1	1	1	2	2
k	4	0	1	1	1	1	1	1	2	3
p	5	0	1	1	1	2	2	2	2	3
a	6	0	1	1	2	2	2	3	3	3
c	7	0	1	1	2	2	2	3	4	4
k	8	0	1	1	2	2	2	3	4	5

Edit distance

The edit distance or Levenshtein distance between string A and B is the minimum number of edits (insertions, deletions, and substitutions) of characters in order to transform the string A into the string B.

E.g.

Input:

- CAT
- GATO

Edit distance

Input:

C	A	T	
G	A	T	O

answer $= 0$

Edit distance

Input:

C	A	T	
G	A	T	O

Substitution:

G	A	T	
G	A	T	O

answer = 1

Edit distance

Input:

C	A	T	
G	A	T	O

Substitution:

~~C~~ G	A	T	
G	A	T	O

Insertion:

~~C~~ G	A	T	O
G	A	T	O

answer $= 2$

Edit distance

$$E(i, j) = \text{Minimum number of edits to transform A into B.}$$

Subproblems?

Where:

- $1 \leq i \leq A$
- $1 \leq j \leq B$

Edit distance

E(i, j) = Minimum number of edits to transform A into B.

Subproblems?

- Cases:

A[i]	-	A[i]
-	B[i]	B[i]

Where:

- $1 \leq \mathrm{i} \leq \mathrm{A}$
- $1 \leq \mathrm{j} \leq \mathrm{B}$

Edit distance

$E(i, j) =$ Minimum number of edits to transform A into B.

Subproblems?

- Deletions:

Where:

- $1 \leq i \leq A$
- $1 \leq j \leq B$

Edit distance

$$E(i, j) = \text{Minimum number of edits to transform A into B.}$$

Subproblems?

- Deletions:

$$E(i, j) = E(i - 1, j) + 1$$

Where:

- $1 \leq i \leq A$
- $1 \leq j \leq B$

Edit distance

$$E(i, j) = \text{Minimum number of edits to transform A into B.}$$

Subproblems?

- **Deletions:**

$$E(i, j) = E(i - 1, j) + 1$$

- **Insertions:**

Where:

- $1 \leq i \leq A$
- $1 \leq j \leq B$

Edit distance

$$E(i, j) = \text{Minimum number of edits to transform A into B.}$$

Subproblems?

\- **Deletions:**

$$E(i, j) = E(i - 1, j) + 1$$

\- **Insertions:**

$$E(i, j) = E(i, j - 1) + 1$$

Where:

- $1 \leq i \leq A$
- $1 \leq j \leq B$

Edit distance

$$E(i, j) = \text{Minimum number of edits to transform A into B.}$$

Subproblems?

- **Deletions:**

$$E(i, j) = E(i - 1, j) + 1$$

- **Insertions:**

$$E(i, j) = E(i, j - 1) + 1$$

- **Substitutions:**

Where:

- $1 \leq i \leq A$
- $1 \leq j \leq B$

Edit distance

$$E(i, j) = \text{Minimum number of edits to transform A into B.}$$

Subproblems?

\- **Deletions:**

$$E(i, j) = E(i - 1, j) + 1$$

\- **Insertions:**

$$E(i, j) = E(i, j - 1) + 1$$

\- **Substitutions:**

$$E(i, j) = E(i - 1, j - 1) + 1$$

Where:

- $1 \leq i \leq A$
- $1 \leq j \leq B$

Edit distance

$$E(i, j) = \text{Minimum number of edits to transform A into B.}$$

Subproblems?

- **Deletions:**

$$E(i, j) = E(i - 1, j) + 1$$

- **Insertions:**

$$E(i, j) = E(i, j - 1) + 1$$

- **Substitutions:**

$$E(i, j) = E(i - 1, j - 1) + 1$$

- **No need to change character:**

Edit distance

$E(i, j) =$ Minimum number of edits to transform A into B.

Subproblems?

- **Deletions:**

$$E(i, j) = E(i - 1, j) + 1$$

- **Insertions:**

$$E(i, j) = E(i, j - 1) + 1$$

- **Substitutions:**

$$E(i, j) = E(i - 1, j - 1) + 1$$

- **No need to change character:**

$$E(i, j) = E(i - 1, j - 1)$$

Edit distance

E(i, j) = Minimum number of edits to transform A into B.

Subproblems?

$$E(i, j) = \min\{\ E(i - 1, j) + 1, E(i, j - 1) + 1, E(i - 1, j - 1) + val\}$$

Where:

- $1 \leq i \leq A$
- $1 \leq j \leq B$
- val is 0 if A[i] == B[i], otherwise is 1

Edit distance

$$E(i, j) = \text{Minimum number of edits to transform A into B.}$$

Subproblems?

$$E(i, j) = \min\{ E(i - 1, j) + 1, E(i, j - 1) + 1, E(i - 1, j - 1) + val\}$$

Where:

- $1 \leq i \leq A$
- $1 \leq j \leq B$
- val is 0 if $A[i] == B[i]$, otherwise is 1

Edit distance

$$E(i, j) = \text{Minimum number of edits to transform A into B.}$$

Subproblems?

$$E(i, j) = \min\{\ E(i - 1, j) + 1, E(i, j - 1) + 1, E(i - 1, j - 1) + val\}$$

Base case?

$$E(i, 0) =$$

$$E(0, j) =$$

Where:

- $1 \leq i \leq A$
- $1 \leq j \leq B$
- val is 0 if A[i] == B[i], otherwise is 1

Edit distance

$$E(i, j) = \text{Minimum number of edits to transform A into B.}$$

Subproblems?

$$E(i, j) = \min\{\ E(i - 1, j) + 1, E(i, j - 1) + 1, E(i - 1, j - 1) + val\}$$

Base case?

$$E(i, 0) = i$$

$$E(0, j) = j$$

Where:

- $1 \leq i \leq A$
- $1 \leq j \leq B$
- val is 0 if $A[i] == B[i]$, otherwise is 1

DP: Applications

- Winning and Score Predictor (WASP)
- TEX paragraph breaking.
- UNIX diff

References I

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein.
Introduction to algorithms.
MIT press, 2009.

Antti Laaksonen.
Competitive Programmers Handbook.
2018.

Dynamic Programming

February 2024