Fundamental Algorithmic Techniques V

February 3, 2026

Outline

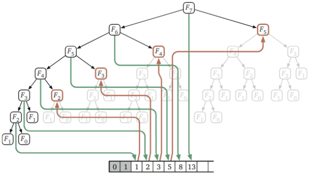
Fibonacci and Memoization

Memoized $\mathcal{O}(n)$, space $O(n)$

```
function IterFibo1(n)
    F[0] ← 0
    F[1] ← 1
    for i = 2 to n do
        F[i] ← F[i − 1] + F[i − 2]
    end for
    return F[n]
end function
```

Bottom-up $\mathcal{O}(n)$, space $O(1)$

```
function IterFibo2(n)
    prev ← 1
    curr ← 0
    for i = 1 to n do
        next ← curr + prev
        prev ← curr
        curr ← next
    end for
    return curr
end function
```



Matrix iteration $\mathcal{O}(\log n)$ (rep. squaring)

$$\begin{bmatrix} F_{n+1} \\ F_n \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Fibonacci identities:

$$F_{2n} = \ldots = F_n(2F_{n-1} + F_n)$$

$$F_{2n-1} = \ldots = F_{n-1} + F_n^2$$

Edit Distances

compare s1, s2 with operations (cost 1):

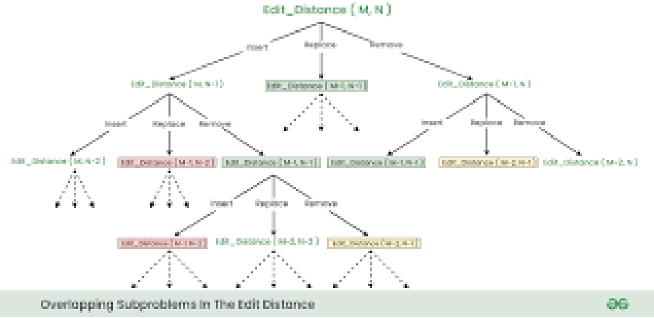
1. insert
2. remove
3. replace

Naive Algo: example of overlappings!

1. Last character of s_1, s_2 are same:
   $\mathrm{ED}(s_1, s_2, m, n) = \mathrm{ED}(s_1, s_2, m-1, n-1)$
2. $$\begin{aligned}\mathrm{ED}(s_1, s_2, m, n) = 1 + \min\big(&\mathrm{ED}(s_1, s_2, m, n-1),\\ &\mathrm{ED}(s_1, s_2, m-1, n),\\ &\mathrm{ED}(s_1, s_2, m-1, n-1)\big)\end{aligned}$$

time complexity: $\mathcal{O}((3)^{n_1+n_2})$ and $\mathcal{O}(n_1 \cdot n_2)$

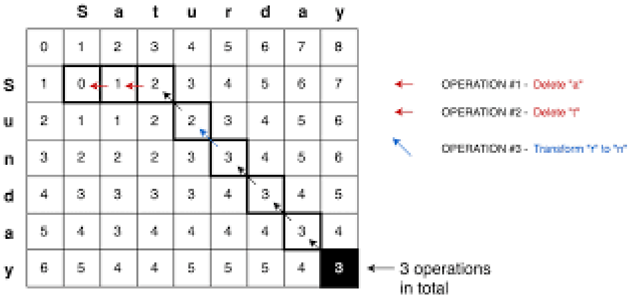
Edit Distances : Memoisation



Memoization Strategies:

- Top-Down ED **Memoisation** - $\mathcal{O}(m \cdot n)$ time and $\mathcal{O}(m \cdot n)$ space
- Bottom-Up ED **Tabulation** - $\mathcal{O}(m \cdot n)$ time and $O(m \cdot n)$ space

Edit Distances : Tabulation



Costs with: insert, remove, replace
Result bottom right!

Other name: **Levenshtein Distance**

Optimal Substructure

Optimal substructure if an optimal solution constructed from optimal solutions of its subproblems.

Examples:

- shortest path on road or graph
- Fibonacci: $F(n) = F(n) + F(n-1)$
- rod sold at prices for subsets

Counterexamples:

- shortest path on a Graph without passing twice same node

If optimal Substructure, you can write:

$$OPT(n) = min(OPT(n-1), OPT(n-2), ...) + f(n)$$

Dynamical Programming $\Leftrightarrow$ Optimal Substructures

Overlapping Subproblems

overlapping subproblems if the same subproblem is solved multiple time Classic examples:

1. Fibonacci numbers
2. shortest paths
3. knapsack problem
4. edit distance.

Overlapping Subproblems $\Leftrightarrow$ memoization/tabulation/others!
No Overlapping $\Leftrightarrow$ No Dynamic Programming!

0/1 Knapsack Problem

- A knapsack with integer capacity $W > 0$,
- n items, where item i has:
  - weight $w_i \in \mathbb{Z}^+$,
  - value $v_i \in \mathbb{R}^+$.

Each item may be taken at most once.
Goal: Maximize total value, weight $\leq W$!

- Overlapping subproblems & Optimal Substructure
  Idea: $ks[i][w]$ = maximum value achievable using the first i items with capacity w,
- Dynamic programming applies!
  complex problem: solvable by simple bottom up tabulation

Iteration:

$$\mathrm{ks}[i][w] = \begin{cases} \mathrm{ks}[i-1][w], & \text{if } w_i > w, \\ \max\left(\mathrm{ks}[i-1][w],\ \mathrm{ks}[i-1][w-w_i] + v_i\right), & \text{if } w_i \leq w. \end{cases}$$

0/1 Knapsack by Tabulation

Capacity: $W = 8$

Items:

i	w_i	v_i
1	2	3
2	3	4
3	4	5
4	5	6

Table $\texttt{ks}[i][w]$:

$i \backslash w$	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3	3	3
2	0	0	3	4	4	7	7	7	7
3	0	0	3	4	5	7	8	9	9
4	0	0	3	4	5	7	8	9	**10**

Optimal value: $\texttt{ks}[4][8] = 10$
Selected items: **2 and 4**
(weight: $3 + 5 = 8$, value: $4 + 6 = 10$)

- **Time complexity**: $O(nW)$
- **Space complexity**: $O(nW)$ (optimizable to $O(W)$)