Basics: Recurrence relations

Basic idea: express the solution to a problem in terms of solutions to smaller problems

Ex: Compute factorial of input n

$$\boxed{T(n) = T(n-1) \cdot n}$$

Algorithm: Break down the problem repeatedly until it is small enough (base case)

Computing edit distance

Defn: 2 strings s and t

Edit distance $ED(s,t)$ = min # operations to $s \to t$ or $t \to s$

(i) delete a character

(ii) insert " "

(iii) swap 2 characters

"cat" "cat" $ED(s,t) = 0$

"cat" "dog" $ED(s,t) = 3$

"cat" "at" $ED(s,t) = 1$

"a cat" "the cats" $ED(s,t) = 4$

the cats

$s_1 \ldots \boxed{s_\ell}$

$t_1, t_2 \ldots \boxed{t_\ell}$

① $s_\ell = t_\ell$

② $s_\ell \neq t_\ell$

3 choices

① swap s_ℓ with t_ℓ + proceed with $s_1 \ldots s_{\ell-1}$, $t_1 \ldots t_{\ell-1}$ smaller

$$\boxed{1 + d(\ell-1, \ell-1)}$$

② delete s_ℓ + proceed with $s_1 \ldots s_{\ell-1}$, $t_1 \ldots t_\ell$ smaller

$$\boxed{1 + d(\ell-1, \ell)}$$

③ delete t_ℓ + proceed with $s_1 \ldots s_\ell$, $t_1 \ldots t_{\ell-1}$ smaller

$$\boxed{1 + d(\ell, \ell-1)}$$

Defn: $d(m,n)$: edit distance of $s_1 \ldots s_m$, $t_1 \ldots t_n$

$$d(m,n) = d(m-1, n-1) \quad \text{if} \quad s_m = t_n$$

$$d(m,n) = 1 + \min \begin{cases} d(m-1, n-1) \\ d(m-1, n) \\ d(m, n-1) \end{cases} \quad s_m \neq t_n$$

Base case:

$$d(0, n) = n$$
$$d(m, 0) = m$$

Ex: Knapsack problem

k objects $\quad \left.\begin{matrix} w_1 \ldots w_k > 0 \\ v_1 \ldots v_k > 0 \end{matrix}\right\}$ ordered

sack of capacity W^*

maxval(k, W^*)

$\boxed{\text{maxVal}(m, n)}$ use objects $1 \ldots m$ + weight constraint $= n$

put m into sack: $v_m + \text{maxVal}(m-1, n - w_m)$

do not put m into sack: $\text{maxVal}(m-1, n)$

$$\text{maxVal}(m, n) = \max \begin{cases} v_m + \text{maxVal}(m-1, n-w_m) \\ \text{maxVal}(m-1, n) \end{cases}$$

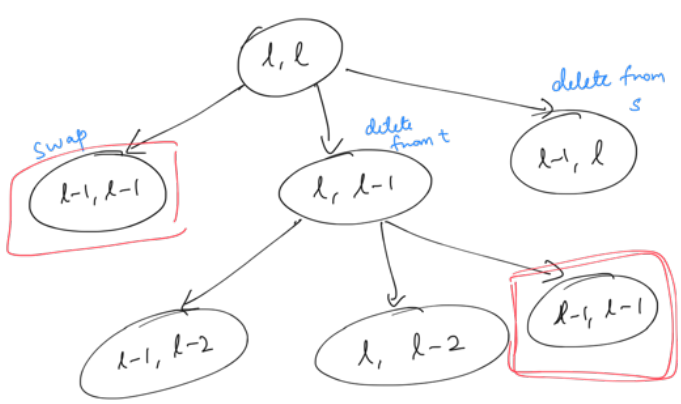
Base case:

$$\text{maxVal}(0, n) = 0$$
$$\text{maxVal}(m, 0) = 0$$

Implementation matters!

Recursive calls: solve some problems repeatedly!

$m = n = l$ (length of strings)



$O(2^{l})$: space : $O(l)$

"cache" computations !

① create "array" [2-D]

$d[i][j] = d(i,j)$ $\nearrow l$ $\nearrow l$

② Populate this array from base case

Dynamic Programming : $O(l^2)$

space : $O(l^2)$