Keyboard Distance

The idea is to keep track of cost in the dp array.

function modified_distance(w_1, w_2):

m = len(w_1), n = len(w_2)

dp = 2D array of (m+1)(n+1)

dp[0][0] = 0

\# All insert
for j from 1 to n, dp[0][j] = dp[0][j-1]+3

\# All delete 1st col.
for i = 1 to m:

if adjacent, dp[i][0] = dp[i-1][0]+1

else

dp[i][0] = dp[i-1][0] + 2

\# Filling the table

for i = 1 to m:

for j = 1 to n:

* if char 1 == char 2:

dp[i][j] = dp[i-1][j-1]

* else:

if same_letter_case_mismatch:

cost = 0.5

if adjacent: cost = 1

else cost = 2

if adj(c_1, c_2): del_c = 1

else: del_c = 2

cost_insert = 3.

dp[i][j] = min

dp[i-1][j-1]+cost,

dp[i-1][j]+ deleting_cost,

dp[i][j-1]+ ins_cost

return dp[m][n].

7

Edit Distance

Distance between "network" and "worth" =

			n	e	t	w	o	r	k
''	0	0	1	2	3	4	5	6	7
w	1	1	1	2	3	4	5	6	7
o	2	2	2	2	3	4	3	4	5
r	3	3	3	3	3	5	4	3	4
t	4	4	4	4	4	4	5	4	5
h	5	5	5	5	5	5	5	5	6

network
↓ 2
work
↓ 1
wort
↓
worth

	0	w	o	r	t	h
						5
		1	2	3	4	
0	0			3	4	5
n	1	1	2			5
e	2	2	2	3	4	6
t	3	3	3	3	4	
w	4	3	4	4	4	4
o	5	4	3	4	5	5
r	6	5	4	3	6	5
k	7	6	5	6	6	5

6

[2, 3, 5, 7, 11, 13] . no shift needed.

S →

n ↓

0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
1	T	F	T	T	F	T	F	T	T	F	F	T	F	T	F	F	F	F	F	F	F	F
2	T	F	T	T	F	T	F	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F
3	T	F	T	T	F	T	F	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F
4	T	F	T	T	F	T	F	T	T	T	T	F	T	F	T	T	F	T	F	F	F	F
5	T	F	T	T	F	T	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	
6	T	F	T	T	F	T	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	
~~7~~	~~T~~	~~F~~	~~T~~	~~T~~	~~F~~	~~T~~	~~F~~	~~T~~	~~T~~	~~T~~	~~T~~	~~T~~	~~T~~	~~T~~	~~T~~	~~T~~	~~T~~	~~T~~	~~T~~	~~T~~	~~T~~	

dp[6][20] = True.

Subset Sum

Given an array of integers, and a target T, design a method to verify if a subset sums to exactly T.

Strategy

~~- keep dp array of subset size from~~ 1 to n

where - dp[i][s] = using first i items, we can form sum s. (boolean) · Initialize all to false.

- s = max possible - min possible sum among all in array.
- Base case: dp[0][s]: True. Sum 0 possible with 0 items.

- Handle negative numbers through a shift
- shift = -(minimum sum ~~of~~ possible from negative numbers).

```
- for i in range (1, n+1):
      for s in range (min_sum, max_sum):

          # skip current
              if dp[i-1][s+shift]:
                  dp[i][s+shift] = True.

          # Take current
              if dp[i-1][s-arr[i-1]+shift]:
                  dp[i][s+shift] = True
```

return dp[n][Target + shift]. Note shift = 0 when all numbers are positive.

0/1/2 problem

for each item you can take 0, 1 or 2 copies.

for standard knapsack, $dp[i][w] = \max(dp[i-1][w],$
$v_i + dp[i-1][w-w_i])$

when we can take two copies max,
we have to chose a maximum of three cases

① Don't chose current (0) = $dp[i-1][w]$

② Chose just one copy : $dp[i-1][w-w_i] + v_i$

③ chose two copies of current = $2v_i + dp[i-1][w_i - 2w_i]$.

pseudocode -

① create 2D dp array - $dp[0 \dots n][0 \dots w]$.

② fill up base cases with $dp[0][w]$ and $dp[n][0]$ for w_i = 0 to w and n_i = 0 to n

③ for i = 1 to n
  for w_i = 1 to w :

    dp[i][w] = max (
      dp[i-1][w] # don't chose
      val[i] + dp[i-1][w-weight[i]] # chose 1
      2 x val[i] + dp[i-1][w - 2 x weight[i]] # chose 2 copy.

④ return dp[n][w]

②

0 1 2 3 4 5 6 7 8 9 10 11 12

i ↓ \ w →	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0		10																		
2	0		10	12		22															
3	0		10	12		25		35	37		47										
4	0		10	12		25		36	37	46	47	46	61	[illegible]	71	73		83			
5	0		10	12		25		36	44	46	54	56	61	69	71	80	81	90	91		105 105
6	0		10	12		25		36	44	~~66~~ 51	54	61		69	76	80	9	95	95	105	105

Hence ~~max table~~

max (6th row) = 105 .

1