Проблем удаљености измјене ниске

Задатак:

Овај проблем се назива проблем удаљености измјене ниске (енг. *the string edit distance problem*) и прилично је користан у многим истраживачким областима. Претпоставите да ријеч *„алгоритам“* желите да претворите у ријеч *„алигатор“*. За свако слово можете копирати слово из једне ријечи у другу по цијени 5, обрисати слово по цијени 20 или уметнути слово по цијени 20. Укупна цијена претварања једне ријечи у другу се користи у програмима за провјеру исправности текста ради пружања приједлога за ријечи које су блиске једна другој. Користити технике динамичког програмирања за развој алгоритма који вам даје најмању удаљеност измјене између било које двије ријечи.

Рјешење:

```python
def min_edit_distance(word1, word2):
    m = len(word1)
    n = len(word2)

    # Pravimo tabelu za chuvanje rezultata podproblema
    dp = [[0 for x in range(n + 1)] for x in range(m + 1)]

    # Popunjavamo d[][] odozdo prema gore
    for i in range(m + 1):
        for j in range(n + 1):

            # Ukoliko je prva niska prazna, jedina opcija
            # jeste umetnuti sva slova iz druge niske
            if i == 0:
                dp[i][j] = j * 20 # cijena umetanja slova

            # Ukoliko je druga niska prazna, jedina opcija
            # jeste ukloniti sva slova iz druge niske
            elif j == 0:
                dp[i][j] = i * 20 # cijena uklanjanja slova

            # Ako su posljednja slova jednaka, ignorishemo posljednje
            # slovo i ponavljamo za ostatak niske
            elif word1[i - 1] == word2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]

            # Ukoliko se posljednje slovo razlikuje, razmatramo
            # sve mogucnosti i trazhimo minimum
            else:
                dp[i][j] = min(dp[i][j - 1] + 20,  # cijena umetanja slova
                               dp[i - 1][j] + 20,  # cijena uklanjanja slova
                               dp[i - 1][j - 1] + 5) # cijena zamjene slova

    return dp[m][n]

# glavni program
if __name__ == '__main__':

    # print(min_edit_distance("algorithm", "alligator"))
    print(min_edit_distance("dragana", "dragon"))
```

Опис:

Ова функција узима два низа *word1* и *word2* и враћа минималну удаљеност измјене између њих, пратећи правила наведена у исказу проблема (цијена копирања знака је 5, цијена уметања или брисања знака је 20).

Функција прво иницијализује *2D* табелу *dp* за складиштење резултата подпроблема. Затим попуњава табелу на начин одоздо према горе, користећи сљедеће случајеве:

- Ако је први низ празан, једина опција је да убаците све знакове другог низа, по цијени од 20 по карактеру.
- Ако је други низ празан, једина опција је да избришете све карактере првог низа, по цијени од 20 по карактеру.
- Ако су посљедњи знакови два низа исти, можемо их занемарити и поновити за преостале низове.
- Ако су посљедњи знакови различити, разматрамо све могућности (убацивање знака, брисање карактера или замјену карактера) и узимамо минималну цијену међу њима.

Коначно, функција враћа минималну удаљеност измјене сачувану у посљедњој ћелији табеле *(dp[m][n])*, гдје су *m* и *n* дужине *word1* и *word2*, респективно.

Временска сложеност овог алгоритма је *O(mn)*, гдје су *m* и *n* дужине унесених ниски. То је због тога што треба да попунимо табелу величине *m* x *n,* а сваку ћелију попуњавамо једном.

Просторна сложеност алгоритма је такође *O(mn),* јер треба да смјестимо у меморију табелу величине *m* x *n.*

dp коначна табела резултата:

		D	R	A	G	O	N
	0	20	40	60	80	100	120
D	20	0	20	40	60	80	100
R	40	20	0	20	40	60	80
A	60	40	20	0	20	40	60
G	80	60	40	20	0	20	40
A	100	80	60	40	20	5	25
N	120	100	80	60	40	25	5
A	140	120	100	80	60	45	25

Коначно, тражено рјешење

Драгана Мајсторовић, 2022. год.