dp2

May 18, 2020

[1]:

```
import time
def measure(f):
    start = time.time()
    f()
    end = time.time()
    return end-start
```

1 Dynamic Programming Examples

1.1 Editing Distance

1.1.1 Recursive

[2]:

```
def lev_r(x,n,y,m):
  """
  recursive distance function that computes the edit distance between strings x␣
 ↪and y
  """
  if n == 0:
    return m
  if m == 0:
    return n
  l1 = lev_r(x,n-1,y,m)+1
  l2 = lev_r(x,n,y,m-1)+1
  l3 = lev_r(x,n-1,y,m-1) + (0 if x[n-1]==y[m-1] else 1)
  return min(l1,l2,l3)

def lev_R(x,y):
    return lev_r(x,len(x),y,len(y))
```

[3]:

```
print(lev_R("ETH","EPFL"))
print(lev_R("ETHZurich","EPFLausanne"))
print(measure(lambda: lev_R("ETHZurich","EPFLausanne")))
print(lev_R("ZIEGE","TIGER"))
```

```
3
9
```

```
2.2370097637176514
3
```

1.1.2 Memoization

[4]:

```python
def lev_m(x,n,y,m,d):
    if (n,m) in d:
        return d[(n,m)]
    else:
        if n == 0:
            return m
        if m == 0:
            return n
    l1 = lev_m(x,n-1,y,m,d)+1
    l2 = lev_m(x,n,y,m-1,d)+1
    l3 = lev_m(x,n-1,y,m-1,d) + (0 if x[n-1]==y[m-1] else 1)
    d[(n,m)] = min(l1,l2,l3)
    return d[(n,m)]

def lev_M(x,y):
    return lev_m(x,len(x),y,len(y),{})
```

[5]:

```python
print(lev_M("ETH","EPFL"))
print(lev_M("ETHZurich","EPFLausanne"))
print(measure(lambda: lev_M("ETHZurich","EPFLausanne")))
print(lev_M("ZIEGE","TIGER"))
```

```
3
9
0.0002186298370361328
3
```

1.1.3 Table-based approach

[6]:

```python
def lev_t(x,y):
    N = len(x)+1
    M = len(y)+1
    d = [[0]*M for i in range(0,N)]
    # do not use [[0]*N]*M because this creates a matrix referencing a single␣
 ↪row!
    for n in range (0,N):
        for m in range (0,M):
            if n == 0:
                d[n][m] = m
            elif m == 0:
                d[n][m] = n
            else:
                l1 = d[n-1][m]+1
                l2 = d[n][m-1]+1
                l3 = d[n-1][m-1]+(0 if x[n-1]==y[m-1] else 1)
                d[n][m] = min(l1,l2,l3)
    return d

def lev_T(x,y):
    d = lev_t(x,y)
    return d[len(x)][len(y)]
```

[7]:

```
print(lev_T("ETH","EPFL"))
print(lev_T("ETHZurich","EPFLausanne"))
print(measure(lambda: lev_T("ETHZurich","EPFLausanne")))
print(lev_T("ZIEGE","TIGER"))
print(lev_t("FISCH","FROSCH"))
```

```
3
9
0.00011157989501953125
3
[[0, 1, 2, 3, 4, 5, 6], [1, 0, 1, 2, 3, 4, 5], [2, 1, 1, 2, 3, 4, 5], [3, 2, 2,
2, 2, 3, 4], [4, 3, 3, 3, 3, 2, 3], [5, 4, 4, 4, 4, 3, 2]]
```

1.2 Which path

[8]:

```
def insert(string,index,c):
    return string[:index] + c + string[index:]
def remove(string,index):
    return string[:index] + string[index+1:]
def replace(string,index,c):
    return string[:index] + c + string[index+1:]

def edit(x,y):
    d = lev_t(x,y)
    word = x
    # reconstruct
    n = len(x)
    m = len(y)
    while n+m > 0:
        if n>0 and d[n][m] == d[n-1][m]+1:
            print("remove: ", x[n-1])
            word = remove(word,n-1)
            print(word)
            n = n-1
        elif m>0 and d[n][m] == d[n][m-1]+1:
            print("insert:", y[m-1])
            word = insert(word,n,y[m-1])
            print(word)
            m = m-1
        else:
            assert(d[n][m]==d[n-1][m-1]+(0 if x[n-1]==y[m-1] else 1))
            if (d[n][m]==d[n-1][m-1]):
                print("keep:",x[n-1])
            else:
                print("replace:",x[n-1]," by ", y[m-1])
            word = replace(word,n-1,y[m-1])
            n = n-1
            m = m-1
            print(word)
```

```
[9]: edit("FISCH","FROSCH")
     #edit("ZIEGE","TIGER")
```

```
keep: H
FISCH
keep: C
FISCH
keep: S
FISCH
insert: O
FIOSCH
replace: I  by  R
FROSCH
keep: F
FROSCH
```