# 最小编辑距离报告


## 人工智能82班 刘志成 2183511589


相关代码已上传至我的github仓库：
https://github.com/zchliu/2020-Fall

## 问题引入

对于两个字符串S1和S2，设其长度分别是n和m，Distance(i,j)或者D(i,j)表示S1[1:i]和S2[1:j]的最小编辑距离，即将S1中前i个字符转化为S2中的前j个字符所需的最小编辑次数，试求S1和S2的最小编辑距离D(m,n)，并打印出相应的操作过程。

编辑可以有如下几种操作：

- 替换：将S1中第x个字符替换为S2中第y个字符，编辑距离+2
- 删除：将S1中第x个字符删去，编辑距离+1
- 增加：在S1中第x位增加一个字符c，编辑距离+1

例如：
输入两个字符串intention和execution：

则输出如下：

```
The minimun editting distance is:  8
intention
entention  替换   6
extention  替换   4
exention  删除   3
exection  替换   1
execution  增加   0
```

## 计算模型

这个问题的求解大体来说可以分成3步，第一步要从两个字符串当中求得状态转换表，第二步要根据状态转换表求出回退的路径，第三步是输出答案

- 求最小编辑距离一个动态规划问题

我们可以定义如下状态转移方程：

1. 首先定义边界条件，对于一个字符串的前i项和另一个字符串的前0项，只能通过增加或减少i个字符来转化，因此

D(i,0) = i

D(0,j) = j

2. 对于每一个D(i,j)来说，他的状态可以由3个地方得到：

首先是增加，由于增加只使得编辑距离+1，因此有D(i,j) = D(i-1,j) + 1

其次是减少，由于减少只使得编辑距离+1，因此有D(i,j) = D(i,j-1) + 1

最后是替换，如果D(i,j)和D(i-1,j-1)对应的字符相同，则D(i,j) = D(i-1,j-1)，如果D(i,j)和D(i-1,j-1)对应的字符不同，由于替换使得编辑距离+2，则有D(i,j) = D(i-1,j-1) + 2

- 回退是一个由有向图产生最小生成树的问题

由于我们还要得到从一个字符串到达另外一个字符串的路径，因此我们在循环的过程中，需要保存每一次状态转移的方向，最后我们会得到一个节点是D(i,j)边是连接两个节点编辑操作的有向图

生成最小生成树有多种算法，我使用bfs（广度优先搜索）实现

1. 以目标节点D(m,n)作为根节点，将根节点压入队列
2. 每次从队列头中取第一个元素，遍历所有与它相连的节点，如果这个节点没有被访问过，则将其压入队列，并且将这个节点设置为已被访问
3. 检查队列是否为空，如果为空则跳出循环，否则回到2

这样我们可以得到一个由D(m,n)作为根节点的最小生成树，我们由D(0,0)向上找父节点就可以得到回退的路径了

- 输出

可以设置一个保留字符串reserve_str和一个位置标记pos，在回退中，我们已经得到了D(0,0)到D(m,n)的路径，以str1作为原始字符串，在路径中：

1. 如果是替换，则将替换的字符加入保留字符串，位置标记+1，打印reserve_str + str1[pos:]
2. 如果是删除，位置标记+1，打印reserve_str + str1[pos:]
3. 如果是增加，则将增加的字符加入保留字符串，位置标记不变，打印reserve_str + str1[pos:]

# 编程实现

```
import numpy as np
from collections import defaultdict


class itemtype:
    def __init__(self, distance, position):
        self.distance = distance
        self.position = position
        self.last_item = []


def edit(str1, str2):
    l1 = len(str1)
    l2 = len(str2)

    D = np.array([[itemtype(0, (i, j)) for j in range(l2 + 10)] for i in range(l1 + 10)], dtype=itemtype)

    for i in range(0, l1 + 1):
        D[i, 0].distance = i

    for j in range(0, l2 + 1):
        D[0, j].distance = j

    # 加上从边缘到D[0,0]的边，防止后面path出错
    for i in range(1, l1 + 1):
        D[i, 0].last_item.append(D[i - 1, 0])

    for j in range(1, l2 + 1):
        D[0, j].last_item.append(D[0, j - 1])

    for i in range(1, l1 + 1):
        for j in range(1, l2 + 1):
            if str1[i - 1] == str2[j - 1]:
                D[i, j].distance = min(D[i - 1, j].distance + 1, D[i, j - 1].distance + 1, D[i - 1, j - 1].distar
                if (D[i, j].distance == D[i - 1, j].distance + 1): D[i, j].last_item.append(D[i - 1, j])
                if (D[i, j].distance == D[i, j - 1].distance + 1): D[i, j].last_item.append(D[i, j - 1])
                if (D[i, j].distance == D[i - 1, j - 1].distance): D[i, j].last_item.append(D[i - 1, j - 1])
            else:
                D[i, j].distance = min(D[i - 1, j].distance + 1, D[i, j - 1].distance + 1, D[i - 1, j - 1].distar
                if (D[i, j].distance == D[i - 1, j].distance + 1): D[i, j].last_item.append(D[i - 1, j])
                if (D[i, j].distance == D[i, j - 1].distance + 1): D[i, j].last_item.append(D[i, j - 1])
                if (D[i, j].distance == D[i - 1, j - 1].distance + 2): D[i, j].last_item.append(D[i - 1, j - 1])

    return D[l1, l2].distance, D


def bfs(s, end):
    queue = []
    path = defaultdict(list)
    queue.append(s)
    seen = set()
```

```
    seen.add(s)
    while (len(queue) > 0):
        vertex = queue.pop(0)
        for i in vertex.last_item:
            print(i.distance)
            if i not in seen:
                queue.append(i)
                path[i] = vertex
                seen.add(i)
    return path


def output(str1, str2, distance, D, path):
    last = D[0, 0]
    end = D[len(str1), len(str2)]
    lst = ""
    pos = 0

    print(str1 + " " + str2)
    while (1):
        if (last == end):
            break

        next = path[last]

        if (next.distance != last.distance):
            # 在下一个位置的距离不相等的时候才做变化
            if (next.position[0] == last.position[0] + 1 and next.position[1] == last.position[1] + 1):
                # 替换
                j = next.position[1] - 1
                lst = lst + (str2[j])
                pos = pos + 1
                print("".join(lst + str1[pos:]), " 替换", " ", distance - next.distance)

            if (next.position[0] == last.position[0] + 1 and next.position[1] == last.position[1]):
                # 删除
                pos = pos + 1
                print("".join(lst + str1[pos:]), " 删除", " ", distance - next.distance)

            if (next.position[0] == last.position[0] and next.position[1] == last.position[1] + 1):
                # 增加
                j = next.position[1] - 1
                lst = lst + (str2[j])
                print("".join(lst + str1[pos:]), " 增加", " ", distance - next.distance)

        else:
            pos = pos + 1
            j = next.position[1] - 1
            lst = lst + (str2[j])
        last = next
```

```
if __name__ == "__main__":

    str1 = input("Enter a string:")
    str2 = input("Enter another string:")

    distance, D = edit(str1, str2)
    path = bfs(D[len(str1), len(str2)], D[0, 0])
    print("The minimun editting distance is: ", distance)
    output(str1, str2, distance, D, path)
```

几种运行结果如下：

```
Enter a string:intention
Enter another string:execution
The minimun editting distance is:  8
intention
entention  替换   6
extention  替换   4
exention  删除   3
exection  替换   1
execution  增加   0

Enter a string:今天你吃了吗
Enter another string:我没吃
The minimun editting distance is:  7
今天你吃了吗
我天你吃了吗  替换   5
我没你吃了吗  替换   3
我没吃了吗  删除   2
我没吃吗  删除   1
我没吃  删除   0

Enter a string:1235487
Enter another string:77788
The minimun editting distance is:  10
1235487
7235487  替换   8
7735487  替换   6
7775487  替换   4
7778487  替换   2
777887  删除   1
77788  删除   0
```

# 评估模型

评估从理论时间复杂度分析，100个长度为50的两个随机字符串的平均运行时间两个角度评判

- 时间复杂度分析

假设字符串的长度分别是m和n，在状态转换表的填写过程中，时间复杂度为O(mn)，在回退过程中，最坏情况下所有节点加入图，bfs遍历每一个节点时间复杂度为O(mn)，输出过程中时间复杂度与路径的长度有关，路径在bfs遍历中为生成树的深度近似表示为O(log(mn))，因此总的时间复杂度为O(mn)

- 平均运行时间分析

testbench

```
import MED
import random
import string
import time

time_start = time.time()

for i in range(100):
    ran_str1 = ''.join(random.sample(string.ascii_letters + string.digits, 5))
    ran_str2 = ''.join(random.sample(string.ascii_letters + string.digits, 5))

    distance, D = MED.edit(ran_str1, ran_str2)
    path = MED.bfs(D[len(ran_str1), len(ran_str2)], D[0, 0])
    print("The minimun editting distance is: ", distance)
    MED.output(ran_str1, ran_str2, distance, D, path)

time_end = time.time()
print('average cost', (time_end - time_start) / 100)
```

结果是平均每一对50个字符的变化大约需要0.06-0.1秒