알고리즘 HW3

2016-19965 김성은

소스 코드:

```
#!/usr/bin/env python
from docx import Document
import argparse, os, time, sys

parser = argparse.ArgumentParser()
parser.add_argument("-f", dest="filePath", type=str, default="./input/seq.txt", help="path of the file, gets seq_short on default")
parser.add_argument("-e", dest="maxerr", type=int, default=15, help="maximum changes allowed in lcs")
parser.add_argument("-l", dest="minlen", type=int, default=50, help="minimum length of lcs(50~100 recommended)")
parser.add_argument("-m", dest="minmatch", type=int, default=3, help="minimum length of consequtive ")
parser.add_argument("-s", dest="save", type=str, default=None, help="store result in filepath")

args = parser.parse_args()

# where output is stored in, show result on terminal if None
OUT_PATH = args.save

# size of the checking window
WIN_SIZE = 250

# used to checks similarity in two lcs(not currently used)
MIN_SIMILARITY = 0.9

# maximum length of addition or deletion
MAX_ERR_LENGTH = args.maxerr

# how many character must continuously match to make it valid lcs
MIN_MATCH_LENGTH = args.minmatch

# minimum length of lcs string
# if found string is longer than this value, declare it as
# hairpin string and print it
MIN_LCS_LENGTH = args.minlen

# if valid lcs isn't found, skip iteration to speed up process
SKIP_DIST = 100

FILE_DEST = args.filePath

# main() will print out all the found lcs based on given parameters and input text file.
# also, insertion and deletion will be shown on second string of lcs with markdown syntax.

# **insertion**
# ~~deletion~~

def main():
    inputType = FILE_DEST[FILE_DEST.rfind('.'):]
    try:
        if inputType == ".txt":
            with open(FILE_DEST) as text:
                gene = text.readlines()[-1]
        elif inputType == ".docx":
            gene = Document(FILE_DEST).paragraphs[-1].text
        else:
            print("input invalid!")
            os._exit(0)
        if OUT_PATH is not None and os.path.exists(OUT_PATH):
            os.remove(OUT_PATH)
    except:
        print("no such file!")
        os._exit(0)
    findHairpin(gene)


def findHairpin(gene):
    genelen = len(gene)
    print("gene length: "+str(genelen))
    start = time.time()
    i = 0
    maxlen = 0
    bestStr = ("", "", "")
    while i < genelen-WIN_SIZE*2:
        out = lcs(gene[i:i+WIN_SIZE], gene[i+WIN_SIZE:i+WIN_SIZE*2])
        percent = str(i * 100 / genelen) + "%"
        sys.stdout.write("calculating:" + percent + "              ₩r")
        sys.stdout.flush()
        if out is not None:
            if maxlen < max(out[1]-out[0], out[3]-out[2]):
                maxlen = max(out[1]-out[0], out[3]-out[2])
                bestStr = (gene[i+out[0]:i+out[1]], gene[i+out[1]:i+WIN_SIZE+out[2]], gene[i+WIN_SIZE+out[2]:i+WIN_SIZE+out[3]], out[4])
                i = i + min(out[1], out[2])
            else:
                printlcs(bestStr);
                i = i + WIN_SIZE
                maxlen = 0
                bestStr = ("","","")
        else:
            i = i + SKIP_DIST

    end = time.time()
    print("time spent:" + str(round(end-start, 2)))

def printlcs(bestStr):
    if OUT_PATH is None:
        print("LCS 1  : " + bestStr[0])
        print("hairpin: " + bestStr[1])
        print("LCS 2  : " + bestStr[2])
        print("change : " + bestStr[3])
        print("")
    else:
        with open(OUT_PATH,"a") as outfile:
            print >> outfile, "LCS 1  : " + bestStr[0]
            print >> outfile, "hairpin: " + bestStr[1]
            print >> outfile, "LCS 2  : " + bestStr[2]
            print >> outfile, "change : " + bestStr[3]
            print >> outfile, ""

# getstring similarity based on levenshtein distance
# similar to lcs algorithm
```

```
def strsim(str1, str2):
    similarity = levdist(str1, str2) * 1.0 / max(len(str1), len(str2))
    return 1 - similarity


# to allow insertion and deletion while removing string with too big gaps,
# decrement dist[][] a little bit when common string is not continuing

# variable for expression where max dist[][]['length'] come from:
START = 0
FROM_I = 1
FROM_J = 2
FROM_MATCH = 3
DEFAULT_DIST = {'length' : 0, 'from' : START, 'cont' : 0}
def lcs(str1, str2):
    retdic = {
        'length' : 0,
        'found' : False
        }
    len1 = len(str1)
    len2 = len(str2)
    maxLength = 0
    bestIndex = (0,0)

    # dist contains a dictionary which consists of length and where lcs come from,
    # and how many character is continously matching
    # if string is not matching, cont variable will turn negative, and too many mismatch will
    # make the lcs start over from 0

    dist = [[DEFAULT_DIST for x in range(len2)] for y in range(len1)]
    for i in range(len1):
        for j in range(len2):
            if j is 0:
                dist[i][j] = {'length' : max(0, dist[i-1][j]['length']), 'from' : START, 'cont' : 0}
            else:
                # if dist[][] gets value from insertion or deletion, decrement its value by 1
                # if str1[i] and str2[j] matches, restore decremented value
                length1 = decrement(dist[i-1][j], FROM_I)
                length2 = decrement(dist[i][j-1], FROM_J)
                if str1[i] is str2[j]:
                    if dist[i-1][j-1]['length'] > 0:
                        matchLength = {'length' : dist[i-1][j-1]['length'] + 1
                                , 'from' : FROM_MATCH
                                , 'cont' : max(0, dist[i-1][j-1]['cont']) + 1
                                }
                    else:
                        matchLength = {'length' : 1
                                , 'from' : START
                                , 'cont' : 1
                                }
                else:
                    # in this case, both of the checking string is from 1 index behind,
                    # so decrement two times since its two times the difference
                    # from single insertion or deletion
                    matchLength = DEFAULT_DIST

                dist[i][j] = getBestLCS(length1, length2, matchLength)
                # because dist[][] looses its value if there's too much insertion
                # in a row, I need to store the index to best index
                if dist[i][j]['length'] > maxLength:
                    maxLength = dist[i][j]['length']
                    bestIndex = (i, j)

    if maxLength > MIN_LCS_LENGTH:
        return LCSindex(str1, str2, dist, bestIndex)
    else:
        return None

MATCH = 0
INSERT = 1
DELETE = 2
def LCSindex(str1, str2, dist, bestIndex):
    i = bestIndex[0]
    j = bestIndex[1]
    # state will be used to mark string for deletion and insertion
    # changestr will store formatted string
    state = MATCH
    changestr = ""
    while True:
        lcsFrom = dist[i][j]['from']
        if lcsFrom == FROM_I:
            if state == MATCH:
                changestr = changestr + "**"
            elif state == DELETE:
                changestr = changestr + "~~**"
            changestr = changestr + str1[i]
            state = INSERT

            i = i - 1
        elif lcsFrom == FROM_J:
            if state == MATCH:
                changestr = changestr + "~~"
            elif state == INSERT:
                changestr = changestr + "**~~"
            changestr = changestr + str2[j]
            state = DELETE

            j = j - 1
        elif lcsFrom == FROM_MATCH:
            if state == INSERT:
                changestr = changestr + "**"
            elif state == DELETE:
                changestr = changestr + "~~"
            changestr = changestr + str2[j]
            state = MATCH

            i = i - 1
            j = j - 1
        elif dist[i][j]['from'] is START:
            changestr = changestr + str2[j]
            break

    return(i, bestIndex[0] , j, bestIndex[1] , changestr[::-1])


# finds the best LCS, using cont value as tiebreaker
def getBestLCS(dict1, dict2, dict3):
    multval = MIN_MATCH_LENGTH + MAX_ERR_LENGTH
    l1 = dict1['length'] * multval + dict1['cont']
    l2 = dict2['length'] * multval + dict2['cont']
    l3 = dict3['length'] * multval + dict3['cont']
    maxLength = max(l1, l2, l3)
    if l1 is maxLength:
        return dict1
    elif l2 is maxLength:
        return dict2
    else:
        return dict3


def decrement(dist, src):
    retlen = dist['length']
    retcont = dist['cont']
    retsrc = START

    # cont value, which is number of continous character that match,
    # must be over MIN_MATCH_LENGTH to be valid lcs
    if retcont > 0 and retcont < MIN_MATCH_LENGTH:
        return DEFAULT_DIST

    # cont will be negative if character is mismatching several times in a row,
    # and if it will be used to discard lcs that has to many insertions or deletion
    if retcont <= -MAX_ERR_LENGTH:
        return DEFAULT_DIST

    retlen = retlen - 1
    retsrc = src
    retcont = min(retcont, 0) - 1

    return {'length' : retlen, 'from' : retsrc, 'cont' : retcont}


def levdist(str1, str2):
    len1 = len(str1)
    len2 = len(str2)
    dist = [[0 for x in range(len2)] for y in range(len1)]
    for i in range(len1):
        for j in range(len2):
            if min(i, j) is 0:
                dist[i][j] = max(i,j)
            else:
                charmismatch = 0 if str1[i] is str2[j] else 1
                dist[i][j] = min(dist[i-1][j] + 1, dist[i][j-1] + 1, dist[i-1][j-1] + charmismatch)
    return dist[len1-1][len2-1]


if __name__ == "__main__":
    main()
```

seq.txt output:

```
LCS 1 :
TAGATGATGATGTTATACGCGTTCTTCTGGCCGCTATTGGTGGAGGATGTAGTACTCCTCTTTTTTTAATAGTGACAT
AGGTCATCCTAGAGGCGGATTCGGACTCGAAGTTTGTGTTTGACGGGGGAATGTTGAGTGACCAGTCC
hairpin:
CCTGGAACATGAATCACAAATGGAGAGCTAACTAATCTATATCACTTTATCTTGCTAATGCAAAGGCCAAATGCAT
AAGTAGTTCAAACCCGGAAAATAATCTACTTTTGG
LCS 2 :
TAGATGATGATGTTATACGCGTTCTTCTGGCCGCTATTGGTGGAGGATGGGTATAGTACTCCTCTTTAATAGTGACA
TATTTTGGTCATCCTAGAGGCGGATTCGGACTCGAAGTTTGGACGGGGGAATGTTGAGTGACCTGTTTAGTCC
change :
TAGATGATGATGTTATACGCGTTCTTCTGGCCGCTATTGGTGGAGGAT~~GG~~GTA~~TA~~GTACTCCTC**TTT
T**TTTAATAGTGACATA~~TTTT~~GGTCATCCTAGAGGCGGATTCGGACTCGAA**GTTTGT**GTTT~~G~~G
ACGGGGGAATGTTGAGTGACC~~TGTTT~~AGTCCC

LCS 1 :
GCGCGGCGGTGCACAAGCAATTGACAACTAACCACCGTGTATTCGTTATGGCATCAGGCAGTTTAAGTCGAGA
CAATAGGGCTCGCAATACACAGTTTACCGCATCTTGCCCTAACTGACAAACTGTGATCGACCACTAGCCATGCC
ATTGCCTCTTAGACACCCGTG
hairpin:
TCGATACTGAACGAATCGATGCACACTCCCTTCCTTGAAAACGCACAATCATACAAGTGGGCACATGATGG
LCS 2 :
GCGCGGCGGTGCACAAGCAATTGACAACTAACCACCGTGTATTCGTTATGGCATCAGTTTAAGTCGAGACAATA
GGGCTCTACACAGTTTGCAAACCGCATCTTGCCCTAACTGACAAACTGTGATCGACCACTAGCCATGCCATTGC
CTCTTAGACACCCTG
change :
GCGCGGCGGTGCACAAGCAATTGACAACTAACCACCGTGTATTCGTTATGGCAT**CAGG**CAGTTTAAGTCG
AGACAATAGGGCT**CGCA**~~C~~**A**TACACAGTTT~~GCAA~~ACCGCATCTTGCCCTAACTGACAAA
CTGTGATCGACCACTAGCCATGCCATTGCCTCTTAGACACCC**G**TGT

LCS 1 :
ACCCGGAAATGGCTGTATTTATTGAGGTATTATACTGTGATATGTTAAAAAAAAAAGGGGAGTAGGTGGATGATT
TTCAAGAAGCTATGCCTAAGCGCGTGAGTACCATCGGCCAGACGCAGTCTTGGCCCAGTACCGACGAATCTAC
TGCAATCGCATGACAGGGCTAC
hairpin:
CATTAGAAACTACATATGAGGAGAATACCAGACGTTATTTTTTTGAACGACCACATACATAGCATACACATAATAA
ATTTAAAA
LCS 2 :
ACCCGGAAATGGCTGTATTTATTGAGGTATTATACAGTTGTGATATGTTAAAAAAAAAAGGGGAGTAGGTGGAT
GATTTTCAAGAAGCTATGCCTAAGCGCGTGAGTACCATCGGCCAGACGCAGTCTTGGCCCAGTACCGACGAAT
CTACTGCCCCAATCGCATGACAGGGCTAC
change :
ACCCGGAAATGGCTGTATTTATTGAGGTATTATAC~~AGT~~TGTGATATGTTAAAAAAAAAAGGGGAGTAGGT
GGATGATTTTCAAGAAGCTATGCCTAAGCGCGTGAGTACCATCGGCCAGACGCAGTCTTGGCCCAGTACCGAC
GAATCTACTG~~CCC~~CAATCGCATGACAGGGCTACC
```

1. Implementation

언어는 Python 을, 알고리즘은 LCS 알고리즘을 사용하였으며, 지나치게 많은 insertion 과 deletion 이 포함되는 것을 막기 위해 LCS 에 약간의 변형을 가해 사용하였다. String 내 변화를 얼마나 허용할지는 MAX_ERR_LENGTH, MIN_MATCH_LENGTH, MIN_MATCH_LENGTH 세개의 변수를 이용해 조절했다. 각각 연속 insertion 또는 deletion 을 최대 몇번까지 허용하는지, 연속 몇번 이상 string 이 일치해야 LCS 로 인정하는지, 그리고 LCS 의 최소 길이를 지정하는 변수이며, 프로그램 실행 시 argument 로 조절 가능하도록 하였다.

결과는 총 3 개의 hairpin structure 을 얻을 수 있었으며, MIN_LEN 과 같은 변수들을 조절하면 insertion 과 deletion 이 약간 지나치게 많은 것까지 4 개의 structure 을 얻을 수 있었다.

Insertion 과 deletion 은 결과값에 change 로 출력되도록 하였다. LCS 의 두번째 string 을 기준으로 어떻게 바꿔야 첫번째 string 이 되는지 markdown 문법으로 보여준다. Insertion 과 deletion 은 위에서부터 총 7 번, 6 번, 2 번 있었다.