Assignment #2

201624548 이재윤

Dynamic Programing 의 동작방식을 이해한다.
염기 서열 분석 알고리즘을 이해한다.
염기 서열 분석 알고리즘을 Smith-Waterman 알고리즘을 이용하여 구현한다.

● 사용 언어

Python 3

● 사용 라이브러리

sys (Command Line input을 위해)
collections.deque (프로그램 실행 속도의 증대를 위해)

● 사용 알고리즘

Dynamic Programing

■ Smith-Waterman Algorithm

● 아이디어

Dynamic Programing을 이용하여 Score Matrix를 만듦.
Score Matrix에 방향을 추가하여 Gap Extension Penalty의 계산을 쉽게 할 수 있도록 구현.
방향은 배열로 선언하여 여러 방향으로의 되추적을 가능하도록 설계.
Score Matrix에서 되추적을 통해 두 부분 Sequence를 구함.
방향이 배열로 선언되어 있기 때문에 가능한 모든 방향을 되추적하여 모든 부분 Sequence를 구함.

● 순서

Command Line에서 입력된 파일을 읽어와 String에 저장.
Smith-Waterman 알고리즘 호출.
Smith-Waterman 알고리즘 내에서 Matrix 계산 함수를 호출
Matrix 계산 함수 내에서 DP를 이용한 Scoring.
반환된 Score 매트릭스를 이용하여 되추적.
파일, 및 화면에 출력.

■ 알고리즘 분석

중요 함수인 mymatrix와 mytrace_back함수만 레포트 pdf에 추가하였다..

또한 나머지 알고리즘에도 주석을 통하여 알고리즘에 대해 설명하였다.

mytrace_back

```
# mymatrix에서 반환한 maxTuple과 H를 이용하여 염기 서열을 되추적하는 함수
# direction을 이용하여 추적함.
# queue를 사용하여 구현
# 5개의 값만 구하도록 구현.
# First in First Out
# 속도를 빠르게 하기 위하여 덱(deque)을 사용
from collections import deque
def mytrace_back(startTuple, H, a, b) :
    queue = deque()
    result = list()
    i,j = startTuple[0], startTuple[1]

    # 0 = 원래 Sequence
    # 1 = 비교 Sequence
    # 2 = 현재 index (이를 통해 Direction과 Sequence를 추가한다)
    queue.append(["","",startTuple])

    while queue and len(result) < 5 :
        nxt = queue.popleft()
        buf_a_word, buf_b_word,nxtTuple = nxt[0], nxt[1], nxt[2]
        i,j = nxtTuple[0], nxtTuple[1]
        nextdirection = H[i][j][1]
        # 가능한 모든 방향에 대해 되추적
        for d in nextdirection :
            # word 원복
            a_word, b_word = buf_a_word, buf_b_word
            # 두 sequence가 같은경우
            if d == 0 :
                a_word = a[i-1] + a_word
                b_word = b[j-1] + b_word
                queue.appendleft([a_word, b_word, [i-1, j-1]])
            # 비교하는 Sequence에 Gap이 필요한 경우
            elif d == 1 :
                a_word = a[j-1] + a_word
                b_word = '-' + b_word
                queue.appendleft([a_word, b_word, [i-1, j]])
            # 비교당하는 Sequence에 Gap이 필요한 경우
            elif d == 2 :
                a_word = '-' + a_word
                b_word = b[i-1] + b_word
                queue.appendleft([a_word, b_word, [i, j-1]])
            # 진행 방향이 없다면 (0을 만난다면)
            # result 값에 추가한다
            elif d == -1 :
                result.append(a_word + ',' + b_word)

    return result
```

i가 될 수 있는 값은 file1의(비교당하는 sequence)의 최대 길이고

j가 될 수 있는 값은 file2의(비교당하는 sequence)의 최대 길이이다.

최악의 경우 O(sequence1 의 최대 길이 * sequence2의 최대 길이)의 시간복잡도를 가진다.

이 함수의 시간 복잡도는 O(sequence1 의 최대 길이 * sequence2의 최대 길이) 이다.

mymatrix

```
# dp를 이용하여 두 sequence를 scoring 하는 함수
# dp 배열을 바로 만들 시 Memory 초과 우려가 있어 dp 배열을 한줄씩 늘려나가도록 구성
def mymatrix(a, b):
    # 길이 선언
    length_a, length_b = len(a)+1, len(b)+1
    # DP 결과값을 저장할 리스트
    H = [[[0,[-1]] for _ in range(length_b)] ]
    # -가 들어갈 경우 Cost
    gap_cost = (5,2)

    # a의 길이동안 반복할 인덱스
    i = 1
    # 5번 이상 감소하면 (불일치 혹은 gap) 루프를 탈출하기 위한 cnt 선언
    cnt = 0
    # 최대값의 위치를 저장하기위한 tuple
    maxtuple = (0,0)
    # 최대값 체크를 위한 변수
    maximum, befmax, aftmax = 0,0,0

    # 끝날 때 까지 반복하거나,
    # 최고점수를 지나고 연속해서 5개가 불일치 혹은 gap이 나오는지 확인.
    while i < length_a and cnt <= 5 :
        H.append([[0,[-1]]])
        aftmax = 0
        for j in range(1,length_b) :
            # 방향을 결정, -1 = 방향없음, 0 대각선, 1 위, 2 왼쪽
            # 동일 값이 여러개일 때를 대비하여 List로 구현
            direction = []
            # 일치 +3 불일치 -2
            match = H[i - 1][j - 1][0] + (3 if a[i-1] == b[j-1] else - 2)
            # Gap penalty -5, -2 위쪽 방향
            # 연속되는 경우 (바로 이전의 direction에 1이 존재하는경우)
            delete = H[i - 1][j][0] - (gap_cost[1] if 1 in H[i - 1][j][1] else gap_cost[0])
            # Gap penalty -5, -2 왼쪽 방향
            # 연속되는 경우 (바로 이전의 direction에 2가 존재하는경우)
            insert = H[i][j - 1][0] - (gap_cost[1] if 2 in H[i][j-1][1] else gap_cost[0])

            tmpmax = max(match, delete, insert, 0)

            # 가능한 모든 방향 추가
            if tmpmax == 0 : direction.append(-1)
            if tmpmax == match : direction.append(0)
            if tmpmax == delete : direction.append(1)
            if tmpmax == insert : direction.append(2)

            # 최대값 갱신
            if tmpmax > aftmax :
                aftmax = tmpmax
                tmptuple = (i,j)

            # 리스트에 추가
            H[i].append( [tmpmax, direction] )

        # 최대값과 최대값이 존재하는 tuple을 갱신
        if maximum < aftmax :
            maximum = aftmax
            maxtuple = tmptuple

        # 불일치, Gap이 생길 경우 값이 내려가기 때문에 이를 이용하여 5회 카운팅
        if befmax > aftmax :
            befmax = aftmax
            cnt += 1
        else :
            befmax = aftmax
            cnt = 0
        i += 1

    return maxtuple, H
```

위 함수 또한 smith-waterman 알고리즘의 일부이다.

알고리즘의 구현방식을 통해 직접 구현하였다.

메모리 관리에 신경 쓰지 못하여 너무 긴 염기서열을 사용할 경우 메모리 초과가 생기는 경우가 있다.

위의 함수 또한 최악의 경우 while문에서 equence1 의 최대 길이, for문에서 sequence2의 최대 길이 만큼 반복하므로 시간복잡도는 O(sequence1 의 최대 길이 * sequence2의 최대 길이) 이다.

■ 결과

시간 복잡도는 O(sequence1 의 최대 길이 * sequence2의 최대 길이) 가 되었다.
아래는 예제 파일을 통한 테스트 결과이다.

```
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#2$ python3 PA02.py pa2-test1.f
asta pa2-test2-1.fasta
 -  서열1 : pa2-test1.fasta, 2462320
 -  서열2 : pa2-test2-1.fasta, 20440
 -  부분서열 : 44

        #1
        seq1 ..CTAAACCCT--AAATTTTAAACCCTAAACCTCTGAATCCTTAATC..
        seq2 ..CGAAGCCGTCCAAA----AAGCCAAAAAACTTTAACGCCTTAATC..

        #2
        seq1 ..CTAAACCCTA-AACTTTAAACCCTAAACCTCTGAATCCTTAATC..
        seq2 ..CGAAGCCGTGCAAA---AAGCCAAAAAACTTTAACGCCTTAATC..

        #3
        seq1 ..CTAAACCCT-AAACTTTAAACCCTAAACCTCTGAATCCTTAATC..
        seq2 ..CGAAGCCGTCGAAA---AAGCCAAAAAACTTTAACGCCTTAATC..

        #4
        seq1 ..CTAAACCCTAAACCTTAAACCCTAAACCTCTGAATCCTTAATC..
        seq2 ..CGAAGCCGTGGAAA--AAGCCAAAAAACTTTAACGCCTTAATC..
```

```
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#2$ ls
PA02.py                              pa2-test1.fasta    pa2-test3-1.fasta
output_pa2-test1_pa2-test2-1.algin   pa2-test2-1.fasta  pa2-test3.fasta
pa2-test1-1.fasta                    pa2-test2.fasta
```

```
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#2$ cat output_pa2-test1_pa2-test2-1.algin
 -  서열1 : pa2-test1.fasta, 2462320
 -  서열2 : pa2-test2-1.fasta, 20440
 -  부분서열 : 44

        #1
        seq1 ..CTAAACCCT--AAATTTTAAACCCTAAACCTCTGAATCCTTAATC..
        seq2 ..CGAAGCCGTCCAAA----AAGCCAAAAAACTTTAACGCCTTAATC..

        #2
        seq1 ..CTAAACCCTA-AACTTTAAACCCTAAACCTCTGAATCCTTAATC..
        seq2 ..CGAAGCCGTGCAAA---AAGCCAAAAAACTTTAACGCCTTAATC..

        #3
        seq1 ..CTAAACCCT-AAACTTTAAACCCTAAACCTCTGAATCCTTAATC..
        seq2 ..CGAAGCCGTCGAAA---AAGCCAAAAAACTTTAACGCCTTAATC..

        #4
        seq1 ..CTAAACCCTAAACCTTAAACCCTAAACCTCTGAATCCTTAATC..
        seq2 ..CGAAGCCGTGGAAA--AAGCCAAAAAACTTTAACGCCTTAATC..
```

```
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#2$ ls
PA02.py                           pa2-test1.fasta    pa2-test3-1.fasta
output_pa2-test1_pa2-test2-1.algin  pa2-test2-1.fasta  pa2-test3.fasta
pa2-test1-1.fasta                 pa2-test2.fasta
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#2$ python3 PA02.py pa2-test3.fasta pa2-test1-1.fasta
 -  서열1 : pa2-test3.fasta, 1246770
 -  서열2 : pa2-test1-1.fasta, 2099
 -  부분서열 : 58

        #1
        seq1 ..AAACACGGAAACTTATATT-GATGAACCCCGTTCAT-CAATTCTCCATAGAAAGG--AGGT---GATCCAGCTGCACCTTCCGATACGG..
        seq2 ..ATAAATGAGATGTTGAATTTGAGGAAC--CTTTGATTCAATGATC-ATAGAAAAAAAAGGTTTTGAGTCAGT-GT-CGTTATGTTATGG..

        #2
        seq1 ..AAACACGGAAACTTATATT-GATGAACCCCGTTCA-TCAATTCTCCATAGAAAGG--AGGT---GATCCAGCTGCACCTTCCGATACGG..
        seq2 ..ATAAATGAGATGTTGAATTTGAGGAAC--CTTTGATTCAATGATC-ATAGAAAAAAAAGGTTTTGAGTCAGT-GT-CGTTATGTTATGG..

        #3
        seq1 ..AAACACGGAAACTTATATT-GATGAACCCCGTTCAT-CAATTCTCCATAGAAAGG--AGGT---GATCCAGCTGCACCTTCCGATACGG..
        seq2 ..ATAAATGAGATGTTGAATTTGAGGAAC--CTTTGATTCAATGAT-CATAGAAAAAAAAGGTTTTGAGTCAGT-GT-CGTTATGTTATGG..

        #4
        seq1 ..AAACACGGAAACTTATATT-GATGAACCCCGTTCA-TCAATTCTCCATAGAAAGG--AGGT---GATCCAGCTGCACCTTCCGATACGG..
        seq2 ..ATAAATGAGATGTTGAATTTGAGGAAC--CTTTGATTCAATGAT-CATAGAAAAAAAAGGTTTTGAGTCAGT-GT-CGTTATGTTATGG..

        #5
        seq1 ..AAACACGGAAACTTATATT-GATGAACCCCGTTCAT-CAATTCTCCATAGAAAG-G-AGGT---GATCCAGCTGCACCTTCCGATACGG..
        seq2 ..ATAAATGAGATGTTGAATTTGAGGAAC--CTTTGATTCAATGATC-ATAGAAAAATAAGGTTTTGAGTCAGT-GT-CGTTATGTTATGG..

lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#2$ ls
PA02.py                           pa2-test1-1.fasta  pa2-test2.fasta
output_pa2-test1_pa2-test2-1.algin  pa2-test1.fasta    pa2-test3-1.fasta
output_pa2-test3_pa2-test1-1.algin  pa2-test2-1.fasta  pa2-test3.fasta
```

```
lep@lep-Virtual-Machine:~/HW/Algorithm/Assignment#2$ cat output_pa2-test3_pa2-test1-1.algin
 -  서열1 : pa2-test3.fasta, 1246770
 -  서열2 : pa2-test1-1.fasta, 2099
 -  부분서열 : 58

        #1
        seq1 ..AAACACGGAAACTTATATT-GATGAACCCCGTTCAT-CAATTCTCCATAGAAAGG--AGGT---GATCCAGCTGCACCTTCCGATACGG..
        seq2 ..ATAAATGAGATGTTGAATTTGAGGAAC--CTTTGATTCAATGATC-ATAGAAAAAAAAGGTTTTGAGTCAGT-GT-CGTTATGTTATGG..

        #2
        seq1 ..AAACACGGAAACTTATATT-GATGAACCCCGTTCA-TCAATTCTCCATAGAAAGG--AGGT---GATCCAGCTGCACCTTCCGATACGG..
        seq2 ..ATAAATGAGATGTTGAATTTGAGGAAC--CTTTGATTCAATGATC-ATAGAAAAAAAAGGTTTTGAGTCAGT-GT-CGTTATGTTATGG..

        #3
        seq1 ..AAACACGGAAACTTATATT-GATGAACCCCGTTCAT-CAATTCTCCATAGAAAGG--AGGT---GATCCAGCTGCACCTTCCGATACGG..
        seq2 ..ATAAATGAGATGTTGAATTTGAGGAAC--CTTTGATTCAATGAT-CATAGAAAAAAAAGGTTTTGAGTCAGT-GT-CGTTATGTTATGG..

        #4
        seq1 ..AAACACGGAAACTTATATT-GATGAACCCCGTTCA-TCAATTCTCCATAGAAAGG--AGGT---GATCCAGCTGCACCTTCCGATACGG..
        seq2 ..ATAAATGAGATGTTGAATTTGAGGAAC--CTTTGATTCAATGAT-CATAGAAAAAAAAGGTTTTGAGTCAGT-GT-CGTTATGTTATGG..

        #5
        seq1 ..AAACACGGAAACTTATATT-GATGAACCCCGTTCAT-CAATTCTCCATAGAAAG-G-AGGT---GATCCAGCTGCACCTTCCGATACGG..
        seq2 ..ATAAATGAGATGTTGAATTTGAGGAAC--CTTTGATTCAATGATC-ATAGAAAAATAAGGTTTTGAGTCAGT-GT-CGTTATGTTATGG..
```