DAFTAR PUSTAKA

Al-Mahmudattussa'adah. (-). *Protein.* Dipetik Maret 6, 2020, dari File Edu: http://file.upi.edu/Direktori/fptk/Jur._pend._kesejahteraan_keluarga/19780 7162006042-AI_mahmudatussa%27adah/protein.pdf

Arifin, A. Y. (2004). Analisa Homologi Gen Penyandi Kode Genetik Sifat-Sifat Produksi Pada Ternak Sapi, Domba dan Kambing. *Skripsi* (pp. 37-38). Bogor: Institut Pertanian Bogor. Retrieved Maret 6, 2020

Arifin, M. A. (2018). Penerapan Algoritma Program Dinamis untuk Memaksimalkan Keuntungan Rumah Makan Padang. *Makalah IF2211 Strategi Algoritma*.

Arifin, Y. (2019). Klasterisasi Kemiripan Data Biner Senyawa Tanaman Jamu Menggunakan Operasi Logika dan Pemrograman Dinamis. *Skripsi*.

Azhar, M. (2016). *Biomolekul Sel : Karbohidrat, Protein dan Enzim.* Padang: UNP Press Padang.

Azizah, U. N. (2013). Perbandingan Detektor Tepi Prewit dan Detektor Tepi Laplacian Berdasarkan Kompleksitas Waktu dan Citra Hasil. 31-39.

Bu'ulölö, I. C., Simamora, N., Tampubolon, S., & Pinem, A. (2010). Sequence Alignment Menggunakan Algoritma Smith Waterman. *Seminar Nasional Politeknik Batam 2010, II*(2).

Budiman, I. C. (2009). Penyejajaran Lokal Barisan DNA dengan menggunakan Metode Smith Waterman. *Skripsi.* Jakarta.

Bustamam, A., Ardaneswari, G., Tasman, H., & Lestari, D. (2014). Performance Evaluation of Fast Smith-Waterman Algorithm for Sequence Database Searches using CUDA GPU-Based Parallel Computing. *Journal of Next Generation Information Technology*, 38-46.

Chan, A. (2004). *An analysis of pairwise sequence alignment algorithm complexities: Needleman-Wuncsh, Smith-Waterman, FASTA, BLAST, and gapped BLAST.* California: Stanford Univercity.

Himawan, F. A. (2013). Penjajaran Lokal Sekuen DNA Menggunakan Algoritme Smith-Waterman. *Skripsi.* Bogor.

Insani, N. (2006). Penerapan Metode Bagi dua (bisection) pada analisis pulang pokok (Break Even). *Seminar Nasional Mipa*.

Keum, J., Yoo, S., Lee, D., & Nam, H. (2015). Prediction of compound-target interactions of natural products using large-scale drug and protein information. *BMC Bioinformatics 2016*, 418-442.

Liu, Y., Hong, Y., Lin, C.-Y., & Hung, C.-L. (2015). Accelerating Smith-Waterman Alignment for Protein Database Search Using Frequency Distance Filtration Scheme Based on CPU-GPU Collaborative System. *International Journal of Genomics, Vol.2015*, 12 Pages.

Munir, R. (2013). *Algoritma Runut Balik*. Retrieved from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2006-2007/Algoritma%20Runut-balik%20(bagian%201).pdf

Nugroho, H. S., & Kusuma, W. A. (2018). Pengukuran Kemiripan Protein pada Ijah Webserver dengan Algoritme Smith Waterman Berbasis Komputasi Paralel Cuda GPU. *Prosiding Seminar Ilmiah Ilmu Komputer*, 2-10. Retrieved Februari 29, 2020

Probosari, E. (2019). Pengaruh protein diet terhadap indeks glikemik. *Journal of Nutrition and Health, 7*(1), 34-39.

Puspitaningrum, D., Pravitasari, A., & Ernawati. (2014). Implementasi Algoritma Smith Waterman pada local alignment daLam pencarian kesamaan pensejajaran barisan DNA (Studi Kasus : DNA tumor wilms). *Jurnal Pseudocode*, 172.

Rivai, M. A. (2011, 04 28). *Analisis Dan Implementasi Algoritma Runut Balik (Backtracking) Pada Permainan Magic Square.* Retrieved from Repositori Institusi Universitas Sumatera Utara: http://repository.usu.ac.id/handle/123456789/23460

Singh, V., Singh, A., Chand, R., & Kushwaha, C. (2011). Role of Bioinformatics in Agriculture and Sustainable Development. *International Journal of Bioinformatics Research, 3*(2), 221-226.

Smith, T., & Waterman, M. (1981). Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, *147*, 195-197.

Xiong, J. (2006). *Essential Bioinformatics.* New York: Cambridge University Press.

LAMPIRAN

```
from __future__ import division, print_function
from pexecute.process import ProcessLoom
import sys
import time
from datetime import datetime
from Bio import SeqIO
import numpy as np
import gzip

pt ={'match': 5, 'mismatch': -3, 'gap': -4}

def mch(alpha, beta):
    if alpha == beta:
        return pt['match']
    elif alpha == '-' or beta == '-':
        return pt['gap']
    else:
        return pt['mismatch']

def water(s1, s2):
    m, n = len(s1), len(s2)
    H = np.zeros((m+1, n+1))
    T = np.zeros((m+1, n+1))
    max_score = 0
    # Score, Pointer Matrix
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            mch_diag = mch(s1[i-1], s2[j-1])
            sc_diag = H[i-1][j-1] + mch_diag
            sc_left = H[i][j-1] + pt['gap']
            sc_up = H[i-1][j] + pt['gap']
            H[i][j] = max(0,sc_left, sc_up, sc_diag)
            if H[i][j] == 0: T[i][j] = 0
            if H[i][j] == sc_up: T[i][j] = 1
            if H[i][j] == sc_left: T[i][j] = 2
            if H[i][j] == sc_diag:
                if mch_diag == pt['match']:
                    T[i][j] = 4
                elif mch_diag == pt['mismatch']:
                    T[i][j] = 3
            if H[i][j] >= max_score:
                max_i = i
                max_j = j
                max_score = H[i][j]
```

```
align1, align2 = '', ''
i,j = max_i,max_j
index_start = (max_i,max_j)

score = 0

#Traceback
while T[i][j] != 0:
    if T[i][j] == 3 or T[i][j] == 4:
        if T[i][j] == 3:
            score -= 3
        elif T[i][j] == 4:
            score += 5
        a1 = s1[i-1]
        a2 = s2[j-1]
        i -= 1
        j -= 1
    elif T[i][j] == 2:
        score -= 4
        a1 = '-'
        a2 = s2[j-1]
        j -= 1
    elif T[i][j] == 1:
        score -= 4
        a1 = s1[i-1]
        a2 = '-'
        i -= 1

    align1 += a1
    align2 += a2

align1 = align1[::-1]
align2 = align2[::-1]
sym = ''
iden = 0
for i in range(len(align1)):
    a1 = align1[i]
    a2 = align2[i]
    if a1 == a2:
        sym += a1
        iden += 1
    elif a1 != a2 and a1 != '-' and a2 != '-':
        sym += ' '
    elif a1 == '-' or a2 == '-':
        sym += ' '
```

```
        return score

def index_maxscore(s1, s2, H, T):
    m, n = len(s1), len(s2)
    max_score = 0
    # Score, Pointer Matrix
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            mch_diag = mch(s1[i-1], s2[j-1])
            sc_diag = H[i-1][j-1] + mch_diag
            sc_left = H[i][j-1] + pt['gap']
            sc_up = H[i-1][j] + pt['gap']
            H[i][j] = max(0,sc_left, sc_up, sc_diag)
            if H[i][j] == 0: T[i][j] = 0
            if H[i][j] == sc_up: T[i][j] = 1
            if H[i][j] == sc_left: T[i][j] = 2
            if H[i][j] == sc_diag:
                if mch_diag == pt['match']:
                    T[i][j] = 4
                elif mch_diag == pt['mismatch']:
                    T[i][j] = 3
            if H[i][j] >= max_score:
                max_i = i
                max_j = j
                max_score = H[i][j]

    i,j = max_i,max_j

    result = np.where(H == max_score)
    index = list(zip(result[0], result[1]))

    return index

def water_local_a(s1, s2):
    m, n = len(s1), len(s2)
    H = np.zeros((m+1, n+1))
    T = np.zeros((m+1, n+1))

    index = index_maxscore(s1,s2,H,T)

    return H, T, index

def water_local_b(s1, s2, x):
    m, n = len(s1), len(s2)
    H = np.zeros((m+1, n+1))
    T = np.zeros((m+1, n+1))
```

```
    for b in range(0, len(H[:,n])):
        H[b][0] = x[b]

    index = index_maxscore(s1,s2,H,T)

    return H, T, index

def water_local_c(s1, s2, x):
    m, n = len(s1), len(s2)
    H = np.zeros((m+1, n+1))
    T = np.zeros((m+1, n+1))

    for b in range(0, len(H[m,:])):
        H[0][b] = x[b]

    index = index_maxscore(s1,s2,H,T)

    return H, T, index

def water_local_d(s1, s2, x, y):
    m, n = len(s1), len(s2)
    H = np.zeros((m+1, n+1))
    T = np.zeros((m+1, n+1))

    half_y = len(y)//2
    if len(y)%2 == 0 :
        half_y = half_y - 1
    for a in range(0, len(H[0,:])):
        H[0][a] = y[a+half_y]

    half_x = len(x)//2
    if len(x)%2 == 0 :
        half_x = half_x - 1
    for a in range(1, len(H[:,0])):
        H[a][0] = x[a+half_x]

    index = index_maxscore(s1,s2,H,T)

    return H, T, index
```

```
def water_new(s1, s2):
    m, n = len(s1), len(s2)

    s11 = s1[:m//2]
    s12 = s1[m//2:]
    s21 = s2[:n//2]
    s22 = s2[n//2:]

    ## Paralel

    X,Y = [],[]

    H_A, T_A, index_A = water_local_a(s11, s21)
    last_column = len(H_A[0])-1
    for a in range(0, len(H_A[:,last_column])):
        X.append(H_A[a][last_column])

    last_row = len(H_A)-1
    for a in range(0, len(H_A[last_row,:])):
        Y.append(H_A[last_row][a])

    loom = ProcessLoom(max_runner_cap=4)
    loom.add_function(water_local_b, [s11, s22, X], {}, 'result_B')
    loom.add_function(water_local_c, [s12, s21, Y], {}, 'result_C')

    output = loom.execute()

    H_B = output['result_B']['output'][0]
    T_B = output['result_B']['output'][1]
    index_B = output['result_B']['output'][2]

    last_row = len(H_B)-1
    for a in range(1, len(H_B[last_row,:])):
        Y.append(H_B[last_row][a])

    H_C = output['result_C']['output'][0]
    T_C = output['result_C']['output'][1]
    index_C = output['result_C']['output'][2]

    for a in range(1, len(H_C[:,last_column])):
        X.append(H_C[a][last_column])

    H_D, T_D, index_D = water_local_d(s12, s22, X, Y)

     ## Paralel
```

```
res = {
    'A' : (0, 0),
    'B' : (0, (n//2)),
    'C' : ((m//2), 0),
    'D' : ((m//2), (n//2))}

trace_index = []
for a in index_A:
    x = a[0]+res['A'][0]
    y = a[1]+res['A'][1]
    trace_index.append((x,y))
for b in index_B:
    x = b[0]+res['B'][0]
    y = b[1]+res['B'][1]
    trace_index.append((x,y))
for c in index_C:
    x = c[0]+res['C'][0]
    y = c[1]+res['C'][1]
    trace_index.append((x,y))
for d in index_D:
    x = d[0]+res['D'][0]
    y = d[1]+res['D'][1]
    trace_index.append((x,y))


H_AB = np.hstack((H_A, H_B))
H_CD = np.hstack((H_C, H_D))
H = np.vstack((H_AB, H_CD))
H = np.delete(H, (m//2)+1, 0)
H = np.delete(H, (n//2)+1, 1)

T_AB = np.hstack((T_A, T_B))
T_CD = np.hstack((T_C, T_D))
T = np.vstack((T_AB, T_CD))
T = np.delete(T, (m//2)+1, 0)
T = np.delete(T, (n//2)+1, 1)
score_list = []
align1_list = []
align2_list = []
for x in range(len(trace_index)):
    i = trace_index[x][0]
    j = trace_index[x][1]
    align1, align2 = '', ''

    score = 0

    #Traceback
```

```
    while T[i][j] != 0:
        if T[i][j] == 3 or T[i][j] == 4:
            if T[i][j] == 3:
                score -= 3
            elif T[i][j] == 4:
                score += 5
            a1 = s1[i-1]
            a2 = s2[j-1]
            i -= 1
            j -= 1
        elif T[i][j] == 2:
            score -= 4
            a1 = '-'
            a2 = s2[j-1]
            j -= 1
        elif T[i][j] == 1:
            score -= 4
            a1 = s1[i-1]
            a2 = '-'
            i -= 1


        align1 += a1
        align2 += a2

    score_list.append(score)
    align1_list.append(align1)
    align2_list.append(align2)

# print('List Score')
# print(score_list)

max_score = max(score_list)
index_max = np.argmax(score_list, axis=0)
index_start = trace_index[index_max]
align1 = align1_list[index_max]
align2 = align2_list[index_max]

align1 = align1[::-1]
align2 = align2[::-1]
sym = ''
iden = 0
for i in range(len(align1)):
    a1 = align1[i]
    a2 = align2[i]
    if a1 == a2:
        sym += a1
```

```
            iden += 1
        elif a1 != a2 and a1 != '-' and a2 != '-':
            sym += ' '
        elif a1 == '-' or a2 == '-':
            sym += ' '


    return max_score

def run_different_sw(seq1, seq2):

    print()
    print('===  Smith Waterman  ===')
    time_start_sw = time.perf_counter()
    water(seq1, seq2)
    time_finish_sw = time.perf_counter()
    print('SW Duration ' + str(time_finish_sw-time_start_sw))

    print()
    print('===  Smith Waterman NEW  ===')
    time_start_swn = time.perf_counter()
    water_new(seq1, seq2)
    time_finish_swn = time.perf_counter()
    print('SW New Duration ' + str(time_finish_swn-time_start_swn))


if __name__ == '__main__':

     allSeqs = []


    with gzip.open("transporter uniprot link.fasta.gz", "rt") as handle:
        for record in SeqIO.parse(handle, "fasta"):
            allSeqs.append(record)


    print('Starting ...')

    allSeqs = allSeqs[:232]

    # start write in file
    original_stdout = sys.stdout
    filename = f'result-{len(allSeqs)}seq.txt'
    sys.stdout = open(filename, 'w')

    print('Total Sequence : ', len(allSeqs))
    time_start = datetime.now()
    for m in range(len(allSeqs)):
        for n in range(len(allSeqs)):
```

```
            if m <= n:
                print()
                print('Sequence [{},{}]'.format(m,n))
                seq1 = allSeqs[m].seq
                seq2 = allSeqs[n].seq

                print('Size ({},{})'.format(len(seq1), len(seq2)))

                time_start_swn = time.perf_counter()
                score_swn, similarity_swn = water_new(seq1, seq2)
                time_finish_swn = time.perf_counter()

                time_start_sw = time.perf_counter()
                score_sw, similarity_sw = water(seq1, seq2)
                time_finish_sw = time.perf_counter()

              print('SWN - score:{:d} - similarity:{:.2f}% - time:{:.3f} sec'.format(score_swn,
similarity_swn, time_finish_swn-time_start_swn))
                print('SW  - score:{:d} - similarity:{:.2f}% - time:{:.3F} sec'.format(score_sw,
similarity_sw, time_finish_sw-time_start_sw))

    time_finish = datetime.now()
    time_total = time_finish-time_start
    print('\nTotal Time :', time_total)

    # stop write in file
    sys.stdout.close()
    sys.stdout = original_stdout

    print('Finish !!!')
    print('Total Time :', time_total)
```