LAPORAN TUGAS KECIL I
IF2211 STRATEGI ALGORITMA
PENYELESAIAN CYBERPUNK 2077 BREACH PROTOCOL DENGAN ALGORITMA BRUTE FORCE

Disusun oleh:

Thea Josephine Halim 13522012

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2024

A. Algoritma Brute Force

Konsep algoritma brute force digunakan untuk penyelesaian *problem Breach Protocol Cyberpunk 2077* dengan mencari semua kemungkinan jawaban yang bisa dibentuk. Dalam penerapan algoritma brute force, saya memutuskan untuk membentuk semua sekuens yang mungkin terlebih dahulu sebelum diproses, dengan begitu saya tidak perlu memikirkan transisi dari sebuah sekuens ke sekuens yang lain, yang juga dibatasi oleh ukuran dari buffer. Program akan menggunakan fungsi new_seq untuk *generate* semua sekuens yang mungkin dengan menggabungkan sekuens-sekuens soal, baik dengan penyambungan secara langsung maupun dengan penyambungan dengan *overlap*. Program akan menampilkan seluruh kemungkinan sekuens dan langsung diproses satu per satu dengan iterasi *while loop.* Perlu ditambahkan juga, supaya setiap token hanya dapat digunakan sekali setiap sekuens, disediakan sebuah *copy* matriks boolean sebagai *checklist.* Program akan mengecek apakah ukuran sekuens tersebut memenuhi ukuran buffer maksimal, jika melebihi akan langsung dilanjutkan ke sekuens berikutnya agar lebih cepat. Dilanjutkan dengan *check_vertical_initial* yang akan menentukan apakah token pada baris pertama berpotensi untuk dimasukkan ke dalam buffer dengan melakukan pengecekan untuk token berikutnya di arah vertikal. Jika token pada baris pertama tidak memenuhi, sebuah *counter* bernama *maincounterstop* akan bertambah, fungsinya untuk menghentikan proses pencarian jika di baris pertama tidak ada token yang memenuhi. Setelah menemukan token yang sesuai, program akan memasukkan token yang mungkin ke dalam *curr_buffer*, list yang akan menyimpan buffer sementara ketika diproses. Perlu juga diperhatikan untuk penanganan khusus buffer berukuran maksimum 1 atau panjang sekuens yang hanya 1. Setiap kasus khusus sisa token seperti itu akan ditangani oleh *check_next_withindex* dan *last_token.* Jika ukuran buffer dan sekuens lebih dari satu, program akan lanjut dengan pengambilan token dari arah vertikal. Proses yang sama terjadi, hanya saja untuk kali ini digunakan *check_next_horizontal*. Begitu pula dengan pengambilan dari arah horizontal, proses tetap sama, terdapat penanganan kasus khusus token terakhir dan

pengambilan token seperti biasa. Perlu dicantumkan juga bahwa setiap memasukkan token ke dalam *curr_buffer,* perlu dilakukan pengecekan poinnya, dan jika point tersebut lebih besar, variabel *final_points, final_buffer,* dan *final_coords* akan diganti dengan sekuens yang lebih optimal. Pada program ini juga dibedakan 2 bagian karena boolean *first,* yang menunjukkan proses awal pengambilan token maupun selanjutnya. Hal ini perlu dibedakan karena adanya perubahan nilai indeks kolom idcol dan j. Akan tetapi, inti proses yang digunakan tetap sama.

Dengan melakukan brute force *generate* semua sekuens yang mungkin dibuat, kita bisa mencari manakah sekuens yang optimal dan mungkin terbentuk. Menurut saya persoalan ini sama seperti pencarian nilai maksimal angka pada suatu list, sebab kita melakukan pengecekan poin setiap sekuens, walaupun kita perlu juga memverifikasi apakah sekuens tersebut mungkin untuk dibentuk atau tidak.

B. Source Code

Catatan: mungkin bisa kurang update, untuk code cek repository, tetapi inti penjelasan sama

1. Fungsi-fungsi

Kode	Keterangan
`# Check if there is a token seq in there -> boolean` `def check_vertical_initial(idcol,token,matrix,sumrow):  # special for first token in a seq` `    for i in range (0,sumrow):` `        if (matrix[i][idcol] == token):` `            return True` `    return False` `def check_next_vertical(idrow,idcol,token,matrix,sumrow):` `    for i in range (sumrow):` `        if (matrix[i][idcol] == token) :` `           return True` `    return False` `def check_next_horizontal(idrow,idcol,token,matrix,sumcol):`	Fungsi boolean menentukan apakah sebuah token dalam matriks dapat kita masukkan ke curr_buffer ketika iterasi

`` for i in range (0,sumcol): if (matrix[idrow][i] == token): return True return False ```	
`` # Ensure use inputs a positive int def positive_input(prompt): while True: try: user_input = int(input(prompt)) if user_input > 0: return user_input else: print("Value has to be bigger than 0") except ValueError: print("Invalid input. Please enter a valid integer.") ```	Fungsi untuk memastikan user input angka yang valid, lebih dari 0.
`` # count total points from a buffer def count_points(buffer,arr_of_seq,arr_of_points): res = 0 for i in range (len(arr_of_seq)): if sublist(buffer,arr_of_seq[i]): res+=arr_of_points[i] return res # Decide to change the max point or not def max_point(curr_buffer,curr_coords,list_of_sequences,points,final_points,final_buffer,final_coords): curr_points = count_points(curr_buffer,list_of_sequences,points) if final_points < curr_points: final_points = curr_points final_coords = curr_coords final_buffer = curr_buffer return final_points,final_buffer,final_coords ```	Fungsi-fungsi untuk menghitung poin dalam sebuah buffer serta apakah perlu melakukan pengubahan final_buffer, final_points, dan final_coords
`` # To know if a sublist is a part of main_list def sublist(main_list, sublist): for i in range(len(main_list) - len(sublist) + 1): if main_list[i:i+len(sublist)] == sublist: return True return False ```	Fungsi untuk menentukan apakah sublist merupakan bagian pada main_list dengan tetap memperhatikan keterurutan

``# for last token needed def last_token(curr_buffer,final_points,list_of_seq uences,token,points,final_buffer,final_coords,c urr_coords): curr_buffer.append(token) final_points, final_buffer,final_coords = max_point(curr_buffer,curr_coords,list_of_seque nces,points,final_points,final_buffer,final_coo rds) return final_buffer,final_points,final_coords # for last token needed, check if the last token is there or not def check_next_withindex(matrix,token,cols,rows,id, type): if type == 0: # check horizontal for i in range (cols): if matrix[id][i]==token: return True,i return False,0 else: # check vertical for i in range (rows): if matrix[i][id]==token: return True,i return False,0```	Fungsi untuk menangani token terakhir pada sebuah sekuens
``# list all possible sequences def new_seq(sequences, curr_sequence=None, remaining_sequences=None, size_limit=None, all_sequences=None): if curr_sequence is None: curr_sequence = [] if remaining_sequences is None: remaining_sequences = sequences.copy() if all_sequences is None: all_sequences = [] if curr_sequence and len(curr_sequence) <= size_limit: all_sequences.append(curr_sequence.copy()) if len(curr_sequence) >= size_limit: return for i, seq in enumerate(remaining_sequences): overlap_len = find_overlap(curr_sequence, seq) if overlap_len > 0: new_sequence = curr_sequence + seq[overlap_len:] new_remaining =```	Fungsi untuk *generate* semua sekuens yang mungkin

<pre>remaining_sequences[:i] + remaining_sequences[i + 1:] new_seq(sequences, new_sequence, new_remaining, size_limit, all_sequences) else: new_sequence = curr_sequence + seq new_remaining = remaining_sequences[:i] + remaining_sequences[i + 1:] new_seq(sequences, new_sequence, new_remaining, size_limit, all_sequences) return all_sequences # count the number of tokens overlapping def find_overlap(seq1, seq2): for i in range(1, min(len(seq1), len(seq2)) + 1): if seq1[-i:] == seq2[:i]: return i return 0</pre>	

Tabel 2.1 Fungsi-fungsi

2. Algoritma Utama

Code	Keterangan
<pre>import time import random print("Welcome to the Cyberpunk 2077 Hacking Minigame Solver! 😈\n") print("Made by Thea Josephine 13522012 :>\n") print("Do you want to use an input file or customized random tokens? (1/2)\n") print("1. Input file (.txt)\n") print("2. Customized and randomized (CLI)\n")</pre>	Opening program, input file/CLI
<pre>while True: byCLI_or_file = input("=> ") if byCLI_or_file == "1": name = input("Enter input file name (ex: input.txt) => ") try: with open(name, 'r') as file: buffer_size = int(file.readline().strip()) dimensions_line = file.readline().strip() rows, cols = map(int, dimensions_line.split()) matrix = [] for i in range(0, rows):</pre>	Proses pembacaan input dari file dan juga random generated CLI. Untuk random points dibatasi dari -30 hingga 100 dengan kelipatan 10

```
matrix.append(list(map(str,
file.readline().split()[:cols])))
                num_of_sequences =
int(file.readline().strip())
                list_of_sequences = []
                points = []
                for i in range(0,
num_of_sequences):

list_of_sequences.append(list(map(str,
file.readline().split())))
                    points.extend([int(x) for
x in file.readline().split()])
                break
        except FileNotFoundError:
            print(f"The input file named
'{name}' was not found.")
    elif (byCLI_or_file == "2"):
        def get_tokens():
            unique_tokens = int(input("Number
of unique tokens: "))
            while True:
                tokens = list(map(str,
input(f"Enter {unique_tokens} tokens
separated by space: ").split()))
                if len(tokens) ==
unique_tokens:
                    return tokens
                else:
                    print(f"Please enter
exactly {unique_tokens} number of unique
tokens. Try again.")

        tokens = None
        matrix_values = None
        num_of_sequences = None
        seq_max_size = None
        buffer_size = None

        questions = [
            {"key": "tokens", "prompt":
"Enter tokens", "values": None},
            {"key": "matrix", "prompt": "The
size of the matrix rows x cols", "values":
None},
            {"key": "num_of_sequences",
"prompt": "How many sequence(s)", "values":
None},
            {"key": "seq_max_size", "prompt":
"Sequence max size", "values": None},
            {"key": "buffer_size", "prompt":
"Buffer size", "values": None}
        ]

        random.shuffle(questions)
```

```
        for question in questions:
            key = question["key"]
            prompt = question["prompt"]

            if "tokens" in key:
                tokens = get_tokens()
            elif "matrix" in key:
                ins = input(f"{prompt}: ")
                inssplit = ins.split()
                rows = int(inssplit[0])
                cols = int(inssplit[1])
            elif "buffer_size" in key:
                buffer_size =
positive_input(f"{prompt}: ")
            elif "num_of_sequences" in key:
                num_of_sequences =
positive_input(f"{prompt}: ")
            elif "seq_max_size" in key:
                seq_max_size =
positive_input(f"{prompt}: ")

        # Randomized matrix
        matrix = [[None for _ in range(cols)]
for _ in range(rows)]

        for i in range (rows):
            for j in range (cols):
                matrix[i][j] =
random.choice(tokens.copy())

        print("Randomized matrix:\n")
        print('\n'.join(['\t'.join([str(cell)
for cell in row]) for row in matrix]))

        # Randomized list of seq
        print(f"There is {num_of_sequences}
amount of sequences:")
        list_of_sequences = []
        for i in range (num_of_sequences):
            seq =  random.sample(tokens,
k=random.randint(2, seq_max_size))
            list_of_sequences.append(seq)
        print('\n'.join(['\t'.join([str(cell)
for cell in row]) for row in
list_of_sequences]))

        # Randomized list of points
        points = []
        for i in range (num_of_sequences):
            pts =
random.choice([-30,-20,-10,0,10,20,30,40,50,6
0,70,80,90,100])
            points.append(pts)
        print("Points: ",points)

        break
```

`` else: print("Wrong input, try again.\n") # Display reading file inputs print("\nBuffer size:", buffer_size) print("Row:", rows) print("Col:", cols) print("\nMatrix:") print('\n'.join(['\t'.join([str(cell) for cell in row]) for row in matrix])) print("\nSequences:", num_of_sequences) print('\n'.join(['\t'.join([str(cell) for cell in row]) for row in list_of_sequences])) print("Points",points) print("") ```	
`` print("*˚*•.¸¸☆•.¸¸*˚*˚*•.¸¸☆•.¸¸*˚* All Sequences *˚*•.¸¸☆•.¸¸*˚*˚*•.¸¸☆•.¸¸*˚*") list_of_sequences = new_seq(list_of_sequences, size_limit=buffer_size) print('\n'.join(['\t'.join([str(cell) for cell in row]) for row in list_of_sequences])) points = [0 for _ in range (len(list_of_sequences))] for j in range (len(old_point)): for i in range (len(list_of_sequences)): if list_of_sequences[i]==old_seq[j] and checklist[j]==True: points[i] = old_point[j] checklist[j] = False print(points) num_of_sequences = len(list_of_sequences) ```	Menyusun semua sekuens yang mungkin dan baru saja dibuat oleh fungsi new_seq. Setiap sekuens baru yang dibuat diberi poin 0 agar tidak terhitung berkali-kali oleh count_points (diiterasi di semua sekuens)
`` for seq in range (num_of_sequences): finish = False id_seq = 0 temp_seq = seq print("") print("Sequence ke-" + str(seq)) print("☆*: .｡. oo .｡.:*☆") print("Here is the last optimal buffer rn:") print(final_buffer) if (len(list_of_sequences[seq])<=buffer_size): curr_buffer = [] stop = False mainstop = False maincounterstop = 0 checklist = [True for _ in range (num_of_sequences)] ```	Algoritma utama, cukup panjang karena adanya perbedaan indeks kolom. Pertama program akan melakukan iterasi untuk sekuens pertama, dimulai dengan pemilihan elemen pada baris pertama. Selanjutnya akan dilanjutkan dengan pemilihan token pada arah vertikal, dilakukan hal yang sama, token yang terverifikasi akan di append ke curr_buffer. Proses akan berlanjut terus-menerus hingga tidak terdapat jawaban (ada counter gagal) atau jawaban ditemukan.

```
        if mainstop:
            print("*´¨`-•-  *,,, Untuk
sequence ini tidak ada solusi\n\n")
            break
        if finish:
            break
        for idcol in range (cols):    # check
mulai dr baris pertama
            if finish:
                break
            curr_buffer = []
            curr_coords = []
            mused = [[True for _ in
range(cols)] for _ in range(rows)]
            cekveriini =
check_vertical_initial(idcol,list_of_sequence
s[seq][0],matrix,rows)
            if maincounterstop==cols:
                mainstop=True
            if cekveriini==False:
                maincounterstop+=1
            # Only one buffer space available
            if ((buffer_size==1) or
(len(list_of_sequences[seq])==1)):
                one_buffer,idcol =
check_next_withindex(matrix,list_of_sequences
[seq][0],cols,rows,0,0)
                if one_buffer:  # token
exists on the first row

final_buffer,final_points,final_coords =
last_token(curr_buffer,final_points,list_of_s
equences,matrix[0][idcol],points,final_buffer
,final_coords,curr_coords)

final_coords.append((0,idcol))
                    checklist[seq]=False
                    finish = True
                    break
                else:           # token
wasn't found on the first row
                    mainstop=True
                    break
            if cekveriini:
                if
(buffer_size==len(list_of_sequences[seq]) and
matrix[0][idcol]==list_of_sequences[seq][0])
or
(buffer_size!=len(list_of_sequences[seq])):

curr_buffer.append(matrix[0][idcol])

curr_coords.append((0,idcol))
                    id_seq+=1
                    mused[0][idcol] = False
                    i = 0
```

```
                    first = True
                    countstop = 0
                    id_seq = len(curr_buffer)
                    if
curr_buffer[0]!=list_of_sequences[seq][0]:
                        id_seq-=1

                    # Choose from vertical
                    while i < rows:
                        if finish:
                            break
                        if
check_seq(curr_buffer,list_of_sequences) or
len(curr_buffer)==buffer_size:
                            finish=True
                            break
                        if first:
                            copy_mused =
mused
                            # Last spot to
avoid getting id_seq indexing error (choosing
from vertical)
                            if
id_seq==len(list_of_sequences[seq])-1: # last
spot

one_buffer,idx =
check_next_withindex(matrix,list_of_sequences
[seq][id_seq],cols,rows,idcol,1)
                                if one_buffer
and mused[idx][idcol]!=False:

curr_coords.append((idx,idcol))

final_buffer,final_points,final_coords =
last_token(curr_buffer,final_points,list_of_s
equences,list_of_sequences[seq][id_seq],point
s,final_buffer,final_coords,curr_coords)

checklist[seq]=False
                                finish = True
                                break
                            cekhori =
check_next_horizontal(i,idcol,list_of_sequenc
es[seq][id_seq],matrix,cols)
                            if cekhori==False
or matrix[i][idcol] !=
list_of_sequences[seq][id_seq]:
                                countstop+=1
                            if
countstop==rows:

maincounterstop+=1
                                break
                            if cekhori and
mused[i][idcol] != False and matrix[i][idcol]
```

```
== list_of_sequences[seq][id_seq]:

curr_buffer.append(matrix[i][idcol])

curr_coords.append((i,idcol))
                                id_seq+=1

mused[i][idcol] = False
                                final_points,
final_buffer, final_coords =
max_point(curr_buffer,curr_coords,list_of_seq
uences,points,final_points,final_buffer,final
_coords)

                                j = 0
                                countstop=0
                                # Choose from
horizontal
                                while j <
cols:
                                    if
check_seq(curr_buffer,list_of_sequences) or
len(curr_buffer)==buffer_size:

finish=True
                                        break
                                if
len(curr_buffer) < buffer_size:

sec_copy_mused = mused
                                        #
Last spot to avoid getting id_seq indexing
error (choosing from horizontal)
                                        if
id_seq==len(list_of_sequences[seq])-1: # last
spot

one_buffer,idx =
check_next_withindex(matrix,list_of_sequences
[seq][id_seq],cols,rows,i,0)

if one_buffer:

finish = True

curr_coords.append((i,idx))

final_buffer,final_points,final_coords =
last_token(curr_buffer,final_points,list_of_s
equences,list_of_sequences[seq][id_seq],point
s,final_buffer,final_coords,curr_coords)

finish=True

break
```

```
cekveri = check_next_vertical(i, j,
list_of_sequences[seq][id_seq], matrix, rows)
                                        if
cekveri==False or matrix[i][j] !=
list_of_sequences[seq][id_seq]:

countstop+=1
                                        if
countstop==cols:

stop = True

break
                                        if
cekveri and mused[i][j] != False and
matrix[i][j] ==
list_of_sequences[seq][id_seq]:

curr_buffer.append(matrix[i][j])

curr_coords.append((i,j))

id_seq+=1

mused[i][j] = False

final_points,final_buffer,final_coords =
max_point(curr_buffer,curr_coords,
list_of_sequences, points,
final_points,final_buffer,final_coords)
                                                i
= -1  # Reset i to 0 in the next iteration

first = False

break
                                            mused
= sec_copy_mused
                                    else:
                                        break
                                    j += 1
                                mused =
copy_mused
                            i += 1
                        else:
                            # Choose from
vertical
                            countstop=0
                            while i < rows:
                                if finish:
                                    break
                                if
check_seq(curr_buffer,list_of_sequences) or
len(curr_buffer)==buffer_size:

finish=True
```

```
                                    break
                                copy_mused =
mused
                                # Last spot
to avoid getting id_seq indexing error
(choosing from vertical)
                                if
id_seq==len(list_of_sequences[seq])-1: # last
spot

one_buffer,idx =
check_next_withindex(matrix,list_of_sequences
[seq][id_seq],cols,rows,j,1)
                                    if
one_buffer:

curr_coords.append((idx,j))

final_buffer,final_points,final_coords =
last_token(curr_buffer,final_points,list_of_s
equences,list_of_sequences[seq][id_seq],point
s,final_buffer,final_coords,curr_coords)

checklist[seq]=False
                                    finish =
True
                                    break
                                cekhori =
check_next_horizontal(i,idcol,list_of_sequenc
es[seq][id_seq],matrix,cols)
                                if
cekhori==False or matrix[i][idcol] !=
list_of_sequences[seq][id_seq]:

countstop+=1
                                if
countstop==rows:

maincounterstop+=1
                                    break


                                if cekhori
and mused[i][j] != False and matrix[i][j] ==
list_of_sequences[seq][id_seq]:

curr_buffer.append(matrix[i][j])

curr_coords.append((i,j))
                                    id_seq+=1

mused[i][j] = False

final_points, final_buffer, final_coords =
max_point(curr_buffer,curr_coords,list_of_seq
uences,points,final_points,final_buffer,final
_coords)
```

```
                                    j = 0

countstop=0
                                    # Choose
from horizontal
                                    while j <
cols:
                                        if
check_seq(curr_buffer,list_of_sequences) or
len(curr_buffer)==buffer_size:

finish=True

break
                                        if
len(curr_buffer) < buffer_size:

sec_copy_mused = mused

if id_seq==len(list_of_sequences[seq])-1: #
last spot

one_buffer,idx =
check_next_withindex(matrix,list_of_sequences
[seq][id_seq],cols,rows,i,0)

if one_buffer:

curr_coords.append((i,idx))

final_buffer,final_points,final_coords =
last_token(curr_buffer,final_points,list_of_s
equences,list_of_sequences[seq][id_seq],point
s,final_buffer,final_coords,curr_coords)

checklist[seq]=False

finish=True

break

cekveri = check_next_vertical(i, j,
list_of_sequences[seq][id_seq], matrix, rows)

if cekveri==False or matrix[i][j] !=
list_of_sequences[seq][id_seq]:

countstop+=1

if countstop==cols:

stop = True

break

if cekveri and mused[j][i] != False and
```

```
matrix[i][j] ==
list_of_sequences[seq][id_seq]:

curr_buffer.append(matrix[i][j])

curr_coords.append((i,j))

id_seq+=1

mused[i][j] = False

final_points,final_buffer,final_coords =
max_point(curr_buffer,curr_coords,
list_of_sequences, points,
final_points,final_buffer,final_coords)

i = -1  # Reset i to 0 in the next iteration

first = False

break

mused = sec_copy_mused
                                        else:

break
                                        j +=
1
                                    mused =
copy_mused
                                i += 1
    print("")
```

```
# Output on terminal
print("ˏˋ°•*⁀➷ Results ˚.·.˚.·.˚ˊˎ\n")

print("Points obtained: " +
str(final_points))
if final_points==0:
    print("Not worth/no possible solution")
else:
    print("Final buffer:")
    print(' '.join(final_buffer))
    print("Final Coordinates:")
    print(final_coords)
print(str(elapsed_time) + " ms\n")
while(True):
    prompt = input("Do you want to save the
result? (y/n) => ")
    if (prompt == "y"):
        output_name = input("Output file
name? (ex: output.txt) => ")
        with open(output_name, "w") as
output:
            output.write(str(final_points) +
```

Output pada terminal dan proses penyimpanan jawaban ke txt

```
"\n")
            output.write(str('
'.join(final_buffer)) + "\n")
            for c in final_coords:
                output.write(f"{c}\n")
            output.write(f"{elapsed_time}
ms")
        output.close()
        print("Saved successfully.\n")
        break
    elif (prompt == "n"):
        print("You did not save the
result.\n")
        break
    else:
        print("Wrong input, try again.\n")
print("Thank you for trying my program :>")
```

C. Contoh Output

1. Tanpa tampilan

No	Input	Output
1.	5 6 6 7A 55 E9 E9 1C 50 53 7A 1C 7A E9 55 51 1C 1C 55 E9 BD BD 1C 7A 1C 55 BD BD 5X BD 7A 1C 1C 1C 51 55 7A 55 7A 3 7A BD 1C 7A E9 20 BD E9 1C E9 30 BD E9 5	---------- Results: Points obtained: 35 Final buffer: 50 BD E9 1C E9 Final Coordinates: [(0, 5), (2, 5), (2, 4), (0, 4), (0, 2)] 0.24570083618164062 ms (tes2.txt)

<table>
<tr>
<td>2.</td>
<td>

```
6
6 6
7A 55 E9 E9 1C 50
53 7A 1C 7A E9 55
51 1C 1C 55 E9 BD
BD 1C 7A 1C 55 BD
BD 5X BD 7A 1C 1C
1C 51 55 7A 55 7A
3
7A BD
20
BD 1C 5X BD 7A 1C
30
BD 1C
5
```

</td>
<td>

```
---------------- Results:

Points obtained: 25
Final buffer:
7A BD 1C
Final Coordinates:
[(0, 0), (3, 0), (3, 1)]
0.035260915756225586 ms
```

(coop.txt)
</td>
</tr>
<tr>
<td>3.</td>
<td>

```
3
6 6
7A 55 E9 E9 1C 50
53 7A 1C 7A E9 55
51 1C 1C 55 E9 BD
BD 1C 7A 1C 55 BD
BD 5X BD 7A 1C 1C
1C 51 55 7A 55 7A
2
7A 51 5X
20
BD E9 1C
30
```

</td>
<td>

```
---------------- Results:

Points obtained: 0
Not worth/no possible solution
0.21530723571777344 ms
```

(nosolulu.txt)
</td>
</tr>
</table>

4.	3 6 6 7A 55 E9 E9 1C 50 53 7A 1C 7A E9 55 51 1C 1C 55 E9 BD BD 1C 7A 1C 55 BD BD 5X BD 7A 1C 1C 1C 51 55 7A 55 7A 3 BD 7A 7A 20 E9 E9 30 1C E9 1C 1C 10	---------- Results: Points obtained: 0 Not worth/no possible solution 0.04777669906616211 ms (nosolulu2.txt)
5.	6 6 6 7A 55 E9 E9 1C 50 53 7A 1C 7A E9 55 51 1C 1C 55 E9 BD BD 1C 7A 1C 55 BD BD 5X BD 7A 1C 1C 1C 51 55 7A 55 7A 4 7A 53 55 BD 1C 50 55 20 BD 1C 10 1C 7A 30	---------- Results: Points obtained: 110 Final buffer: 7A 53 55 BD 1C 7A Final Coordinates: [(0, 0), (1, 0), (1, 5), (2, 5), (2, 1), (1, 1), (0, 1)] 1.8804693222045898 ms (gap.txt)

6.

```
3
6 6
7A 55 E9 E9 1C 50
53 7A 1C 7A E9 55
51 1C 1C 55 E9 BD
BD 1C 7A 1C 55 BD
BD 5X BD 7A 1C 1C
1C 51 55 7A 55 7A
4
7A
20
BD 1C
10
7A BD
-30
BD 5X
20
```

```
---------------- Results:

Points obtained: 20
Final buffer:
7A
Final Coordinates:
[(0, 0)]
0.2487952709197998 ms
```

(spc.txt)

7.

```
5
6 6
7A 55 E9 E9 1C 50
53 7A 1C 7A E9 55
51 1C 1C 55 E9 BD
BD 1C 7A 1C 55 BD
BD 5X BD 7A 1C 1C
1C 51 55 7A 55 7A
3
7A BD
20
BD 1C
30
BD 7A
35
```

```
---------------- Results:

Points obtained: 85
Final buffer:
7A BD 7A BD 1C
Final Coordinates:
[(0, 0), (3, 0), (3, 2), (4, 2), (4, 4)]
0.8093645572662354 ms
```

(spc2.txt)

8.

```
3
5 5
AA AA EE BB BB
DD DD AA CC BB
DD BB AA CC BB
CC BB AA BB EE
DD BB DD EE CC
3
CC BB AA EE
70
BB CC AA
80
DD BB CC AA
90
```

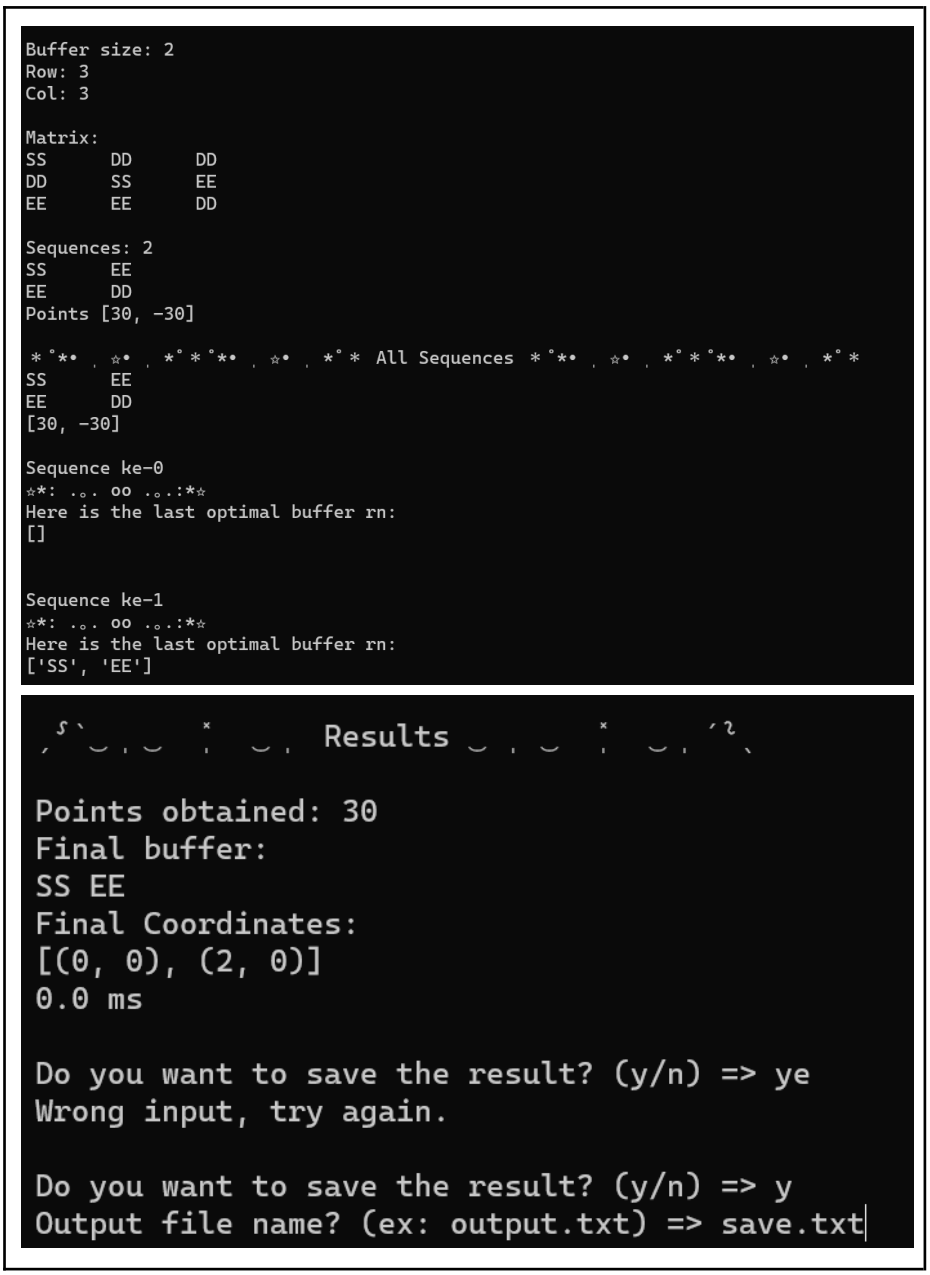
```
---------------- Results:

Points obtained: 80
Final buffer:
BB CC AA
Final Coordinates:
[(0, 3), (1, 3), (1, 2)]
0.02710723876953125 ms
```

(CLI.txt)

2. Full program

CLI Input

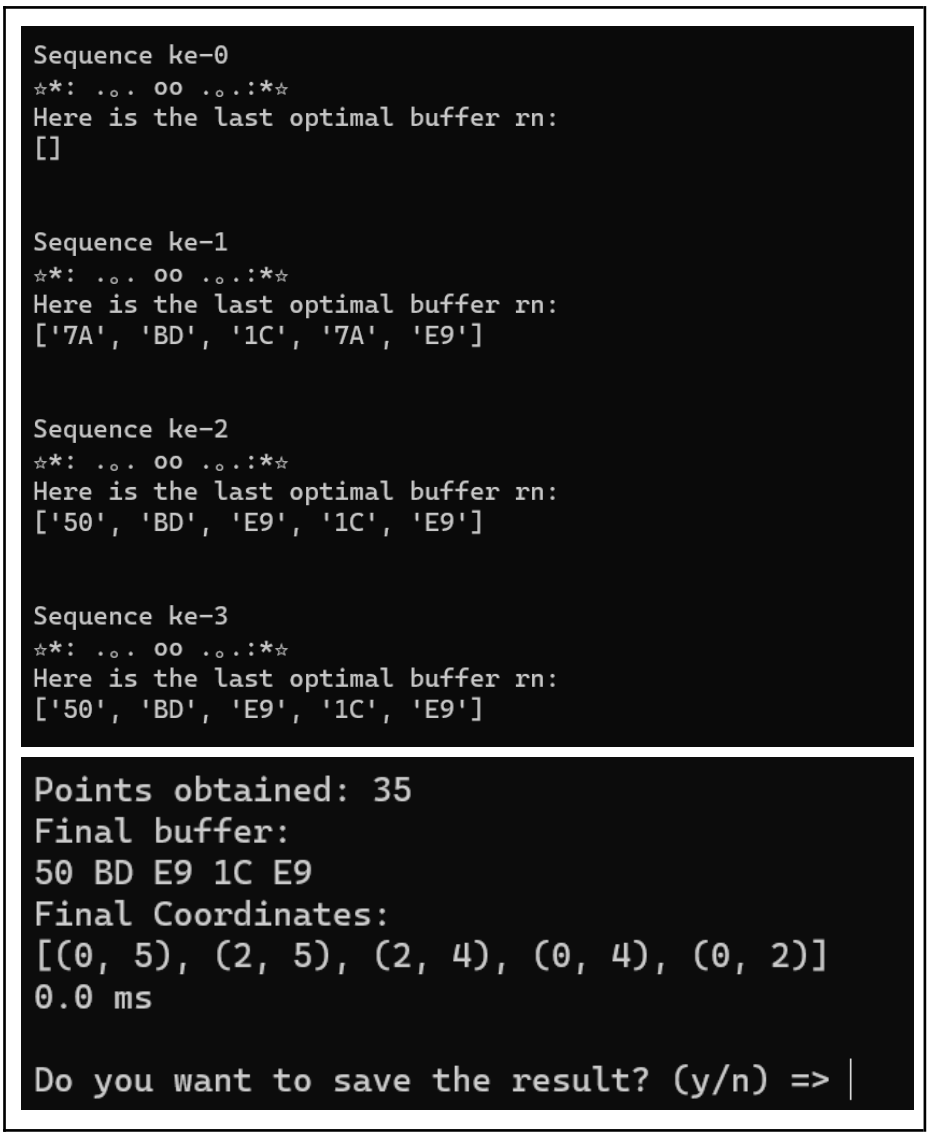
```
Welcome to the Cyberpunk 2077 Hacking Minigame Solver! 

Made by Thea Josephine 13522012 :>

Do you want to use an input file or customized random tokens? (1/2)

1. Input file (.txt)

2. Customized and randomized (CLI)

=> 2
The size of the matrix rows x cols: 3 3
How many sequence(s): 2
Sequence max size: 2
Number of unique tokens: 3
Enter 3 tokens separated by space: EE DD SS
Buffer size: 2
Randomized matrix:

SS      DD      DD
DD      SS      EE
EE      EE      DD
There is 2 amount of sequences:
SS      EE
EE      DD
Points:  [30, -30]
```


Buffer size: 2
Row: 3
Col: 3
Matrix:
SS DD DD
DD SS EE
EE EE DD
Sequences: 2
SS EE
EE DD
Points [30, -30]
All Sequences
SS EE
EE DD
[30, -30]
Sequence ke-0
Here is the last optimal buffer rn:
[]
Sequence ke-1
Here is the last optimal buffer rn:
['SS', 'EE']
Results
Points obtained: 30
Final buffer:
SS EE
Final Coordinates:
[(0, 0), (2, 0)]
0.0 ms
Do you want to save the result? (y/n) => ye
Wrong input, try again.
Do you want to save the result? (y/n) => y
Output file name? (ex: output.txt) => save.txt

File Input

```
Welcome to the Cyberpunk 2077 Hacking Minigame Solver! 🤖

Made by Thea Josephine 13522012 :>

Do you want to use an input file or customized random tokens? (1/2)

1. Input file (.txt)

2. Customized and randomized (CLI)

=> 1
Enter input file name (ex: input.txt) => kucing.txt
The input file named 'kucing.txt' was not found.
=> tes.txt
Wrong input, try again.

=> 1
Enter input file name (ex: input.txt) => tes.txt

Buffer size: 5
Row: 6
Col: 6
```

```
Matrix:
7A      55      E9      E9      1C      50
53      7A      1C      7A      E9      55
51      1C      1C      55      E9      BD
BD      1C      7A      1C      55      BD
BD      5X      BD      7A      1C      1C
1C      51      55      7A      55      7A

Sequences: 3
7A      BD      1C      7A      E9
BD      E9      1C      E9
BD      E9
Points [20, 30, 5]

 * ˚*•  ,  ☆•  ,  *˚ * ˚*•  ,  ☆•  ,  *˚ *  All Sequences  * ˚*•  ,  ☆•  ,  *˚ * ˚*•  ,  ☆•  ,  *˚ *
7A      BD      1C      7A      E9
BD      E9      1C      E9
BD      E9
BD      E9      1C      E9
[20, 30, 5, 0]
```



D. Checklist

Poin	Ya	Tidak
Program berhasil dikompilasi tanpa kesalahan	v	
Program berhasil dijalankan	v	
Program dapat membaca masukan berkas .txt	v	
Program dapat menghasilkan masukan secara acak	v	
Solusi yang diberikan program	v	

optimal		
Program dapat menyimpan solusi dalam berkas .txt	v	
Program memiliki GUI (Bonus)		v

E. Link Akses

1. Repository: https://github.com/pandaandsushi/Tucil1_13522012
2. References:
   https://stackoverflow.com/questions/30055830/pythonic-way-to-merge-two-overlapping-lists-preserving-order
   https://stackoverflow.com/questions/5458048/how-can-i-make-a-python-script-standalone-executable-to-run-without-any-dependen
   https://www.w3schools.com/python/python_howto_remove_duplicates.asp