BLG336E - Analysis of Algorithms II, Spring 2021 Homework 3

Local Sequence Alignment with Smith-Waterman

Student Name: Bahadır LÜLECİ
Student Number: 504201511

Date: 16/05/2021

IMPORTANT NOTE: Please compile your sour code with these flags, do NOT remove or change the flags.
Step 1:
```
g++ -std=c++11 -Wall -Werror 504201511.cpp -o 504201511
```
Step 2
(with no arguments):
```
./504201511
```

(with default values):
```
./504201511 strings.txt output.txt
```

(with specified values) val1=match_val, val2= mismatch_val, val3=gap_penalty
```
./504201511 strings.txt output.txt val1 val2 val3
```

1) Problem Fomulation:
a. Pseudo-code

```
class MatrixPair
public items
    string pairOne
    string pairTwo
    int score
    vector<vector<int>> matrixOfPairs
public items
    MatrixPair(string pairOne, string pairTwo)
    void MatrixFiller(int match_val, int mismatch_penalty, int gap_penalty)
    void ScoreFinder(int match_val)
    void traceBack(int match_val)
endclass

Funtion MatrixPair
Pass In: pairOne, pairTwo
    Object score value = 0
    Object pairOne value= pairOne
    pairOneSize = pairOne's size()
    Object pairTwo value= pairTwo
    pairTwoSize = pairTwo's size()
    create matrixOfPairs  matrix  (pairTwoSize + 1  X pairOneSize + 1 )
intiliaze matrixOfPairs with zeros
Endfunction

Function MatrixFiller
Pass In: int match_val, int mismatch_penalty, int gap_penalty
  Define variables : left_box, right_box, diagonal
    for(i=1; i<pairTwo' size+1; i++){
        for(j=1; j<pairOne' size+1; j++){

            left_box = matrixOfPairs[i][j-1] + gap_penalty
            right_box = matrixOfPairs[i-1][j] + gap_penalty
            diagonal = matrixOfPairs[i-1][j-1]

     if(pairTwo[i-1] == pairOne[j-1]){
                diagonal += match_val
                matrixOfPairs[i][j] = max_val(left_box, right_box, diagonal)
                if(matrixOfPairs[i][j] < 0)
                matrixOfPairs[i][j] =  0
            }
            else{
                diagonal += mismatch_penalty
                matrixOfPairs[i][j] = max_val(left_box, right_box, diagonal)
                if(matrixOfPairs[i][j] < 0)
                    matrixOfPairs[i][j] =  0
            }
        }
    }
Endfunction

Function ScoreFinder
Pass In : matching_val
    for( i=0; i< pairTwo’s size+1); i++){
        it = find max_element(begin(matrixOfPairs[i]), end(matrixOfPairs[i]))
        if(it> score)
            score = it
    }
}
score = score / matching_val
Endfunction

Function traceBack
Pass In:  int mathcing_val
    count_of_longest_items = 0;

    for(i=0; i< pairTwo’s size + 1; i++){
        for(j=0; j< paiOne’s size + 1; j++){
            if(matrixOfPairs[i][j] == score*mathcing_val)
                count_of_longest_items ++
        }
    }
   Create  tracebackVector ( size = count_of_longest_items) //each elementof vector shall be character
vector

    temp_i = 0
    temp_j = 0
    counter = 0
    define int t_i , t_j

    while(count_of_longest_items != 0){
        for(i=temp_i; i< pairTwo’s size + 1; i++){
            forj=temp_j; j< paiOne’s size + 1; j++){
                if(matrixOfPairs[i][j] == score*matching_val){
                    int k = j-1
                    t_i = i
                    t_j = j
                     while(matrixOfPairs[t_i][t_j] != 0){
                        tracebackVector[counter].push_back(pairOne[k])
                        k --
                        t_i --
                        t_j --
                    }
                    temp_i = i+1
                    temp_j = 0
                    goto finish label
                }
            }
        }
        Finish label:
        Reverse tracebackVector[counter]
        counter ++
        count_of_longest_items --
    }

    Sort tracebackVector alphabetically

    iter = unique(tracebackVector)// find unique items
    tracebackVector.erase(iter, tracebackVector.end())  // delete not unique items
    for( i=0; i<tracebackVector' size ; i++){
        if(tracebackVector[i].size() == 0)
            tracebackVector.erase (tracebackVector.begin()+i)      //delete items in vector which hase 0
size.
    }
    //print part of function
    For (i=0 ; i< tracebackVector' size ; i++){
        if(tracebackVector[i]' size == 0)
            continue
        print << "\""
        for (vector<char>::const_iterator j = tracebackVector[i].begin(); j != tracebackVector[i].end(); ++j)
        {
                print << j
        }
        print << "\" "
    }
        print << "\n"
}

Function main
  Pass In: 5 argument  = input file name, output file name, match_val, mismatch_penalty, gap_penalty
    //default values. It would be more correct to use
    int match_val = 1;
    int mismatch_penalty = -2;
    int gap_penalty = -4;

if argument count equals to 6
match_val = argument 3
mismatch_penalty = argument 4
gap_penalty = argument 5

if argument count smaller then 2
out_f_name = “output.txt”
else
 out_f_name = argument 2


    Create string vector named inputStrings
    string fname

    //get file name into fname
if argument count smaller then 2
fname =”strings.txt”
else
    fname = argument [1]
    read file
    string line
    //reading every strings in the file inside line
        inputStrings push(line)
    //sort the vector alphabetically
    sort(inputStrings , compareFunction)

    vector<MatrixPair> AllObjectPairs;
    //add all string pairs in object vector
    for (i=0; i<inputStrings' size; i++){
        for(j=i+1; j<inputStrings' size j++)
        AllObjectPairs.push_back(MatrixPair(inputStrings[i], inputStrings[j]))
    }

    //@Start: traverse all objects and find their substrings
    for(i=0; i< AllObjectPairs'size; i++){
        AllObjectPairs[i].MatrixFiller()
        AllObjectPairs[i].ScoreFinder()
        print << AllObjectPairs[i].pairOne << " - "<<AllObjectPairs[i].pairTwo << newline
        print << "Score: " << AllObjectPairs[i].score << " " << "Sequence(s): "
        AllObjectPairs[i].traceBack();
    }
     //@End
    Exit succesfully

  Pass Out: objects and their scores and sequence(s)
Endfunction
```

b. Complexity of algorithm

First, let's examine all functions one by one.
n is the lenght of pair one string
m is the lenght of pair two string

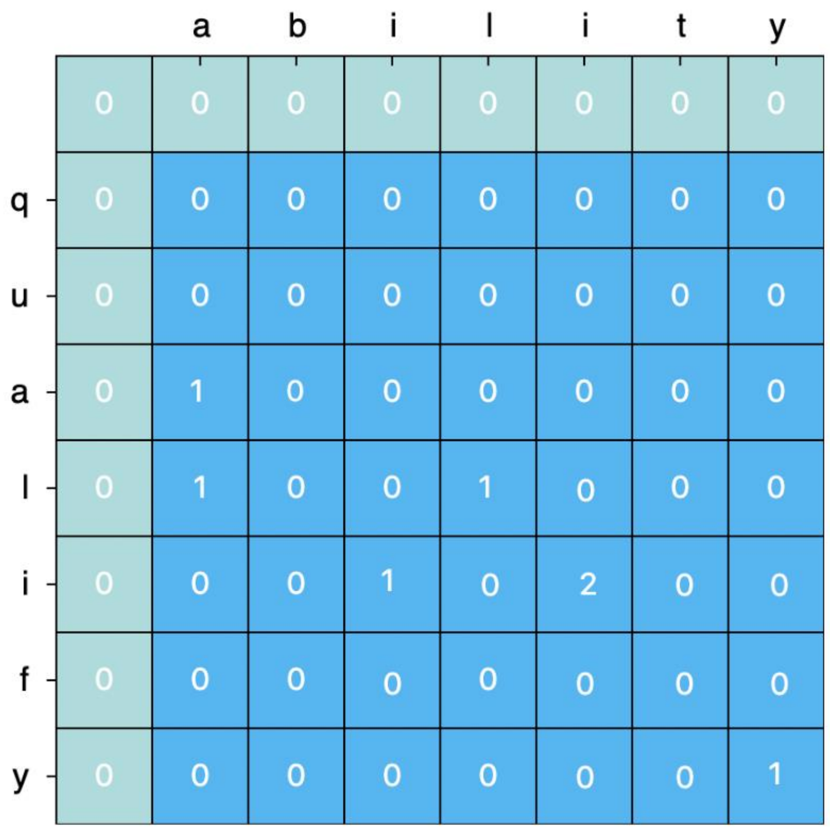
1) Function MatrixPair : has only 5 basic operations O(1)
2) Function MatrixFiller : has 2 nested for loops O(n*m)
3) Function ScoreFinder : O(m)
4) Function traceBack : has 1 for loop O(2*n*m)
5) Function main: the time complexity is O(n*m)

2) Analyze and compare the algorithm results in terms of:
a. The calculations made

Our goal is to find an optimal alignment between two substrings that maximizes the scoring function.
One approach to finding an optimal alignment would be to generate every single possible substring of the two strings and compare them against each other. A substring can be created from a longer string by choosing a starting and ending position within the longer string, and treating that as a shorter string. Creating all possible substrings of a string is computationally expensive, as for every starting position we choose, we can choose any ending position along the string. The longer the original string, the more possible substrings can be created from it. In addition to generating every possible substring, it would also be necessary to generate every possible variation of each of these substrings, in which one or more dashes are inserted in order to simulate the insertion/deletion of a character, to match one string against another. Once we have all of these potential substrings generated, we then have to go through all possible pairings of two substrings to calculate their alignment score, according to the scoring function we use. Overall, we can see that calculating all of these alignment scocres would be computationally expensive because of the sheer amount of potential substrings and combinations.
The essence of the Smith-Waterman Algorithm is to improve this naive approach using dynamic programming. This algorithm takes the two strings that we want to find optimal substring alignments for and aligns them into a matrix. This might seem strange at first, but this approach actually makes the problem much more efficient.

The problem actually desires longest common sequences. Therefore it is better to give very low number to mismatch penalty and gap. The important thing here is to add only the matching points. Finding the longest common sub-string will be more efficient with this way. The most important point is determining the score and traceback operations. In order to find them, the character that is common to one another must be kept in memory in dinamically. 3 important operations should be performed, their order is not important. The gap penalty score should be added to the value in the left box and the top box and recorded. If there is a match in the area on the matrix, the matching value should be added to the value in the diagonal area and kept in memory. If there is no match, the mismatching score should be added and kept in memory. It should be written in the relevant location of the matrix working on the largest value among these 3 values. Then after the score is found and the trackback function is run and the longest common sub-strings are found.
An example is given below:

Matching value = 1, mismatch_penalty = -2, gap_penalty= -4
ability - qualify
Score: 2 Sequence(s): "li"



		a	b	i	l	i	t	y
	0	0	0	0	0	0	0	0
q	0	0	0	0	0	0	0	0
u	0	0	0	0	0	0	0	0
a	0	1	0	0	0	0	0	0
l	0	1	0	0	1	0	0	0
i	0	0	0	1	0	2	0	0
f	0	0	0	0	0	0	0	0
y	0	0	0	0	0	0	0	1

Figure 1 Example of a Matching Matrix

This graph displays a comparison between a brute-force approach to find optimal substring alignment and the Smith-Waterman algorithm. The naive, brute-force approach entails comparing every single possible edit (i.e. insertion of a dash) of every single possible substring of the two strings against each other. When we consider the lengths of the two strings to be n and m, the naive approach has a runtime of O(n3m3), while the Smith-Waterman algorithm has a runtime O(nm).

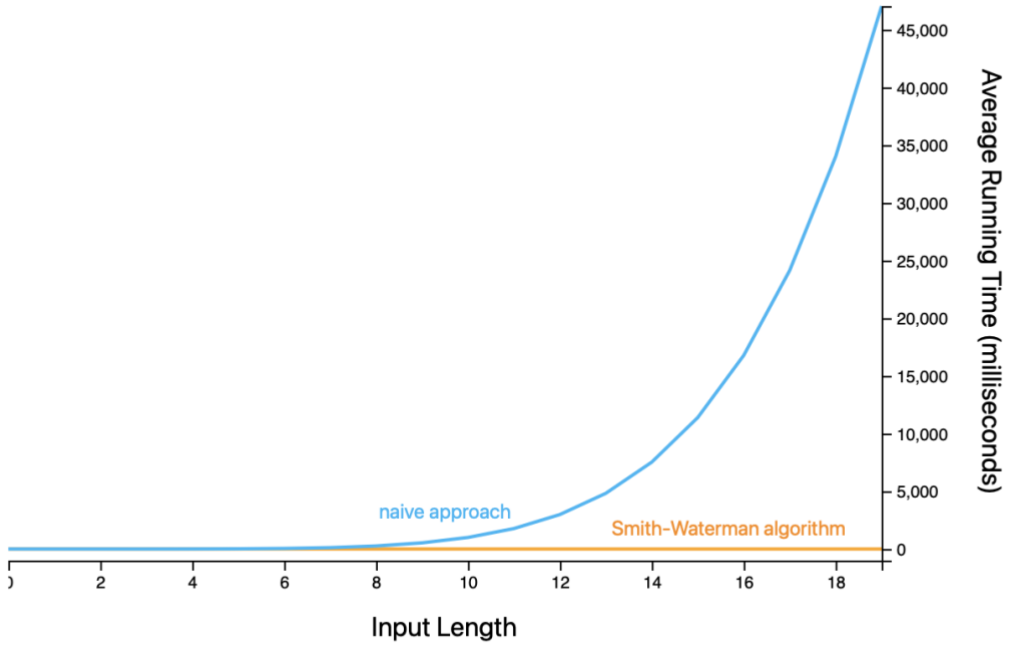


Figure 2 Runtime Comparison: Naive Approach (Brute-Force) vs Smith-Waterman Algorithm

b. The maximum number of calculation results kept in the memory

```
for(int i=0; i<int(AllObjectPairs.size()); i++){
    AllObjectPairs[i].MatrixFiller();   //190 calculation
    AllObjectPairs[i].ScoreFinder();  //190 calculation
    cout << AllObjectPairs[i].pairOne << " - "<<AllObjectPairs[i].pairTwo << "\n";
    cout << "Score: " << AllObjectPairs[i].score << " " << "Sequence(s): ";
    AllObjectPairs[i].traceBack(); // 190 calculation
}
```

The maximum number of calculation results kept in the memory is 190 + 190 + 190 = 570.
if we want to formulize this:

n = number of input strings

Maximum number of calculation = 3* (n)*(n+1)/2

c. The running time

The time complexity has calculated in chapter 2.b as O(n*m). The following code has been added to the beginning and end of the main function to find the running time.

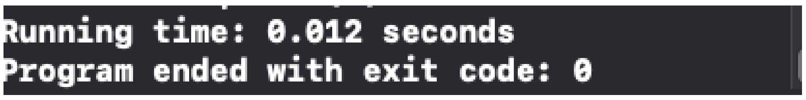
```
chrono::time_point<chrono::high_resolution_clock> start = chrono::high_resolution_clock::now();

.
.
.
.

chrono::time_point<chrono::high_resolution_clock> stop = chrono::high_resolution_clock::now();

auto duration = chrono::duration_cast<chrono::microseconds>(stop - start);

cout << "Running time: " << setprecision(2) << duration.count() * (1e-6) << " seconds" << endl;
```



```
Running time: 0.012 seconds
Program ended with exit code: 0
```

Figure 3 Running Time

d. Results of the code

Input: Strings.txt

information
fanimatio
malfunction
automative
medication
paramedic
automata
flammable
compatibility
ability
certificate
directive
conformity
component
examination
predict
disctrict
fountain
personality
qualify

Output:

```
ability - automata

Score: 1 Sequence(s): "a" "t"

ability - automative
```

Score: 1 Sequence(s): "a" "i" "t"

ability - certificate

Score: 1 Sequence(s): "a" "i" "t"

ability - compatibility

Score: 6 Sequence(s): "bility"

ability - component

Score: 1 Sequence(s): "t"

ability - conformity

Score: 3 Sequence(s): "ity"

ability - directive

Score: 1 Sequence(s): "i" "t"

ability - disctrict

Score: 1 Sequence(s): "i" "t"

ability - examination

Score: 1 Sequence(s): "a" "i" "t"

ability - fanimatio

Score: 1 Sequence(s): "a" "i" "t"

ability - flammable

Score: 2 Sequence(s): "ab"

ability - fountain

Score: 1 Sequence(s): "a" "i" "t"

ability - information

Score: 1 Sequence(s): "a" "i" "t"

ability - malfunction

Score: 1 Sequence(s): "a" "i" "l" "t"

ability - medication

Score: 1 Sequence(s): "a" "i" "t"

ability - paramedic

Score: 1 Sequence(s): "a" "i"

ability - personality

Score: 4 Sequence(s): "lity"

ability - predict

Score: 1 Sequence(s): "i" "t"

ability - qualify

Score: 2 Sequence(s): "li"

automata - automative

Score: 7 Sequence(s): "automat"

automata - certificate

Score: 2 Sequence(s): "at"

automata - compatibility

Score: 2 Sequence(s): "at" "om"

automata - component

Score: 2 Sequence(s): "om"

automata - conformity

Score: 1 Sequence(s): "m" "o" "t"

automata - directive

Score: 1 Sequence(s): "t"

automata - disctrict

Score: 1 Sequence(s): "t"

automata - examination

Score: 2 Sequence(s): "at"

automata - fanimatio

Score: 3 Sequence(s): "mat"

automata - flammable

Score: 2 Sequence(s): "ma"

automata - fountain

Score: 2 Sequence(s): "ta"

automata - information

Score: 3 Sequence(s): "mat"

automata - malfunction

Score: 2 Sequence(s): "ma"

automata - medication

Score: 2 Sequence(s): "at"

automata - paramedic

Score: 1 Sequence(s): "a" "m"

automata - personality

Score: 1 Sequence(s): "a" "o" "t"

automata - predict

Score: 1 Sequence(s): "t"

automata - qualify

Score: 1 Sequence(s): "a" "u"

automative - certificate

```
Score: 2 Sequence(s): "at" "ti"

automative - compatibility

Score: 3 Sequence(s): "ati"

automative - component

Score: 2 Sequence(s): "om"

automative - conformity

Score: 1 Sequence(s): "i" "m" "o" "t"

automative - directive

Score: 4 Sequence(s): "tive"

automative - disctrict

Score: 1 Sequence(s): "i" "t"

automative - examination

Score: 3 Sequence(s): "ati"

automative - fanimatio

Score: 4 Sequence(s): "mati"

automative - flammable

Score: 2 Sequence(s): "ma"

automative - fountain

Score: 1 Sequence(s): "a" "i" "o" "t" "u"

automative - information

Score: 4 Sequence(s): "mati"

automative - malfunction

Score: 2 Sequence(s): "ma" "ti"

automative - medication

Score: 3 Sequence(s): "ati"

automative - paramedic

Score: 1 Sequence(s): "a" "e" "i" "m"

automative - personality

Score: 1 Sequence(s): "a" "e" "i" "o" "t"

automative - predict

Score: 1 Sequence(s): "e" "i" "t"

automative - qualify

Score: 1 Sequence(s): "a" "i" "u"

certificate - compatibility

Score: 2 Sequence(s): "at" "ti"

certificate - component
```

```
Score: 1 Sequence(s): "c" "e" "t"
certificate - conformity
Score: 1 Sequence(s): "c" "f" "i" "r" "t"
certificate - directive
Score: 2 Sequence(s): "ti"
certificate - disctrict
Score: 2 Sequence(s): "ic"
certificate - examination
Score: 2 Sequence(s): "at" "ti"
certificate - fanimatio
Score: 2 Sequence(s): "at" "ti"
certificate - flammable
Score: 1 Sequence(s): "a" "e" "f"
certificate - fountain
Score: 1 Sequence(s): "a" "f" "i" "t"
certificate - information
Score: 2 Sequence(s): "at" "ti"
certificate - malfunction
Score: 2 Sequence(s): "ti"
certificate - medication
Score: 4 Sequence(s): "icat"
certificate - paramedic
Score: 2 Sequence(s): "ic"
certificate - personality
Score: 2 Sequence(s): "er"
certificate - predict
Score: 2 Sequence(s): "ic"
certificate - qualify
Score: 2 Sequence(s): "if"
compatibility - component
Score: 4 Sequence(s): "comp"
compatibility - conformity
Score: 3 Sequence(s): "ity"
compatibility - directive
Score: 2 Sequence(s): "ti"
compatibility - disctrict
```

```
Score: 1 Sequence(s): "c" "i" "t"
compatibility - examination
Score: 3 Sequence(s): "ati"
compatibility - fanimatio
Score: 3 Sequence(s): "ati"
compatibility - flammable
Score: 1 Sequence(s): "a" "b" "l" "m"
compatibility - fountain
Score: 1 Sequence(s): "a" "i" "o" "t"
compatibility - information
Score: 3 Sequence(s): "ati"
compatibility - malfunction
Score: 2 Sequence(s): "ti"
compatibility - medication
Score: 3 Sequence(s): "ati"
compatibility - paramedic
Score: 2 Sequence(s): "pa"
compatibility - personality
Score: 4 Sequence(s): "lity"
compatibility - predict
Score: 1 Sequence(s): "c" "i" "p" "t"
compatibility - qualify
Score: 2 Sequence(s): "li"
component - conformity
Score: 2 Sequence(s): "co" "on"
component - directive
Score: 1 Sequence(s): "c" "e" "t"
component - disctrict
Score: 1 Sequence(s): "c" "t"
component - examination
Score: 2 Sequence(s): "on"
component - fanimatio
Score: 1 Sequence(s): "m" "n" "o" "t"
component - flammable
Score: 1 Sequence(s): "e" "m"
component - fountain
```

Score: 2 Sequence(s): "nt"

component - information

Score: 2 Sequence(s): "on"

component - malfunction

Score: 2 Sequence(s): "on"

component - medication

Score: 2 Sequence(s): "on"

component - paramedic

Score: 1 Sequence(s): "c" "e" "m" "p"

component - personality

Score: 2 Sequence(s): "on"

component - predict

Score: 1 Sequence(s): "c" "e" "p" "t"

component - qualify

Score: 0 Sequence(s):

conformity - directive

Score: 1 Sequence(s): "c" "i" "r" "t"

conformity - disctrict

Score: 1 Sequence(s): "c" "i" "r" "t"

conformity - examination

Score: 2 Sequence(s): "mi" "on"

conformity - fanimatio

Score: 1 Sequence(s): "f" "i" "m" "n" "o" "t"

conformity - flammable

Score: 1 Sequence(s): "f" "m"

conformity - fountain

Score: 2 Sequence(s): "fo"

conformity - information

Score: 5 Sequence(s): "nform"

conformity - malfunction

Score: 2 Sequence(s): "on"

conformity - medication

Score: 2 Sequence(s): "on"

conformity - paramedic

Score: 1 Sequence(s): "c" "i" "m" "r"

conformity - personality

```
Score: 3 Sequence(s): "ity"
conformity - predict
Score: 1 Sequence(s): "c" "i" "r" "t"
conformity - qualify
Score: 1 Sequence(s): "f" "i" "y"
directive - disctrict
Score: 2 Sequence(s): "ct" "di"
directive - examination
Score: 2 Sequence(s): "ti"
directive - fanimatio
Score: 2 Sequence(s): "ti"
directive - flammable
Score: 1 Sequence(s): "e"
directive - fountain
Score: 1 Sequence(s): "i" "t"
directive - information
Score: 2 Sequence(s): "ti"
directive - malfunction
Score: 3 Sequence(s): "cti"
directive - medication
Score: 2 Sequence(s): "di" "ti"
directive - paramedic
Score: 2 Sequence(s): "di"
directive - personality
Score: 1 Sequence(s): "e" "i" "r" "t"
directive - predict
Score: 2 Sequence(s): "ct" "di" "re"
directive - qualify
Score: 1 Sequence(s): "i"
disctrict - examination
Score: 1 Sequence(s): "i" "t"
disctrict - fanimatio
Score: 1 Sequence(s): "i" "t"
disctrict - flammable
Score: 0 Sequence(s):
disctrict - fountain
```

```
Score: 1 Sequence(s): "i" "t"
disctrict - information
Score: 1 Sequence(s): "i" "r" "t"
disctrict - malfunction
Score: 2 Sequence(s): "ct"
disctrict - medication
Score: 2 Sequence(s): "di" "ic"
disctrict - paramedic
Score: 2 Sequence(s): "di" "ic"
disctrict - personality
Score: 1 Sequence(s): "i" "r" "s" "t"
disctrict - predict
Score: 3 Sequence(s): "ict"
disctrict - qualify
Score: 1 Sequence(s): "i"
examination - fanimatio
Score: 4 Sequence(s): "atio"
examination - flammable
Score: 2 Sequence(s): "am"
examination - fountain
Score: 2 Sequence(s): "in"
examination - information
Score: 5 Sequence(s): "ation"
examination - malfunction
Score: 4 Sequence(s): "tion"
examination - medication
Score: 5 Sequence(s): "ation"
examination - paramedic
Score: 2 Sequence(s): "am"
examination - personality
Score: 2 Sequence(s): "na" "on"
examination - predict
Score: 1 Sequence(s): "e" "i" "t"
examination - qualify
Score: 1 Sequence(s): "a" "i"
fanimatio - flammable
```

Score: 2 Sequence(s): "ma"

fanimatio - fountain

Score: 1 Sequence(s): "a" "f" "i" "n" "o" "t"

fanimatio - information

Score: 5 Sequence(s): "matio"

fanimatio - malfunction

Score: 3 Sequence(s): "tio"

fanimatio - medication

Score: 4 Sequence(s): "atio"

fanimatio - paramedic

Score: 1 Sequence(s): "a" "i" "m"

fanimatio - personality

Score: 1 Sequence(s): "a" "i" "n" "o" "t"

fanimatio - predict

Score: 1 Sequence(s): "i" "t"

fanimatio - qualify

Score: 1 Sequence(s): "a" "f" "i"

flammable - fountain

Score: 1 Sequence(s): "a" "f"

flammable - information

Score: 2 Sequence(s): "ma"

flammable - malfunction

Score: 2 Sequence(s): "ma"

flammable - medication

Score: 1 Sequence(s): "a" "e" "m"

flammable - paramedic

Score: 2 Sequence(s): "am"

flammable - personality

Score: 1 Sequence(s): "a" "e" "l"

flammable - predict

Score: 1 Sequence(s): "e"

flammable - qualify

Score: 1 Sequence(s): "a" "f" "l"

fountain - information

Score: 2 Sequence(s): "fo" "in"

fountain - malfunction

```
Score: 2 Sequence(s): "un"
fountain - medication
Score: 1 Sequence(s): "a" "i" "n" "o" "t"
fountain - paramedic
Score: 1 Sequence(s): "a" "i"
fountain - personality
Score: 1 Sequence(s): "a" "i" "n" "o" "t"
fountain - predict
Score: 1 Sequence(s): "i" "t"
fountain - qualify
Score: 1 Sequence(s): "a" "f" "i" "u"
information - malfunction
Score: 4 Sequence(s): "tion"
information - medication
Score: 5 Sequence(s): "ation"
information - paramedic
Score: 1 Sequence(s): "a" "i" "m" "r"
information - personality
Score: 2 Sequence(s): "on"
information - predict
Score: 1 Sequence(s): "i" "r" "t"
information - qualify
Score: 1 Sequence(s): "a" "f" "i"
malfunction - medication
Score: 4 Sequence(s): "tion"
malfunction - paramedic
Score: 1 Sequence(s): "a" "c" "i" "m"
malfunction - personality
Score: 2 Sequence(s): "al" "on"
malfunction - predict
Score: 2 Sequence(s): "ct"
malfunction - qualify
Score: 2 Sequence(s): "al"
medication - paramedic
Score: 5 Sequence(s): "medic"
medication - personality
```

```
Score: 2 Sequence(s): "on"
medication - predict
Score: 4 Sequence(s): "edic"
medication - qualify
Score: 1 Sequence(s): "a" "i"
paramedic - personality
Score: 1 Sequence(s): "a" "e" "i" "p" "r"
paramedic - predict
Score: 4 Sequence(s): "edic"
paramedic - qualify
Score: 1 Sequence(s): "a" "i"
personality - predict
Score: 1 Sequence(s): "e" "i" "p" "r" "t"
personality - qualify
Score: 3 Sequence(s): "ali"
predict - qualify
Score: 1 Sequence(s): "i"
Program ended with exit code: 0
```

References:

1) https://cse442-17f.github.io/Prims-and-A-Star/   (16/05/2021)