

Advanced Data Structures and Algorithms

Assignment III

Venkatesh E
Indian Institute of Technology
Hyderabad
AI20MTECH14005
ai20mtech14005@iith.ac.in

Table of Contents

| | | |
|----------|--|-----------|
| 1 | PROBLEM DESCRIPTION | 2 |
| 2 | SEQUENCE ALIGNMENT PROBLEM: | 5 |
| 2.1 | Why Sequence Alignment ? | 5 |
| 2.2 | Computation Time Complexity : | 6 |
| 3 | CONCEPT OF DYNAMIC PROGRAMMING | 7 |
| 3.1 | Dynamic Programming : | 7 |
| 3.2 | Key Characteristics of Dynamic Programming : | 7 |
| 3.3 | Methods to solve a Dynamic Programming problem : | 7 |
| 4 | APPROACH | 8 |
| 4.1 | Optimal Substructure and Overlapping Subproblems | 8 |
| 4.2 | Algorithm | 9 |
| 4.3 | Pseudo Code | 10 |
| 4.4 | Explanation with Example : | 11 |
| 5 | PROCEDURE TO COMPILE AND RUN THE CODE | 19 |
| 5.1 | cpp Files | 19 |
| 5.2 | Makefile | 24 |
| 6 | ADDITIONAL INFORMATION | 25 |
| 7 | REFERENCES | 26 |

Abstract

This assignment provides the solution to Global DNA Sequence Alignment problem using Dynamic Programming approach to align DNA sequences.

OS : MAC

Compiler : g++

1 PROBLEM DESCRIPTION

In this assignment, we will look at an application of Dynamic Programming to align DNA sequences. In the global DNA sequence alignment problem, you are given as input two DNA sequences S_1 , S_2 , which are strings using the four characters A, C, T, G. The strings are of lengths m , n , where possibly $m \neq n$. Given two DNA sequences, the goal is to compute the best possible (minimum) alignment cost between them. We will soon specify how to compute the alignment cost between two sequences.

Application :

Why find the alignment cost? Suppose you want to find the role played by a DNA sequence you find in an organism. Instead of carrying out experiment after experiment trying to determine the function, we can use prior information to our advantage: there is usually a good chance that it is a variant of a known gene in a previously studied organism. DNA sequences and their functions in many species have been studied and published in publicly available databases. Thus, a more efficient way would be to compare our sequence for a match with a sequence in an already studied organism. However, DNA strands are subject to minor random mutations from organism to organism, so we cannot expect that we will find an exact match. Nevertheless, two sequences that are more or less similar can be expected to perform the same function. We try to align our (query) sequence with these target sequences, and compute an alignment cost; finding a low alignment cost with a target sequence would give us a reasonable guess about the role played by the query sequence in our organism.

Alignment and Scoring :

Consider the target and query sequences (our first sequence will always be called the target and second the query): AGGCCT, TGGCA respectively. Note that these are of different lengths, so one natural way to pair them would be:

```
A G G G C C T
T G G _ _ C _ _ T
```

Here, `_` is called a gap, that can be inserted anywhere in either string, to make the rest of sequences align better. There is a penalty $s(x, y)$ for matching character x to character y . Consider the following scoring function:

$$s(x, y) = \begin{cases} 0, & \text{if } x = y. \\ 2, & \text{if } x \neq y. \\ 1, & \text{if } x = _ \text{ or } y = _ \end{cases}$$

Note that a smaller cost means better alignment. The cost of aligning a character with the gap character is usually denoted by γ , and is called the gap-penalty. According to the above scoring rules, the pairing given above has an alignment cost of $2 + 1 + 1 = 4$.

Consider aligning the target string: AACAGTTACC and query string: TAAGGTCA. Two possible alignments and corresponding scores are given:

| | | | | | | | | | | | |
|--------|---|---|---|---|---|---|---|---|---|---|------|
| | A | A | C | A | G | T | T | A | C | C | |
| | T | A | A | G | G | T | C | A | _ | _ | |
| Score: | 2 | 0 | 2 | 2 | 0 | 0 | 2 | 0 | 1 | 1 | = 10 |

| | | | | | | | | | | | |
|--------|---|---|---|---|---|---|---|---|---|---|-----|
| | A | A | C | A | G | T | T | A | C | C | |
| | T | A | _ | A | G | G | T | _ | C | A | |
| Score: | 2 | 0 | 1 | 0 | 0 | 2 | 0 | 1 | 0 | 2 | = 8 |

Thus, the second alignment is preferred. Note we may decide to insert gap characters in both target or query string to minimize the alignment scores, and not just in the string with smaller length.

Now consider another example: two possible alignments of AGTACG and ACATAG. Both have an alignment cost of 4, which, you can check, is the minimum alignment cost between the two strings.

Alignment 1:

A_GTACG
ACATA_G

Alignment 2:

AG__TACG
A_CATA_G

Problem to solve :

Given two sequences, write code to compute the cost and alignment pattern of the optimal alignment. The scoring $s(x, y)$ function will be specified as part of the input, along with the target and query sequences. The output should be (each on a new line):

1. The optimal alignment score (a single integer)
2. The pattern with gaps of the target sequence that is matched (without spaces)
3. The pattern with gaps of the query sequence that is matched (without spaces)

Important Criterion :

As seen in the last example above, there may be multiple output patterns possible with the same alignment score. You are to output the alignment pattern in which the matched query pattern is lexicographically the least, when read from left to right. You should consider the character ordering: $_ < A < C < G < T$. The gap character is considered the smallest. e.g. ACC__TG is smaller than ACCT__G, because $_ < T$. This ordering makes sense even for strings of unequal length: AT__CGG is smaller than ATCC. In the third example given above, the second alignment should be given as output, and not the first.

2 SEQUENCE ALIGNMENT PROBLEM:

Many types of biological objects can be represented as sequences. For example, DNA sequences are commonly represented as string where each character in the string would be either A(Adenine),C(Cytosine),G(Guanine) or T(Thymine). Similarly RNA can be represented as a string of four characters such as A,C,G or U.

Our application is DNA Sequence alignment and possibility of string with characters should be A,C,G or T.

2.1 Why Sequence Alignment ?

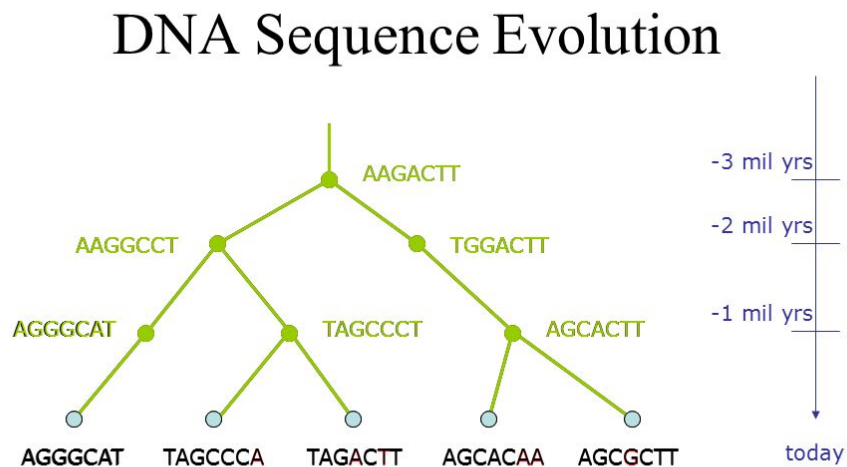


Figure 3: Evolution of DNA Sequence

Studying about the sequences there is a concept of evolution of sequences due to mutations. Consider the example , at time t_0 there are two sequences S_1 and S_2

which are identical at this time.

$$S_1 : ATCTATACAATTAG$$

$$S_2 : ATCTATACAATTAG$$

As time changes, some mutations may happen so that some of the nucleotides will be replaced by some other nucleotides i.e some insertion and deletion could happen in any of the sequences S_1 and S_2 . Therefore at time t i.e at present, the modified sequences may be as follows.

$$S_{1new} : ACTATTACATTAG$$

$$S_{2new} : ATTATACATTG$$

The above two sequences S_{1new} and S_{2new} were similar but not exactly same like before at time t_0 . Therefore we will try to find out how similar the given two DNA sequences are so that we would be clear that both the sequences came from same origin at time t_0 .

2.2 Computation Time Complexity :

Let S_1 and S_2 be the two given sequences of length n and m .
Number of possible arrangements($N_{arrangements}$) is given by,

$$N_{arrangements} = \binom{n+m}{n} = \frac{(n+m)!}{(n!)^2}$$

| Length of sequence 1(m) | Length of Sequence 2(n) | Computations Required |
|-------------------------|-------------------------|---|
| 10 | 5 | $\frac{1.307674368E+12}{(120)^2}$ |
| 50 | 100 | $\frac{5.713383956E+262}{(9.332621544E+157)^2}$ |
| 100 | 500 | $\frac{1.265572316E+1408}{(1.220136825E+1134)^2}$ |

Table 1: Computations Required

From the above table we could say that even for small input of sequence 1 of length 10 and sequence 2 of length 5 we need to compute many possible arraignment and find the best possible score. But this will take lot of time and this need to be optimized which could be done by dynamic programming concept.

3 CONCEPT OF DYNAMIC PROGRAMMING

3.1 Dynamic Programming :

1. Dynamic Programming is an optimization method which divides the main problem into many overlapping sub-problems and the solution to all the sub-problem will yield the solution to the main problem.
2. If there exists a optimal solution of main problem which we obtain using optimal solution of sub-problems then it indicates the main problem has Optimal Substructure property.
3. Here we will store the result of sub-problems so that there is no need to re-compute the same result when needed for later use. This will reduce the overall time complexity.

3.2 Key Characteristics of Dynamic Programming :

1. Optimal Substructure
2. Overlapping Subproblems

3.3 Methods to solve a Dynamic Programming problem :

1. Top Down Method

Also called as Memoization. It is an optimization technique which is primarily used to speed up the computer programs by storing the results of expensive function calls and returning the cache result when the same inputs occur again when the recursive calls were made to find the solution to sub-problems.

2. Bottom Up Method

Bottom Up method avoids recursion. All the sub-problems results were stored in table to avoid future computations.

4 APPROACH

4.1 Optimal Substructure and Overlapping Subproblems

Optimal Substructure :

If an optimal solution of the subproblems leads to an overall optimal solution then it is said to be **Optimal Substructure**.

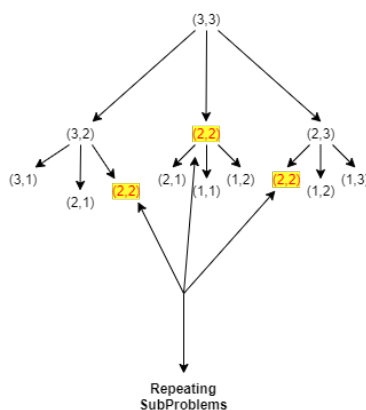
Assume, we need to align the first i characters in target T and first j characters in query Q . Let Optimal Alignment Cost $OAC(i,j)$ of the alignment between $Target_i$ and $Query_j$.

Here let g be gap cost and m be mismatch cost.

$$OAC(i, j) = \begin{cases} j * g & \text{if } i = 0. \\ i * g & \text{if } j = 0. \\ \min \begin{cases} g + OAC(i - 1, j) \\ g + OAC(i, j - 1) \\ m + OAC(i - 1, j - 1) \end{cases} & \text{Otherwise.} \end{cases}.$$

The above equation leads to the optimal solution.

Overlapping Subproblems :

Figure 4: **Overlapping Subproblems**

If the same subproblem is repeated many times, we need not calculate the same many times. Instead we will store the result and use it in future. Such a case where the same subproblem is repeating is termed to be overlapping subproblem.

Consider the target of length 3 and query of length 3. To construct the table using bottom-up approach. We could see the overlapping subproblems as shown in above figure.

4.2 Algorithm

Consider two sequences as follows :

$$\begin{aligned} \text{Target} &: t_1 t_2 \dots t_n \\ \text{Query} &: q_1 q_2 \dots q_m \end{aligned}$$

where Target sequence of length n and Query sequence of length m.

1. Create an table of size $n + 1 \times m + 1$ to store the computations so that re-computations will be avoided.
2. Initialize the first row and first column of the table. If both targetchar and querychar were "»" then assign the particular entry as 0. Else, compute $(i * \text{gapcost})$ for i in range of 1 to n along row and similarly compute the same along column aswell.
3. If $t_x == q_y$ then store the value to table(x,y) as table(x-1,y-1).
4. Else get the minimum among table(x-1,y-1)+mismatchcost, table(x-1,y)+gapcost, table(x,y-1)+gapcost and store it in table.
5. Computing recursively and storing the entries in the table will give alignment score and it is given by table[n][m].
6. Once the table is computed backtrack to get the target and query alignments.
7. Start from the table[m][n] entry and see from where it could have come. There could be 3 possibilities. It could be from table[m-1][n] or table[m][n-1] if both the target and query characters were not equal else the table[m][n] should have come from table[m-1][n-1].
8. Recursively do the step 7 till we reach the beginning of target or query index.
9. Once the starting index of any of the target or query gets reached print the remaining elements of other sequence to their respective output(either targetoutput or queryoutput).

$$\text{TimeComplexity} : O(nm)$$

$$\text{SpaceComplexity} : O(nm)$$

Time taken to compute the entire table will take $O(nm)$ time and to store the computed results it will take $O(nm)$ space storage.

4.3 Pseudo Code

```

target_length = length of target
query_length = length of query
Gap cost = gap cost entered by user
Mismatch cost = mismatch cost entered by user
Get target string input
Get query string input
Create a table of size target_length+1 x query_length+1
Initialize the table entry
    for(int i=0;i<=target_length;i++){
        table[i][0]=i*gapcost
    }
    for(int j=0;j<=query_length;j++){
        table[0][j]=j*gapcost
    }

    for(int i=1;i<=target_length;i++){
        for(int j=1;j<=query_length;j++){
            if(target[i-1]==query[j-1]):
                table[i][j]=table[i-1][j-1]
            Else:
                table[i][j]=std::min({table[i-1][j-1]+mismatchcost,table[i-1][j]+gapcost,table[i][j-1]+gapcost})
        }
    }
Optimal_alignment_score = table[target_length][query_length]
Perform backtracking to get the target_output and query_output using the table.

```

Figure 5: PseudoCode

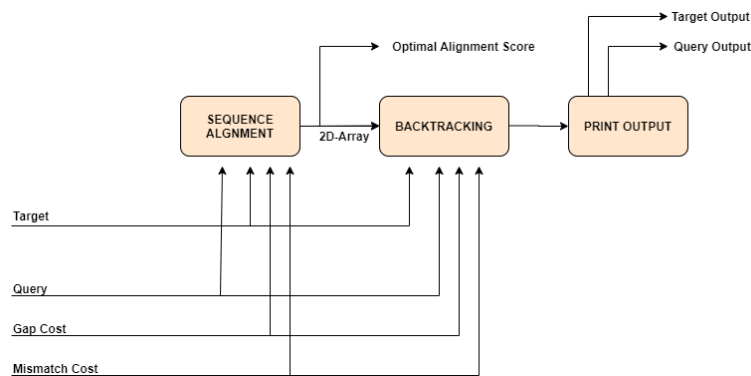


Figure 6: Block Diagram

4.4 Explanation with Example :

Lets consider a example,

Target : ATGACT

Query : ATTGAG

gapcost = 1

mismatchcost = 2

Constructing the Table

Step 1: Create a table of size 7×7 as the target and query of length 6 as shown in table 7.

| QUERY \ TARGET | TARGET | | | | | | |
|----------------|--------|---|---|---|---|---|---|
| | -- | A | T | G | A | C | T |
| -- | | | | | | | |
| A | | | | | | | |
| T | | | | | | | |
| T | | | | | | | |
| G | | | | | | | |
| A | | | | | | | |
| G | | | | | | | |

Figure 7: Create Table of size 7×7

Step 2: Compute the table entries corresponding to '-' query row and target column. The first entry where target string is '-' and query string is '-', the corresponding cost will be 0 as there is no target and query string. Initializing the first row of the table is explained in figure (9). Similarly first column of the table can be implemented keeping the target as '-' and query as string input given. Entire initialization of table is shown in table 8.

| QUERY \ TARGET | TARGET | | | | | | |
|----------------|--------|---|---|---|---|---|---|
| | -- | A | T | G | A | C | T |
| -- | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| A | 1 | | | | | | |
| T | 2 | | | | | | |
| T | 3 | | | | | | |
| G | 4 | | | | | | |
| A | 5 | | | | | | |
| G | 6 | | | | | | |

Figure 8: Initializing the first row and first column as base condition


| TARGET | -- | A | AT | ATG | ATGA | ATGAC | ATGACT |
|---------------|----|-----------|-----------|-----------|-----------|-----------|-----------|
| QUERY | -- | -- | -- | -- | -- | -- | -- |
| TARGET OUTPUT | -- | A | AT | ATG | ATGA | ATGAC | ATGACT |
| QUERY OUTPUT | -- | - | -- | --- | ---- | ----- | ----- |
| SCORE | 0 | 1 gaps | 2 gaps | 3 gaps | 4 gaps | 5 gaps | 6 gaps |

Figure 9: Computing the table


Step 3:Computing the intermediate step. To compute the intermediate step we will consider the computation that were being done before this part which is stored in the table aswell.For example if the target string is ACTE and ABTE and if we were comparing the last character E in both query and target, we would use the computation that we did for ACT as target and ABT as query and on top of this we

will compute the result obtained on comparing the present character of target and query. This indicates that we need not compute the previous results again as we have computed earlier and stored. This reduces the time complexity. This is the optimal substructure where the solution to the subproblem will lead to solution to the main problem.

| QUERY \ TARGET | | | | | | | |
|----------------|----|---|---|---|---|---|---|
| | -- | A | T | G | A | C | T |
| -- | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| A | 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| T | 2 | 1 | 0 | 1 | 2 | 3 | 4 |
| T | 3 | 2 | 1 | 2 | 3 | 4 | 3 |
| G | 4 | 3 | 2 | 1 | 2 | 3 | 4 |
| A | 5 | 4 | 3 | 2 | 1 | 2 | 3 |
| G | 6 | 5 | 4 | 3 | 2 | 3 | 4 |



P1



P2

Figure 10: Computation

To compute P1:

$$score(targetchar, querychar) = \begin{cases} 0, & \text{if } targetchar=querychar. \\ 2, & \text{if } targetchar \neq querychar. \\ 1, & \text{if } targetchar==_ \text{ or } querychar==_ \end{cases}$$

IF(targetchar==querychar)

Update IF :

P1(targetindex,queryindex)=P1(targetindex-1,queryindex-1)

ELSE

Update ELSE :

$P1(i,j) = \min(P1(i-1,j-1) + \text{mismatchcost}, P1(i-1,j) + \text{gapcost}, P1(i,j-1) + \text{gapcost})$

In similar way the P2 can also be computed, the only difference between both the computations is targetchar and querychar were same while computing P1 and it is different while computing P2. Entire table is compute in this way and it is shown in figure 11.

| QUERY \ TARGET | | | | | | | |
|----------------|----|---|---|---|---|---|---|
| | -- | A | T | G | A | C | T |
| -- | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| A | 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| T | 2 | 1 | 0 | 1 | 2 | 3 | 4 |
| T | 3 | 2 | 1 | 2 | 3 | 4 | 3 |
| G | 4 | 3 | 2 | 1 | 2 | 3 | 4 |
| A | 5 | 4 | 3 | 2 | 1 | 2 | 3 |
| G | 6 | 5 | 4 | 3 | 2 | 3 | 4 |




Figure 11: Final Table – BOTTOM UP Approach

Backtracking:

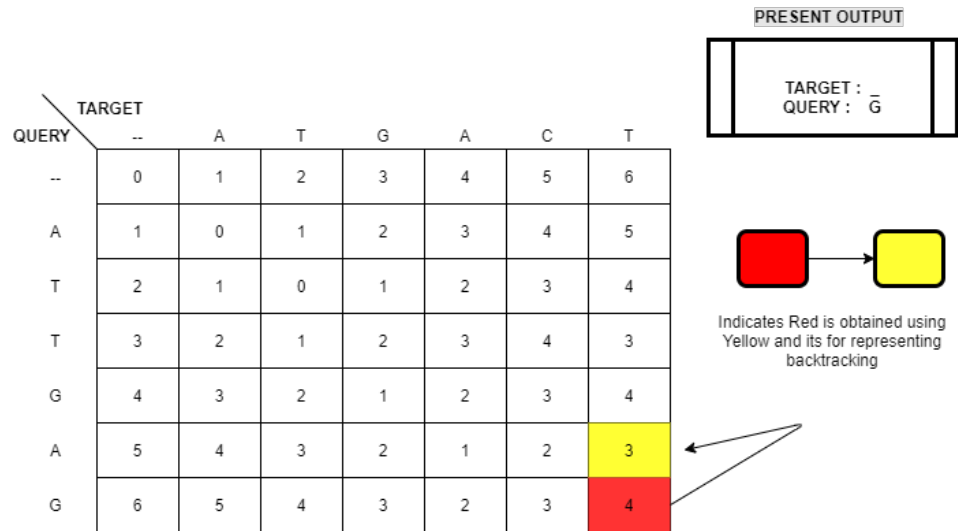


Figure 12: Backtracking-Step 1 along with output at this stage

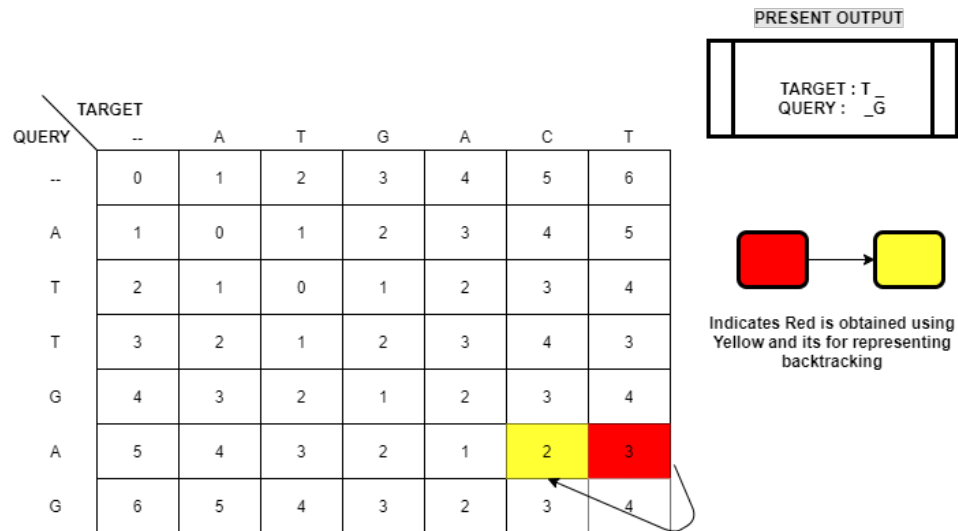


Figure 13: Backtracking-Step 2 along with output at this stage

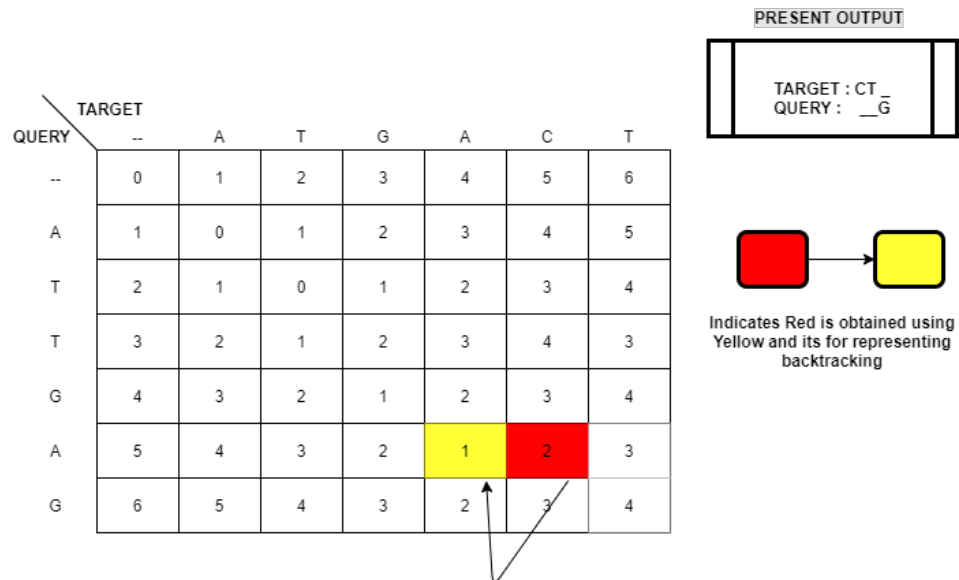


Figure 14: Backtracking-Step 3 along with output at this stage

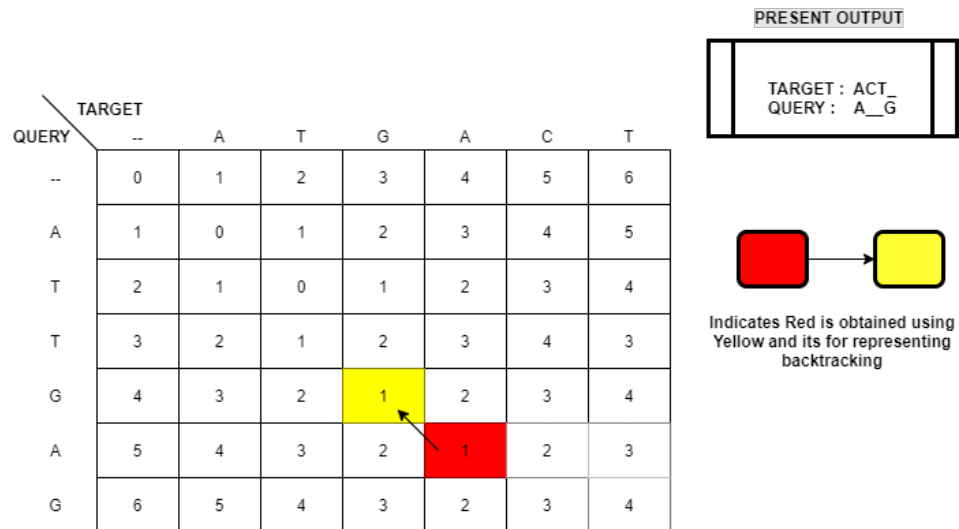


Figure 15: Backtracking-Step 4 along with output at this stage

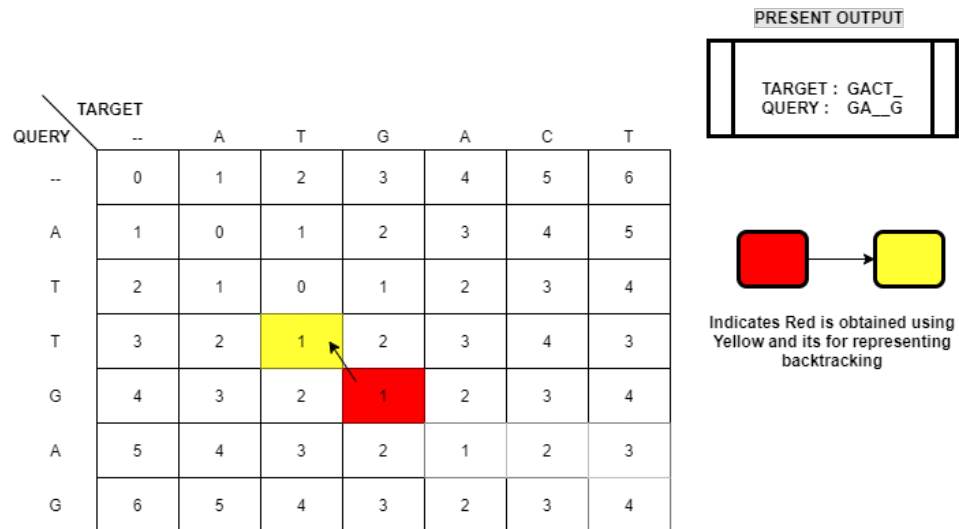


Figure 16: Backtracking-Step 5 along with output at this stage

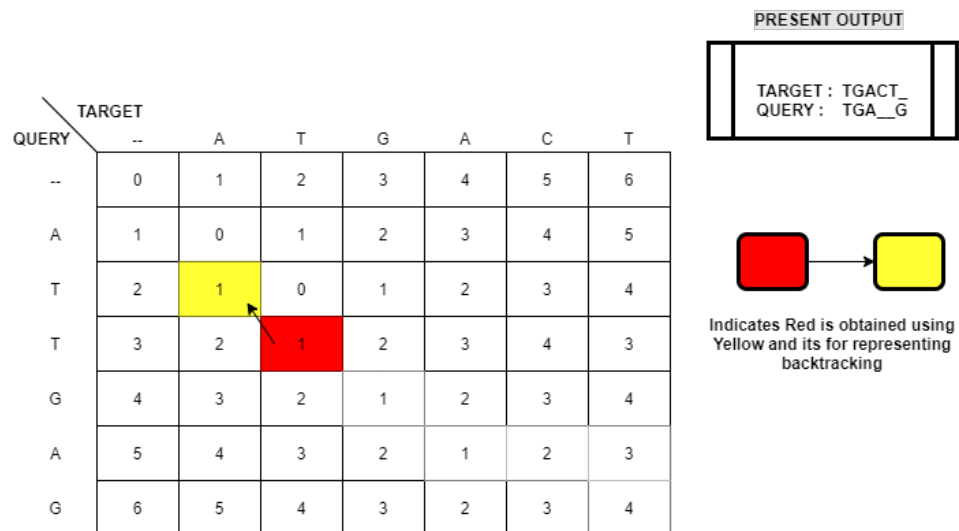


Figure 17: Backtracking-Step 6 along with output at this stage

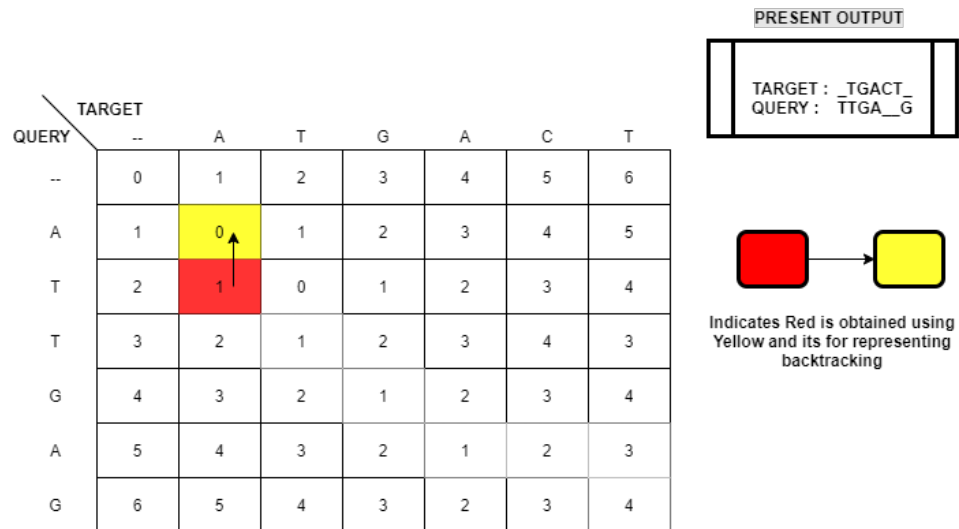


Figure 18: Backtracking-Step 1 along with output at this stage

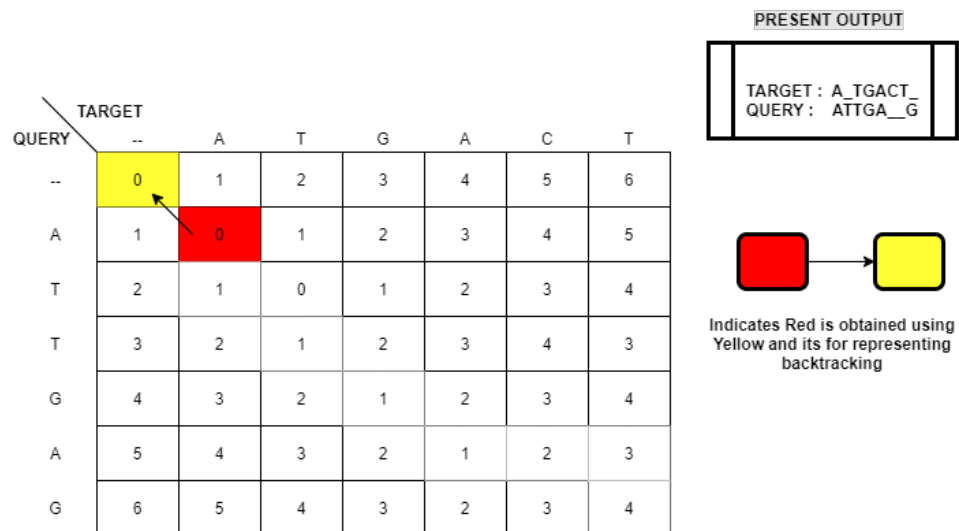


Figure 19: Backtracking-Step 8 along with output at this stage

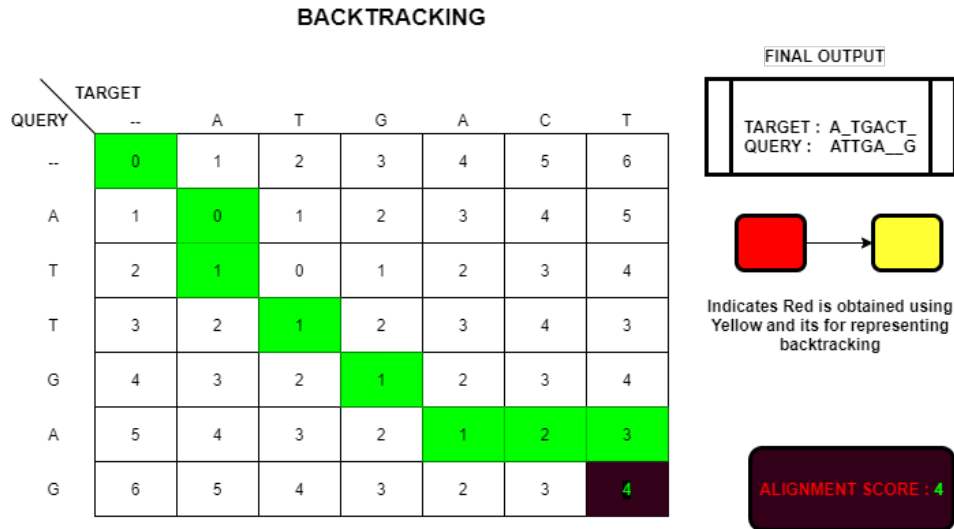


Figure 20: Backtracking Results

Disadvantages :

Input dimensions i.e length of target and query string may be very large. In such a case huge storage space and more computational complexity will be the problem.

5 PROCEDURE TO COMPILE AND RUN THE CODE

5.1 cpp Files

```

1 #include <iostream>
2 #include <algorithm>
3 #include <cstdio>
4 #include <vector>
5 using namespace std;
6
7 /*
8 1. sequence_alignment
9 Input :
10     target : input target of type string
11     query  : input query of type string
12         nt : length of the input target string of type int
13         nq : length of the input query string of type int
14     gcost : gap cost of type int
15     mcost : mismatch cost of type int

```

```

16     output : 2D array of type vector which stores the result of
           alignment score
17 Returns :
18     output 2D array of type vector
19 Description :
20     This function takes the input as mentioned above and returns
           the 2D array of type vector which stores the
           alignment score.
21 */
22 */
23 vector< vector<int> > sequence_alignment(string target,string
           query,int nt,int nq,int gcost,int mcost,vector< vector<int> >
           output);
24
25 /*
26 2. BackTracking
27 Input :
28     target : input target of type string
29     query : input query of type string
30     nt : length of the input target string of type int
31     nq : length of the input query string of type int
32     gcost : gap cost of type int
33     mcost : mismatch cost of type int
34     output_dp : 2D array of type vector which stores the result of
           alignment score
35 Prints :
36     optimal alignment score
37 Calls :
38     print function to print the target result and query result
39 Description :
40     This function takes the input as mentioned above and prints
           the optimal alignment score and calls the
           function to print the target result and query result.
41 */
42 */
43 void BackTracking(string target,string query,int nt,int nq,int
           gcost,int mcost,vector< vector<int> > output_dp);
44
45 /*
46 3. print_out
47 Input :
48     target_result : string of length of targetlen+querylen
49     query_result : string of length of targetlen+querylen
50 Prints :
51     target output - The pattern with gaps of the target
           sequence that is matched (without spaces)
52     query output - The pattern with gaps of the query sequence
           that is matched (without spaces)
53 Description :
54     This function takes the input as target_result and
           query_result which does certain modifications to

```

```

55         that string and returns the expected output.
56     */
57     void print_out(string target_result,string query_result);
58
59     void print_out(string target_result,string query_result){
60         // get the index from which the string to be printed as output
61         int start_index;
62         for(int i=0;i<target_result.length();i++){
63             // if one of the char is either not "_" break and get that
64             // index
65             if((target_result[i]!='_')||(query_result[i]!='_')){
66                 start_index=i;
67                 break;
68             }
69             cout<<target_result.substr(start_index)<<"\n"; // substring
70             // starting from start_index
71             cout<<query_result.substr(start_index)<<"\n"; // substring
72             // starting from start_index
73             return;
74         }
75     }
76
77     vector< vector<int> > sequence_alignment(string target,string
78     query,int nt,int nq,int gcost,int mcost,vector< vector<int> >
79     output){
80         // initializing the dynamic 2D output array
81         for(int i=0;i<=nt;i++){
82             output[i][0]=i*gcost;
83         }
84         for(int j=0;j<=nq;j++){
85             output[0][j]=j*gcost;
86         }
87         // to calculate the minimum penalty
88         for(int i=1;i<=nt;i++){
89             for(int j=1;j<=nq;j++){
90                 // if target character and query character are equal
91                 if(target[i-1]==query[j-1]){
92                     output[i][j]=output[i-1][j-1];
93                 }
94                 else{
95                     output[i][j]=min(output[i-1][j-1]+mcost,min(output
96                     [i-1][j]+gcost,output[i][j-1]+gcost));
97                 }
98             }
99         }
100         return output; // return 2D array of type vector
101     }
102
103     // function to display the output target and query string

```

```

98 void BackTracking(string target,string query,int nt,int nq,int
    gcost,int mcost,vector< vector<int> > output_dp){
99     // initialize the variables
100     int output_len=nt+nq; // output len to store the output target
        and query
101     int t=nt,q=nq; // to iterate through the target and query
102     string target_out(output_len+1,'_'); // initialize the output
        target string
103     string query_out(output_len+1,'_'); // initialize the output
        query string
104     int i=output_len,j=output_len;
105     // Steps involved in backtracking and store the results in
        target_out and query_out
106     while(!((t==0)|| (q==0))){
107         if(target[t-1]==query[q-1]){
108             // if target and query character are equal, store the
            char in output string
109             target_out[i]=target[t-1];
110             query_out[j]=query[q-1];
111             i-=1;
112             j-=1;
113             t-=1;
114             q-=1;
115         }
116         else if(output_dp[t][q-1]+gcost==output_dp[t][q]){
117             // we first perform this to confirm lexicographically
            the least for query
118             query_out[j]=query[q-1];
119             i-=1;
120             j-=1;
121             q-=1;
122         }
123         else if(output_dp[t-1][q]+gcost==output_dp[t][q]){
124             target_out[i]=target[t-1];
125             i-=1;
126             j-=1;
127             t-=1;
128         }
129
130         else if(output_dp[t-1][q-1]+mcost==output_dp[t][q]){
131             target_out[i]=target[t-1];
132             query_out[j]=query[q-1];
133             i-=1;
134             j-=1;
135             t-=1;
136             q-=1;
137         }
138     }
139     // copy the remaining content of target string to output

```

```

140     target
141     while(i>0){
142         target_out[i--]=(t>0)?target[--t] : '_';
143     }
144     // copy the remaining content of query string to output query
145     while(j>0){
146         query_out[j--]=(q>0)?query[--q]:'_';
147     }
148     // calls the function print_out to get target_output and
149     query_out
150     print_out(target_out,query_out);
151     return ;
152 }
153 int main() {
154     int target_length,query_length,gap_cost,mismatch_cost;
155     string target_input,query_input;
156     cin>>target_length>>query_length; // get the input of target
157     length and query length
158     cin>>gap_cost>>mismatch_cost; // get the input of gap cost and
159     mismatch cost
160     cin>>target_input; // get the target input of type string
161     cin>>query_input; // get the query input of type string
162     std::string target = target_input.substr(0,target_length);
163     std::string query = query_input.substr(0,query_length);
164     vector< vector<int> >sequence_dp(target_length+1, vector<int>(
165     query_length+1)); // 2D Array to store alignment score
166     sequence_dp=sequence_alignment(target,query,target_length,
167     query_length,gap_cost,mismatch_cost,sequence_dp);
168     //prints the optimal alignment score
169     cout<<sequence_dp[target_length][query_length]<<"\n";
170     // BackTracking to get the target and query output
171     BackTracking(target,query,target_length,query_length,gap_cost,
172     mismatch_cost,sequence_dp);
173     return 0;
174 }

```

Listing 1: C++ Code for Global DNA Sequence Alignment Problem

Compilation command to create the executable : **make**

Run command :**make run**

Output is executed in terminal and it is as shown below in figure 22. Output is highlighted using yellow color.

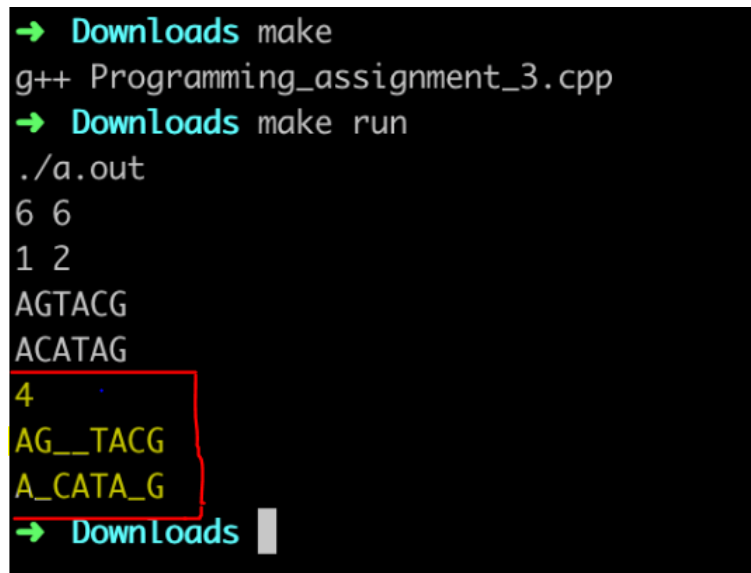
5.2 Makefile

```
Program: Programming_assignment_3.cpp
        g++ Programming_assignment_3.cpp

run:
        ./a.out

clean:
        rm a.out
```

Figure 21: **makefile**



```
→ Downloads make
g++ Programming_assignment_3.cpp
→ Downloads make run
./a.out
6 6
1 2
AGTACG
ACATAG
4
AG__TACG
A_CATA_G
→ Downloads
```

Figure 22: **Output**

6 ADDITIONAL INFORMATION

Description of Functions :

1. *vector < vector < int >>* **sequencealignment**(string target,string query,int nt,int nq,int gcost,int mcost,*vector < vector < int >>* output)

Input :

target : input target of type string

query : input query of type string

nt : length of the input target string of type int

nq : length of the input query string of type int

gcost : gap cost of type int

mcost : mismatch cost of type int

output : 2D array of type vector which stores the result of alignment score

Returns : output 2D array of type vector

Description : This function takes the input as mentioned above and returns the 2D array of type vector which stores the alignment score.

2. **void BackTracking**(string target,string query,int nt,int nq,int gcost,int mcost,*vector < vector < int >>* outputdp)

Input :

target : input target of type string

query : input query of type string

nt : length of the input target string of type int

nq : length of the input query string of type int

gcost : gap cost of type int

mcost : mismatch cost of type int

outputdp : 2D array of type vector which stores the result of alignment score

Prints : optimal alignment score

Calls : print function to print the target result and query result

Description : This function takes the input as mentioned above and prints the optimal alignment score and calls the function to print the target result and query result.

3. void printout(string targetresult,string queryresult)

Input :

targetresult : string of length of targetlen+querylen

queryresult : string of length of targetlen+querylen

Prints :

target output - The pattern with gaps of the target sequence that is matched (without spaces)

query output - The pattern with gaps of the query sequence that is matched (without spaces)

Description : This function takes the input as targetresult and queryresult which does certain modifications to that string and returns the expected output.

7 REFERENCES

[IIT-Kgp Lecture](#)

[CMU - Lecture](#)

[MIT - Lecture](#)

[Sequence Alignment - Wikipedia](#)

[Sequence Alignment - Slide Share](#)

[Stanford - Lecture](#)