1.4
X =dcdcbacbbb
Y =acdccabdbb
X’ =cdcbacb_bb
Y’ =cdcca_bdbb

-	0	a	c	d	c	c	a	b	d	b	b
0	0	0	0	0	0	0	0	0	0	0	0
d	0	0 ↖	0 ↖	2 ↖	1 ↑	0 ↖	0 ↖	0 ↖	2 ↖	1 ↑	0 ↖
c	0	0 ↖	2 ↖	1 ↑	4 ↖	3 ↖	2 ↑	1 ↑	1 ←	1 ↖	0 ↖
d	0	0 ↖	1 ←	4 ↖	3 ↑	3 ↖	2 ↖	1 ↖	3 ↖	2 ↑	1 ↑
c	0	0 ↖	2 ↖	3 ←	6 ↖	5 ↖	4 ↑	3 ↑	2 ↑	2 ↖	1 ↖
b	0	0 ↖	1 ←	2 ←	5 ←	5 ↖	4 ↖	6 ↖	5 ↑	4 ↖	4 ↖
a	0	2 ↖	1 ↑	1 ←	4 ←	4 ↖	7 ↖	6 ↑	5 ↖	4 ↖	3 ↖
c	0	1 ←	4 ↖	3 ↑	3 ↖	6 ↖	6 ←	6 ↖	5 ↖	4 ↖	3 ↖
b	0	0 ↖	3 ←	3 ↖	2 ↖	5 ←	5 ↖	8 ↖	7 ↑	7 ↖	6 ↖
b	0	0 ↖	2 ←	2 ↖	2 ↖	4 ←	4 ↖	7 ↖	7 ↖	9 ↖	9 ↖
b	0	0 ↖	1 ←	1 ↖	1 ↖	3 ←	3 ↖	6 ↖	6 ↖	9 ↖	11 ↖

-	0	a	c	d	c	c	a	b	d	b	b
0	0	0	0	0	0	0	0	0	0	0	0
d	0	1	1	1	2	1	1	1	1	2	1
c	0	1	1	2	1	1	2	2	3	1	1
d	0	1	3	1	2	1	1	1	1	2	2
c	0	1	1	3	1	1	2	2	2	1	1
b	0	1	3	3	3	1	1	1	2	1	1
a	0	1	2	3	3	1	1	2	1	1	1
c	0	3	1	2	1	1	3	1	1	1	1
b	0	1	3	1	1	3	1	1	2	1	1
b	0	1	3	1	1	3	1	1	1	1	1
b	0	1	3	1	1	3	1	1	1	1	1

H[i][j] = max{p1,p2,p3}
p1 = H[i-1][j-1]-1
p2 = H[i-1][j]-1
p3 = H[i][j-1]-1
similarly other H matrix values are computed for P matrix corresponding p values are considered are directions are entered.
Pseudocode is

```
H[i][j] = max{p1,p2,p3}
if (H[i][j] = p1) then
    P[i][j] = ↖
else
  If  (H[i][j] = p2) then
     P[i][j] = ↑
else
      P[i][j] = ←
return(H,P)
```

2. Optimal binary search tree

w	0	1	2	3	4	5
1	0	0.21	0.36	0.64	0.76	1.0
2		0	0.15	0.43	0.55	0.79
3			0	0.28	0.40	0.64
4				0	0.12	0.36
5					0	0.24
6						0

e	0	1	2	3	4	5
1	0	0.21	0.51	0.13	1.39	1.99
2		0	0.15	0.58	0.82	1.42
3			0	0.28	0.52	1.12
4				0	0.12	0.48
5					0	0.24
6						0

r	1	2	3	4	5
1	1	1	2	3	3
2		2	3	3	3
3			3	3	3
4				4	5
5					5

For i =1, j =2

w[i , j] = w[i , j-1] + Pj = w[1 , 1] + P2 = 0.21 + 0.15 = 0.36

- r = 1 :

e[1, 2] = e[1, 0] + e[2, 2] + w[1, 2] = 0 + 0.15 + 0.36 = 0.51

- r = 2 :

e[1, 1] = e[1, 1] + e[3, 2] + w[1, 2] = 0.21 + 0 + 0.36 = 0.57

For i = 2, j = 3

w[2, 3] = w[2, 2] + P3 = 0.15+ 0.28 = 0.43

- r = 2 :

e[2, 3] = e[2, 1] + e[3, 3] + w[2, 3] = 0 + 0 + 0.43 = 0.71

- r = 3 :

e[2, 3] = e[2, 2] + e[4, 3] + w[2, 3] = 0.15 + 0 + 0.43 = 0.58

For i =1, j =3

$w[1, 3] = w[1, 2] + P3 = 0.36 + 0.28 = 0.64$

- $r = 1$:

$e[1, 3] = e[1, 0] + e[2, 3] + w[1, 3] = 0 + 0.58 + 0.64 = 1.22$

- $r = 2$:

$e[1, 3] = e[1, 1] + e[3, 3] + w[1, 3] = 0.15 + 0 + 0.64 = 1.13$

- $r = 3$:

$e[1, 3] = e[1, 2] + e[4, 3] + w[1, 3] = 0.51 + 0 + 0.64 = 1.15$

For $i = 3, j = 4$

$w[3, 4] = w[3, 3] + P4 = 0.28 + 0.12 = 0.4$

- $r = 3$:

$e[3, 4] = e[3, 2] + e[4, 4] + w[3, 4] = 0 + 0.12 + 0.4 = 0.52$

- $r = 4$:

$e[3, 4] = e[3, 3] + e[5, 4] + w[3, 4] = 0.28 + 0 + 0.4 = 0.68$

For $i = 2, j = 4$

$w[2, 4] = w[2, 3] + P4 = 0.43 + 0.12 = 0.55$

- $r = 2$:

$e[2, 4] = e[2, 1] + e[3, 4] + w[2, 4] = 0 + 0.52 + 0.55 = 1.07$

- $r = 3$:

$e[2, 4] = e[2, 2] + e[4, 4] + w[2, 4] = 0.15 + 0.12 + 0.55 = 0.82$

- $r = 3$:

$e[2, 4] = e[2, 3] + e[5, 4] + w[2, 4] = 0.58 + 0 + 0.55 = 1.13$

For $i = 1, j = 4$

$w[1, 4] = w[1, 3] + P4 = 0.64 + 0.12 = 0.76$

- $r = 1$:

$e[1, 4] = 0 + 0.82 + 0.76 = 1.58$

- $r = 2$:

$e[1, 4] = 0.21 + 0.52 + 0.76 = 1.49$

- $r = 3$:

$e[1, 4] = 0.51 + 0.12 + 0.76 = 1.39$

- $r = 4$:

$e[1, 4] = 1.13 + 0 + 0.76 = 1.89$

For $i = 4, j = 5$

$w[4, 5] = w[4, 4] + P5 = 0.12 + 0.24 = 0.36$

- $r = 4$:

$e[4, 5] = 0 + 0.24 + 0.36 = 0.6$

- $r = 5$:

$e[4, 5] = 0.12 + 0 + 0.36 = 0.48$

For $i = 3, j = 5$

$w[3, 5] = w[3, 4] + P5 = 0.4 + 0.24 = 0.64$

- r = 3 :

e[3, 5] = 0 + 0.48 +0.64 = 1.12

- r = 4 :

e[1, 4] = 0.28 + 0.24 + 0.64 = 1.16

- r = 5 :

e[1, 4] = 0.52 + 0 + 0.64 = 1.16

For i = 2, j =5
w[2, 5] = w[2, 4] + P5   = 0.55 + 0.24 = 0.79

- r = 1 :

e[1, 4] = 0 + 0.16 + 0.79 = 1.95

- r = 2 :

e[1, 4] = 0.15 + 0.48 + 0.79 = 1.42

- r = 3 :

e[1, 4] = 0.58 + 0.24 + 0.79 = 1.61

- r = 4 :

e[1, 4] = 0.82 + 0 + 0.79 = 1.61

For i = 1, j =5
w[1, 5] = w[1, 4] + P5   = 0.76 + 0.24 = 1.0

- r = 1 :

e[1, 5] = 0 + 1.42 + 1 = 2.42

- r = 2 :

e[1, 5] = 0.21 + 1.16 + 1 = 2.37

- r = 3 :

e[1, 5] = 0.51 + 0.48 + 1 = 1.99

- r = 4 :

e[1, 5] = 1.13 + 0.24 + 1 = 2.37

- r = 5 :

e[1, 5] = 1.39 + 0 + 1 = 2.39

code:

```
#include <cstring>
#include <iostream>
#include <limits.h>

#include "dynprog.h"

using namespace std;

/*
```

```
 * Bottom up implementation of Smith-Waterman algorithm
 */
void SW_bottomUp(char* X, char* Y, char** P, int** H, int n, int m){
  // Initialize the matrix H with zeros
  for (int i = 0; i <= n; ++i) {
    for (int j = 0; j <= m; ++j) {
      H[i][j] = 0;
    }
  }

  // Fill in the matrix H and P
  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
      if (X[i - 1] == Y[j - 1]) {
        H[i][j] = H[i - 1][j - 1] + 1;
        P[i][j] = 'D'; // Diagonal arrow
      } else {
        H[i][j] = max(H[i - 1][j], H[i][j - 1]);
        if (H[i][j] == H[i - 1][j])
          P[i][j] = 'U'; // Up arrow
        else
          P[i][j] = 'L'; // Left arrow
      }
    }
  }
}

/*
 * Top-down with memoization implementation of Smith-Waterman algorithm
 */
int memoized_SW_AUX(char* X, char* Y, char** P, int** H, int n, int m) {
  if (n == 0 || m == 0)
    return 0;
  if (H[n][m] != -1)
    return H[n][m];

  int score = 0;
  if (X[n - 1] == Y[m - 1]) {
    score = memoized_SW_AUX(X, Y, P, H, n - 1, m - 1) + 1;
    P[n][m] = 'D';
  } else {
    int left = memoized_SW_AUX(X, Y, P, H, n, m - 1);
    int up = memoized_SW_AUX(X, Y, P, H, n - 1, m);
    if (left >= up) {
```

```
            score = left;
            P[n][m] = 'L';
        } else {
            score = up;
            P[n][m] = 'U';
        }
    }
    H[n][m] = score;
    return score;
}


void memoized_SW(char* X, char* Y, char** P, int** H, int n, int m){
    for (int i = 0; i <= n; ++i) {
        for (int j = 0; j <= m; ++j) {
            H[i][j] = -1; // Initialize memoization matrix with -1
        }
    }
    memoized_SW_AUX(X, Y, P, H, n, m);
}

void print_Seq_Align_X(char* X, char** P, int n, int m){
    // Print X' using P matrix
    int i = n, j = m;
    while (i > 0 && j > 0) {
        if (P[i][j] == 'D' || P[i][j] == 'L') {
            cout << X[i - 1];
            --i;
        } else {
            cout << '-';
            --j;
        }
    }
    while (i > 0) {
        cout << X[i - 1];
        --i;
    }
}

/*
 * Print Y'
 */
void print_Seq_Align_Y(char* Y, char** P, int n, int m){
    // Print Y' using P matrix
```

```
    int i = n, j = m;
    while (i > 0 && j > 0) {
        if (P[i][j] == 'D' || P[i][j] == 'U') {
            cout << Y[j - 1];
            --j;
        } else {
            cout << '-';
            --i;
        }
    }
    while (j > 0) {
        cout << Y[j - 1];
        --j;
    }
}


#ifndef __DYNPROG_H__
#define __DYNPROG_H__

void SW_bottomUp(char*, char*, char**, int**, int, int);
void print_Seq_Align_X(char*, char**, int, int);
void print_Seq_Align_Y(char*, char**, int, int);

void memoized_SW(char*, char*, char**, int**, int, int);
int memoized_SW_AUX(char*, char*, char**, int**, int, int); // Updated return type

#endif
```