HOMEWORK 05

INTRODUCTION:

One method of solving problems is to divide them into smaller subproblems and solve each subproblem only once, a process known as Dynamic programming. When the problem can be broken down into overlapping subproblems and the answers to these subproblems are saved and utilized again to cut down on computation that would otherwise be necessary, it is especially helpful.

There are two primary categories of dynamic programming: The top-down approach and the Bottom-up approach.

- In Top-down dynamic programming, sometimes referred to as memoization, solves larger subproblems recursively after completing the main problem. To prevent duplicating computation, the solutions are stored in a table.
- In Bottom-up dynamic programming, the smallest subproblems are solved first, and the larger subproblems are solved iteratively using the solutions to the smaller subproblems until the original problem is solved.

For local sequence alignment, the algorithm known as Smith-Waterman is a dynamic programming technique. Even in the case of gaps and mismatches, it is especially helpful for locating similar regions between two biological patterns such as DNA, RNA, or protein sequences.
It works in the following ways:

a. Initialization: Make a matrix in which the rows and columns represent the characters in the two sequences that are being compared. This matrix is frequently referred to as the score/ similarity matrix. Put zeros in the first row and column and start.
b. Scoring methods: Assign points for gaps, mismatches, and matches. A match usually receives a positive score, a mismatch receives a negative score, and opening and extending a gap results in penalties.
c. Completing the matrix: Work through the matrix, assigning a score to each cell based on the scoring scheme and the adjacent cells. Each cell's score is the highest possible score for all alignments that could end at that particular position.
d. Traceback: Upon filling the matrix, identify the cell that has the highest score. The alignment ends with this cell. From this cell, trace backward along the path of the highest scores until it comes to a cell that has a score of zero, designating the alignment's beginning. The ideal local alignment is obtained from this traceback.

1.1 Bottom-Up Approach for SmithWaterman Algorithm:

```
void SW_bottomUp(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] = 0;
            P[i][j] = '-';
        }
    }
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            int match = H[i - 1][j - 1] + (X[i - 1] == Y[j - 1] ? 2 : -1);
            int gapX = H[i - 1][j] - 1;
            int gapY = H[i][j - 1] - 1;

            if (match >= gapX && match >= gapY && match >= 0) {
                H[i][j] = match;
                P[i][j] = '\\';
            } else if (gapX >= gapY && gapX >= 0) {
                H[i][j] = gapX;
                P[i][j] = '|';
            } else if (gapY >= 0) {
                H[i][j] = gapY;
                P[i][j] = '-';
            } else {
                H[i][j] = 0;
                P[i][j] = '-';
            }
        }
    }
}
```

1.2 Top-Down Approach for Smith-Waterman Algorithm:

```
void memoized_SW(char* X, char* Y, char** P, int** H, int n, int m) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            memoized_SW_AUX(X, Y, P, H, i, j);
        }
    }
}

int memoized_SW_AUX(char* X, char* Y, char** P, int** H, int n, int m) {
    if (n == 0 || m == 0) {
        H[n][m] = 0;
        return 0;
    }
    int match;
    if (X[n - 1] == Y[m - 1]) {
        match = memoized_SW_AUX(X, Y, P, H, n - 1, m - 1) + 2;
    } else {
        match = memoized_SW_AUX(X, Y, P, H, n - 1, m - 1) - 1;
    }
    int gap1 = memoized_SW_AUX(X, Y, P, H, n - 1, m) - 1;
    int gap2 = memoized_SW_AUX(X, Y, P, H, n, m - 1) - 1;
    int score = (match > 0 ? match : 0);
    if (score >= gap1 && score >= gap2) {
        H[n][m] = score;
        P[n][m] = '\\';
    } else if (gap1 >= gap2) {
        H[n][m] = gap1;
        P[n][m] = '|';
    } else {
        H[n][m] = gap2;
        P[n][m] = '-';
    }
    return score;
}
```

1.3 Printing PRINT-SEQ-ALIGN-X & PRINT-SEQ-ALIGN-Y

```
void print_Seq_Align_X(char* X, char** P, int n, int m) {
    if (n == 0 || m == 0) {
        return;
    }
    if (P[n][m] == '\\') {
        print_Seq_Align_X(X, P, n - 1, m - 1);
        std::cout << X[n - 1];
    } else if (P[n][m] == '|') {
        print_Seq_Align_X(X, P, n - 1, m);
        std::cout << '-';
    } else if (P[n][m] == '-') {
        print_Seq_Align_X(X, P, n, m - 1);
        std::cout << '-';
    }
}

void print_Seq_Align_Y(char* Y, char** P, int n, int m) {
    if (n == 0 || m == 0) {
        return;
    }
    if (P[n][m] == '\\') {
        print_Seq_Align_Y(Y, P, n - 1, m - 1);
        std::cout << Y[m - 1];
    } else if (P[n][m] == '|') {
        print_Seq_Align_Y(Y, P, n - 1, m);
        std::cout << '-';
    } else if (P[n][m] == '-') {
        print_Seq_Align_Y(Y, P, n, m - 1);
        std::cout << '-';
    }
}
```

1.4 Testing the Algorithm

The H and P are initilzed to zero completely. Then looking up from the previous values on the table the directions and costs are put back to the table. The conditions used are

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j\,, \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

Result for Smith Waterman for:

X= dcdcbacbbb

Y = acdccabdbb

H		a	c	d	c	c	a	b	d	b	b
	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

P	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	d	d	d	l	d	d	d	d	l	l
c	0	d	d	l	d	d	l	l	u	d	d
d	0	d	u	d	l	l	d	d	d	l	l
c	0	d	d	u	d	d	l	l	l	d	d
b	0	d	u	u	u	d	d	d	l	d	d
a	0	d	l	u	u	d	d	l	d	d	d
c	0	u	d	l	d	d	u	d	d	d	d
b	0	u	u	d	d	u	d	d	l	d	d
b	0	d	u	d	d	u	d	d	d	d	d
b	0	d	u	d	d	u	d	d	d	d	d

The values of X= dcdcbacbbb
The values of Y = acdccabdbb
The values of X'= cdcbacb_bb
The values of Y'= cdcca_bdbb

The maximum alignment for X and Y are solved and X' and Y' are obtained. The maximum cost is 11.

2. OPTIMAL BINARY SEARCH TREE:

An optimal binary search tree (BST) is a data structure that aims to reduce the mean length of search for a given set of keys, each with its probability of being searched. Dynamic programming is used in the setting up of an optimal BST, where the lowest expected search cost of building subtrees is calculated using a cost matrix. The average search time is lowered by placing keys with higher probability closer to the root. Optimal BSTs offer a balance between search time and storage capacity.

Given problem states,

i	1	2	3	4	5
p_i	0.21	0.15	0.28	0.12	0.24

➔ The matrix, w,e, and R are initialized accordingly. The w is the weight matrix, e is the Expected cost matrix and the R is the root matrix.

w	0	1	2	3	4	5
1	0					
2		0				
3			0			
4				0		
5					0	
6						0

e	0	1	2	3	4	5
1	0					

2		0				
3			0			
4				0		
5					0	
6						0

R	1	2	3	4	5
1	1				
2		2			
3			3		
4				4	
5					5

The matrix then goes into a main for-loop that iterates for l works on the subtrees with l keys. There the order of the subtree size from smaller to larger is iterated and the values are calculated.

Calculations are shown below:

For i= 1, j = 2:

w[1, 2] = w[1,1] + p_2 = 0.21 + 0.15 = 0.36
e[1, 2] = e[1,0] + e[2,2] + w[1,2] = 0 + 0.15 + 0.36 = 0.51 → r = 1
e[1, 2] = e[1,1] + e[3,2] + w[1,2] = 0.21 + 0 + 0.36 = 0.57 → r = 2
e[1, 2] = min (0.51, 0.57) = 0.51 → R[1,2] = 1

For i= 2, j = 3:

w[2, 3] = w[2, 2] + p_3 = 0.15 + 0.28 = 0.43
e[2, 3] = e[2, 1] + e[3,3] + w[1,2] = 0 + 0.28 + 0.43 = 0.71 → r = 2
e[2, 3] = e[2, 2] + e[4,3] + w[1,2] = 0.15 + 0 + 0.43 = 0.58 → r = 3
e[2, 3] = min (0.71, 0.58) = 0.58 → R[2, 3] = 3

For i= 3, j = 4:

w[3, 4] = w[3, 3] + p_4 = 0.28 + 0.12 = 0.4
e[3, 4] = e[3, 2] + e[4,4] + w[3,4] = 0 + 0.12 + 0.4 = 0.52 → r = 3
e[3, 4] = e[3, 3] + e[5,4] + w[3,4] = 0.28 + 0 + 0.4 = 0.68 → r = 4
e[3, 4] = min (0.52, 0.68) = 0.52 → R[3, 4] = 3

For i= 4, j = 5:

w[4, 5] = w[4, 4] + p_5 = 0.12 + 0.24 = 0.36
e[4, 5] = e[4, 3] + e[5,5] + w[4,5] = 0 + 0.24 + 0.36 = 0.6 → r = 4
e[4, 5] = e[4, 4] + e[6,5] + w[4,5] = 0.12 + 0 + 0.36 = 0.48 → r = 5
e[4, 5] = min (0.6, 0.48) = 0.58 → R[4,5] = 5

For i= 3, j = 5:

w[3, 5] = w[3, 4] + p_5 = 0.4 + 0.24 = 0.64
e[3, 5] = e[3, 2] + e[4,5] + w[4,5] = 0 + 0.48 + 0.64 = 1.12 → r = 3

e[3, 5] = e[3, 3] + e[5,5] + w[4,5] = 0.28 + 0.24 + 0.64 = 1.16 → r = 4
e[3, 5] = e[3, 4] + e[6,5] + w[4,5] = 0.58+ 0 +0.64 = 1.16 → r = 5
e[3, 5] = min (1.12,1.16,1.16) = 1.12 → R[3,5] = 3

For i= 2, j = 4:

w[2, 4] = w[2, 3] + p_4 = 0.43 + 0.12 = 0.55
e[2,4] = e[2, 1] + e[3,4] + w[2,4] = 0 + 0.52 + 0.55 = 1.07 → r = 2
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 → r = 4
e[2, 4] = min (1.07,0.82,1.13) = 0.82 → R[2,4] = 3

For i= 1, j = 3:

w[1, 3] = w[1, 2] + p_3 = 0.36 + 0.28 = 0.64
e[1, 3] = e[1, 0] + e[2,3] + w[1, 3] = 0 + 0.58 + 0.64 = 1.22 → r = 1
e[1, 3] = e[1, 1] + e[3,3] + w[1, 3] = 0.21 + 0.28 + 0.64= 1.13 → r = 2
e[1, 3] = e[1, 2] + e[4,3] + w[1, 3] = 0.51+ 0 +0.64 = 1.15 → r = 3
e[1, 3] = min (1.22,1.13,1.15) = 1.13 → R[1,3] = 2

For i= 2, j = 5:

w[2, 5] = w[2, 4] + p_5 = 0.55 + 0.24 = 0.79
e[2, 5] = e[2, 1] + e[3, 5] + w[2, 5] = 0 + 1.12 + 0.79 = 1.91 → r = 2
e[2, 5] = e[2,2] + e[4, 5] + w[2, 5] = 0.15 + 0.48 + 0.79= 1.42 → r = 3
e[2, 5] = e[2, 3] + e[5, 5] + w[2, 5] = 0.58+ 0.24 +0.79= 1.61→ r = 4
e[2, 5] = e[2, 4] + e[6, 5] + w[2, 5] = 0.82+ 0 +0.79= 1.61 → r = 5
e[2, 5] = min (1.91,1.42,1.61, 1.61) = 1.42 → R[2,5] = 3
For i= 1, j = 4:

w[1, 4] = w[1, 3] + p_3 = 0.64 + 0.12 = 0.76
e[1, 4] = e[1, 0] + e[2, 4] + w[1, 4] = 0 + 0.82+ 0.76 = 1.58 → r = 1
e[1, 4] = e[1,1] + e[3, 4] + w[1, 4] = 0.21 + 0.52 + 0.76= 1.49 → r =2
e[1, 4] = e[1, 2] + e[4,4] + w[1, 4] = 0.51+ 0.12 +0.76= 1.39→ r = 3
e[1, 4] = e[1, 3] + e[5,4] + w[1, 4] = 1.13+ 0 +0.76= 1.89 → r = 4
e[1, 4] = min (1.58,1.49,1.48, 1.89) = 1.48 → R[1,4] = 3

For i= 1, j = 5:

w[1, 5] = w[1, 4] + p_5 = 0.76 + 0.24 = 1.0
e[1, 5] = e[1, 0] + e[2, 5] + w[1, 5] = 0 + 1.42 +1.0 = 2.42 → r = 1
e[1, 5] = e[1,1] + e[3, 5] + w[1, 5] = 0.21+1.12+1=2.33 → r = 2
e[1, 5] = e[1, 2] + e[4, 5] + w[1, 5] = 0.51 + 0.48 + 1= 1.99 → r = 3
e[1, 5] = e[1, 3] + e[5, 5] + w[1, 5] = 1.13 + 0.24+ 1 = 2.37 → r = 4
e[1, 5] = e[1, 4] + e[6, 5] + w[1, 5] = 1.48 + 0 + 1 = 2.48 → r = 5
e[1, 5] = min (2.42,2.33,1.99,2.37,2.48) = 1.99 → R[1,5] = 3

The final w, e and R matrix are shown below:

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	0.4	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	1.13	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

The cost of the given BST is 1.99.

The tree gets k3 as the root and the k2 as left child and k5 as the right child of the root. the k1 will be the left child of k2 and k4 will be the left child of k5.

REFERENCES:

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009) *Introduction to Algorithms* (3rd ed.). The MIT Press.

[2] Sedgewick, R., & Wayne, K. (2011). *Algorithms(4th Edition)*. Addison-Wesley Professional.

x = d c d c b a c b b b

y = a c d c c a b d b b

i		0	1	2	3	4	5	6	7	8	9	10
	y \ x	—	d	c	d	c	b	a	c	b	b	b
0	—	0	0	0	0	0	0	0	0	0	0	0
1	a	0	-1	-1	-1	-1	-1	2	1	0	-1	-1
2	c	0	-1	1	0	1	0	1	4	3	2	1
3	d	0	2	1	3	2	1	0	3	3	2	1
4	c	0	1	4	3	5	4	3	2	2	2	1
5	c	0	0	3	3	5	4	3	5	4	3	2
6	a	0	-1	2	2	4	4	6	5	4	3	2
7	b	0	-1	1	1	3	6	5	5	7	6	5
8	d	0	2	1	3	2	5	5	4	6	6	5
9	b	0	1	1	2	2	4	4	4	6	8	8
10	b	0	0	0	1	1	4	3	3	6	8	10

x = d c d c b a c b b b

y = a c d c c a b d b b

~~(8,1)~~ (1,2) (3,1)

P_1 $(1-1)(2-1) = (0,1) = 0$ $(3)(0) = 0-1 = -1$

P_2 $(0)(2) - 1 = -1$ (

P_3 $(1)($

$(i-1)(j-1)-1$ $(i-1)(j)-1$ $(i)(j-1)-1$

	P_1 ↖	P_2 ↑	P_3 ←
(1,1)	-1	-1	-1
(2,1)	-1	-1-1 = -2	0-1 = -1
(3,1)	-1	-1-1 = -2	0-1 = -1
(4,1)	-1	-1-1 = -2	0-1 = -1
(5,1)	-1	-1-1 = -2	-1
(6,1)	0+2 = 2	—	—
(7,1)	0-1 = -1	2-1 = 1	0-1 = -1
(8,1)	0-1 = -1	1-1 = 0	0-1 = -1
(9,1)	0-1 = -1	0-1 = -1	0-1 = -1
(10,1)	0-1 = -1	-1-1 = -2	0-1 = -1
(1,2)	(0,1)-1 = -1	(0,2)-1 = -1	(1,1)-1 = -2
(2,2)	(1,1) = -1+2 = 2	~~(1)(2) = 1 = 3~~	
	-1+2 = 1		
(3,2)	(2,1) = -1-1 = -2	(2,2)-1 = 1-1 = 0	(3,1)-1 = -2
(4,2)	(3,1)+2 = -1+2 = 1	(4,2)-1 = 1-1 = 0	(5,1)-1 = -2
(5,2)	(4,1) = -1-1 = -2	(5,2)-1 = 0-1 = -1	(6,1)-1 = 2-1 = 1
(6,2)	(5,1) = -1-1 = -2		
(7,2)	(6,1) = 2+2 = 4	(7,2)-1 = 4-1 = 3	(8,1) = 0-1 = -1
(8,2)	(7,1) = 1-1 = 0	(8)(2) = 3-1 = 2	(9,1)-1 = -1-1 = -2
(9,2)	(8,1) = 0-1 = -1	(9)(2) = 2-1 = 1	(10,1)-1 = -1-1 = -2
(10,2)	(9,1) = -1-1 = -2		
(1,3)	(0,2)+2 = 2	—	—
(2,3)	(1,2)-1 = -3	(1,3)-1 = 1	(2,2)-1 = 0
(3,3)	(2,2)+2 = 1+2 = 3	—	—
(4,3)	(3,2)-1 = 0-1 = -1	(3,3)-1 = 2	(4,2)-1 = 0

	P_1 $(i-1)(j-1)$	$P_2(i-1)(j)$	$P_3(i)(j-1)$
(5,3)	(4,2)-1 = 0	(4,3)-1 = 2-1 = ①	(5,2)-1 = -1
(6,3)	(5,2)-1 = -1	(5,3)-1 = ⓪	(6,2)-1 = 0
(7,3)	(6,2)-1 = 0	(6,3)-1 = 0-1 = -1	(7,2)-1 = ③
(8,3)	(7,2)-1 = 4-1 = ③	(7,3)-1 = 3-1 = 2	(8,2)-1 = 3-1 = 2
(9,3)	(8,2)-1 = 3-1 = ②	(8,3)-1 = 3-1 = 2	(9,2)-1 = 2-1 = 1
(10,3)	(9,2)-1 = 2-1 = ①	(9,3)-1 = 2-1 = 1	(10,2)-1 = 0
(1,4)	(0,3)-1 = -1	(0,4)-1 = -1	(1,3)-1 = 2-1 = ①
(2,4)	(1,3)+2 = ④	—	—
(3,4)	(2,3)-1 = 0	(2,4)-1 = ③	(2,3)-1 = 2
(4,4)	(3,3)+2 = ⑤	—	—
(5,4)	(4,3)-1 = 2-1 = 1	(4,4)-1 = ④	(5,3)-1 = 1-1 = 0
(6,4)	(5,3)-1 = 0	(5,4)-1 = ③	(6,3)-1 = 0-1 = -1
(7,4)	(6,3)+2 = ②	—	—
(8,4)	(7,3)-1 = ②	(7,4)-1 = 1	(8,3)-1 = 3-1 = 2
(9,4)	(8,3)-1 = ②	(8,4)-1 = 1	(9,3)-1 = 2-1 = 1
(10,4)	(9,3)-1 = ①	(9,4)-1 = 1	(10,3)-1 = 1-1 = 0
(1,5)	(0,4)-1 = -1	(0,5)-1 = -1	(1,4)-1 = ⓪
(2,5)	(1,4)+2 = ③	—	—
(3,5)	(2,4)-1 = ③	(2,5)-1 = 2	(3,4)-1 = 2
(4,5)	(3,4)+2 = ⑤	—	—
(5,5)	(4,4)-1 = ④	(4,5)-1 = 4	(5,4)-1 = 3
(6,5)	(5,4)-1 = ③	(5,5)-1 = 3	(6,4)-1 = 2
(7,5)	(6,4)+2 = ⑤	—	—
(8,5)	(7,4)-1 = 1	(7,5)-1 = ④	(8,4)-1 = 1
(9,5)	(8,4)-1 = 1	(8,5)-1 = ③	(9,4)-1 = 1
(10,5)	(9,4)-1 = 1	(9,5)-1 = ②	(10,4)-1 = 0

	$P_1(i-1)(j-1)$	$P_2(i-1)(j)$	$P_3(i)(j-1)$
(1,6)	(0,5)-1 = (-1)	(0,6)-1 = -1	(1,5)-1 = -1
(2,6)	(1,5)-1 = -1	(1,6)-1 = -2	(2,5)-1 = (2)
(3,6)	(2,5)-1 = (2)	(2,6)-1 = 1	(3,5)-1 = 2
(4,6)	(3,5)-1 = 2	(3,6)-1 = 1	(4,5)-1 = (4)
(5,6)	(4,5)-1 = (4)	(4,6)-1 = 3	(5,5)-1 = 3
(6,6)	(5,5)+2 = (6)	~~(5,6)~~	—
(7,6)	(6,5)-1 = 2	(6,6)-1 = (5)	(7,5)-1 = 4
(8,6)	(7,5)-1 = (4)	(7,6)-1 = 4	(8,5)-1 = 3
(9,6)	(8,5)-1 = (3)	(8,6)-1 = 3	(9,5)-1 = 2
(10,6)	(9,5)-1 = (2)	(9,6)-1 = 2	(10,5)-1 = 1
(1,7)	(0,6)-1 = (-1)	(0,7)-1 = -1	(1,6)-1 = -2
(2,7)	(1,6)-1 = -2	(1,7)-1 = -2	(2,6)-1 = (1)
(3,7)	(2,6)-1 = (1)	(2,7)-1 = 0	(3,6)-1 = 1
(4,7)	(3,6)-1 = 1	(3,7)-1 = 0	(4,6)-1 = (3)
(5,7)	(4,6)+2 = (6)	~~(5,6)~~	—
(6,7)	(5,6)-1 = 3	(5,7)-1 = (5)	(6,6)-1 = 5
(7,7)	(6,6)-1 = (5)	(6,7)-1 = 4	(7,6)-1 = 4
(8,7)	(7,6)+2 = (7)	—	—
(9,7)	(8,6)+2 = (6)	—	—
(10,7)	(9,6)+2 = (5)	—	—
(1,8)	(0,7)+2 = (2)	—	—
(2,8)	(1,7)-1 = -2	(1,8)-1 = (1)	(2,7)-1 = 0
(3,8)	(2,7)+2 = (3)	—	—
(4,8)	(3,7)-1 = 0	(3,8)-1 = (2)	(4,7)-1 = 2
(5,8)	(4,7)-1 = 2	(4,8)-1 = 1	(5,7)-1 = (5)
(6,8)	(5,7)-1 = (5)	(5,8)-1 = 4	(6,7)-1 = 4
(7,8)	(6,7)-1 = (4)	(6,8)-1 = 4	(7,7)-1 = 4
(8,8)	(7,7)-1 = 4	(7,8)-1 = 3	(8,7)-1 = (6)
(9,8)	(8,7)-1 = (6)	(8,8)-1 = 5	(9,7)-1 = 5
(10,8)	(9,7)-1 = (5)	(9,8)-1 = 5	(10,7)-1 = 4

	$P_1(i-1)(j-1)$	$P_2(i-1)(j)$	$P_3(i)(j-1)$
(1,9)	(0,8)-1 = -1	(0,9)-1 = -1	(1,8)-1 = 1
(2,9)	(1,8)-1 = 1	(1,9)-1 = 0	(2,8)-1 = 0
(3,9)	(2,8)-1 = 0	(2,9)-1 = 0	(3,8)-1 = 2
(4,9)	(3,8)-1 = 2	(3,9)-1 = 1	(4,8)-1 = 1
(5,9)	(4,8)+2 = 4	—	—
(6,9)	(5,8)-1 = 4	(5,9)-1 = 3	(6,8)-1 = 4
(7,9)	(6,8)-1 = 4	(6,9)-1 = 3	(7,8)-1 = 3
(8,9)	(7,8)+2 = 6	—	—
(9,9)	(8,8)+2 = 8	—	—
(10,9)	(9,8)+2 = 8	—	—

(1,10)	-1	-1	0
(2,10)	0	-1	0
(3,10)	0	-1	1
(4,10)	1	0	1
(5,10)	1	~~0~~	~~3~~ 1
(6,10)	~~4~~	2	3
(7,10)	3	2	3
(8,10)	3	~~2~~	~~5~~
(9,10)	~~6~~ 5	~~4~~	~~7~~
(10,10)	7	~~6~~	~~7~~