알고리즘분석 실습 자료

6주차

실습 소요 시간 100분

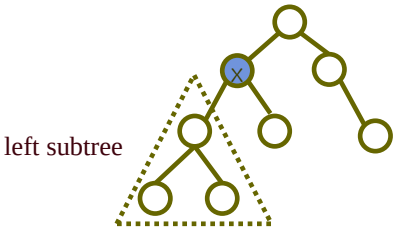
동적계획법

실습프로그램

- ✓ 최적이진검색트리 구축 알고리즘
- ✓ DNA 서열 맞춤 알고리즘

최적 이진검색 트리

- left(right) subtree: 이진트리에서 어떤 마디의 왼쪽(오른쪽)자식마디가 뿌리마디가 되는 부분트리
- 이진검색트리(binary search tree): 순서가능집합(ordered set)에 속한 아이템(키)으로 구성된 이진 트리
  - ✓ 각 마디는 하나의 키만 가지고 있다
  - ✓주어진 마디의 왼쪽(오른쪽) 부분트리에 있는 키는 그 마디의 키보다 작거나(크거나) 같다.



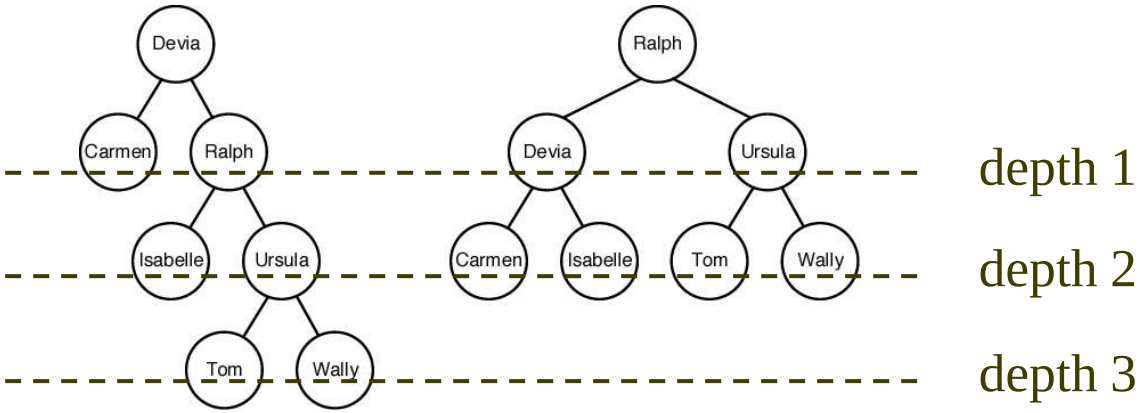


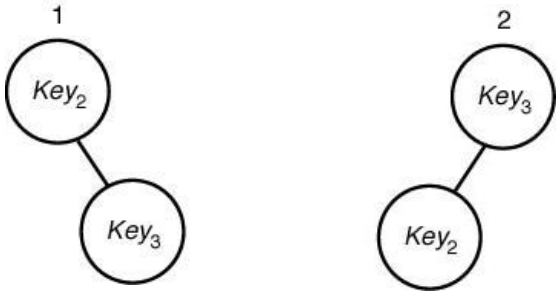
그림 3.10 두 이진검색 트리의 예

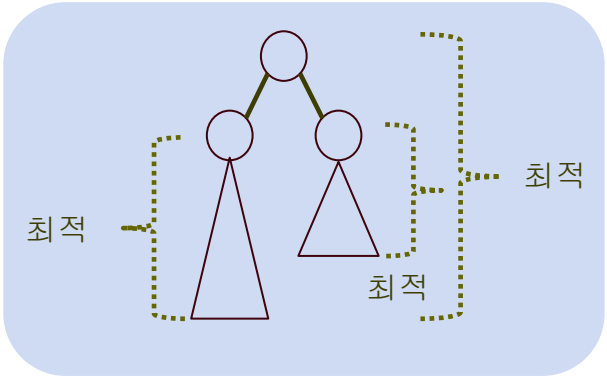
동적계획법

- Key_i 부터 Key_j 까지 키를 포함하는 최적이진검색트리는 $\sum_{m=i}^{j} c_m p_m$ 를 최소화해야 함.
- 검색시간 최적값을 A[*i*][*j*]로 표시. A[*i*][*i*]=p_i
- (예 3.8)

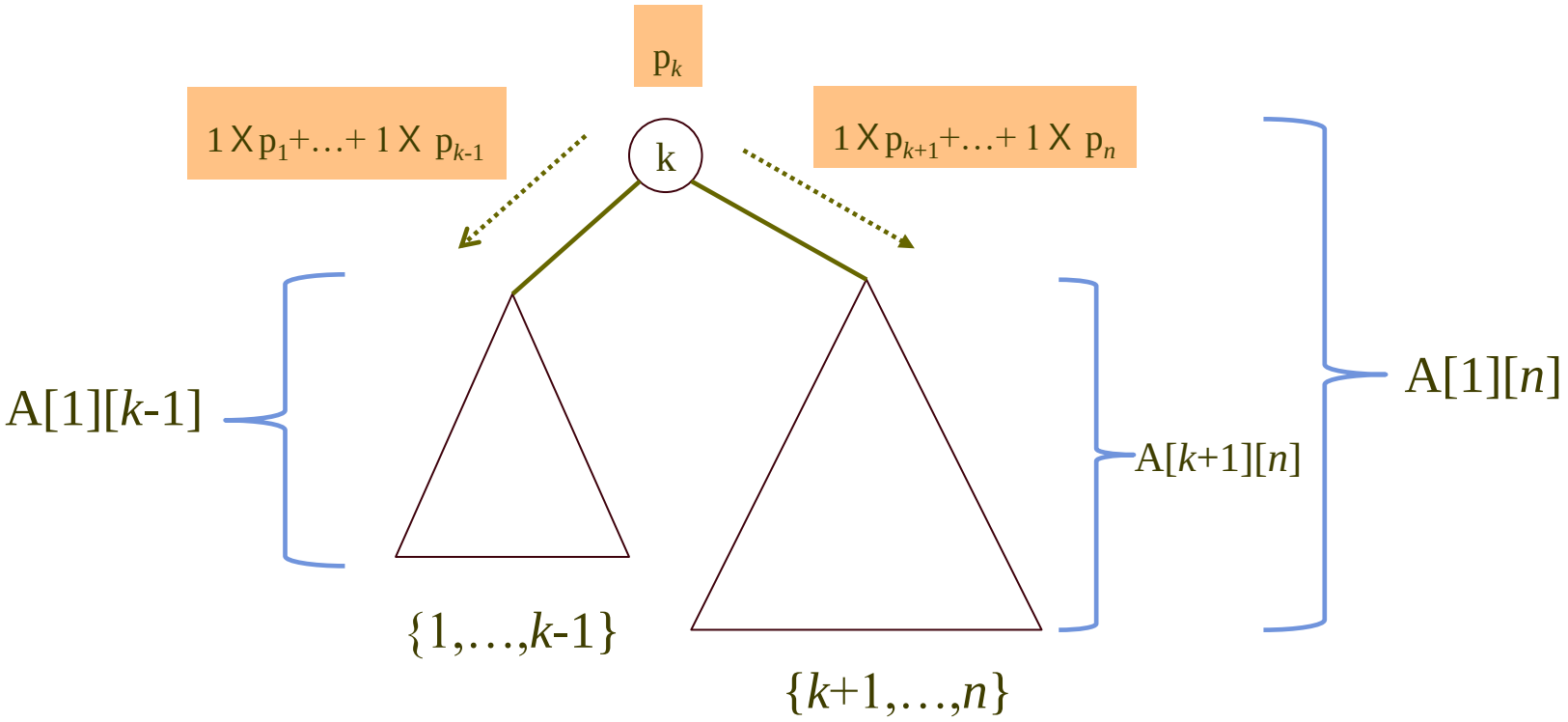
  p_1=0.7, p_2=0.2, p_3= 0.1일 때 A[2][3]?

  1. $1(p_2)+2(p_3) = 1(0.2) + 2(0.1) = 0.4$
  2. $2(p_2)+1(p_3) = 2(0.2) + 1(0.1) = 0.5$





- 최적트리의 부분트리는 그 부분트리안에 있는 키들에 대해서 반드시 최적이어야 한다. → 최적의 원칙 적용



$$
\begin{aligned}
A[1][n] &= A[1][k-1] && /* \text{왼쪽 부분트리에서 평균시간} \\
&+ p_1 + \ldots + p_{k-1} && /* \text{뿌리에서 비교하는데 드는 추가시간} \\
&+ p_k && /* \text{뿌리를 검색하는 평균시간} \\
&+ A[k+1][n] && /* \text{오른쪽 부분트리에서 평균시간} \\
&+ p_{k+1} + \ldots + p_n && /* \text{뿌리에서 비교하는데 드는 추가시간} \\
&= A[1][k-1] + A[k+1][n] + \sum_{m=1}^{n} p_m
\end{aligned}
$$

- 문제: 최적이진검색트리 구하기

```
void optsearchtree(int n, const float p[], float& minavg
                   index R[][] ){
   index i, j, k, diagonal;
   float A[1..n+1][0..n];

   for( i=1; i<=n; i++){
         A[i][i-1]=0; A[i][i]=p[i]; R[i][i]=i; R[i][i-1]=0;
   }
   A[n+1][n]=0;
   R[n+1][n]=0;
   for(diagonal=1; diagonal<=n-1; diagonal++){
         for(i=1; i<=n-diagonal; i++){
             j = i+diagonal;
             A[i][j] = min_{i≤k≤j}(A[i][k-1]+A[k+1][j])+ Σ_{m=i}^{j} p_m ;
             R[i][j] = 최소값을 주는 k의 값
             }
   minavg = A[1][n];
}
```

- 예 3.9

$p_1=3/8, p_2=3/8, p_3=1/8, p_4=1/8$

	0	1	2	3	4
1	0	3/8	9/8	11/8	7/4
2		0	3/8	5/8	1
3			0	1/8	3/8
4				0	1/8
5					0

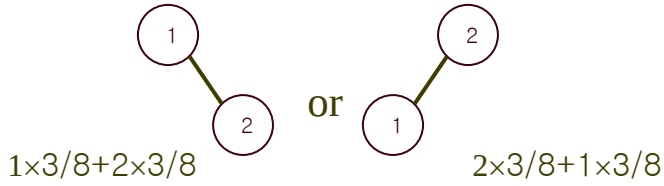
diagonal=3

diagonal=2

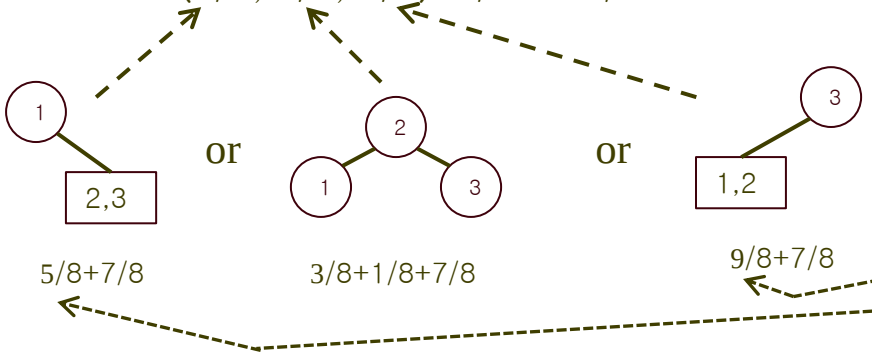
diagonal=1

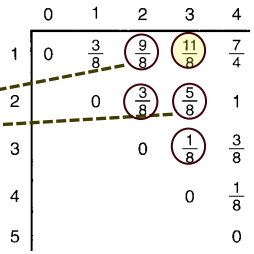
진행순서

A[1][2] = min(A[1][0]+A[2][2],A[1][1]+A[3][2])+6/8
= min(3/8,3/8)+6/8 = 9/8



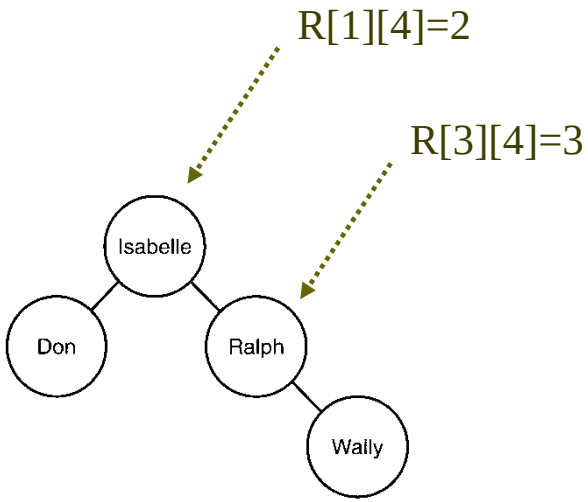
A[1][3] = min(A[1][0]+A[2][3], A[1][1]+A[3][3], A[1][2]+A[4][3])
+7/8
= min(5/8, 4/8, 9/8)+7/8 = 11/8





	0	1	2	3	4
1	0	1	1	2	2
2		0	2	2	2
3			0	3	3
4				0	4
5					0

R



- 문제: 최적이진검색트리 구축

```
node_pointer tree (index i, index j) {
    index k;
    node_pointer p;

    k = R[i][j];
    if (k == 0)
        return NULL;
    else {
        p = new nodetype;
        p->key = Key[k];
        p->left = tree(i, k-1);
        p->right = tree(k+1, j);
        return p;
    }
}
```

- root = tree(1, n)

[실습프로그램] 최적이진검색트리 구축 알고리즘 구현

```
import utility

class Node:
    def __init__(self,data):
        self.l_child=None
        self.r_child=None
        self.data = data

def tree(key,r,i,j):
    k=r[i][j]
    if(k==0):
        return
    else:
        p=Node(key[k])
        p.l_child=tree(key,r,i,k-1)
        p.r_child=tree(key,r,k+1,j)
        return p
```

```
key=[" ","A","B","C","D"]
p=[0,0.375, 0.375, 0.125,0.125]
n=len(p)-1

a=[[0 for j in range(0,n+2)] for i in range(0,n+2)]
r=[[0 for j in range(0,n+2)] for i in range(0,n+2)]

for i in range (1,n+1):
    a[i][i-1]=0
    a[i][i]=p[i]
    r[i][i]=i
    r[i][i-1]=0
a[n+1][n]=0
r[n+1][n]=0
```

$p_1=3/8, p_2=3/8, p_3=1/8, p_4=1/8$

구현

```
utility.printMatrixF(a)
print()
utility.printMatrix(r)

root=tree(key,r,1,n)
utility.print_inOrder(root)
print()
utility.print_preOrder(root)
```

[실습프로그램] 최적이진검색트리 구축 알고리즘 output

```
>>>
 0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.38  1.12  1.38  1.75  0.00
 0.00  0.00  0.38  0.62  1.00  0.00
 0.00  0.00  0.00  0.12  0.38  0.00
 0.00  0.00  0.00  0.00  0.12  0.00
 0.00  0.00  0.00  0.00  0.00  0.00

   0     0     0     0     0     0
   0     1     1     2     2     0
   0     0     2     2     2     0
   0     0     0     3     3     0
   0     0     0     0     4     0
   0     0     0     0     0     0
A
B
C
D

B
A
C
D
>>>
```

inorder

preorder

utility 내 함수들

```
# when use them, from utility. utility.printMatrix(m)
#print integer matrix
def printMatrix(d):
    m = len(d)
    n=len(d[0])

    for i in range(0,m):
        for j in range(0,n):
            print(f'{d[i][j]:4d}',end=" ")
        print()

#print float matrix
def printMatrixF(d):
    n=len(d[0])
    for i in range(0,n):
        for j in range(0,n):
            print(f'{d[i][j]:5.2f}',end=" ")
        print()
```

```
def print_inOrder(root):
    if not root:
        return
    print_inOrder(root.l_child)
    print(root.data)
    print_inOrder(root.r_child)

def print_preOrder(root):
    if not root:
        return
    print(root.data)
    print_preOrder(root.l_child)
    print_preOrder(root.r_child)

def print_postOrder(root):
    if not root:
        return

    print_postOrder(root.l_child)
    print_postOrder(root.r_child)
    print(root.data)
```

DNA 서열 맞춤(sequence alignment)

- 분자유전학(molecular genetics)의 한 분야인 동족(homologous) DNA 서열 맞춤문제를 동적계획법으로 해결하는 방법 소개
- 먼저 divide-and-conquer 방법을 설명하고, 동적계획법 방법을 설명한다.

- DNA 서열 맞춤 문제:

두 개의 서열을 최소비용으로 맞추는 방법을 찾는다.

서열을 배열로 표시

x[ ]	0	1	2	3	4	5	6	7	8	9
	A	A	C	A	G	T	T	A	C	C

y[ ]	0	1	2	3	4	5	6	7
	T	A	A	G	G	T	C	A

- opt(i,j): 부분 서열 x[i,...,9]와 y[j,...,7]의 최적 맞춤 비용
- opt(0,0): 부분 서열 x[0,...,9]와 y[0,...,7]의 최적 맞춤 비용, 즉, 우리가 구하려는 최종 비용

틈: 2 불일치: 1

opt(*i*,*j*): 부분 서열 *x*[*i*,…,9]와 *y*[*j*,…,7]의 최적 맞춤 비용

$$opt(0,0)=\min(opt(1,1)+penalty,\ opt(1,0)+2,\ opt(0,1)+2)$$

① ② ③

- *penalty*=0 if x[0]=y[0]
- *penalty*=1 if x[0]≠y[0]

①

x[ ]	0	1	2	3	4	5	6	7	8	9
	A	A	C	A	G	T	T	A	C	C

y[ ]	0	1	2	3	4	5	6	7
	T	A	A	G	G	T	C	A

penalty=1

②

x[ ]	0	1	2	3	4	5	6	7	8	9
	A	A	C	A	G	T	T	A	C	C

y[ ]		0	1	2	3	4	5	6	7
	-	T	A	A	G	G	T	C	A

틈 1개에 의해 비용 2 발생

③

x[ ]		0	1	2	3	4	5	6	7	8	9
	-	A	A	C	A	G	T	T	A	C	C

y[ ]	0	1	2	3	4	5	6	7
	T	A	A	G	G	T	C	A

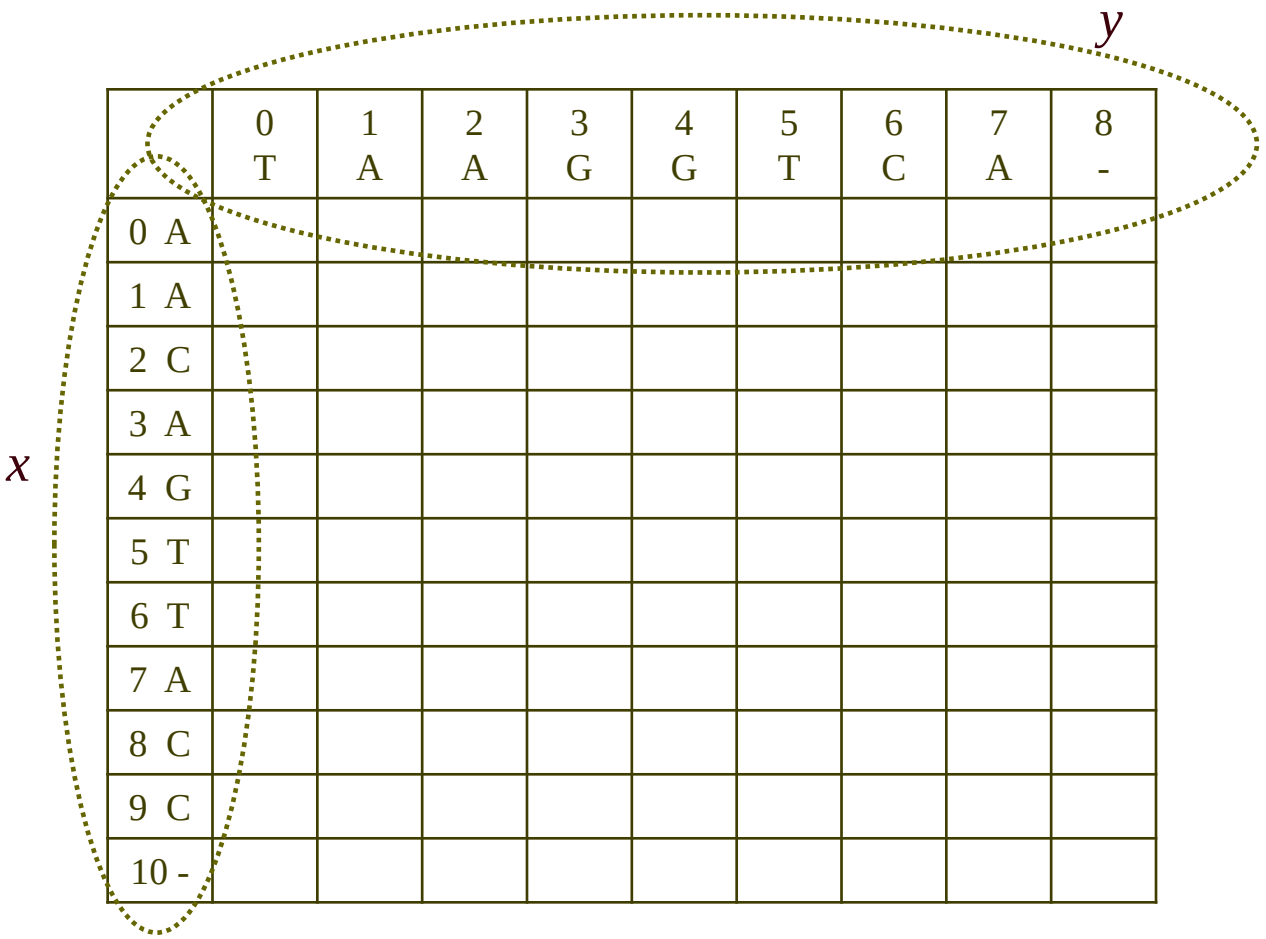
틈 1개에 의해 비용 2 발생

- 일반식은

$$opt(i,j)=\min(opt(i+1,j+1)+penalty,\ opt(i+1,j)+2,\ opt(i,j+1)+2)$$

동적계획법을 이용한 최적맞춤 방법 찾기

- m+1, n+1 크기의 2차원 배열. m=10, n=8
- 마지막 행, 마지막 열에 '– ' 틈 문자 추가

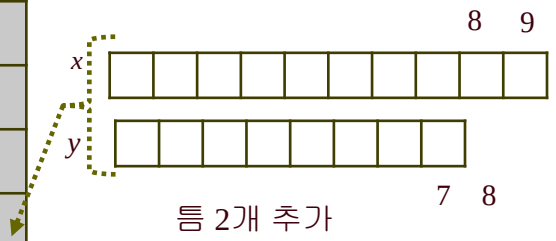


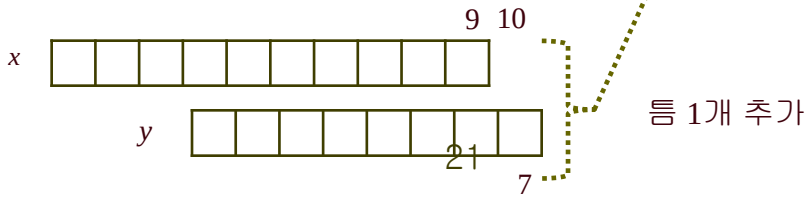
	0 T	1 A	2 A	3 G	4 G	5 T	6 C	7 A	8 -
0 A									
1 A									
2 C									
3 A									
4 G									
5 T									
6 T									
7 A									
8 C									
9 C									
10 -									

1. 틈 행과 틈 열을 채워 넣는다

$opt(10,j)=2(8-j)$, $opt(i,8)=2(10-i)$, 틈의 비용=2

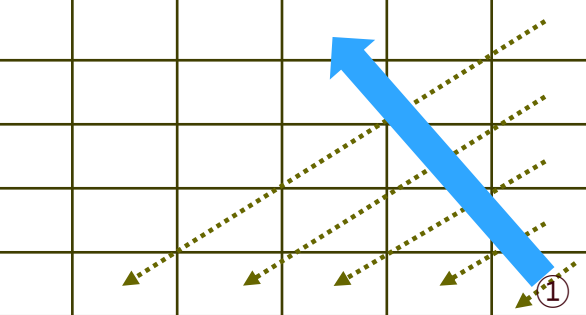
i \ j	0 T	1 A	2 A	3 G	4 G	5 T	6 C	7 A	8 -
0 A									20
1 A									18
2 C									16
3 A									14
4 G									12
5 T									10
6 T									8
7 A									6
8 C									4
9 C									2
10 -	16	14	12	10	8	6	4	2	0





2. 우측 아래 부터 대각원소들을 채워 넣는다

j / i	0 T	1 A	2 A	3 G	4 G	5 T	6 C	7 A	8 -
0 A									20
1 A									18
2 C									16
3 A									14
4 G									12
5 T									10
6 T									8
7 A									6
8 C									4
9 C								①	2
10 -	16	14	12	10	8	6	4	2	0



x[9]와 y[7]의 일치여부에 의한 penalty

① opt(9,7)= min(opt(9+1,7+1)+penalty, opt(9+1,7)+2, opt(9,7+1)+2)

= min(0+1, 2+2, 2+2)

=1

C과 A로 다르므로

i \ j	0 T	1 A
0 A	7	8
1 A	6	6

- 최적서열맞춤을 찾아가는 방법
  - ✓ opt(0,0)에서 출발하여, 3가지 가능성을 조사

(1) [0][0] 선택

(2) 경로에 넣을 둘째 항목을 찾는다

1) [0][1]을 검사

opt(0,1)+2=8+2=10 ≠7

2) [1][0]을 검사

opt(1,0)+2=6+2=8 ≠7

3) [1][1]을 검사

opt(1,1)+1=6+1=7. 찾음

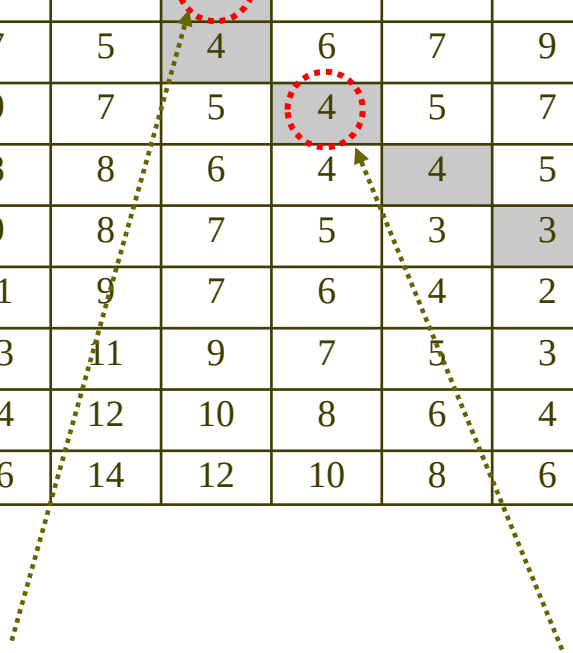
([0][0]이 A, T로 서로 다르므로 penalty=1)

i \ j	0 T	1 A	2 A	3 G	4 G	5 T	6 C	7 A	8 -
0 A	7	8	10	12	13	15	16	18	20
1 A	6	6	8	10	11	13	14	16	18
2 C	6	5	6	8	9	11	12	14	16
3 A	7	5	4	6	7	9	11	12	14
4 G	9	7	5	4	5	7	9	10	12
5 T	8	8	6	4	4	5	7	8	10
6 T	9	8	7	5	3	3	5	6	8
7 A	11	9	7	6	4	2	3	4	6
8 C	13	11	9	7	5	3	1	3	4
9 C	14	12	10	8	6	4	2	1	2
10 -	16	14	12	10	8	6	4	2	0

✓최적서열맞춤 해는 해당 cell을 음영으로 표시했다.

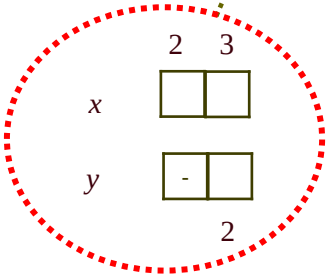
- 배열값을 채워 넣을 때 어디로 부터 min을 구했는지의 정보를 저장할 수 있다.

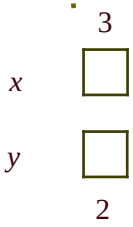
j i	0 T	1 A	2 A	3 G	4 G	5 T	6 C	7 A	8 -
0 A	7	8	10	12	13	15	16	18	20
1 A	6	6	8	10	11	13	14	16	18
2 C	6	5	6	8	9	11	12	14	16
3 A	7	5	4	6	7	9	11	12	14
4 G	9	7	5	4	5	7	9	10	12
5 T	8	8	6	4	4	5	7	8	10
6 T	9	8	7	5	3	3	5	6	8
7 A	11	9	7	6	4	2	3	4	6
8 C	13	11	9	7	5	3	1	3	4
9 C	14	12	10	8	6	4	2	1	2
10 -	16	14	12	10	8	6	4	2	0



- min(6+1, 4+2, 8+2)
- min(4+0, 4+2, 5+2)

j i	0 T	1 A	2 A	3 G	4 G	5 T	6 C	7 A	8 -
0 A	7	8	10	12	13	15	16	18	20
1 A	6	6	8	10	11	13	14	16	18
2 C	6	5	6	8	9	11	12	14	16
3 A	7	5	4	6	7	9	11	12	14
4 G	9	7	5	4	5	7	9	10	12
5 T	8	8	6	4	4	5	7	8	10
6 T	9	8	7	5	3	3	5	6	8
7 A	11	9	7	6	4	2	3	4	6
8 C	13	11	9	7	5	3	1	3	4
9 C	14	12	10	8	6	4	2	1	2
10 -	16	14	12	10	8	6	4	2	0





x[2]와 y[2]를 맞추는 방법은 y에 – 를 넣은 것이 최적

- 경로를 찾은 후 맞춤된 서열을 구성하는 방법

1. 최적배열의 오른쪽 맨 아래 구성에서 시작하여 표시해둔 경로를 따라간다.
2. 배열의 [*i*][*j*]칸에 도달하기 위해서 대각선으로 이동할 때 마다, *i*째 행에 해당하는 문자를 *x*서열에 넣고, *j*째 열에 해당하는 문자를 *y*서열에 넣는다.
3. 배열의 [*i*][*j*]칸에 도달하기 위해서 위로 이동할 때 마다, *i*째 행에 해당하는 문자를 *x*서열에 넣고, 틈 문자를 *y*서열에 넣는다.
4. 배열의 [*i*][*j*]칸에 도달하기 위해서 왼쪽으로 이동할 때 마다, *j*째 열에 해당하는 문자를 *y*서열에 넣고, 틈 문자를 *x*서열에 넣는다.

j / i	0 T	1 A	2 A	3 G	4 G	5 T	6 C	7 A	8 -
0 A	7	8	10	12	13	15	16	18	20
1 A	6	6	8	10	11	13	14	16	18
2 C	6	5	6	8	9	11	12	14	16
3 A	7	5	4	6	7	9	11	12	14
4 G	9	7	5	4	5	7	9	10	12
5 T	8	8	6	4	4	5	7	8	10
6 T	9	8	7	5	3	3	5	6	8
7 A	11	9	7	6	4	2	3	4	6
8 C	13	11	9	7	5	3	1	3	4
9 C	14	12	10	8	6	4	2	1	2
10 -	16	14	12	10	8	6	4	2	0

- 최적해

A A C A G T T A C C

T A - A G G T - C A

1. 최적배열의 오른쪽 맨 아래 구성에서 시작하여 표시해둔 경로를 따라간다.
2. 배열의 [*i*][*j*]칸에 도달하기 위해서 대각선으로 이동할 때 마다, *i*째 행에 해당하는 문자를 *x*서열에 넣고, *j*째 열에 해당하는 문자를 *y*서열에 넣는다.
3. 배열의 [*i*][*j*]칸에 도달하기 위해서 위로 이동할 때 마다, *i*째 행에 해당하는 문자를 *x*서열에 넣고, 틈 문자를 *y*서열에 넣는다.
4. 배열의 [*i*][*j*]칸에 도달하기 위해서 왼쪽으로 이동할 때 마다, *j*째 열에 해당하는 문자를 *y*서열에 넣고, 틈 문자를 *x*서열에 넣는다.

[실습프로그램] DNA 서열 맞춤 알고리즘 구현

```
import utility
a=['A','A','C','A','G','T','T','A','C','C']
b=['T','A','A','G','G','T','C','A']

m=len(a)
n=len(b)
table=[[0 for j in range(0,n+1)] for i in range(0,m+1)]
minindex = [[ (0,0) for j in range(0,n+1)] for i in range(0,m+1)]

for j in range(n-1,-1,-1):
    table[m][j] =table[m][j+1]+2

for i in range(m-1,-1,-1):
    table[i][n] =table[i+1][n]+2


        테이블 생성 구현

utility.printMatrix(table)
x=0
y=0

while (x <m and y <n):
    tx, ty = x, y
    print(minindex[x][y])
    (x,y)= minindex[x][y]
    if x == tx + 1 and y == ty+1:
        print(a[tx],"  ",  b[ty])
    elif x == tx and y == ty+1:
        print(" - ", " ", b[ty])
    else:
        print(a[tx], " " , " -")
```

DNA 서열 맞춤 알고리즘 output

```
  7    8   10   12   13   15   16   18   20
  6    6    8   10   11   13   14   16   18
  6    5    6    8    9   11   12   14   16
  7    5    4    6    7    9   11   12   14
  9    7    5    4    5    7    9   10   12
  8    8    6    4    4    5    7    8   10
  9    8    7    5    3    3    5    6    8
 11    9    7    6    4    2    3    4    6
 13   11    9    7    5    3    1    3    4
 14   12   10    8    6    4    2    1    2
 16   14   12   10    8    6    4    2    0
```


```
(1, 1)
A   T
(2, 2)
A   A
(3, 2)
C    -
(4, 3)
A   A
(5, 4)
G   G
(6, 5)
T   G
(7, 6)
T   T
(8, 6)
A    -
(9, 7)
C   C
(10, 8)
C   A
>>>
```