3장 DP 정리

이항 계수

이항 계수 식

$$C(n,k) = \frac{n!}{k!(n-k)!} \quad \text{for} \quad 0 \le k \le n$$

$$\rightarrow C(n,k) = \begin{cases} C(n-1,k) + C(n-1,k-1) & \text{for} \quad 0 < k < n \\ 1 & \text{for } k=0 \text{ or } k=n \end{cases}$$

분할 정복 이항 계수

분할 정복.

```
public static int bin(int n, int k)
{
    if (k==0 || k==n)
        return 1 ;

    else
        return bin(n-1,k-1) + bin(n-1,k) ;

}
```

DP 슈도 코드

for문 j 조건 min(i,k) 주의

Basic Operation: an assignment of B[i][j]
Input Size: n, k

	i	#of passes
$\frac{k(k+1)}{2}$	0	1
	1	2
	2	3
	…	…
$(n-k+1)\cdot(k+1)$	k	k+1
	k+1	k+1
	…	…
	n	k+1

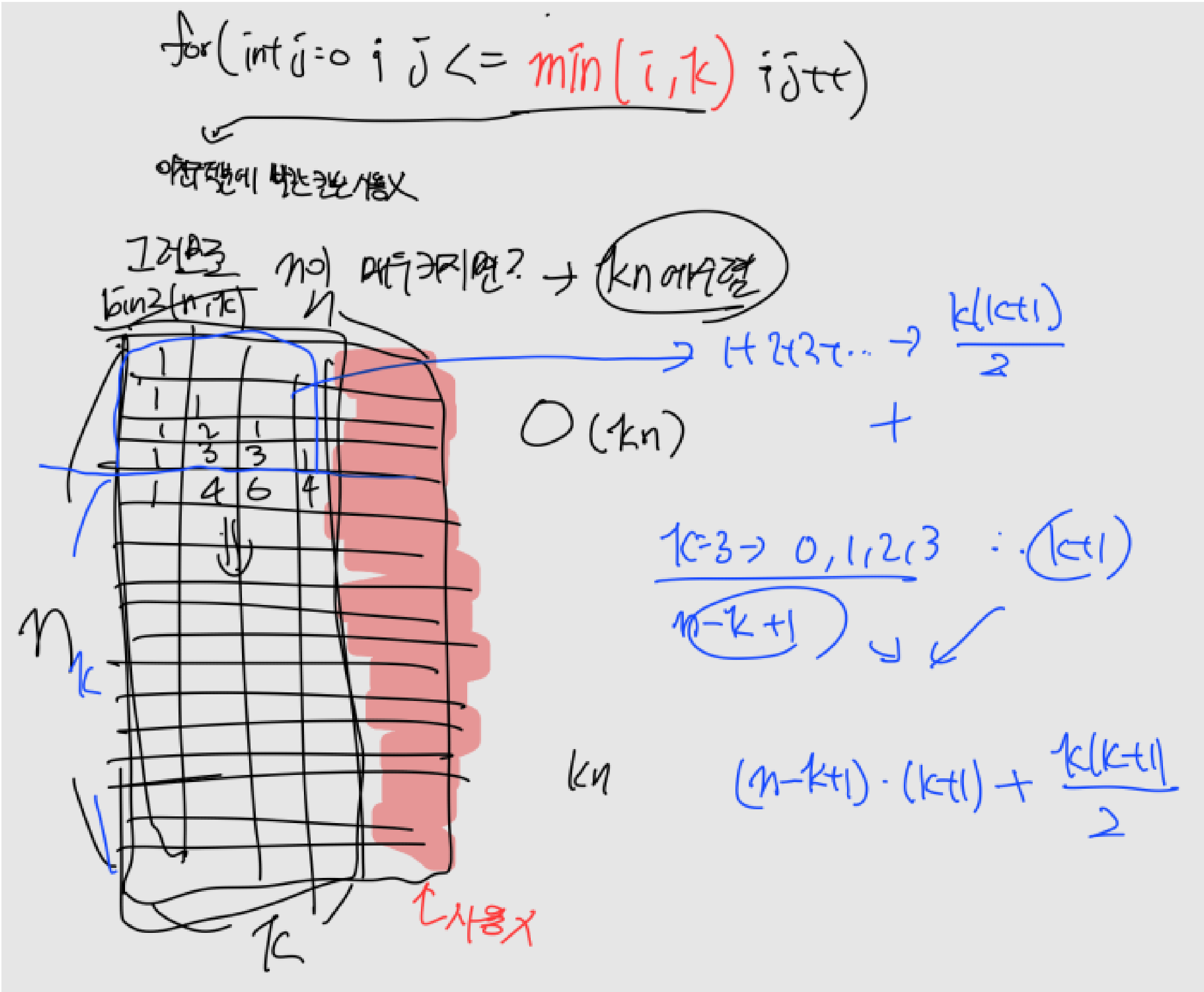
$\rightarrow$ **Total number of passes**

$= 1+2+3+ \ldots + (k+1) + (k+1) \ldots + (k+1)$

$= k(k+1)/2 \quad + (n-k+1)(k+1)$

$= (2n-k+2)(k+1)/2$

$\in \Theta(nk)$



플로이드 알고리즘

❑ Floyd's Algorithm

```
void floyd(int n, const number W[ ][ ],
                     const number D[ ][ ])
{
   index i,j,k ;

   D = W ;
   for (k=1; k<= n ;k++)
     for (i=1; i <= n ; i++)
       for (j=1; j <= n ; j++)
         D[i][j]=min(D[i][j],D[i][k]+D[k][j]);
}
```

Every-Case Complexity:

Basic Operation:

the assignment operation in for–J loop

Input Size :

n, number of vertices

➔ $T(n) = n \times n \times n = n^3$

간단한 설명

다익스트라 - 모든 정점 ~ 모든 정점의 최단거리.

시간복잡도 $O(n^3)$

공간복잡도 $O(n^2)$ ← 2차원 배열 사용

1. 거쳐가는 노드 중심 (k)

2. 기존의 x~y로 가는 비용 vs x~k로 가는 비용 + k~y로 가는 비용.

```
// k= 거쳐가는 노드.
for(int k=0 ; k < 노드수 ; k++){
    // i = 출발 노드
    for(int i=0; i < 노드수 ; i++){
        // j = 도착노드.
        for(int j=0; j<노드수; j++){
            if(arr[i][k]+arr[k][j] < arr[i][j]){
                arr[i][j] = arr[i][k] + arr[k][j]
            }
        }
    }
}
```

최단 경로만 따로 저장해둔 P 배열

❑ Floyd's Algorithm Producing the Shortest Paths → 최단거리 경로까지 저장.

```
void floydPath(int n, const number W[ ][ ], number D[ ][ ],
                                             number P[ ][ ])
{     index i,j,k ;

      for (i=1; i <= n ; i++)
        for (j=1; j <= n ; j++)
           P[i][j]=0 ;
      D = W ;
      for (k=1; k<= n ;k++)
        for (i=1; i <= n ; i++)
          for (j=1; j <= n ; j++)
                if (D[i][k]+D[k][j] < D[i][j]) {
                     D[i][j] = D[i][k]+D[k][j] ;   P[i][j] = k ;
                }
}
```

최적화 원칙

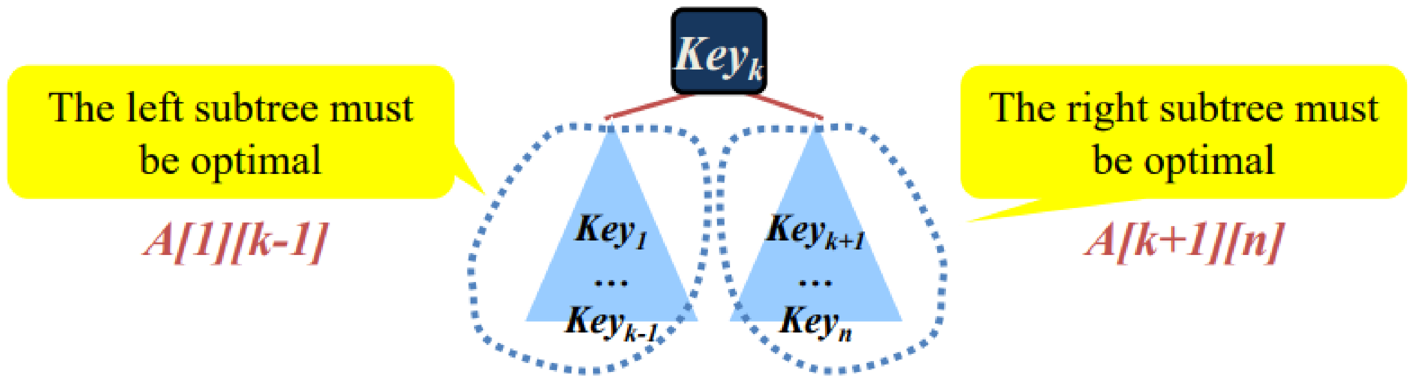
어떤 문제의 입력에 대한 최적 해가 그 입력을 나눈 부분에 대해서도 최적해를 포함하고 있으면 그 문제는 최적의 원칙이 적용된다.

최적 이진 트리 알고리즘

Binary Search Tree Algorithm - 노가다로 다 계산해보기 → 너무 오래걸림! → DP 사용

❑ Dynamic Programming Approach

- Now, consider an optimal binary search tree with key *Key_k* ***at the root*** : 서브트리에 k가 생기면 모든 노드의 탐색 횟수 더함.



$$\sum_{m=1}^{n} P_m C_m = \textbf{A[1][k-1]} + \sum_{m=1}^{k-1} P_m + \boldsymbol{P_k} + \textbf{A[k+1][n]} + \sum_{m=k+1}^{n} P_m$$

$$\textbf{A[1][n]} = \min_{1 \le k \le n}\Big(\textbf{A[1][k-1]} + \sum_{m=1}^{k-1} P_m + \boldsymbol{P_k} + \textbf{A[k+1][n]} + \sum_{m=k+1}^{n} P_m\Big)$$

$$= \min_{1 \le k \le n} (\textbf{A[1][k-1]} + \textbf{A[k+1][n]}) + \sum_{m=1}^{n} P_m$$

Pi = 검색 확률, 검색 빈도

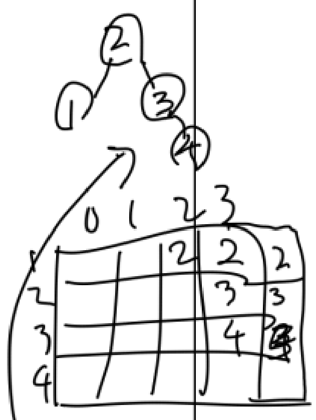
Ci = 검색 수

A[i][i-1] = 0

A[i][i] = Pi, i부터 i까지 노드 중 최적 이진트리는 그냥 그 노드 자체이므로 Pi이다.

아래에서 R에는 i~j까지 노드를 최적값으로 만드는 k를 저장한다.

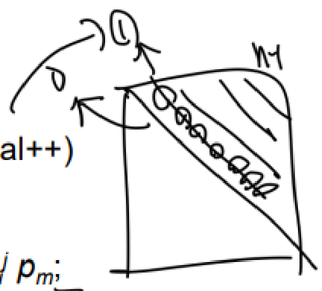
❑ Dynamic Programming Approach




```
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(A[i][k-1]+A[k+1][j])+∑m=i j pm;
                        i≤k≤j
                R[i][j]=a value of k giving the minimum
            }
        minavg = A[1][n] ;
}
```

i≤k≤j 범위 ㄴ 사실상 for문이 하나 더 있는거다.

여기서 최소가 되도록 하는 k를 R[i][j]에 저장한다.



A[3][4] = 3

ㄴ 3~4노드 중 3이 루트일 때 최적값이고 그 값은

Basic Operation:

- the instructions executed for each value of k

Input Size :

- n, number of keys

```
for (diagonal=1;diagonal <= n-1;diagonal++)
  for (i=1; i<= n-diagonal ; i++) {  → (n-digonal)-1 +1
    j = i + diagonal ;   → 이 반복문에서 수행되는 줄이 B.O
    A[i][j]=min(A[i][k-1]+A[k+1][j]) + Σ_{m=i}^{j} p_m;
             i≤k≤j
    R[i][j]=a value of k giving the minimum
  }
```

i ~ j 개씩 루프가 돈다.

Within k-loop: * j= i+diagonal

$$j\text{-}i+1 = (i+\text{diagonal})\text{-}i+1 = (\text{diagonal} + 1) \text{ times}$$

Within i-loop:

$$\underbrace{(n\text{-diagonal})}_{i\text{-loop}}\underbrace{(\text{diagonal} + 1)}_{k\text{-loop}}$$

Within diagonal-loop:

$$\sum_{diagonal=1}^{n-1}(n\text{-diagonal})(\text{diagonal} + 1) = \mathbf{n(n\text{-}1)(n\text{+}4)/6} \in \Theta(n^3)$$

38

1. min(~)의 범위는 i ≤ k ≤ j 이므로 j-1+1 만큼 반복한다. → j = 1+ d 이므로 → **d+1**로 표현
2. for i 는 (n-d) - (1) + 1 이므로 n-d가된다.

시그마 d=1 부터 n-1까지(n-d)(d+1) 을 간략화하면 n(n-1)(n+4)/6 → 세타 n^3

```
node_pointer tree(index i,j) {

    index k ;     node_pointer p ;

    k = R[i][j] ;  — 루트의 키값
    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 ;
    }
}
```

위 그림은 R의 k를 이용해서 최적 이진 트리를 구성하는 슈도코드

TSP(Traveling SalesPerson Problem)

그냥 노가다로 찾으면 (n-1)이 된다.