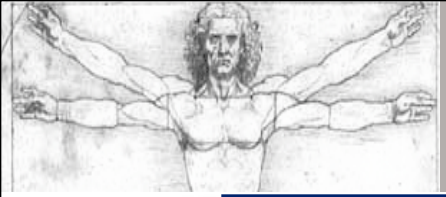
KNU KYUNGPOOK NATIONAL UNIVERSITY

Lecture #15: Dynamic programming

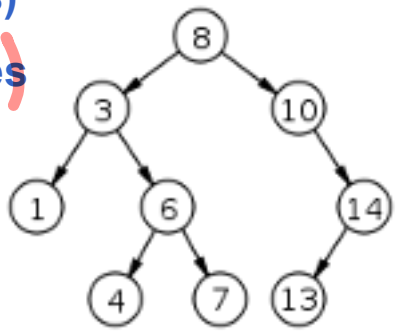
School of Computer Science and Engineering
Kyungpook National University (KNU)

Woo-Jeoung Nam

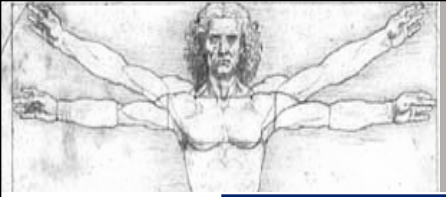


Optimal Binary Search Trees

- **Binary search tree**
  - A rooted binary tree data structure
    - Internal nodes each store a key
      greater than all the keys in the node's left subtree
      and less than those in its right subtree
- **Optimal binary search tree**
  - Provides the smallest possible search time (or expected search time) for a given sequence of accesses (or access probabilities)
  - to minimize the number of nodes visited in all searches

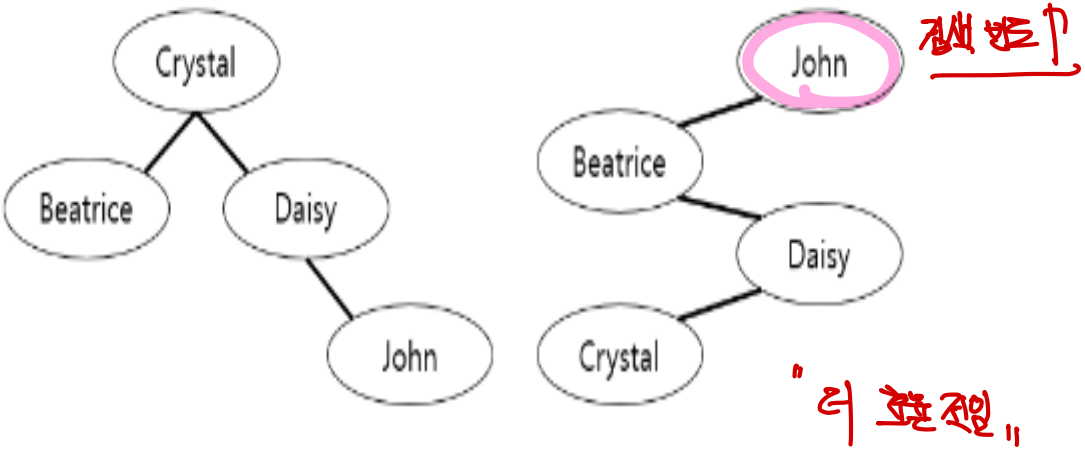


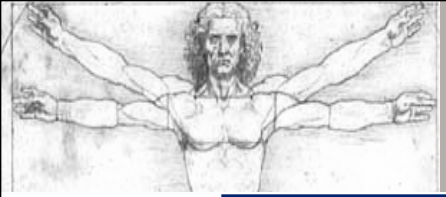
[A binary search tree of size 9 and depth 3, with 8 at the root]



Optimal Binary Search Trees

- **Binary search tree (BST)**
  - 크다, 작다의 기준은 알파벳 ABC순 정렬
  - 둘다 BST
  - 왼쪽의 트리가 오른쪽보다 층의 수가 작으므로 왼쪽이 더 효율적이다

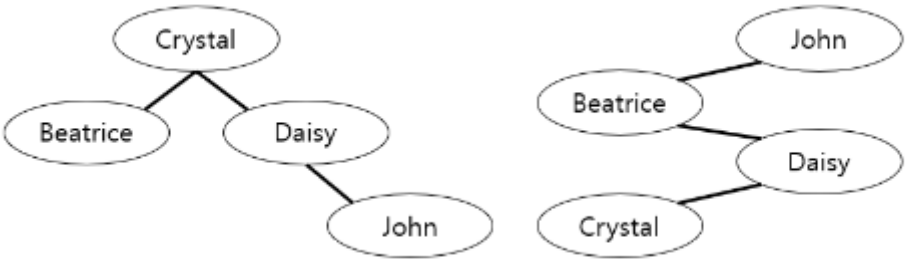


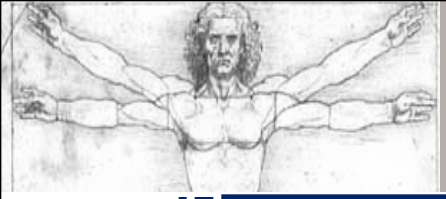


Optimal Binary Search Trees

- **Binary search tree (BST)**
  - ➢ 현실에서는?
    - **Crystal 보다는 John이라는 이름이 훨씬 더 많이 호출된다**
    - 단어별 검색 빈도가 다음과 같다고 가정
      - Crystal = 0.1 / Daisy = 0.2 / Beatrice = 0.3 / John = 0.4
    - BST 평균 검색시간 = 검색빈도 * 필요탐색횟수 = $\sum_{i=1}^{n} c_i p_i$

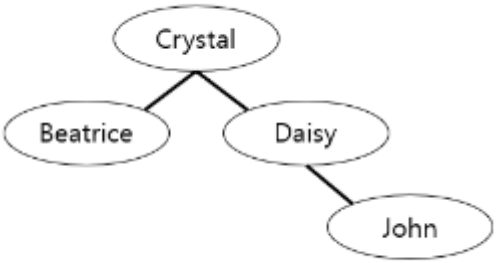
      (p_i: 검색 빈도, c_i: 필요 탐색 횟수)
      - **C: 단어 검색하는데 호출함수 횟수, p : 검색 빈도, n: 단어 수**

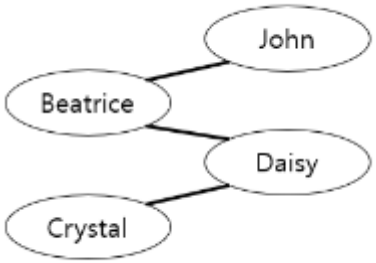


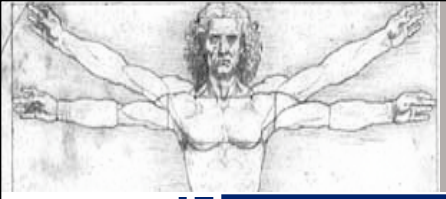


Optimal Binary Search Trees

- **Binary search tree (BST)**
  - ➢ 현실에서는?
    - Crystal = 0.1 / Daisy = 0.2 / Beatrice = 0.3 / John = 0.4
    - **왼쪽: (0.1*1) + (0.2*2) + (0.3*2) + (0.4*3) = 2.3**
    - **오른쪽: (0.1*4) + (0.2*3) + (0.3*2) + (0.4*1) = 2.0**
    - **오른쪽 BST가 효율이 좋다**
    - **가능한 BST 모양들 중에서 평균검색시간이 가장 낮은, 가장 효율적인 최적이진탐색 트리는 뭘까? (optimal binary search trees)**
    - **가능한 경우의 수를 모두 구현 후 비교? -> 너무 비효율적이다**

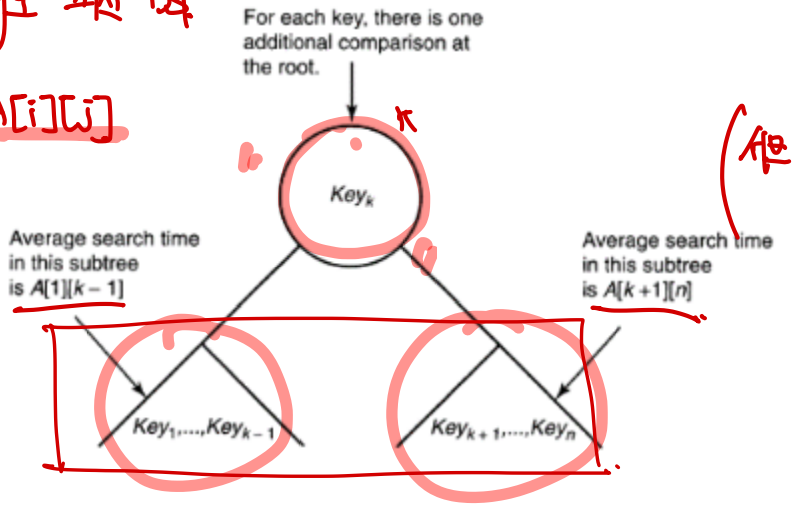


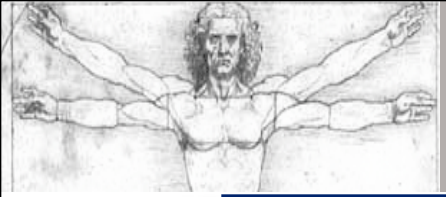




Optimal Binary Search Trees

- BST의 특성상 어떤 BST의 왼쪽, 오른쪽 서브트리가 각각 최적 BST여야 해당 BST도 최적
  - Optimal substructure 성립
  - 최적 이진 검색 트리는 단순히 높이를 균형있게 하는 것이 아니라 탐색시간을 최소로 하여 최적의 탐색시간을 갖도록 하는 것





Optimal Binary Search Trees (cont.)

- N개의 서로다른 키 $K = <k_1, k_2, \ldots, k_n>$, sorted $(k_1 < k_2 < \ldots < k_n)$
  - 각 key에 대해 k_i, 검색확률 p_i that a search is for k_i
  - 검색을 했는데 k에 존재하지 않는 값에 대해 일어날 수 있음
- N+1 "dummy keys(가상 키)" $d_0, d_1, d_2, \ldots, d_n$ representing values not in K
  - Some searches may be for values not in K
  - 가상키 d_i 에 대해 검색확률 q_i that a search is for d_i
- Every search is either successful (finding some key k_i) or unsuccessful (finding some dummy key d_i), and so we have

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$$

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

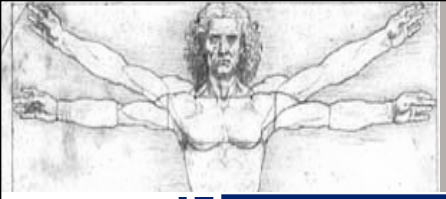
실패 확률
→가상 키

$p_1 + p_2 + p_{k-1}$

$m \neq k$

k_m은 비교하지 않음

A[][k-1]
+A[k+1][n]
$+\sum_{m=1}^{n} p_m$



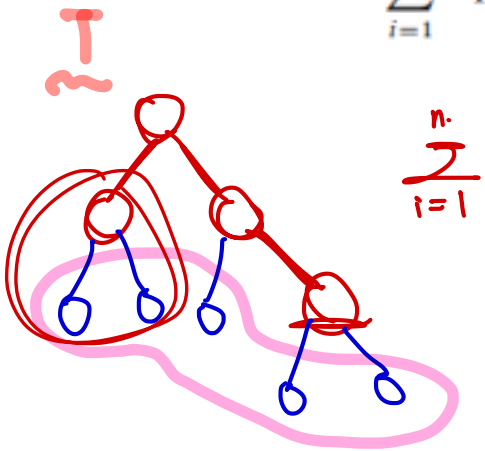
Optimal Binary Search Trees (cont.)

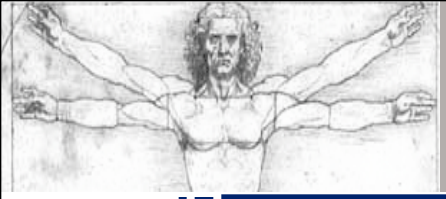
- 각 키와 각 가상 키의 확률이 존재, 이진검색트리 T의 기댓값 결정 가능

- **Expected cost of a search in a given binary search tree *T***
  - 검색 한번에 필요한 실제 비용 = 검사한 노드의 수
    - T를 검색하여 발견한 노드의 깊이 +1 이라고 가정
  - T에서 한번의 검색에 필요한 기대값은

$$\text{E[search cost in } T] = \sum_{i=1}^{n} (\text{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (\text{depth}_T(d_i) + 1) \cdot q_i$$

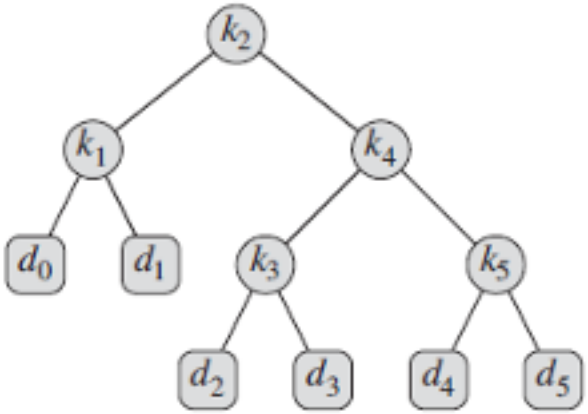
$$= 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \text{depth}_T(d_i) \cdot q_i \,, \qquad (15.11)$$



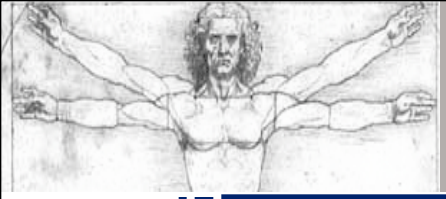


Example

- 다음 트리의 노드별 기댓값은 다음 테이블과 같이 계산 가능



node	depth	probability	contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.05	0.15
d_1	2	0.10	0.30
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
Total			2.80

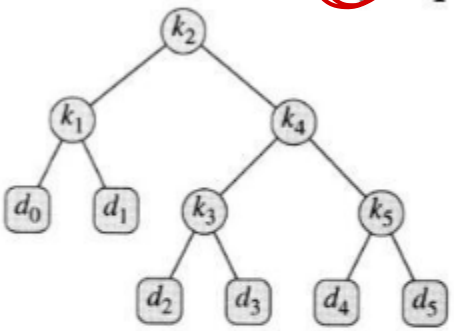


Example

- 다음 트리의 노드별 기댓값은 다음 테이블과 같이 계산 가능
  - ➢ B기 전체 높이가 더 큼에도 최직인 이진트리이다

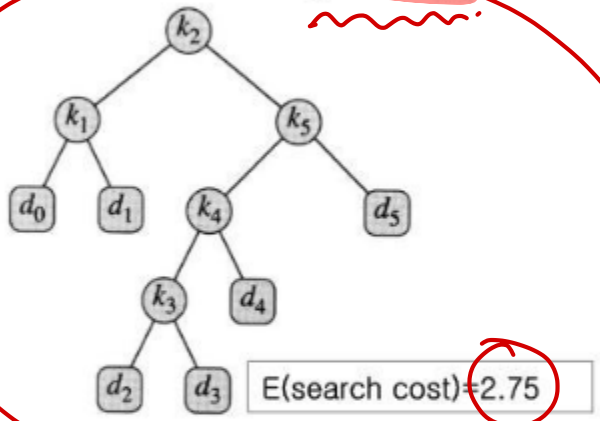
$K = \langle k_1, k_2, \cdots, k_n \rangle$ where $k_1 < k_2 < \cdots < k_n$

d_i represents all values between k_i and k_{i+1}



E(search cost)=2.80

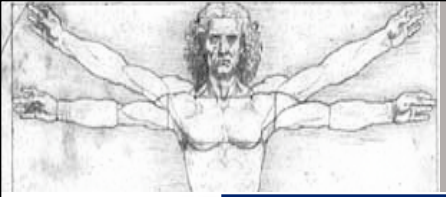
(a)



E(search cost)=2.75

(b)

i	0	1	2	3	4	5	
p_i		0.15	0.10	0.05	0.10	0.20	entry key
q_i	0.05	0.10	0.05	0.05	0.05	0.10	dummy key

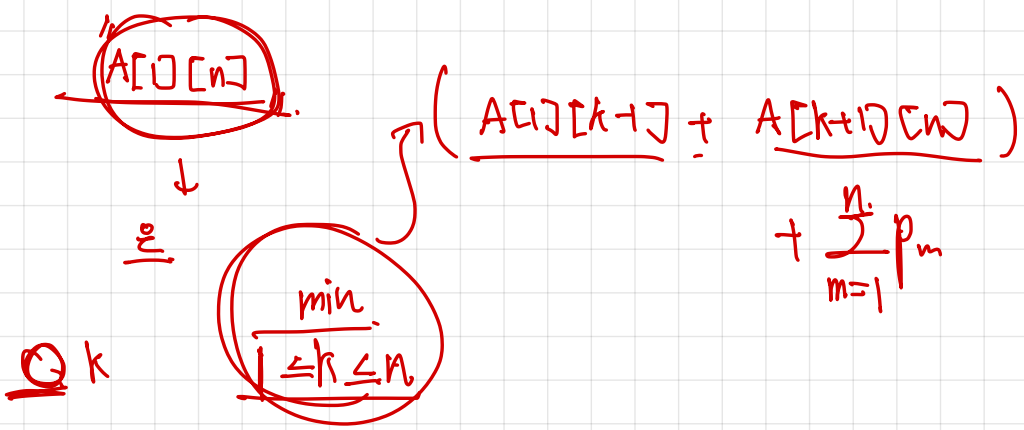


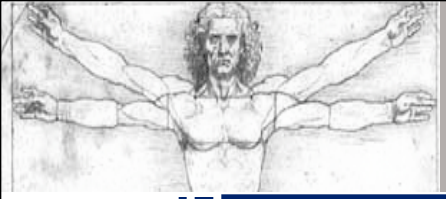
Optimal Substructure

- 먼저 i번째 key부터 j번째 key까지로 최적 BST를 만들었을 때
  - 평균 검색 시간, 즉 최적값을 A[i][j]
- 평균 검색 시간은 루트노드가 k번째 key일 때
  - 왼쪽, 오른쪽 각각 A[1][k-1], A[k+1][n]
- k개의 BST 중 하나는 반드시 최적
  - 서브트리들은 이미 최적이라고 했기 때문에 루트노드를 k번 바꿔보고 그 중 최적의 값을 구하면 그게 전체에 대한 최적의 값

$$A[1][n] = \underset{1 \le k \le n}{\text{minimun}} \left(A[1][k-1] + A[k+1][n]\right) + \sum_{m=1}^{n} p_m$$

왼쪽 서브트리 탐색시간 / 오른쪽 서브트리 탐색시간 / 루트노드 탐색시간


A[0][n]
=
min
1≤k≤n
A[1][k-1] + A[k+1][n]
+ Σ p_m
m=1
n
Q k



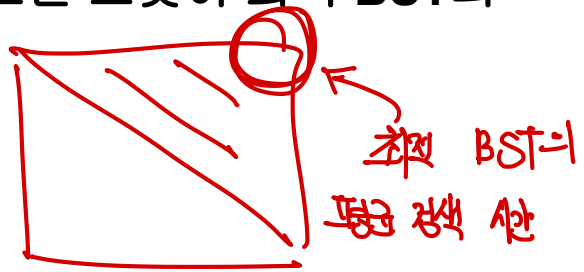
Optimal Substructure

- 1부터 n까지가 아니라 i번째부터 j번째까지의 key로 일반화

$$A[i][j] = \underset{i<=k<=j}{minimum}(A[i][k-1] + A[k+1][j]) + \sum_{m=i}^{j} p_m \quad (i < j)$$

$A[i][j] = \underset{i \le k \le j}{\text{minimum}} \left(A[i][k-1] + A[k+1][j] \right) + \sum_{m=i}^{j} p_m \quad (i<j)$

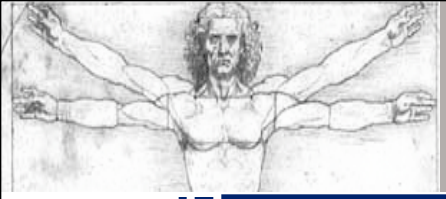
- 2차원 배열 A를 채워나가서, A[1][n] 의 값을 얻으면 그것이 최적 BST의 평균 검색 시간 (최적값)



- 최적 BST의 모양을 알 수가 없다
  - R이라는 2차원 배열을 하나 더 만드는 것
  - R의 각 칸에 들어가는 값은 A 배열에 최적값을 넣는 순간의 k값으로, 이는 즉 i번째 key부터 j번째 key까지를 이용해 BST를 만들 때 루트가 되는 노드의 번호를 의미

i~j 중 k

    - R[1][n] 을 구하면 최적 BST의 루트노드를 구할 수 있다.



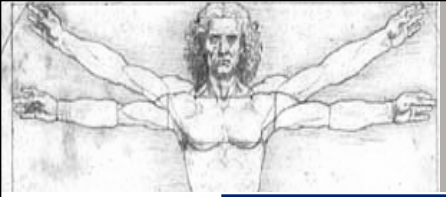
Optimal Substructure – example

- 4개의 key, 검색빈도 p_i
- i번째부터 j번째 key를 이용해서 만든 최적 BST 에 대한 내용을 담고 있으므로, i<=j인 경우만 생각
- 대각선 칸은 0, 점화식을 이용해서 칸을 채운다

Don	Isabelle	Ralph	Wally
Key[1]	Key[2]	Key[3]	Key[4]

$p_1 = \frac{3}{8}$ $p_2 = \frac{3}{8}$ $p_3 = \frac{1}{8}$ $p_4 = \frac{1}{8}$.

$$A[i][j] = \underset{i<=k<=j}{minimum}(A[i][k-1] + A[k+1][j]) + \sum_{m=i}^{j} p_m \quad (i < j)$$



Optimal Substructure – example

- A[1][2]의 경우, k=1, 2 중 작은값 + $\sum_{m=i}^{j} p_m$ (i < j)
- A[1][0] + A[2][2] (k=1인 경우) = 0 + 3/8 + 3/8*2
- A[1][1] + A[3][2] (k=2인 경우) = 3/8 + 0 + 3/8 * 2
  - 둘다 값이 같다 -> 작은 값을 우선으로 R배열에 삽입

Don	Isabelle	Ralph	Wally
Key[1]	Key[2]	Key[3]	Key[4]

$p_1 = \frac{3}{8}$ $p_2 = \frac{3}{8}$ $p_3 = \frac{1}{8}$ $p_4 = \frac{1}{8}$.

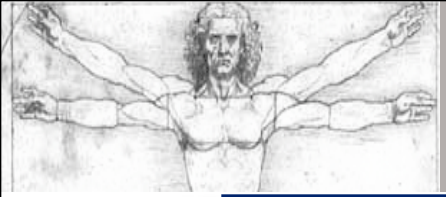
	0	1	2	3	4
1	0	$\frac{3}{8}$	$\frac{9}{8}$	$\frac{11}{8}$	$\frac{7}{4}$
2		0	$\frac{3}{8}$	$\frac{5}{8}$	1
3			0	$\frac{1}{8}$	$\frac{3}{8}$
4				0	$\frac{1}{8}$
5					0

A

	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

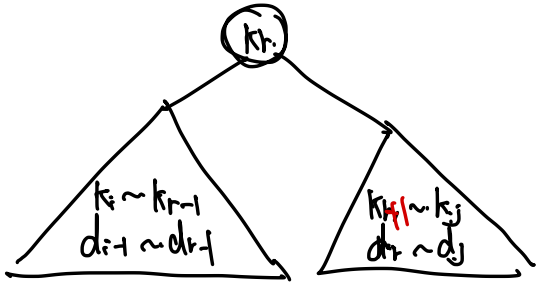
R

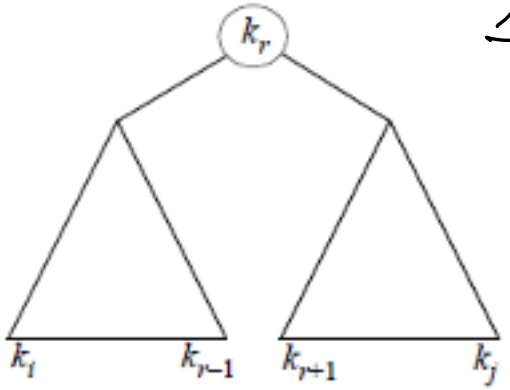
$$A[i][j] = \underset{i<=k<=j}{minimum}(A[i][k-1] + A[k+1][j]) + \sum_{m=i}^{j} p_m \ (i < j)$$

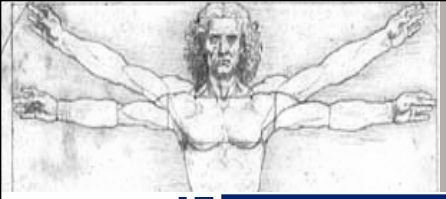


Optimal Substructure (교재)

- **Use optimal substructure to construct an optimal solution to the problem from optimal solutions to subproblems:**
  - Given keys $k_i, \ldots, k_j$ (the problem)
  - One of them, k_r, where $i \le r \le j$, must be the root
  - Left subtree of k_r contains $k_i, \ldots, k_{r-1}$
  - Right subtree of k_r contains $k_{r+1}, \ldots, k_j$

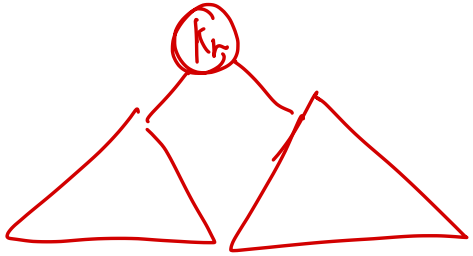


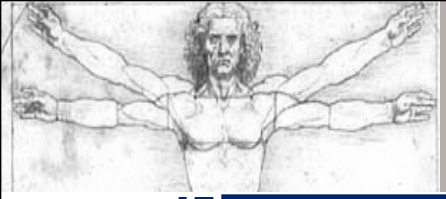




Recursive Solution

- **Subproblem domain:**
  - Find optimal BST for $k_i, \ldots, k_j$, where $i \geq 1, j \leq n, j \geq i - 1$
  - When $j = i - 1$, there are no actual keys; just the dummy key d_{i-1}
- **Define $e[i, j]$ = expected search cost of optimal BST for $k_i, \ldots, k_j$**
  - If $j = i - 1$, then $e[i, j] = q_{i-1}$
  - If $j \geq i$,
    - Select a root k_r for some $i \leq r \leq j$
    - Make an optimal BST with $k_i, \ldots, k_{r-1}$ as the left subtree
    - Make an optimal BST with $k_{r+1}, \ldots, k_j$ as the right subtree





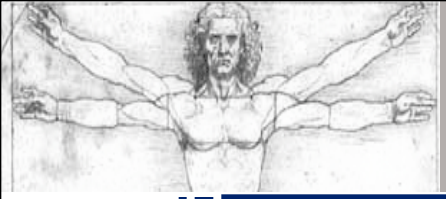
Recursive Solution (cont.)

- When a subtree becomes a subtree of a node:
  - Depth of every node in subtree goes up by 1
  - Expected search cost increases by $w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$
- If k_r is the root of an optimal BST for $k_i, \ldots, k_j$
  - $e[i,j] = p_r + (e[i,r-1] + w(i,r-1)) + (e[r+1,j] + w(r-1,j))$

    But $w(i,j) = w(i,r-1) + p_r + w(r+1,j)$

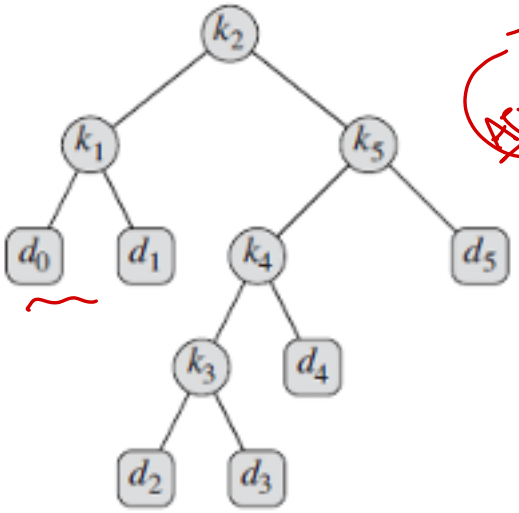
    Therefore, $e[i,j] = e[i,r-1] + e[r+1,j] + w(i,j)$

$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1, \\ \min_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j. \end{cases}$$



Computing an Optimal Solution

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10



e

i \ j	0	1	2	3	4	5
1	0.45	0.90	0.70	0.60	0.90	5
2		0.40	0.25	0.30	0.50	6
3			0.05	0.05	0.05	0.10
4				0.05	0.05	0.10
5					0.10	0.10

w

i \ j	0	1	2	3	4	5
1	0.30	0.45	0.35	0.30	0.50	5
2		0.25	0.15	0.20	0.35	6
3			0.05	0.05	0.05	0.10
4				0.05	0.05	0.10
5					0.10	0.10

$root$

i \ j	0	1	2	3	4	5
1	2	2	2	4	5	5
2		1	2	4	5	5
3			2	4	5	5
4				4	5	5
5					5	5