최적이진검색트리

강지훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터공학과

Static
vs
Dynamic

Static vs. Dynamic

Static Search Structure

- No insertion / No deletion
- Identifiers are known in advance.
  - When each occurrence has *equal* probability:
    - Sort the names and store them sequentially.
      And use binary search.
      $\Rightarrow O(\log n)$ for a search.
  - When each occurrence has *different* probability:
    - Optimal binary search tree

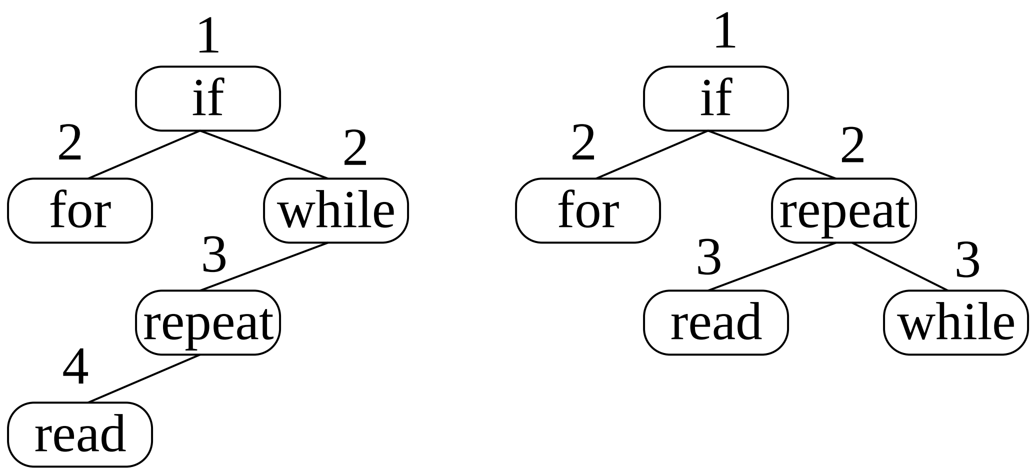
Dynamic Search Structure

- All operations are possible.
  - Hash Table
  - AVL(Height Balanced) Tree

What is Optimal in Binary Search Trees?

What is optimal?

- The search cost must be optimal.
  - # of iterations of the while loop must be minimum.
  - This number corresponds to the level number of a node.



	Number of iterations	
Worst case	4	3
Average case (with equal probability)	(1+2+2+3+4)/5 = 2.4	(1+2+2+3+3)/5 = 2.2 *This tree is better!*

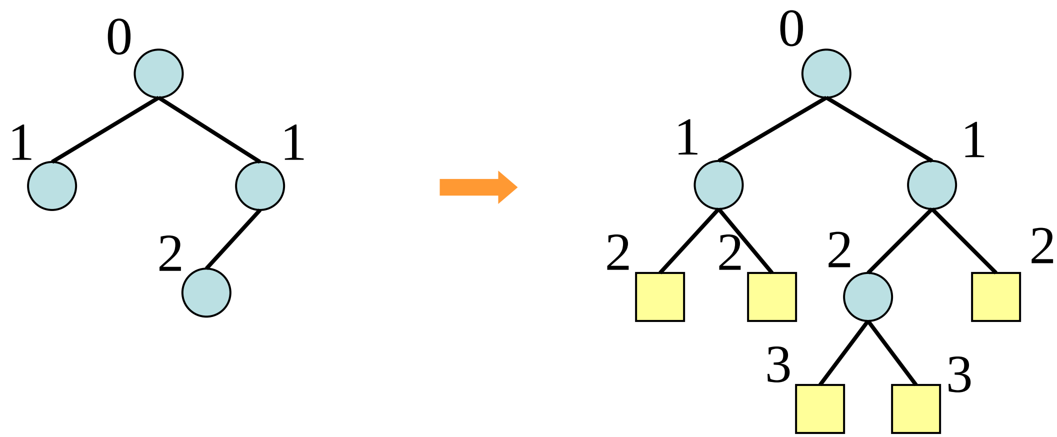
"Failure" of search

- We need to consider the "failure" of search.
  - During searching the binary search tree, we fails the search when we meet any null link.
  - We can extend the binary tree by considering a special node that is attached into each null link field of the original binary tree.
  - These special nodes are called External node / Failure node.
  - The original nodes can be considered as Internal nodes compared to the external nodes.
  - Every binary tree with n nodes has $(n+1)$ null links.

    $\Rightarrow$ $(n+1)$ external nodes.

Extended Binary Trees [2]

- An extended binary tree is a binary tree with external nodes.
  - Example

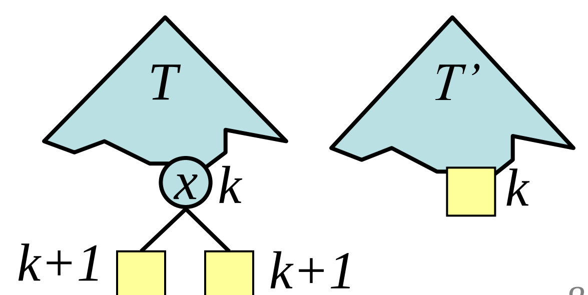


  - External Path Length :
    - $E = 2 + 2 + 3 + 3 + 2 = 12$
  - Internal Path Length
    - $I = 0 + 1 + 1 + 2 = 4$

Relationship between E and I

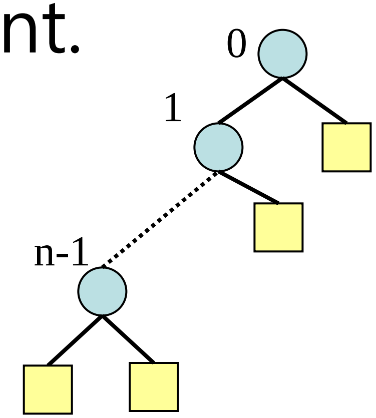
- $E = I + 2n$ (n is the number of internal nodes).
- (Proof) By induction on n.

1. Basis: $n = 0$.
   The binary tree is empty. Then $E = I = n = 0$.
   Therefore, $E = I + 2n$.
2. Hypothesis:
   Assume it is true for all binary trees with less than n nodes.
3. Step :
   Let T be the tree to be considered. Let x be an internal node whose left and right children are both external nodes. Such x must exist, otherwise the tree would be infinite. Let k be the path length of x from the root.
   We can consider the tree T' by replacing x and its external nodes to one external node.
   Then T' has (n-1) nodes. So $E' = I' + 2(n-1)$.
   $E = E' + 2(k+1) - k$.
   $I = I' + k$.
   $\Rightarrow E-I = (E'-I') + 2 = 2(n-1) + 2 = 2n$.
   Therefore, $E = I + 2n$.

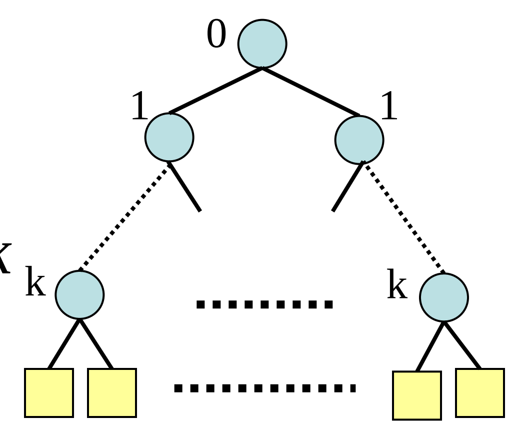


Worst / Best Cases

- From the fact $E = I + 2n$, we can know that E is minimized when I is minimized since the number n of the internal nodes is a constant.
  - Worst Case: I is maximized.
    $\Rightarrow$ Skewed tree.
    - $I = \sum_{i=1}^{n}(i-1) = n(n-1)/2 = O(n^2)$



  - Best Case: I is minimized.
    $\Rightarrow$ Complete binary tree.
    - $k = \lfloor \log_2 n \rfloor$
      $I = 0+1+1+2+2+2+2+\ldots+k+\ldots+k$
      $= \sum_{i=1}^{n} \log_2 i = O(n \log_2 n)$



Optimal Binary Search Trees

Optimal Binary Search Trees [1]

- Given a set of identifiers $\{a_1, a_2, \ldots, a_n\}$ with $\{a_1 < a_2 < \ldots < a_n\}$, we want to find an optimal binary search tree. Here, we assume that the following probabilities are known in advance:
  - p_i = the probability of searching for a_i $(1 \le i \le n)$,
  - q_i = the probability of searching for identifiers between a_i and a_{i+1} $(0 \le i \le n)$, (i.e., the probability of unsuccessful search for identifiers between a_i and a_{i+1}).
    - More formally, let
      $E_0 = \{x \mid x < a_1\}$
      $E_i = \{x \mid a_i < x < a_{i+1}\}, i = 1, 2, \ldots, n\text{-}1$
      $E_n = \{x \mid x > a_n\}$
      Then, q_i = probability(E_i)

Optimal Binary Search Trees [2]

Total Cost of Binary Search Trees

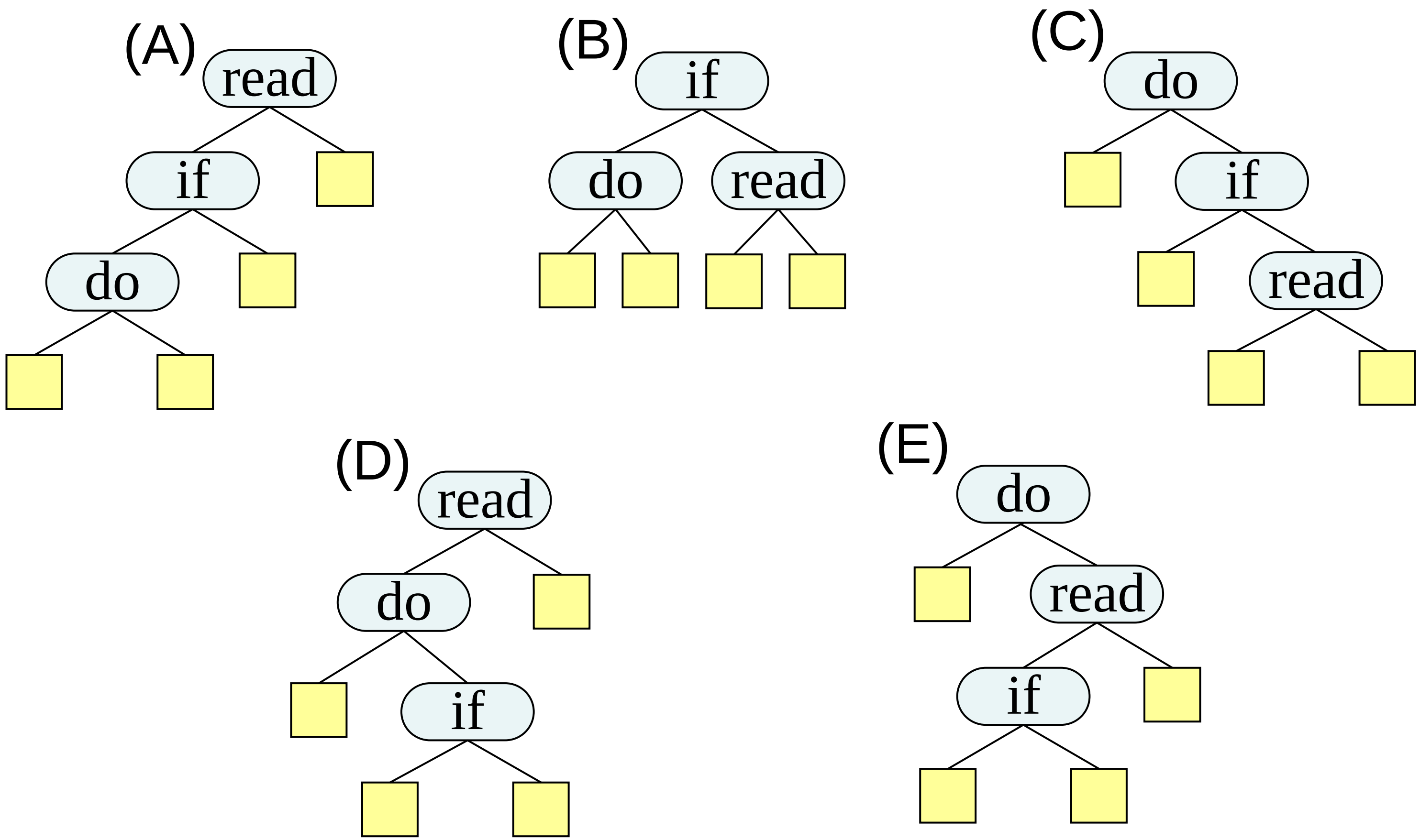
- a_i : internal node with p_i, $1 \leq i \leq n$.
- E_i: external node with q_i, $0 \leq i \leq n$.
- If the level of a_i is $l(a_i)$ in a BST,
  then $p_i \cdot l(a_i)$ is the cost of searching for a_i over all search.
- If the level of E_i is $l(E_i)$ in a BST,
  then, $q_i \cdot (l(E_i)-1)$ is the cost of searching the identifiers in E_i.
- Total cost = $\sum p_i \cdot l(a_i) + \sum q_i \cdot (l(E_i) - 1)$

How can we get OBST?

- An OBST is the one which minimizes the "total cost" among all kinds of the binary search trees.
  - There are $\frac{1}{n+1}\binom{2n}{n}$ kinds of binary trees with n nodes.
  - One possible method:
    - Generate all BSTs and calculate their costs.
    - And select one with minimum cost as an OBST.

Example:

- (a_1, a_2, a_3) = (‘do’, ‘if’, ‘read’)
  - There are 5 possible BSTs.



- Let $p_i = q_i = 1/7$
  - cost(A) = $(1/7 * 3 + 1/7 * 2 + 1/7 * 1)$
    $+ (1/7 * 3 + 1/7 * 3 + 1/7 * 2 + 1/7 * 1)$
    $= 15/7$
  - cost(B) = 13/7 ← Optimal
  - cost(C) = 15/7, cost(D) = 15/7, cost(E) = 15/7

- Let $p_1 = 0.5$, $p_2 = 0.1$, $p_3 = 0.05$,
  $q_0 = 0.15$, $q_1 = 0.1$, $q_2 = 0.05$, $q_3 = 0.05$.
  - cost(A) = $(0.5 \cdot 3 + 0.1 \cdot 2 + 0.05 \cdot 1)$
    $+ (0.15 \cdot 3 + 0.1 \cdot 3 + 0.05 \cdot 2 + 0.05 \cdot 1)$
    $= 2.65$
  - cost(B) = 1.9
  - cost(C)= 1.5 ← Optimal
  - cost(D)= 2.05
  - cost(E)= 1.6

What's the problem?

- $\frac{1}{n+1}\binom{2n}{n}$ kinds of binary trees with n nodes.
- But, $$\frac{1}{n+1}\binom{2n}{n} = O(4^n / n^{1.5})$$
- So, this method is very significant.

Very Slow !!

Efficient Algorithm

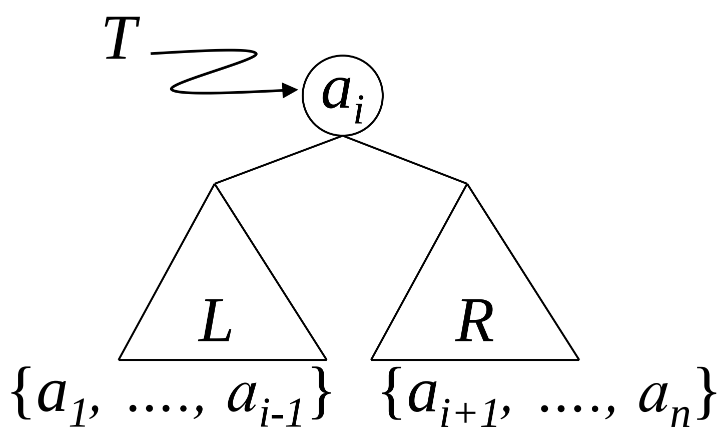
- We can find a fairly efficient algorithm by making some observations regarding the properties of OBST's

- Key Observation:
  - So many same calculations

- Dynamic Programming Approach
  - ➡ 동일한 계산이 반복되는 것을 막자.
  - 한번 계산된 것은 기억
  - 동일한 계산이 필요하면, 기억된 것을 재활용

Idea for finding OBSTs

- The OBST's root is one of $\{a_1, ..., a_n\}$. Let a_i be the root of the OBST T.



- Since T is OBST, both L and R must be OBST.
- Similar concept can also be applied to L and R.
- $\text{cost}(T) = \min_{0<i\leq n}\{\text{cost}(L)+\text{cost}(R)+weight(L)+weight(R)+p_i\}$,

  where $weight(L) = q_0 + (p_1 + q_1) + ... + (p_{i-1} + q_{i-1})$, and
  $weight(R) = q_i + (p_{i+1} + q_{i+1}) + ... + (p_n + q_n)$.

Notations

- T_{ij} : an OBST for $a_{i+1}, \ldots, a_j$, $i < j$
  - $T_{ii} = 0$ for $0 \le i \le n$
  - T_{ij} is undefined if $(i > j)$

- $w_{ij} = q_i + \sum_{k=i+1}^{j} (p_k + q_k)$
  - Clearly, $w_{ii} = q_i$

- c_{ij} = the cost of T_{ij} $(c_{ii} = 0)$
- r_{ij} = the root of T_{ij} $(r_{ii} = 0)$

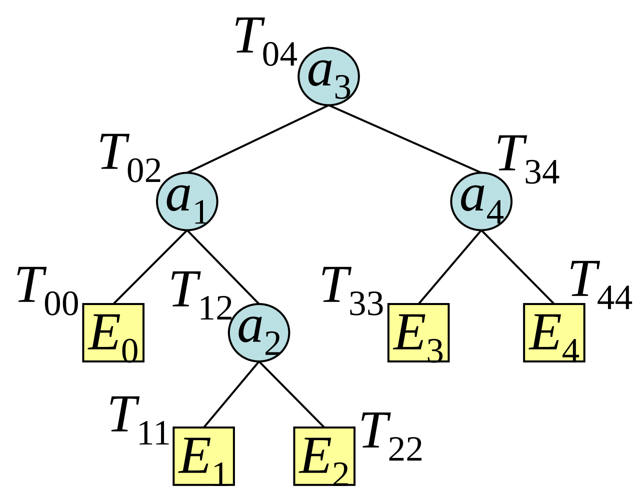
OBST $T_{0,n}$

- OBST can be represented as $T_{0,n}$, with $c_{0,n}$, $w_{0,n}$, and $r_{0,n}$.
  - $c_{0,n} = \min_{0<i\leq n}\{c_{0,i\text{-}1} + c_{i,n} + w_{0,i\text{-}1} + w_{i,n} + p_i\}$
  - In general,
    - $c_{ij} = \min_{i<l\leq j}\{c_{i,l\text{-}1} + c_{l,j} + w_{i,l\text{-}1} + w_{l,j} + p_l\}$
      $= \min_{i<l\leq j}\{c_{i,l\text{-}1} + c_{l,j}\} + w_{i,j}$
    - $r_{i,j} = k$ if $c_{i,j}$ is minimum when $l = k$

Example:

- p_1=5, p_2=2, p_3=7, p_4=6, q_0=9, q_1=3, q_2=4, q_3=9, q_4=1.

	0	1	2	3	4
0	$w_{00} = 9\ (=q_0)$ $c_{00} = 0$ $r_{00} = 0$	$w_{11} = 3\ (=q_1)$ $c_{11} = 0$ $r_{11} = 0$	$w_{22} = 4\ (=q_2)$ $c_{22} = 0$ $r_{22} = 0$	$w_{33}=9(=q_3)$ $c_{33}=0$ $r_{33}=0$	$w_{44}=1(=q_4)$ $c_{44}=0$ $r_{44}=0$
1	$w_{01} = 17$ $c_{01} = 17$ $r_{01} = 1$	$w_{12} = 9$ $c_{12} = 9$ $r_{12} = 2$	$w_{23} = 20$ $c_{23} = 20$ $r_{23} = 3$	$w_{34} = 16$ $c_{34} = 16$ $r_{34} = 4$	
2	$w_{02} = 23$ $c_{02} = \min\{c_{00}+c_{12} = 9,$ $c_{01}+c_{22}=17\}+ w_{02} = 32$ $r_{02} = 1$	$w_{13} = 25$ $c_{13} = \min\{c_{11}+c_{23}=20,$ $c_{12}+c_{33}=9\}+ w_{13}=34$ $r_{13} = 3$	$w_{24} = 27$ $c_{24} = \min\{c_{22} + c_{34} =16,$ $c_{23}+c_{44}=20\}+ w_{24}= 43$ $r_{24} = 3$		
3	$w_{03} = w_{02}+p_{3}+ q_3 =39$ $c_{03} = \min\{c_{00}+c_{13}=34,$ $c_{01}+c_{23}=37,$ $c_{02}+c_{33}=32\}+ w_{03} =71$ $r_{03} = 3$	$w_{14} = 32$ $c_{14} = \min\{c_{11}+c_{24}=43,$ $c_{12}+c_{34}=25,$ $c_{13}+c_{44}=34\}+w_{14}=57$ $r_{14} = 3$			
4	$w_{04} = 46$ $c_{04} = \min\{c_{00}+c_{14}=57,$ $c_{01}+c_{24}=60,$ $c_{02}+c_{34}=48,$ $c_{03}+c_{44}=71\}+ w_{04}=94$ $r_{04} = 3$				



Analysis

- Compute c_{ij} for $(j-i)=1,2,\cdots,n.$

  When $(j\text{-}i)=m$, there are $(n\text{-}m+1)$ c_{ij}'s computed.

  For each c_{ij}, we must find the minimum from m values.

  Total computation is $\sum_{m=1}^{n}(n-m+1)m=\mathrm{O}(n^3)$

- Better Method (by D.E. Knuth)

  Original one: $c_{ij}=\min_{i<l\le j}\{c_{i,l-1}+c_{l,j}\}+w_{ij}$

  Better one: $c_{ij}=\min_{r_{i,j-1}\le l\le r_{i+1,j}}\{c_{i,l-1}+c_{l,j}\}+w_{ij}$

  The total computing time is:

$$\sum_{m=1}^{n}\sum_{m\le j\le n,i=j-m}(r_{i+1,j}-r_{i,j-1}+1)$$

$$=\sum_{m=1}^{n}[(r_{n-m+1,n}-r_{0,m-1})+n-m+1]$$

$$\le\sum_{m=1}^{n}(n-1+n-m+1)\ \left(\because 1\le r_{i,j}\le n \text{ for any } i \text{ and } j.\right)$$

$$=\sum_{m=1}^{n}(2n-m)=\mathrm{O}(n^2)$$

Fibonacci Number
by
Dynamic Programming Approach

Recursive Algorithm

- public int fibo (int n)

```
{
    if (n == 0)
        return 0 ;
    else if (n == 1)
        return 1 ;
    else
        return fibo(n-1)+fibo(n-2) ;
}
```

- Time Complexity: Exponential !!!

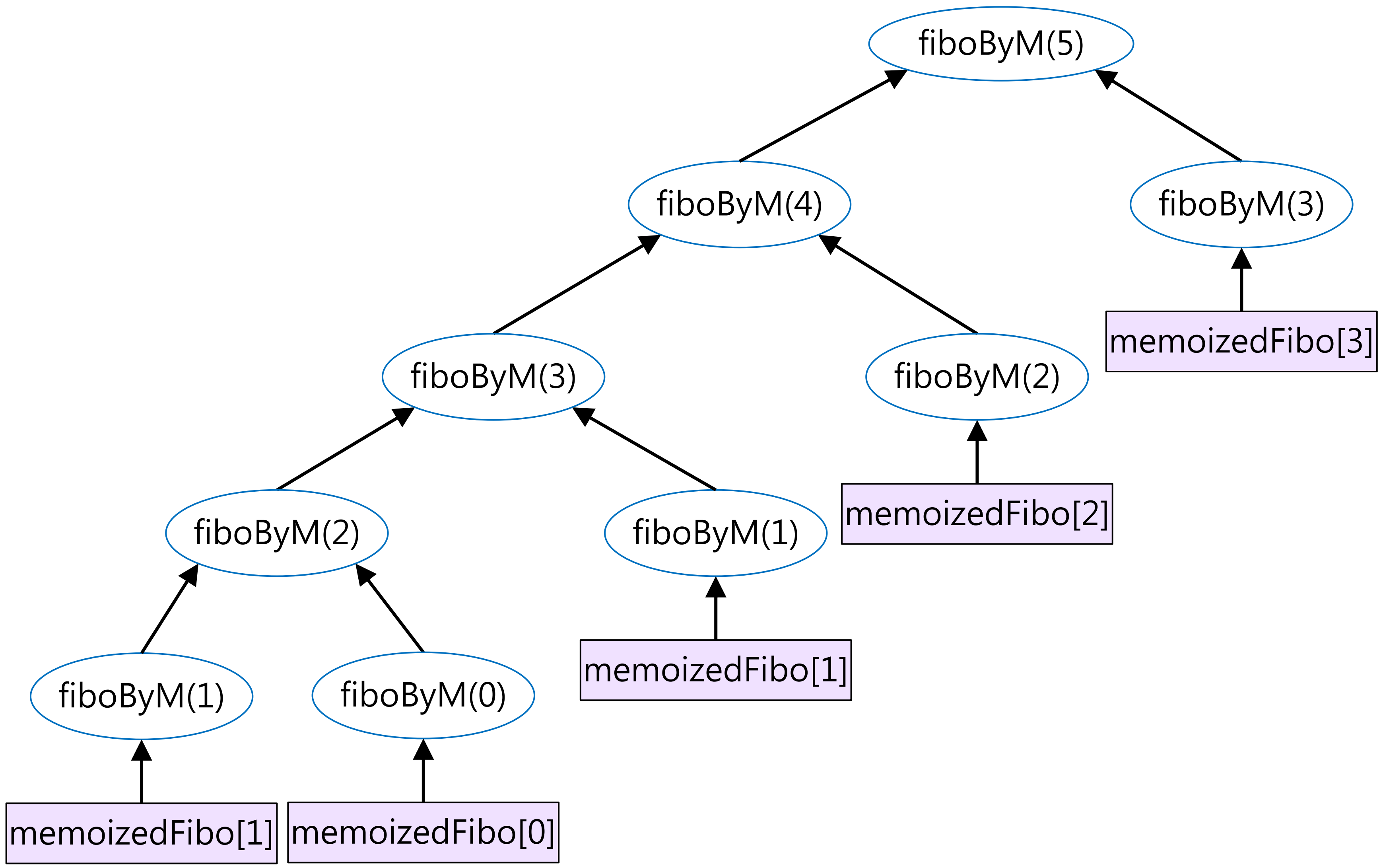
Revised Recursive Algorithm

```
private static final int MaxInputNumber = 1000;
private int[] memoizedFibo = new int[MaxInputNumber] ;

public int  fibo (int  n) {
    for (int i = 0 ; i < n ; i++)
        memoizedFibo[i] = -1 ; // 아직 계산되지 않은 상태의 값
    memoizedFibo[0] = 0 ; // F0
    memoizedFibo[1] = 1 ; // F1
    return  fiboByMemoized(n) ;
}

private int  fiboByMemoized (int  n) {
    if (memoizedFibo[n] < 0) {
        memoizedFibo[n]  = fiboByMemoized(n-1)+ fiboByMemoized(n-2)  ;
    }
    return  memoizedFibo[n]  ;
}
```

- Time Complexity: O(n)


fiboByM(5)
fiboByM(4)
fiboByM(3)
fiboByM(3)
fiboByM(2)
memoizedFibo[3]
fiboByM(2)
fiboByM(1)
memoizedFibo[2]
fiboByM(1)
fiboByM(0)
memoizedFibo[1]
memoizedFibo[1]
memoizedFibo[0]

End
of
Optimal Binary Search Trees