- Data Encoding
  - Fixed-Length Binary Code

    a: 00 b: 01 c: 11. fixed code

    ababcbbbc → 000100011101010111

  - Variable-Length Binary Code

    a: 10 b: 0 c: 11 variable length code

    ababcbbbc → 1001001100011

  - Data Compression
    - Problem of finding an efficient method for encoding a data file.
  - Optimal Binary Code Problem
    - Problem of finding a binary character code for the characters in the given file such that the file is represented by a least number of bits.

◆ Prefix Code

- ✦ A code that no codeword for one character constitutes the beginning of the codeword for another character.

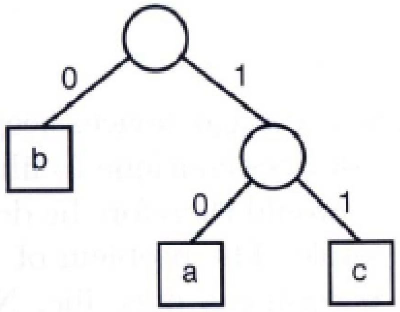
a

b

≠

- ✦ Any prefix code can be represented by a binary tree.

a : 10, b : 0, c : 11 ⟷



text file ⇒ 각 char 의 frequency 계산

encoded string

0 1 1 0 1 1 0 1 0 ---

b c b c b a ---

⇓ decoding (decompression)

- ✦ If prefix code, the parsing (decoding) is easy.

measure?

encoding efficiency.

- ✦ The number of bits to encode a file given the binary tree T corresponding to some code.

$$bits(T) = \sum_{i=1}^{n} frequency(v_i)depth(v_i),$$

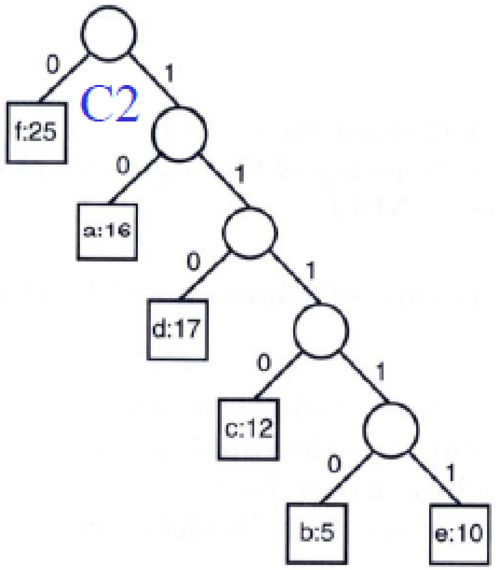
where $\{v_1, v_2, \ldots v_n\}$ is the set of characters in the file, $frequency(v_i)$ is the number of times v_i occurs in the file, and $depth(v_i)$ is the depth of v_i in T.

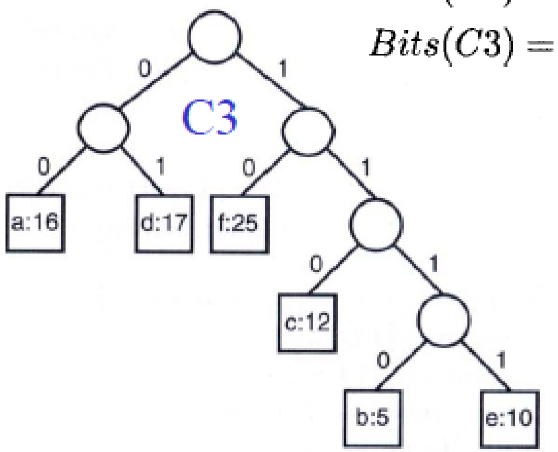
Huffman Tree

- A prefix code tree T such that bits(T) is minimum.
- Example

Character	Frequency	C1(Fixed-Length)	C2	C3 (Huffman)
a	16	000	10	00
b	5	001	11110	1110
c	12	010	1110	110
d	17	011	110	01
e	10	100	11111	1111
f	25	101	0	10
		255 bits	231 bits	212 bits

$$Bits(C1) = 16(3) + 5(3) + 12(3) + 17(3) + 10(3) + 25(3) = 255$$
$$Bits(C2) = 16(2) + 5(5) + 12(4) + 17(3) + 10(5) + 25(1) = 231$$
$$Bits(C3) = 16(2) + 5(4) + 12(3) + 17(2) + 10(4) + 25(2) = 212$$





→ Huffman tree

◆ Huffman's Algorithm

- ✦ The algorithm actually construct the Huffman tree.
- ✦ Nodetype

```
struct nodetype{
   char symbol;          // The value of a character.
   int frequency;        // The number of times the character
   nodetype* left;       // is in the file.
   nodetype* right;
};
```

초기 heap → 각 char 별 빈도.

min heap에

빈도가 작은 2개를 delete

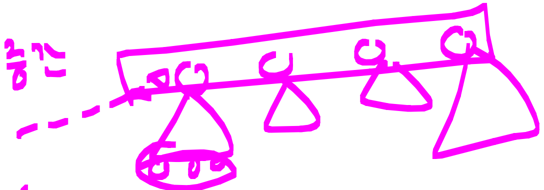
merge

heap에 insert

until a single element in the heap.

- ✦ First construct a priority tree (a heap) PQ
  - ◆ Initially n nodes, where n is the number of distinct characters.
  - ◆ The lower frequency node has higher priority.
  - ◆ Initial node setting

Alg 중간



자손들에 해당하는 char들

char

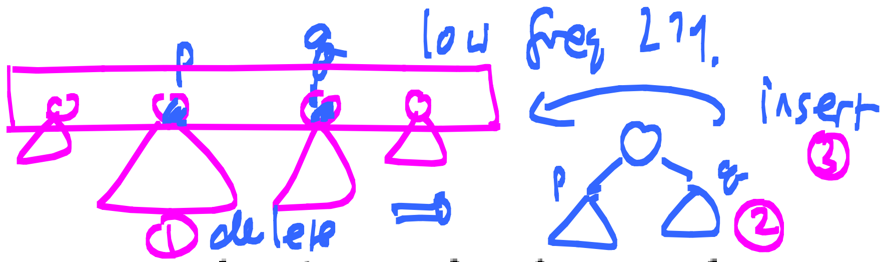
$p\text{->}\ symbol$ = a distinct character in the file;

$p\text{->}\ frequency$ = the frequency of that character in the file;

빈도수 합..

$p\text{->}\ left = p\text{->}\ right$ = NULL;

✦ The Algorithm




```
      for (i=1; i <= n-1; i++) {
O(logn)   remove(PQ, p);          // There is no solution check; rather,
          remove(PQ, q);          // solution is obtained when i = n - 1.
          r = new  nodetype;      // There is no feasibility check.
O(1)      r->left = p;
          r->right = q;
          r->frequency = p->frequency + q->frequency;
O(logn)   insert(PQ, r);
      }
n iterations
      remove(PQ, r);
      return r;
```

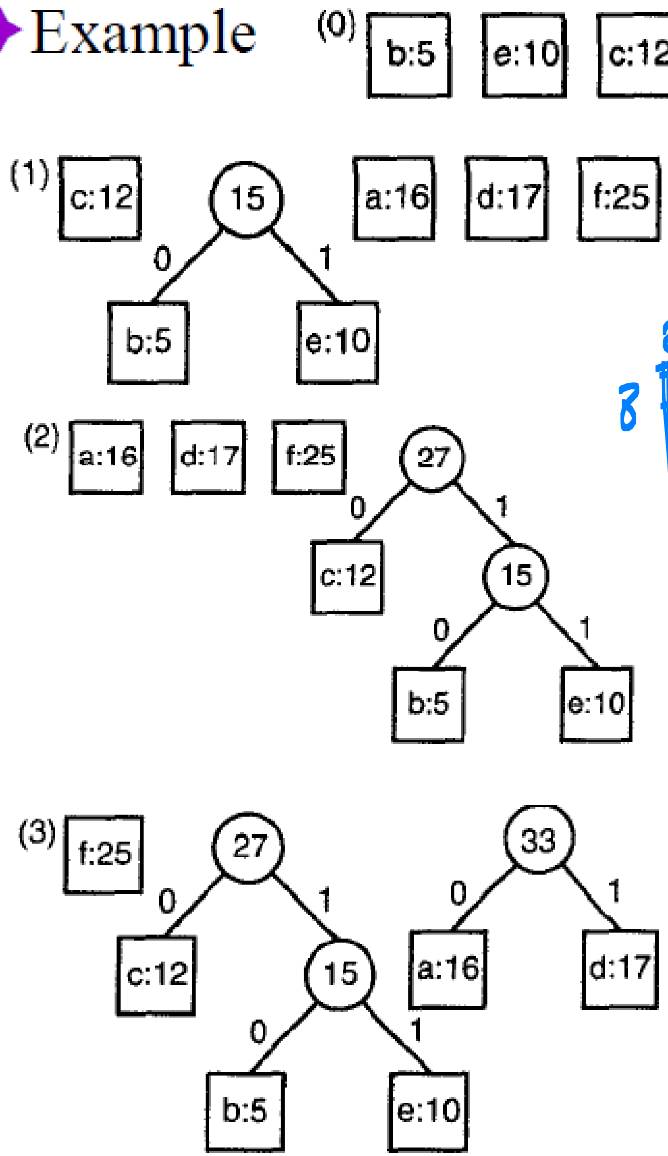
Total O(n*log*n) time

✦ Optimality Proof

▲ Lemma 4.4 The binary tree corresponding to an optimal binary prefix code is full. That is, every nonleaf has two children.

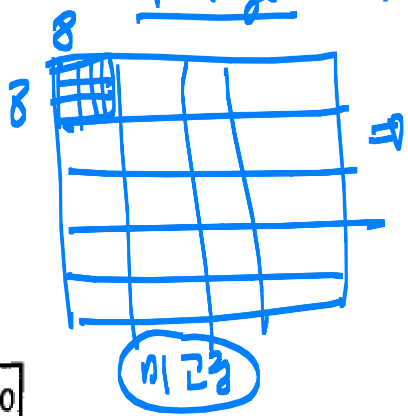
► Theorem 4.5 Huffman's algorithm produces an optimal binary code.

✦Example

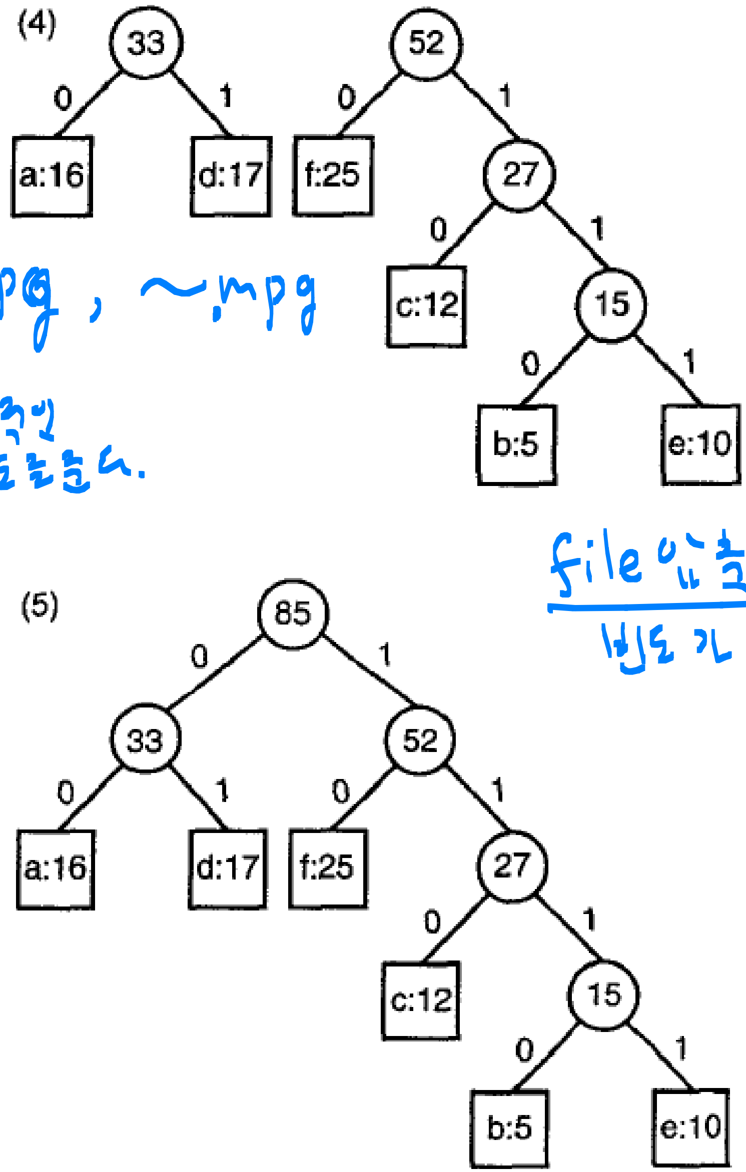


image

~.jpg , ~.mpg



⇒ 통계적인
빈도를 준다.



file 압축

빈도가 고정 :: 효적 된
세상.

table

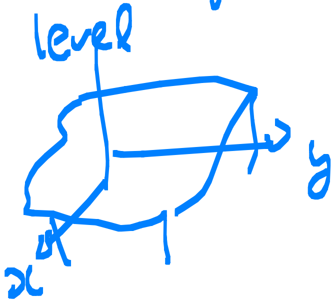
a 00

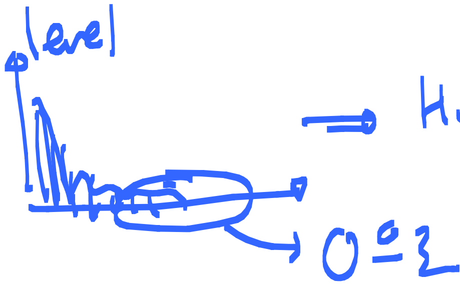
b 1110

⋮

table lookup

Digital Cosine Transform(DCT)

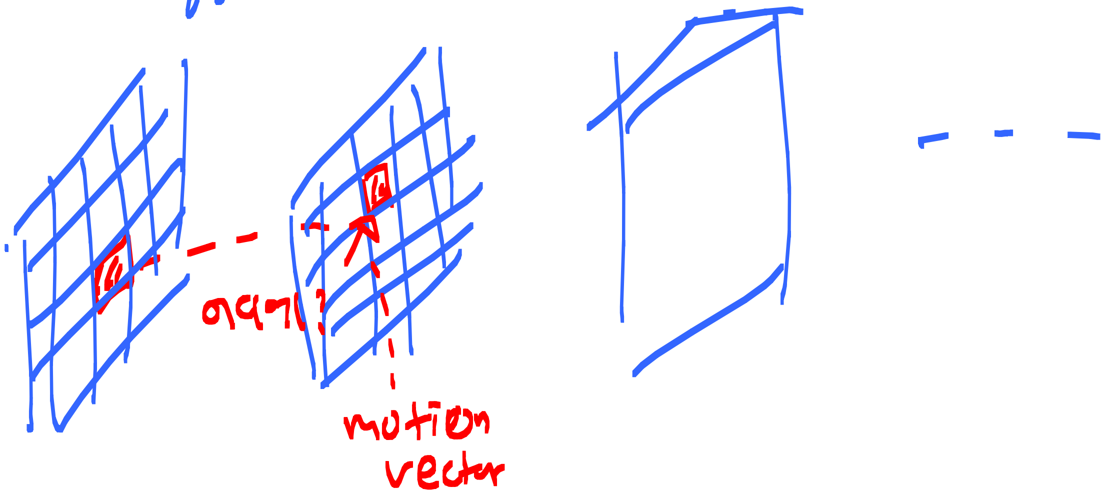




→ Huffman code

Moving Image

신호와 시스템



Dijkstra Alg : Fibonacci Heap ⇒ complexity를 더 줄일 수 있다.

complexity는 amortization 사용.

◆ Amortization

- An analysis method to get more accurate time complexity.
- Example
  - Suppose that insert and deletion operations are done as follows in some data structure:

	I1	I2	D1	I3	I4	I5	I6	D2	I7	
time spent (# of operations) →	1	1	8	1	1	1	1	10	1	= 25

가상적으로

  - The operations for deletion may be very large and/or irregular which prevents from estimating the accurate execution time.
  - We may be able to distribute some of operations for deletion to the operations for insertion.

I1	I2	D1	I3	I4	I5	I6	D2	I7	
2	2	6	2	2	2	2	6	1	= 25

  - Then, we may say that the number of operations is 2i+6d by amortization if the above distribution is possible.

- Aggregate Analysis
  - Calculate the total operations T(n) to perform all the n tasks.
  - The amortized cost per task = T(n)/n.
  - Example : Stack Operations

```
push(S, x);      // add x to the stack S
pop(S);          // remove an element from S
multipop(S, k); // remove k element from S
```

- Time for `push()`, `pop()` : O(1)

pu pu pu ---- m pu -- pu --

O(n)

- Time for `multipop()` : O(min(S, k))
- Assume that n operations of push, pop and multipop are done for an initially empty stack S.
  - In the worst case, the time for multipop() is O(n).
  - Therefore, we may say that the time for each operation is O(n), which implies that the total time is $O(n^2)$ (too much!)

- Suppose that n times of push, pop and multipop operations are executed.
  - Since an element pushed into S is poped exactly once, the number pop operations are the same as that of the push operations (this is true even if multipop operations exist).
  - Therefore, the total time for these operations are O(n)
  - The amortized cost for each operation = O(n)/n = O(1).
- Other Methods for Amortized Complexity
  - Accounting method
  - Potential method

Backtracking

brute force

모든 feasible solution을 본다.

↳ best를 선택.

필요 없으면 안 본다.

search 방법 2가지

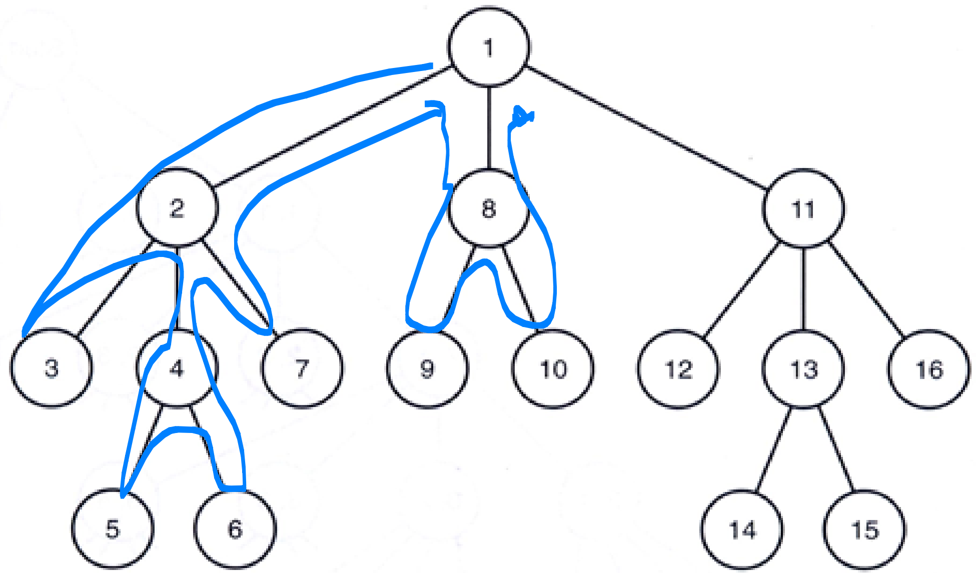
① DFS style — backtracking

② BFS style — branch & bound

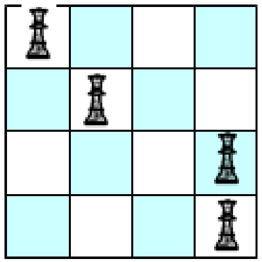
- Depth First Tree Searching
  - The same as the preorder tree traversal.
  - The Procedure

```
void depth_first_tree_search (node v){
   node u;
   visit v;
   for (each child u of v)
       depth_first_tree_search(u);
}
```

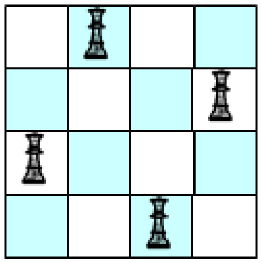
  - An Example



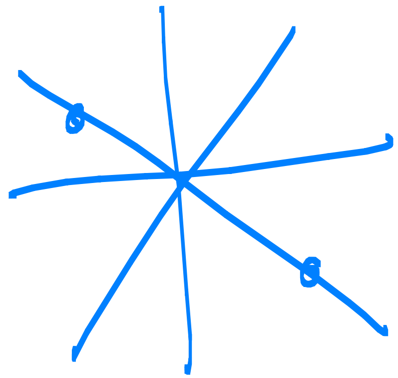
- n-Queens Problem
  - We have n × n chess board and n queens.
  - The goal is to position n queens on the chessboard so that no two queens are in the same row, column or diagonal.
  - Example:



not a solution



a solution

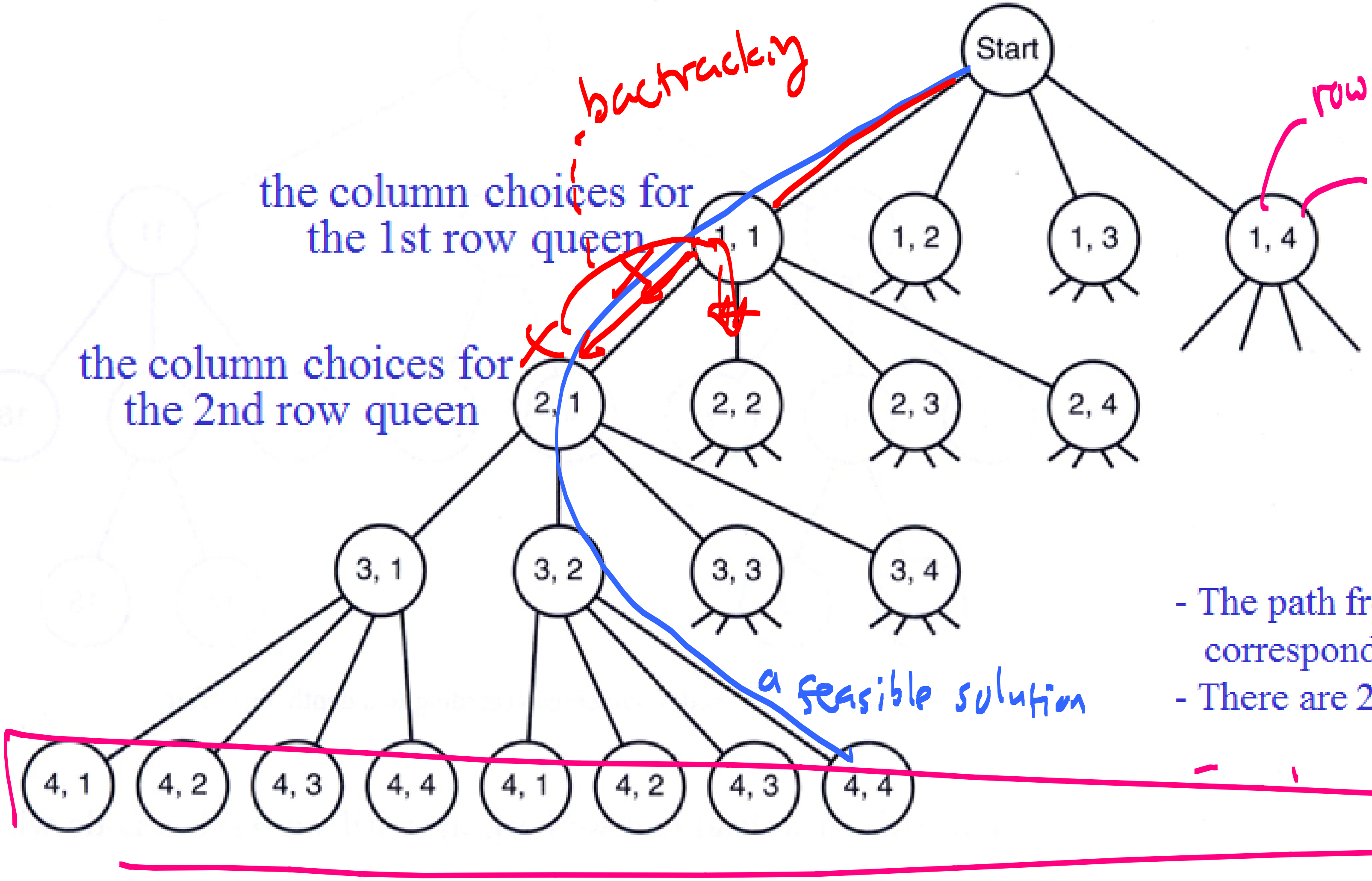


  - Observation
    - No two queens can be in the same row.
    - Assign each queen a different row and check which column combinations yields solutions.
    - We have 4×4×4×4 = 256 candidate solutions for n=4.

row 마다 1개의 queen을 놓는 경우 n-proble의 feasible sol의 개수 = n^n

- The State Space Tree (가상적인 tree)
  - A tree representing all possible candidate solutions.
  - Example: The 4-queens problem

solution space는 엄기하는
모두 search? No.
필요한 부분만 search



- The path from the root to a leaf node corresponds to a candidate solution.
- There are 256 such path.

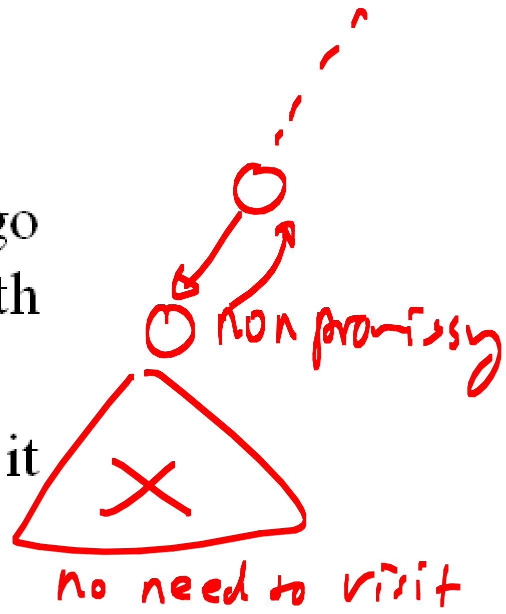
n^n
256개

◆ Backtracking

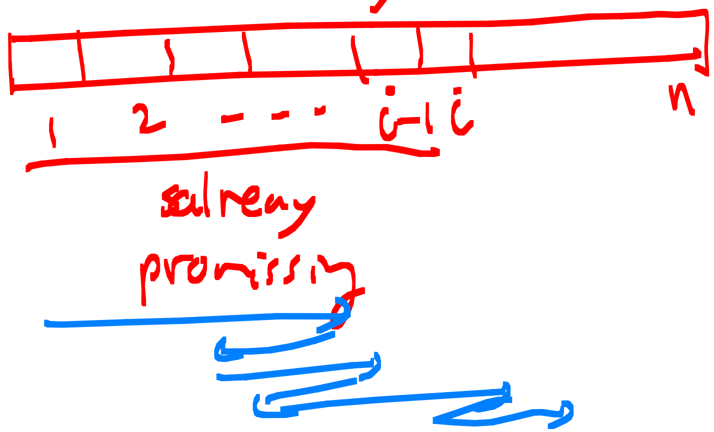
- A procedure to search the state space tree.
- If a node can not leads to a solution (nonpromising), go back ('backtrack') to the node's parent and proceed with the search on the next child. This is called pruning.
- If a node has a possibility to lead a solution, we call it promising.
- A generic procedure :

```
void checknode (node v){
  node u;
  if (promising(v))
     if (there is a solution at v)
        write the solution;
     else
        for (each child u of v)
           checknode(u);
}
```

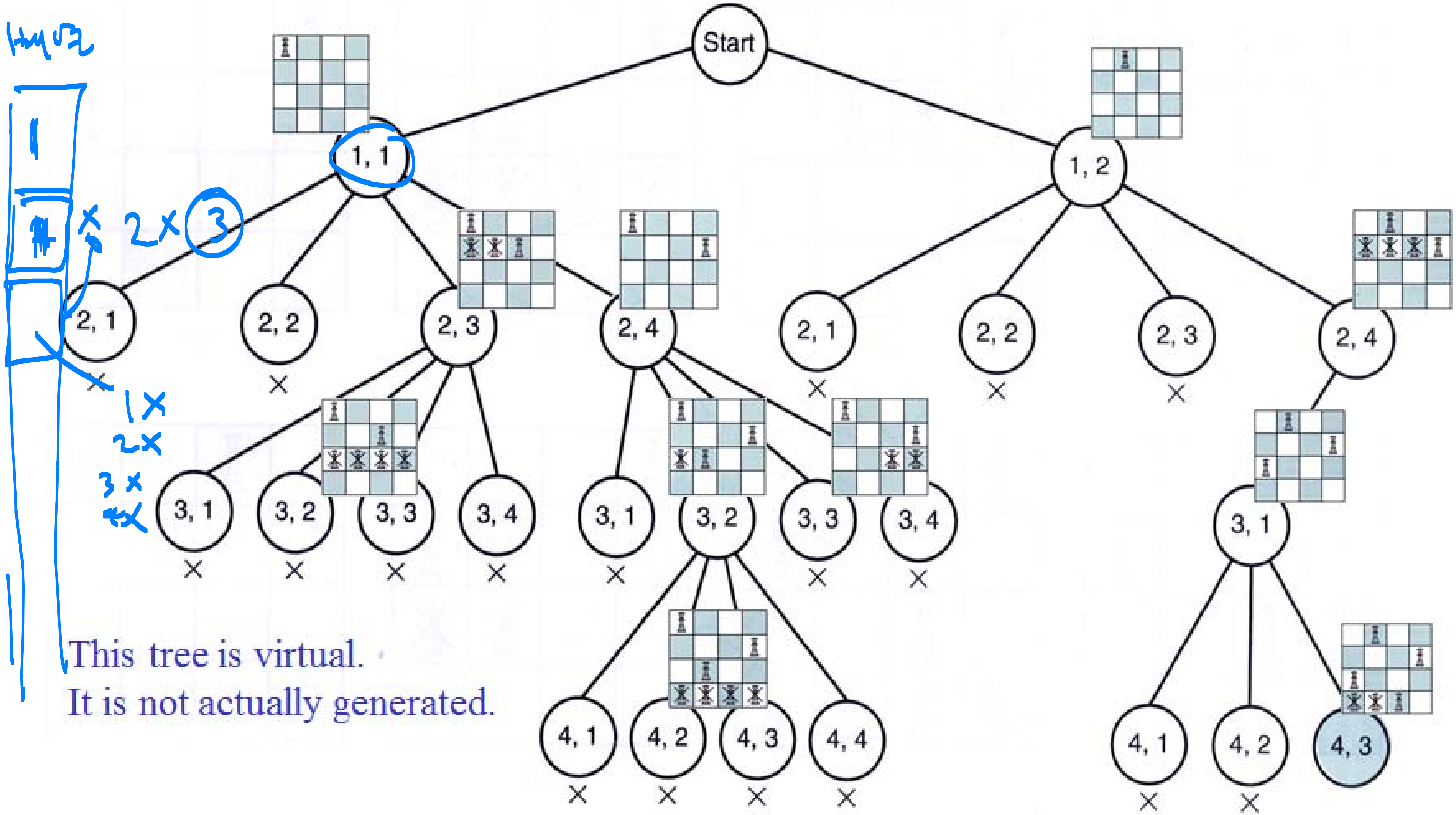
Promising function : application dependant



solution (array 로 나옴.



◆ An Example (The 4-queen problem)



This tree is virtual.
It is not actually generated.

Backtracking 에 의한 Alg 설계

⓪ 주어진 문제 : NP hard 인지 check → yes 이면 backtracking 사용 가능.

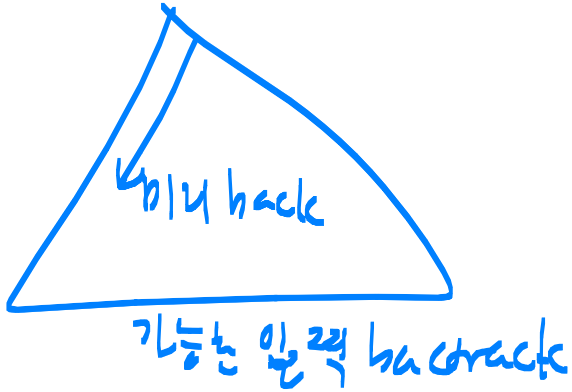
① state space tree 구성. (가상적)

② promising 함수. (가장 중요)

③ 자료구조 (거의 배열 사용)

④ programming

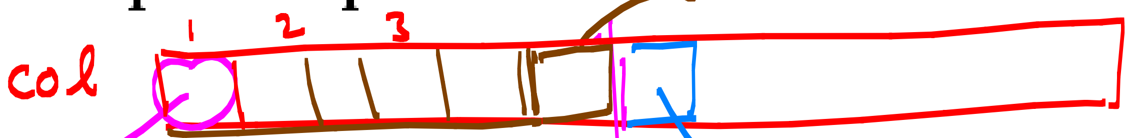
⑤ test. → 그러면 얼마나 효율적인가?



어떤 instance : quickly finished
" : need very very long time.
instance가 주어졌을때 reasonable 한 시간에 끝낼수 있는가?
미리 test 하는 방법? ⇒ 교재에 있음.
Monte Carlo method 사용.

Implementation of the n-queen problem

row index




```
void queens (index i){
   index j;
   if (promising(i))
      if (i == n)
         cout << col[1] through col [n];
      else
         for (j = 1; j <= n; j++){ // See if queen in (i + 1)st
            col[i + 1] = j;         // row can be positioned
            queens(i + 1);          // in each of the n columns.
         }
}
```

column index

i-1 까지

fixed

promising

promising 체크 ①

1, 2, ---, n check → call queen

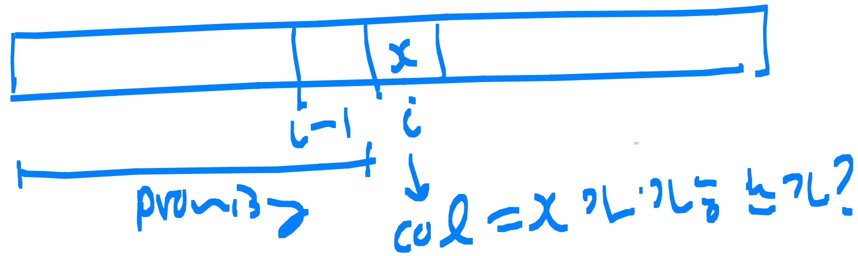
계속 진행

모두 check 했는데

backtrack

- col[i] : the column where the queen in the *i*-th row is located.
- col[1], col[2], …, col[i–1] : store the current promising queen's location.

✦ The Function *promising(i)*




```
bool promising (index i) {
    index k;   bool switch;
    k = 1;
    switch = true;                    // Check if any queen threatens
    while (k < i && switch) {         // queen in the ith row.
        if (col[i] == col[k] || abs(col[i] - col[k]) == i - k)
            switch = false;
        k++;
    }
    return switch;
}
```

하는 call?   queens(0)

i=0 이면 true return?

- col[i] == col[k] : checking if the two queens are at the same column.
- abs(col[i] – col[k]) == i – k : checking if diagonally located.

Analysis

- The total number of nodes in the state space tree

$$1 + n + n^2 + n^3 + \cdots + n^n = \frac{n^{n+1} - 1}{n - 1}$$

19,173,961 nodes for n=8

- The promising nodes are at most

$$1 + n + n(n-1) + n(n-1)(n-2) + \cdots + n!$$

109,601 nodes for n=8

- Actual number of nodes

n	Number of Nodes Checked by Algorithm 1†	Number of Candidate Solutions Checked by Algorithm 2‡	Number of Nodes Checked by Backtracking	Number of Nodes Found Promising by Backtracking
4	341	24	61	17
8	19,173,961	40,320	15,721	2057
12	9.73×10^{12}	4.79×10^{8}	1.01×10^{7}	8.56×10^{5}
14	1.20×10^{16}	8.72×10^{10}	3.78×10^{8}	2.74×10^{7}

Trying n! possible solutions.

DFS without backtracking. The number of nodes in the state space tree.