ALOHA 알고리즘반
09회차: Backtracking

강사: 우한샘

ALOHA
The algorithm club.

1장: 제약 충족 문제

Backtracking을 배우기 전에…

ALOHA
The algorithm club.

제약 충족 문제란?

모든 해(Solution) 중에서 제시되는 조건을 만족하는
최적해를 구하는 문제.

ALOHA
The algorithm club.

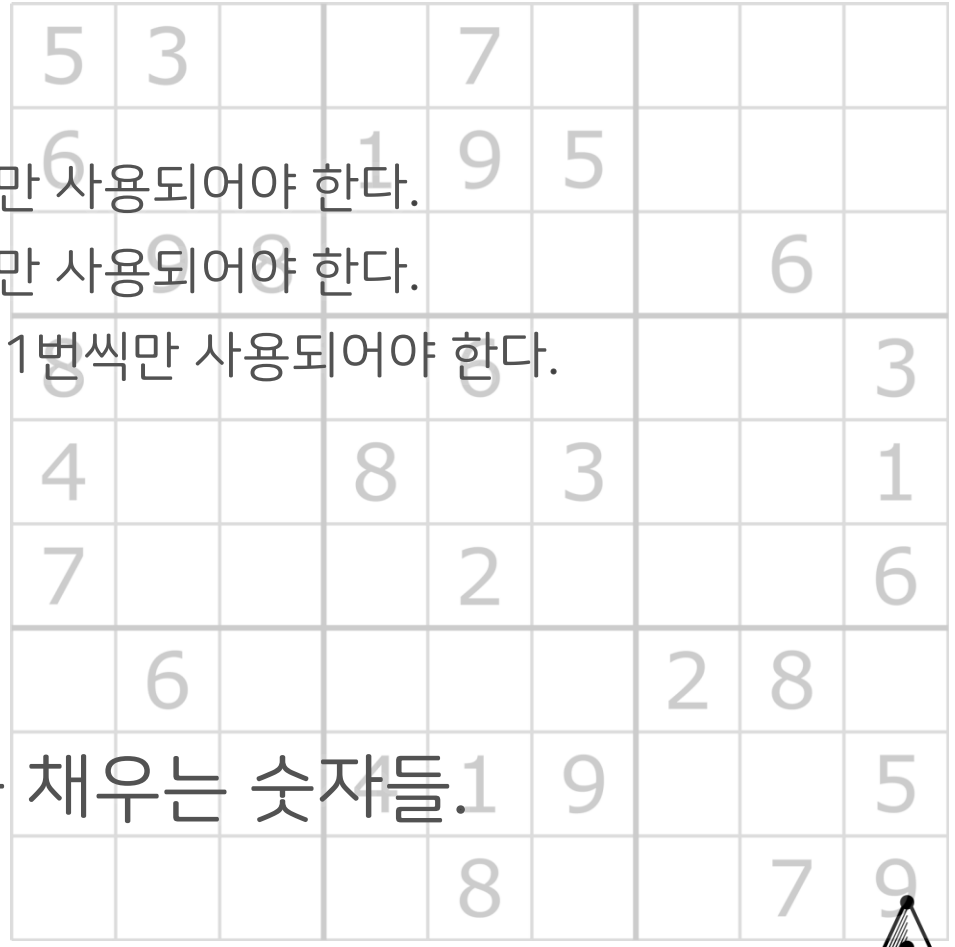
제약 충족 문제란?

예시 – Sudoku

조건 1 : 1개의 세로줄에는 1부터 9까지의 숫자가 1번씩만 사용되어야 한다.
조건 2 : 1개의 가로줄에는 1부터 9까지의 숫자가 1번씩만 사용되어야 한다.
조건 3 : 3×3 크기의 한 구획에는 1부터 9까지의 숫자가 1번씩만 사용되어야 한다.
+ 조건 : 고정된 숫자들.

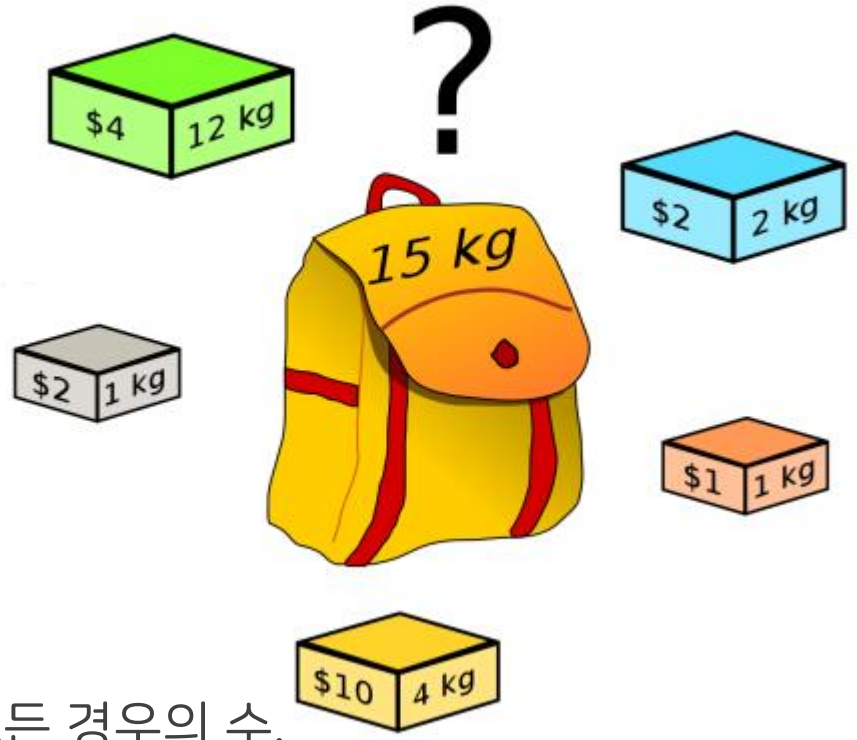
모든 해 : 1부터 9까지의 숫자들.

최적해 : 조건을 충족하면서 남은 빈 칸을 채우는 숫자들.



ALOHA
The algorithm club.

제약 충족 문제란?



예시 – 0/1 Knapsack

조건 1 : 각 보석들은 최대 1개씩만 가방에 담을 수 있다.
조건 2 : 가방에 담을 보석의 무게의 합은 w 이하여야 한다.

모든 해 : n개의 보석에 대해 넣을지 말지를 고려한 2^n 개의 모든 경우의 수.

최적해 : 조건을 만족하면서 최대한 많은 이익을 얻을 수 있는 보석들.

제약 충족 문제란?

Q1. 제약 충족 문제를 해결하는데 최적화된 알고리즘이 있나요?

A1. 최적화된 알고리즘이 있는 문제도 있고, 그렇지 않은 문제도 있어요.
(ex. 0/1 Knapsack 문제는 $O(Nw)$ 시간복잡도의 DP 알고리즘이 존재)

Q2. 그렇다면 최적화된 알고리즘이 없는 문제는 어떡해요?

A2. -다음장-

ALOHA
The algorithm club.

2장: Backtracking

Backtracking과 이를 활용하는 문제 풀이

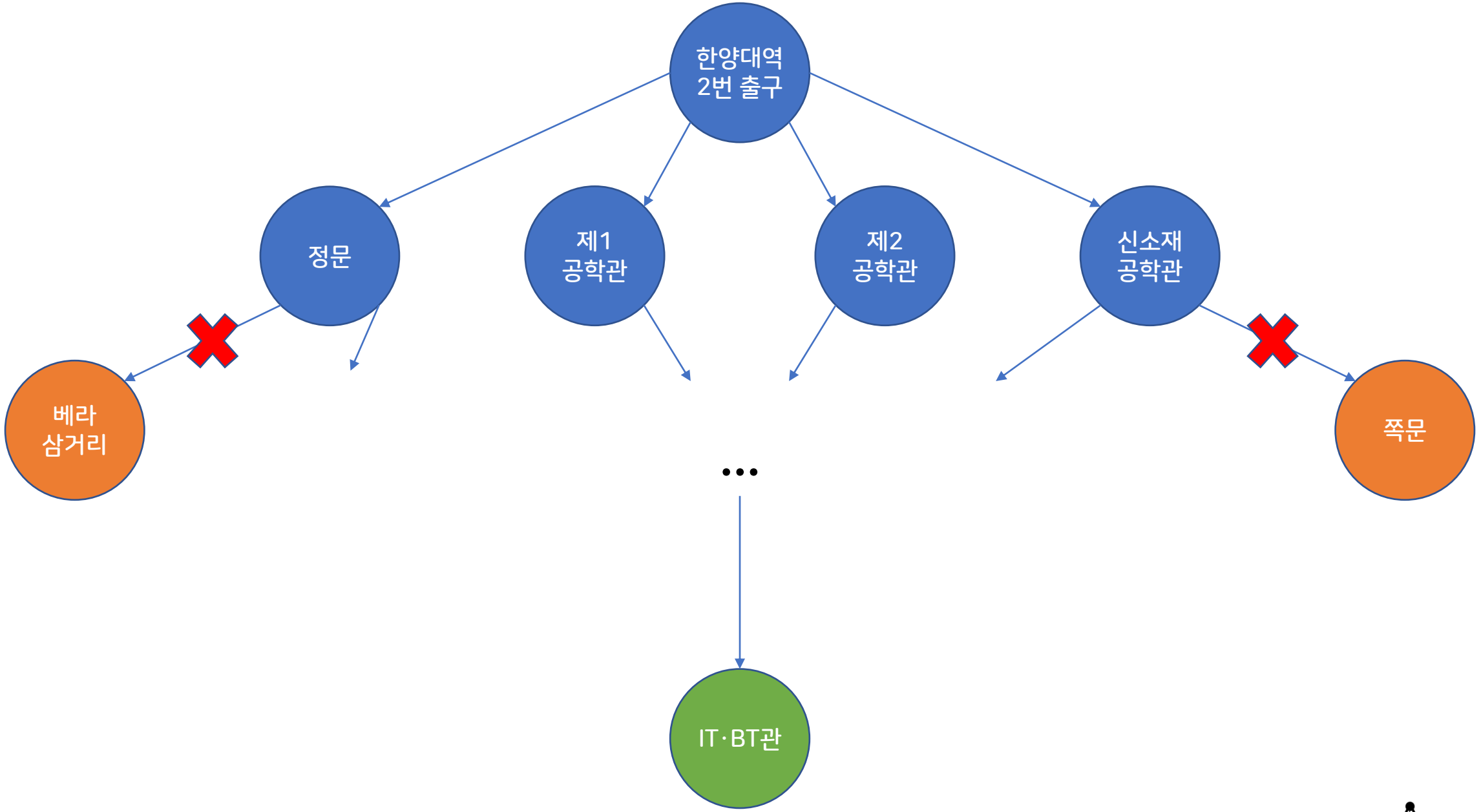
ALOHA
The algorithm club.

Backtracking이란?

Backtracking (퇴각검색)

- 제약 충족 문제에서 최적해를 찾기 위한 전반적인 알고리즘
- 최적해를 향해 계속 **가지를 뻗어나가고**, 그와 동시에 가능성이 없는 해는 **가지를 제거**하면서 최적해를 찾는다.

어떻게?



ALOHA
The algorithm club.

Backtracking의 구현

최적해가 될지도 모르는 **후보**들의 집합을 **트리**로 나타내고,

이를 **순회**하면서 최적해를 찾는다!

Q1. 해를 자료형으로 어떻게 나타내나요?

Q2. 해의 집합을 어떻게 트리로 표현하나요?

Q3. 순회는 어떻게 이루어지나요?

ALOHA
The algorithm club.

해의 표현

일반적으로 제약 충족 문제에서 해를 n-tuple로 표현

- **Sudoku**: 각 빈칸에 들어갈 숫자를 수열 x_i 에 대입하여 tuple로 나타낸다

$$\{x_1, x_2, \dots, x_n \mid 1 \leq x_i \leq 9\}$$

- **0/1 Knapsack**: 각 보석을 넣을 때는 1, 넣지 않을 때는 0으로 표현, 수열 x_i 에 대입하여 tuple로 나타낸다

$$\{x_1, x_2, \dots, x_n \mid x_i \in \{0, 1\}\}$$

ALOHA
The algorithm club.

해의 집합의 트리 표현

Depth = 0: 아무런 값도 확정되지 않은 tuple을 root node로!

Depth = 1: x_1까지의 값이 확정된 tuple을 node로!

Depth = 2: x_2까지의 값이 확정된 tuple을 node로!

…

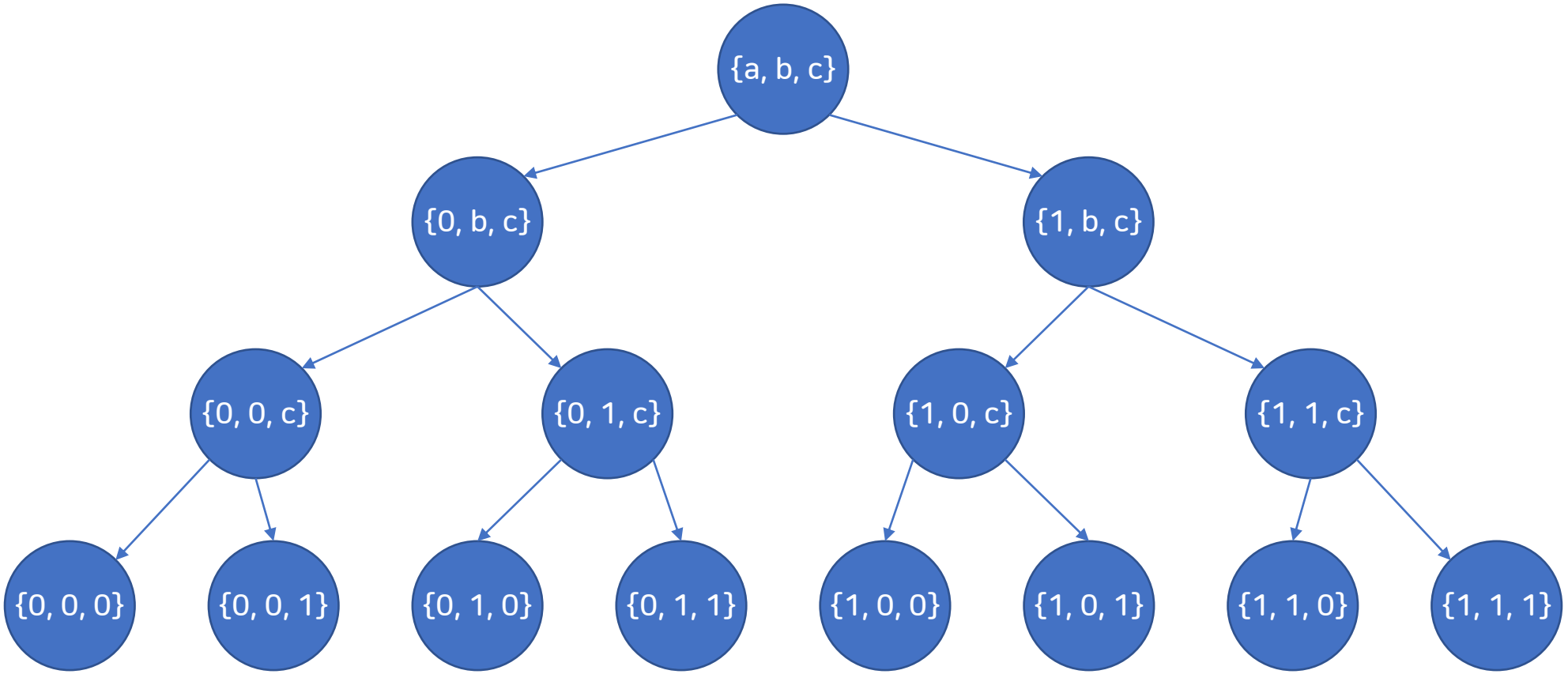
Depth = n: x_n까지의 값이 모두 확정된 tuple을 node로 나타내며,

동시에 이 값이 최적해인지를 검사!

상태 공간 트리 (State Space Tree)

ALOHA
The algorithm club.

해의 집합의 트리 표현



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

상태 공간 트리의 순회

트리 ∈ 그래프!

그래프의 순회 방법 두 가지를 기억하시나요…?

BFS(너비 우선 탐색): queue 사용, Level-order-traversal

DFS(깊이 우선 탐색): stack 사용, Pre-order-Traversal

그럼 상태 공간 트리는 어떤 방법을 이용해서 탐색해야 할까요?

ALOHA
The algorithm club.

상태 공간 트리의 순회

- 공간복잡도(Space Complexity)

프로그램을 실행시킨 후 완료하는 데 필요로 하는 **자원 공간**

➔ 프로그램이 차지하는 메모리 공간!

상태 공간 트리는 그 특성상 노드의 개수가 아주아주아주아주아주 많을 수밖에 없기 때문에, 알고리즘의 공간복잡도에 민감합니다!

ALOHA
The algorithm club.

상태 공간 트리의 순회

BFS: x_i의 범위가 2라고 해도, 해를 구성하는 숫자가 많아질수록 공간 복잡도가 빠르게 증가 $O(2^N)$

DFS: x_i의 범위와 상관 없이, 해를 구성하는 숫자가 많아져도 공간 복잡도가 천천히 증가 $O(\log_2 N)$

➔ Backtracking의 트리 순회를 위해서는 DFS를 사용하는 게 좋다!

ALOHA
The algorithm club.

상태 공간 트리의 순회

그래프의 DFS처럼 후보해를 **노드-간선 자료형**으로 저장할 경우,

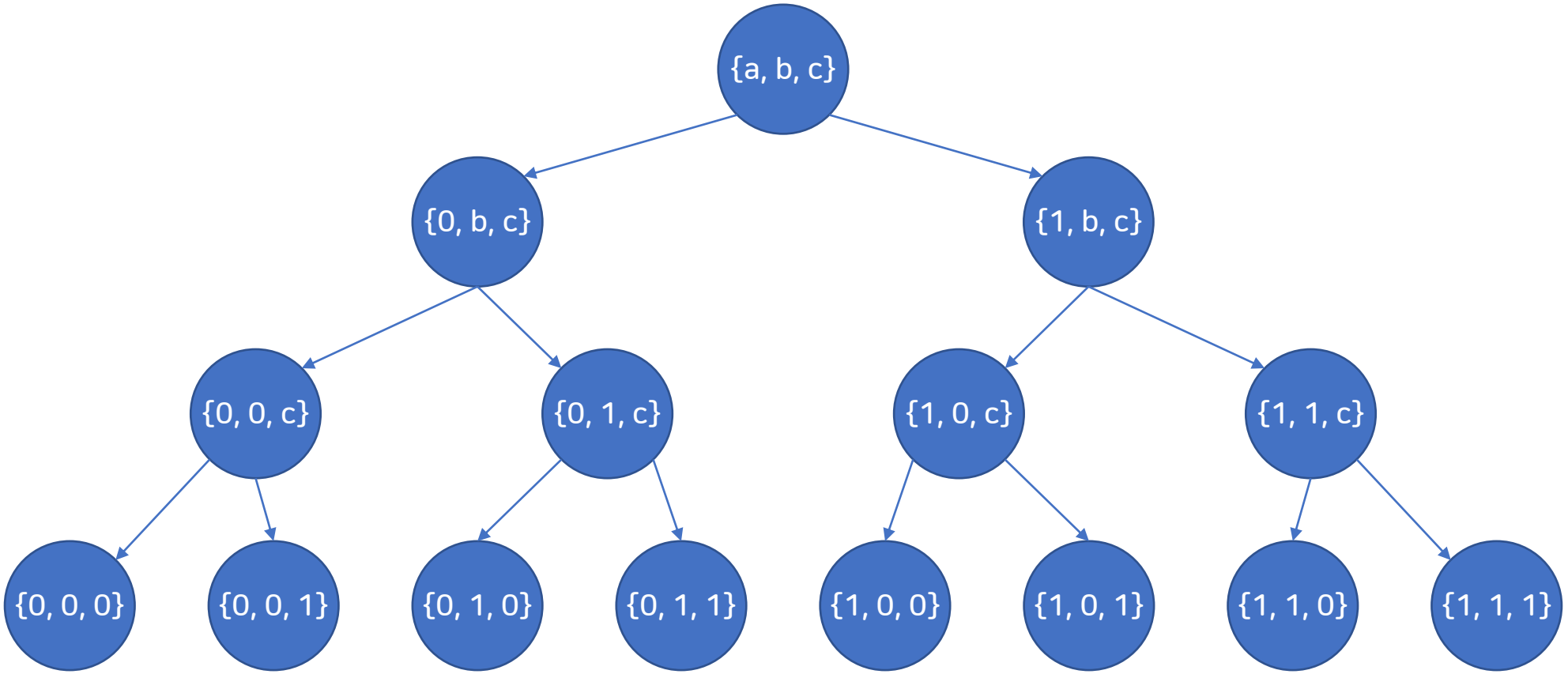
후보해의 개수가 너무 많기 때문에 **메모리 초과**가 발생합니다!

하지만 우리는 이미 **해의 범위**를 알고 있기 때문에,

한 노드와 그 노드가 가진 해만을 이용해서 자식 노드를 만들어낼 수 있습니다.

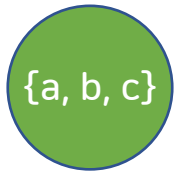
따라서, 노드와 간선의 정보를 저장하지 않고 순회를 합니다!

상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

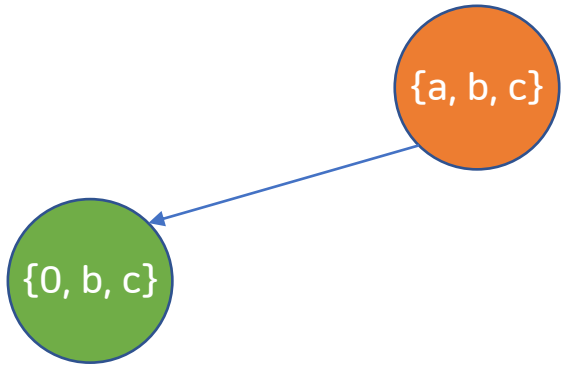
ALOHA
The algorithm club.

상태 공간 트리의 순회

보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

ALOHA
The algorithm club.

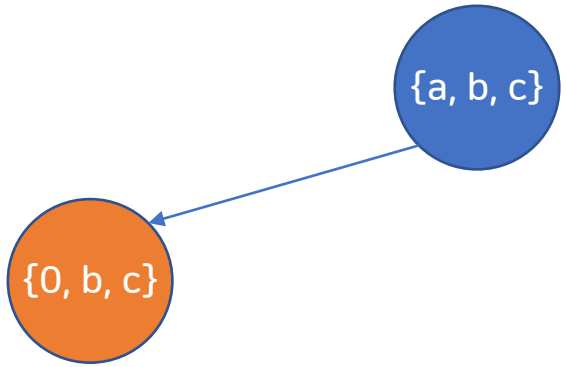
상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

ALOHA
The algorithm club.

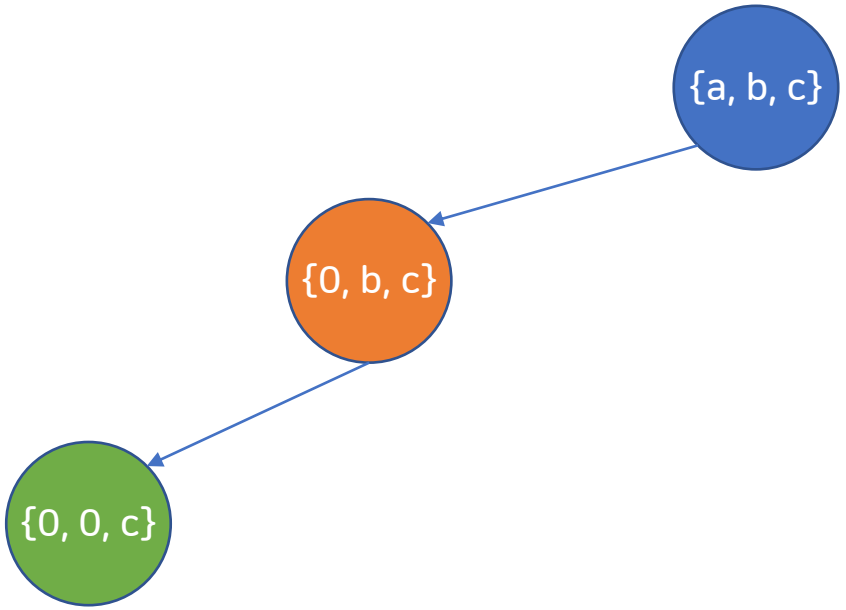
상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

ALOHA
The algorithm club.

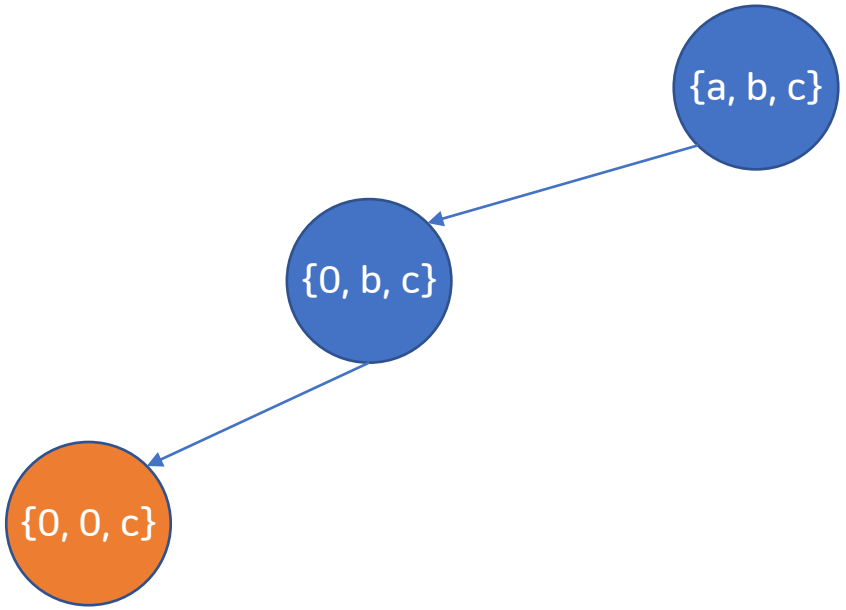
상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

ALOHA
The algorithm club.

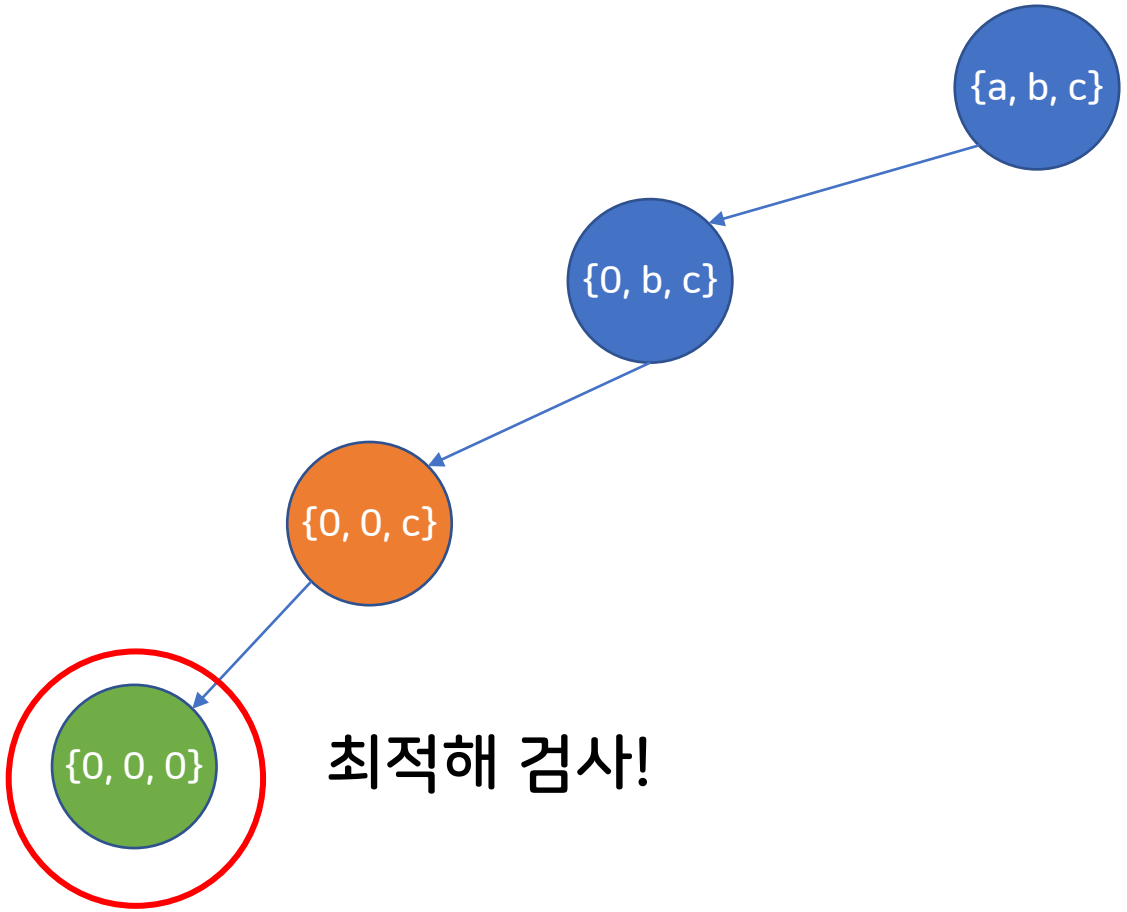
상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

ALOHA
The algorithm club.

상태 공간 트리의 순회

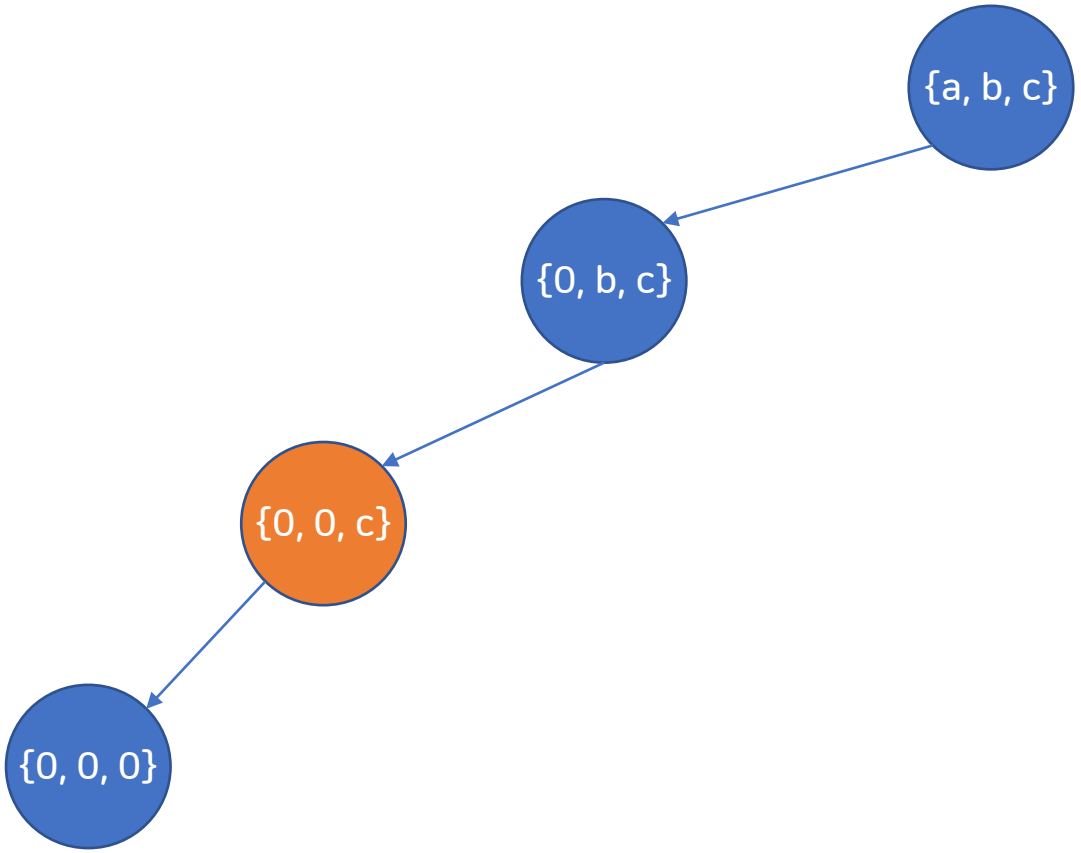


최적해 검사!

보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

ALOHA
The algorithm club.

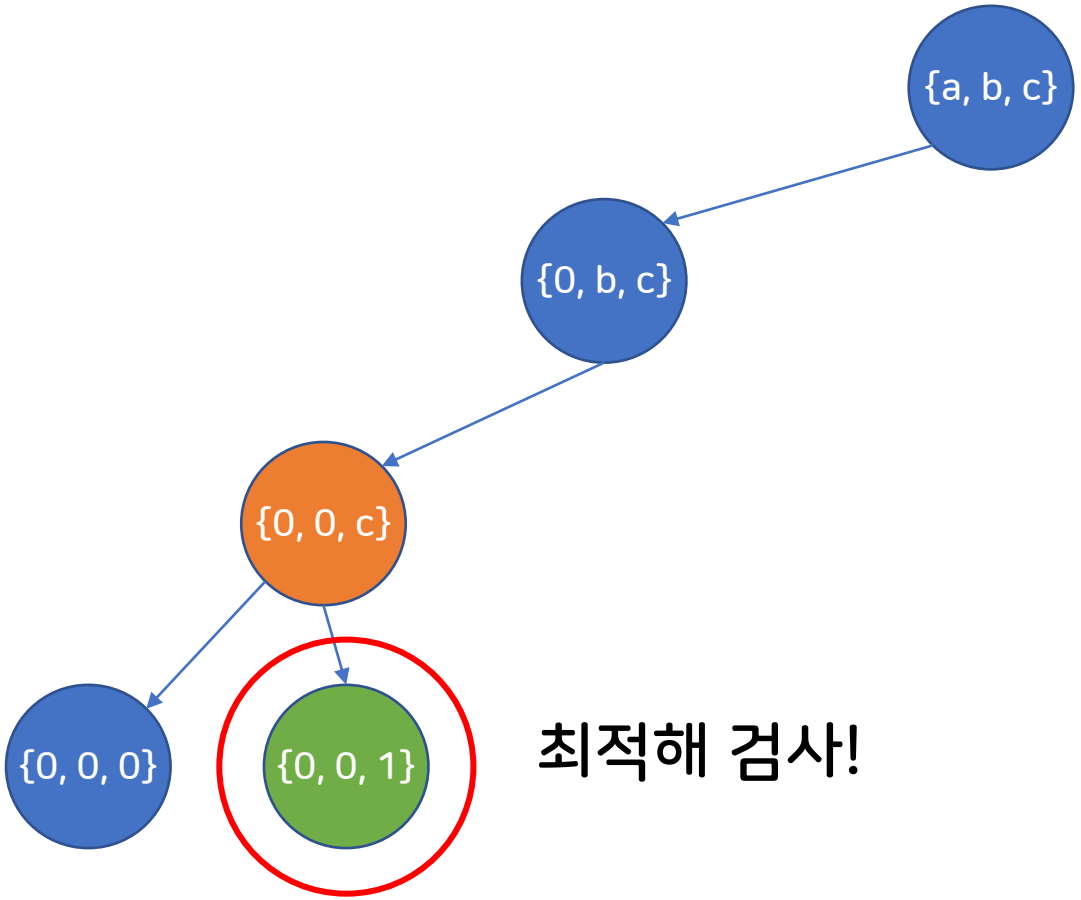
상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

ALOHA
The algorithm club.

상태 공간 트리의 순회

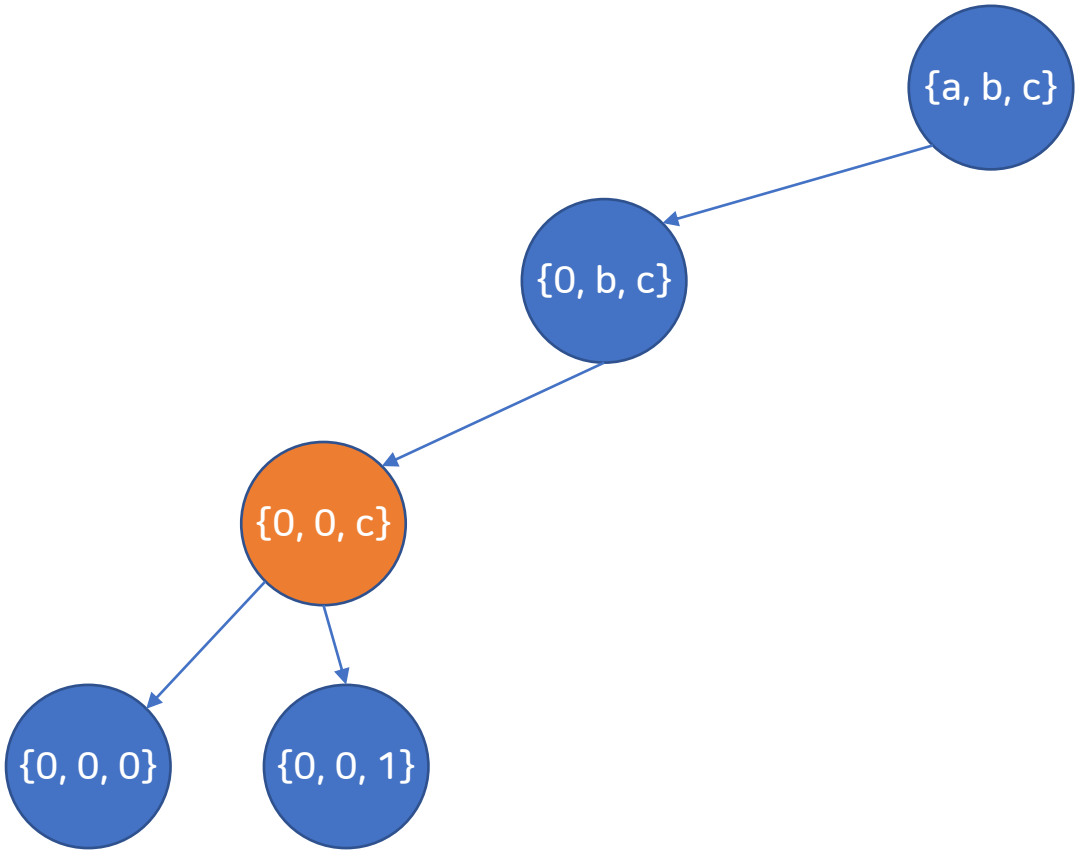


최적해 검사!

보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

ALOHA
The algorithm club.

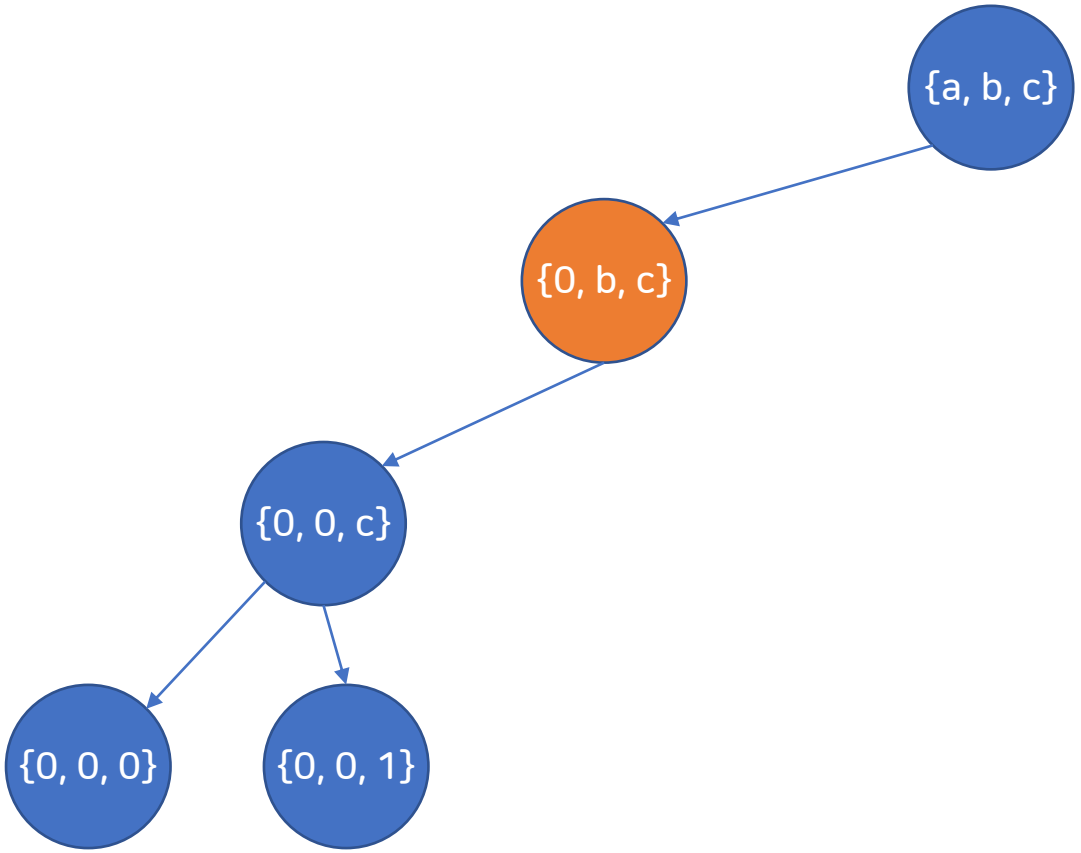
상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

ALOHA
The algorithm club.

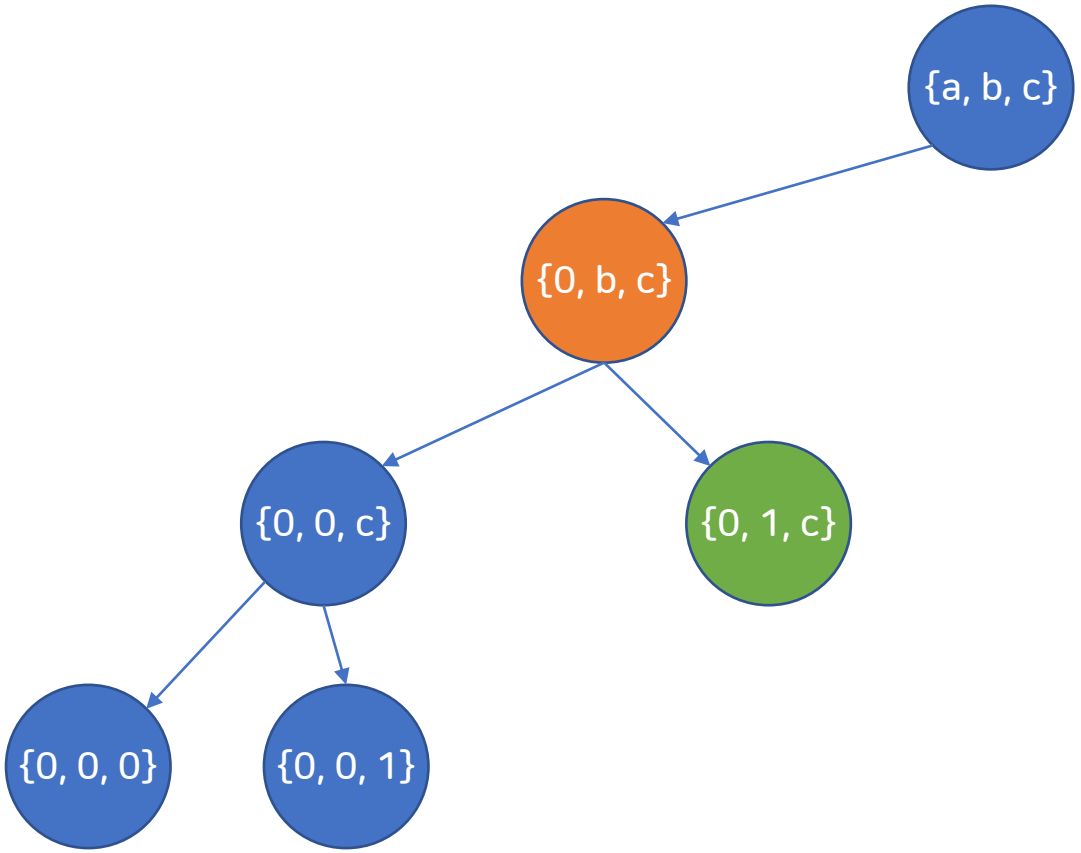
상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

ALOHA
The algorithm club.

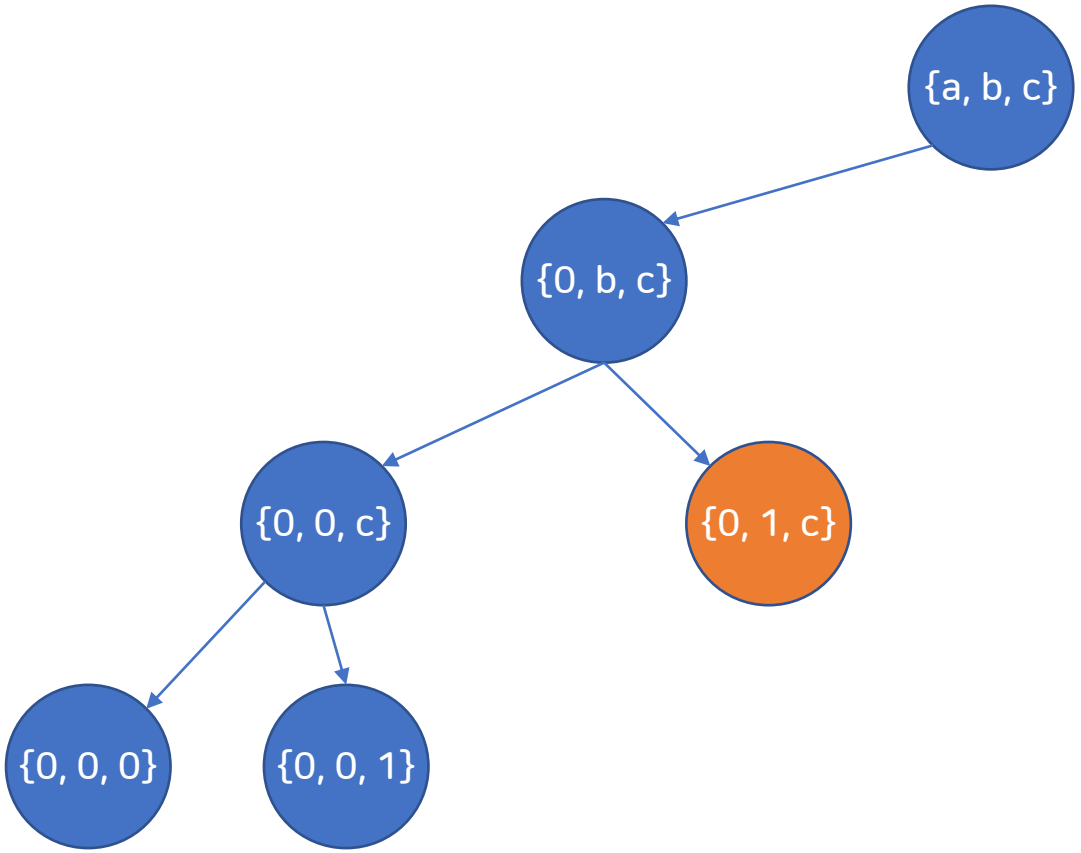
상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

ALOHA
The algorithm club.

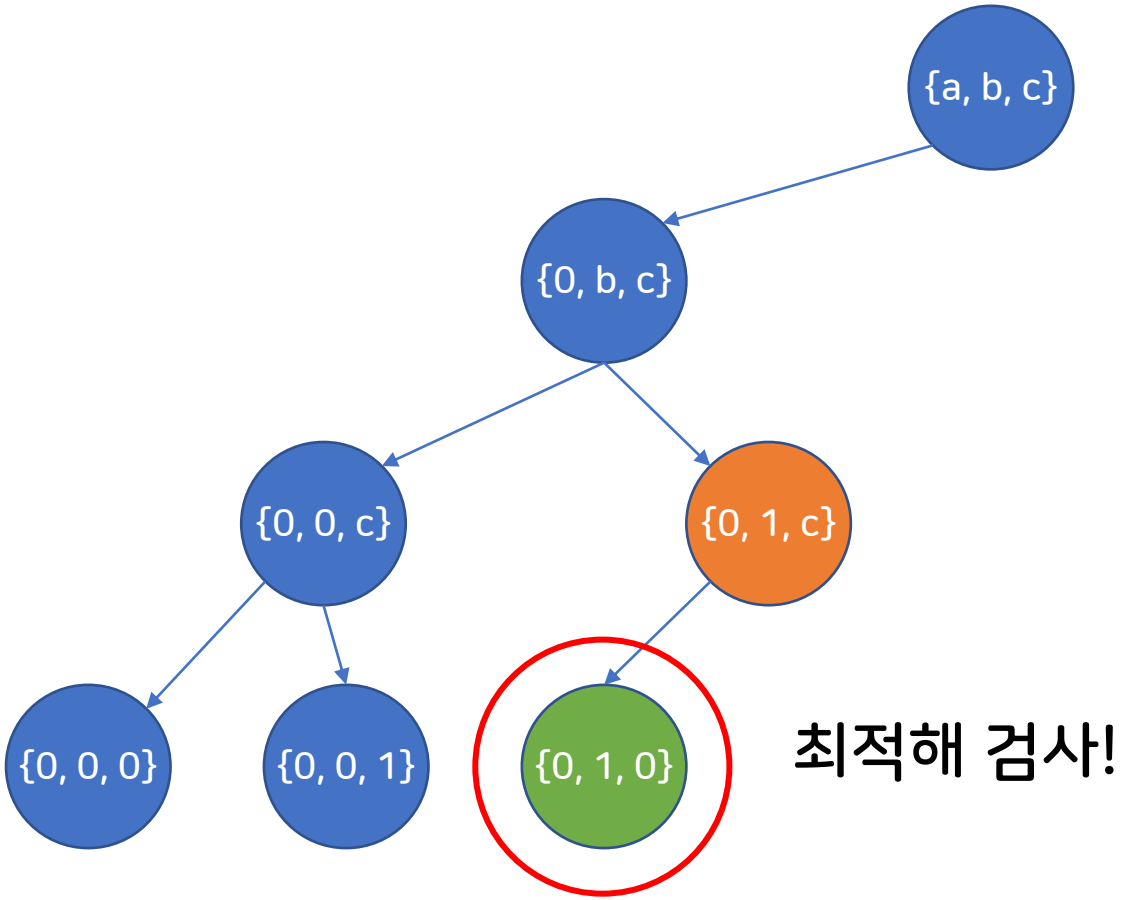
상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

ALOHA
The algorithm club.

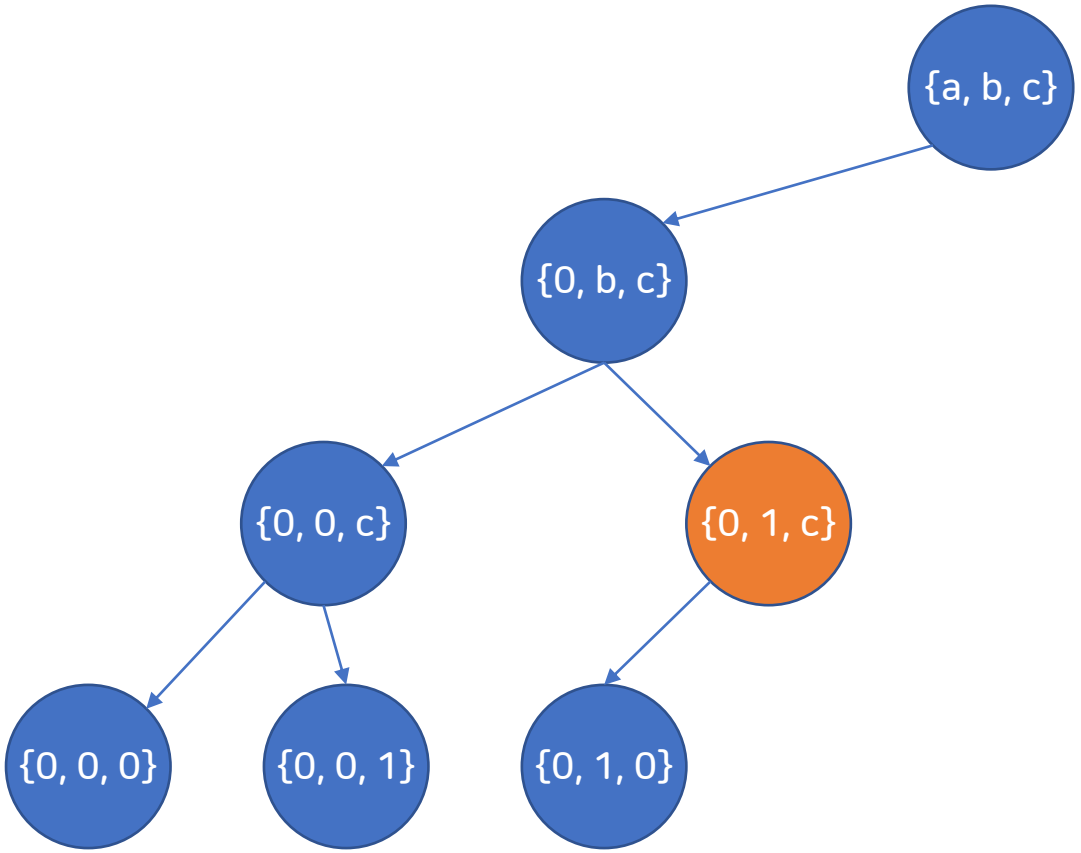
상태 공간 트리의 순회



최적해 검사!

보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

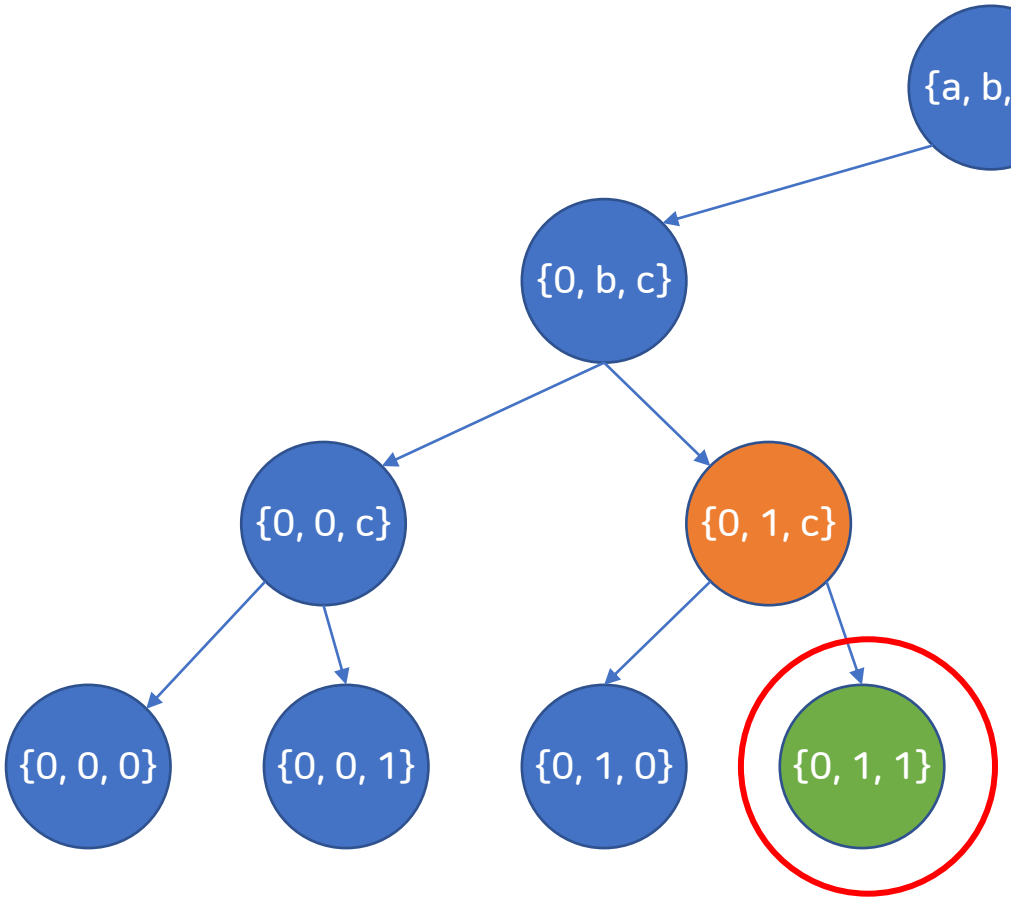
상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

ALOHA
The algorithm club.

상태 공간 트리의 순회



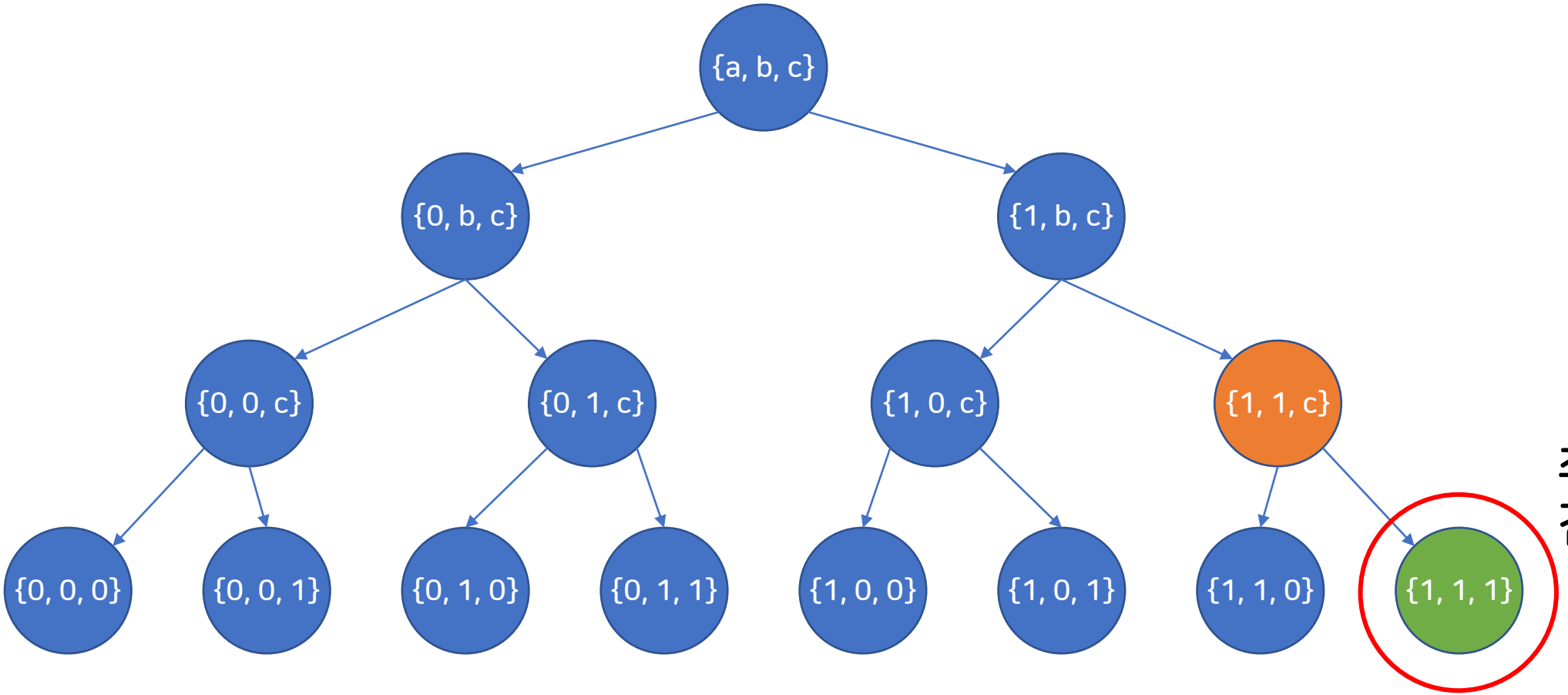
최적해 검사!

보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

ALOHA
The algorithm club.


A FEW
MOMENTS LATER

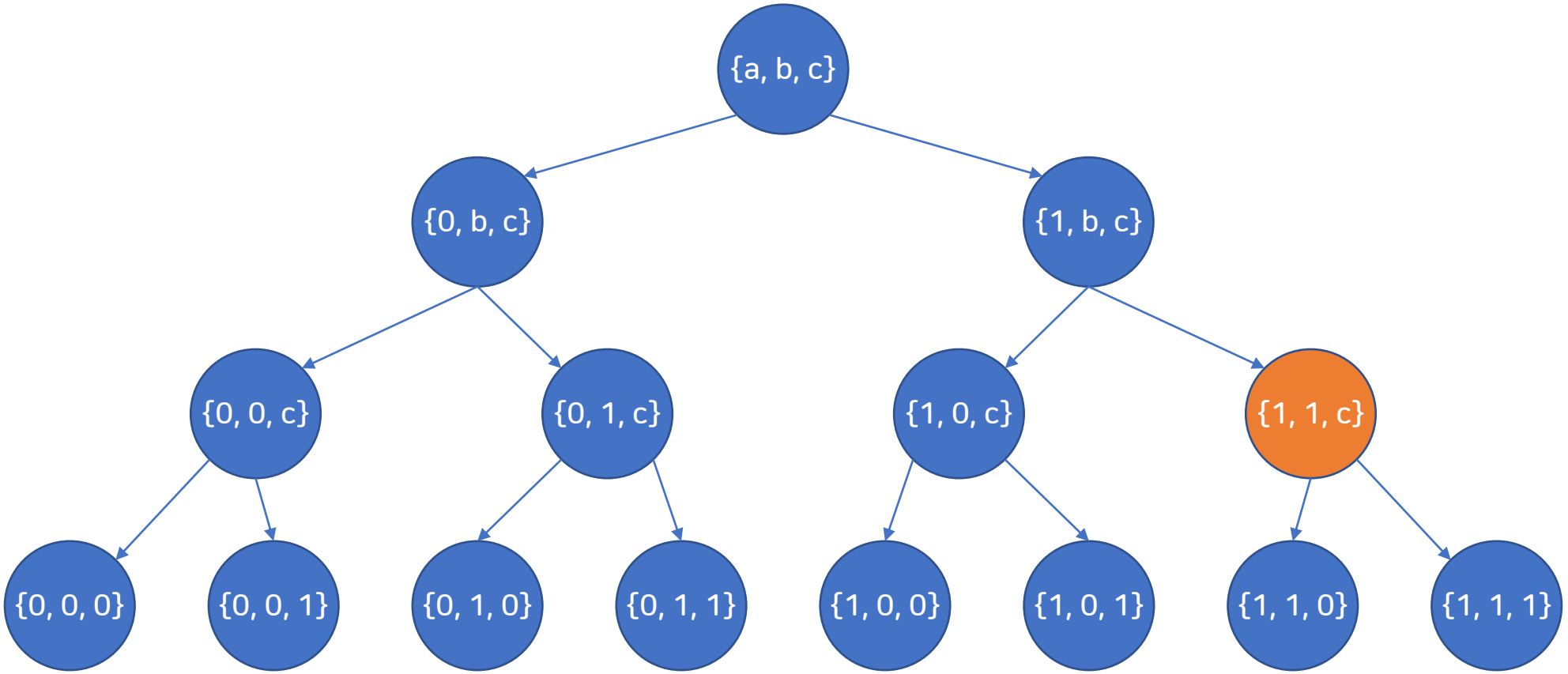
상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

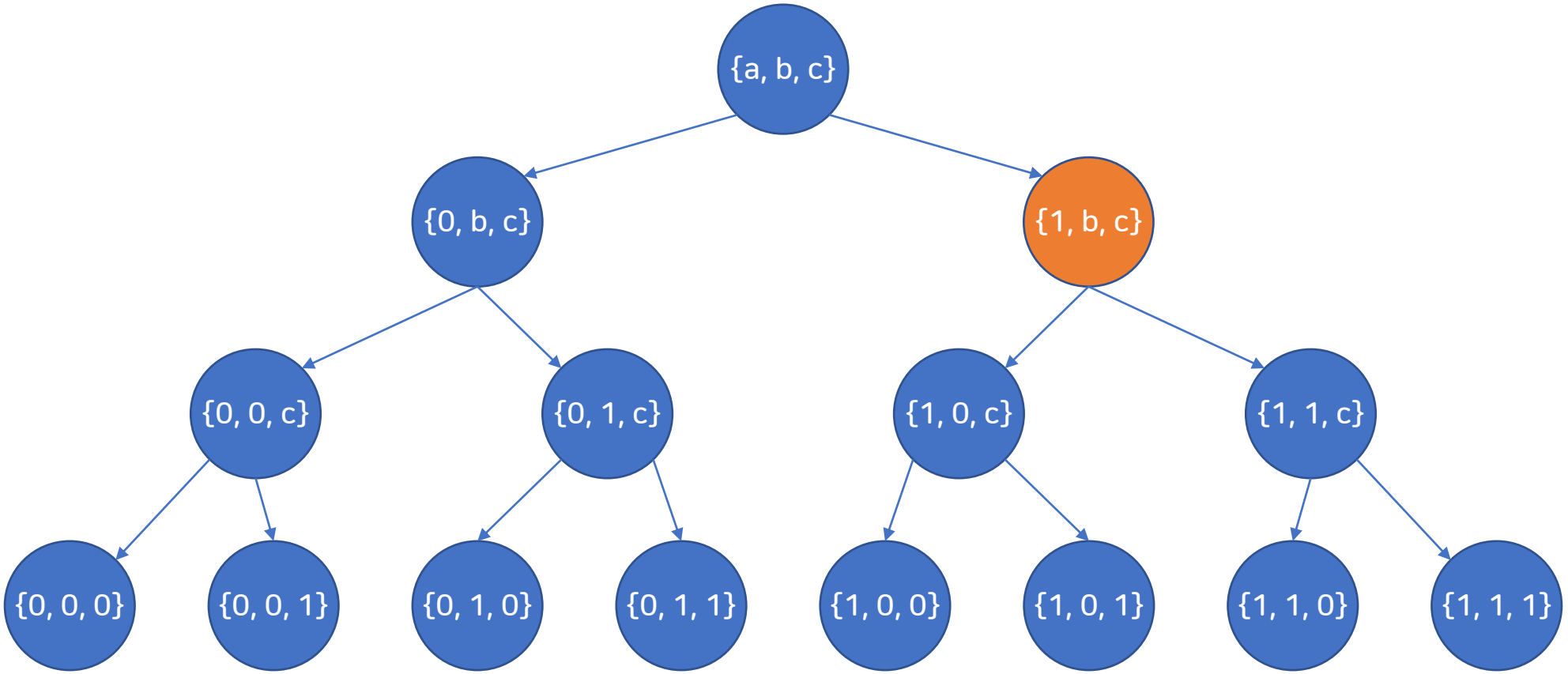
ALOHA
The algorithm club.

상태 공간 트리의 순회



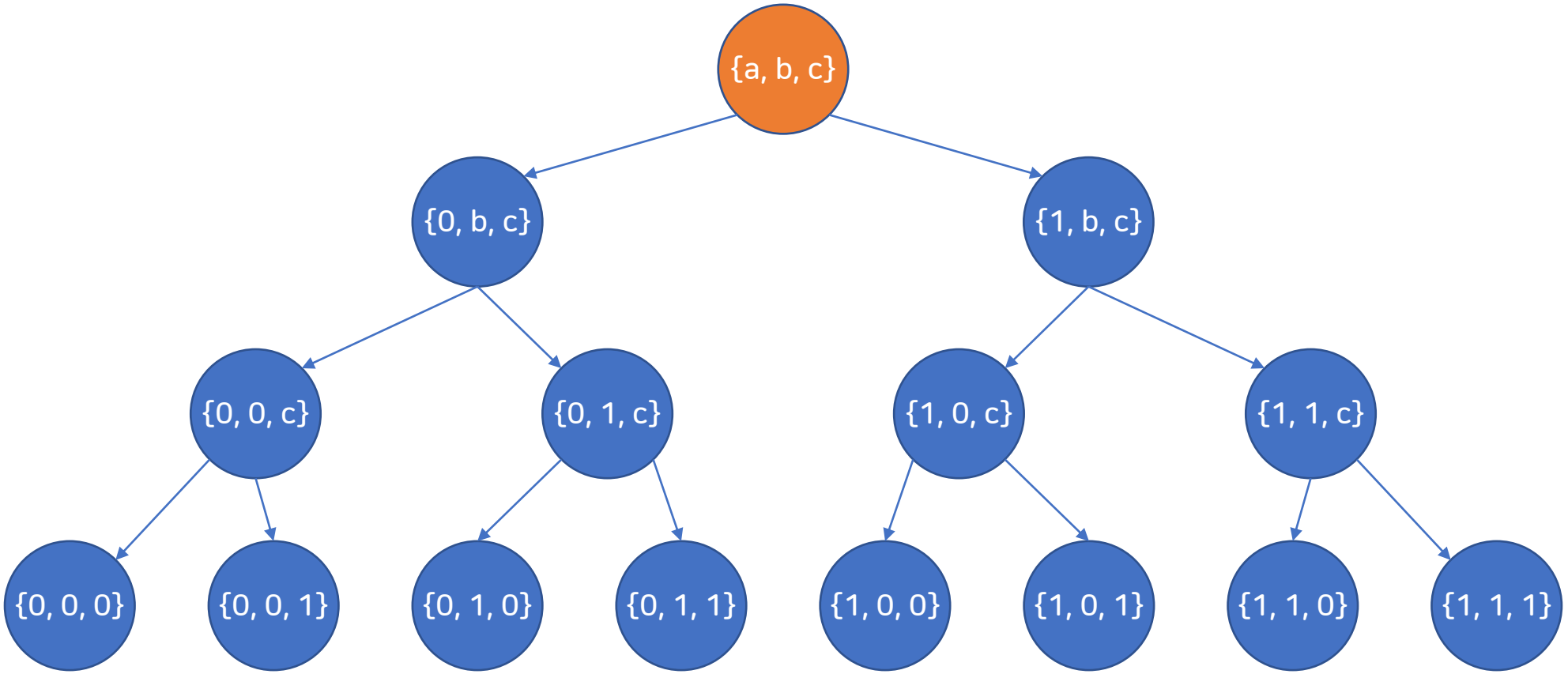
보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

상태 공간 트리의 순회



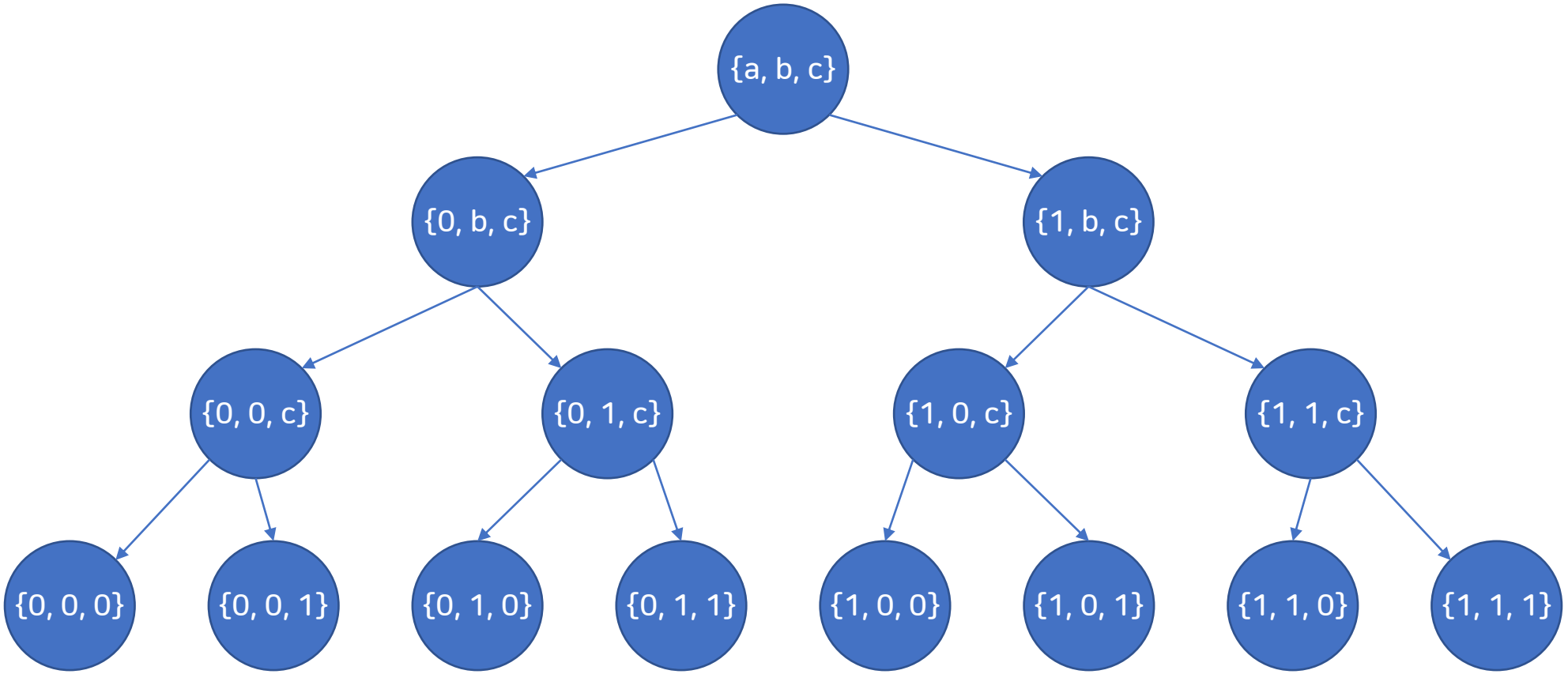
보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

상태 공간 트리의 순회



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리

상태 공간 트리의 순회

```
#define MAX_DEPTH 3
int solution[MAX_DEPTH];

void Backtracking(int depth) {
  if (depth == MAX_DEPTH) { // 해를 완성한 경우
    if (IsOK())             // 현재의 해가 최적해인지 확인
      DoSomething();
    return;
  }

  for (int i = 0; i <= 1; i++) { // 해를 완성하지 못한 경우
    // 해를 구성하는 숫자 중 depth번째 숫자를 i로 설정
    solution[depth] = i;
    Backtracking(depth + 1);     // 자식 노트로 이동
  }
}
```

연습 문제

BOJ 1759
암호 만들기

ALOHA
The algorithm club.

연습 문제 – 암호 만들기

조건 1 : 모든 암호는 사전순으로 정렬되어야 한다
조건 2 : 모든 암호는 최소 1개의 모음과 최소 2개의 자음이 있어야 한다.

모든 해 : C개의 문자를 이용하여 만들어진 길이 L의 문자열 $_{L}P_{C}$개.

최적해 : 모든 해 중 조건을 만족하는 비밀번호들의 집합.

ALOHA
The algorithm club.

연습 문제 – 암호 만들기

- Point 1. 가능성 있는 암호를 증가하는 순서로 만들어야 함!

➔ **정렬**된 문자를 이용하여

➔ 길이가 L인 **LIS(최장 증가 순열)**를 구성해야 함

- Point 2. 만든 암호를 사전 순으로 출력해야 함!

➔ 앞에서 사용한 문자를 배재한 후, 다음 노드로 이동해야 함

➔ 따라서, `Backtracking()` 함수에서 **이전 노드에서 사용한 문자를 기억**해야 최적해를 찾을 수 있음

ALOHA
The algorithm club.

연습 문제 – 암호 만들기

```
char arr[15];
int ans[15];

void Backtracking(int depth) {
  if (depth == L) {  // 해를 완성한 경우
    if (IsOK())      // 최적해인지 확인
      DoSomething();
    return;
  }

  for (int i = 0; i < C; i++) {                // 해를 완성하지 못한 경우
    if (depth != 0 && i <= ans[depth - 1])  // 사전 순으로 앞에 있는 문자를 배제
      continue;
    ans[depth] = i;  // 해를 구성하는 depth번째 문자를 arr의 i번째 문자로 설정
    Backtracking(depth + 1);  // 자식 노드로 이동
  }
}
```

연습 문제 – 암호 만들기

```
bool IsOK() {
  int vowelNum = 0, consonantNum = 0;
  for (int i = 0; i < L; i++) {
    if (arr[ans[i]] == 'a' || arr[ans[i]] == 'e' || arr[ans[i]] == 'i' ||
        arr[ans[i]] == 'o' || arr[ans[i]] == 'u')
      vowelNum++;
    else
      consonantNum++;
  }
  if (vowelNum >= 1 && consonantNum >= 2)
    return true;
  else
    return false;
}
```

ALOHA
The algorithm club.

연습 문제

BOJ 15649
N과 M (1)

ALOHA
The algorithm club.

연습 문제 – N과 M (1)

- 주어진 범위 내의 수들을 이용하여 **순열(Permutation)**을 출력하는 문제

- 암호 만들기와 유사한 방법으로 풀되, `Backtracking()`의 작동 방법과 `IsOK()`의 조건 확인 부분을 바꿔주면 됩니다.

- N과 M은 (1)부터 (12)까지 총 12문제가 있는데, 전부 다 Backtracking 연습에 도움이 되니 시간 나면 꼭 다 풀어보는 걸 추천 드려요!

ALOHA
The algorithm club.

3장: 좀 더 어려운 Backtracking

아까보다 더 머리 아파지는 Backtracking 문제들 모음…

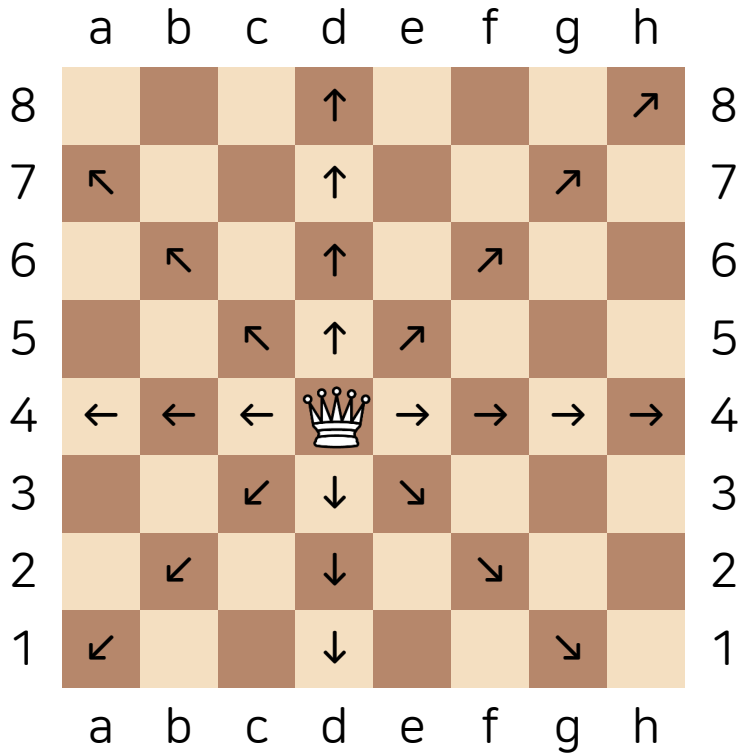
ALOHA
The algorithm club.

n-Queen Problem

체스의 퀸(Queen)

- 체스에서 가장 강력한 기물
- 한 번의 턴에 원하는 만큼의 거리를 이동시킬 수 있음
- 이 때 상하좌우는 물론, 대각선 방향까지도 이동 가능

갑자기 웬 체스 이야기냐구요…?



퀸의 행마법

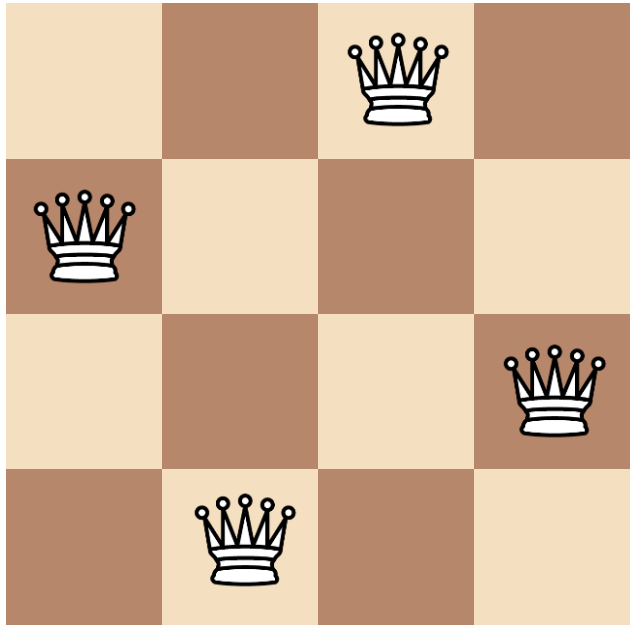
ALOHA
The algorithm club.

n-Queen Problem

n-Queen 문제

N×N 크기의 체스판 위에서, N개의 퀸을 서로가 서로를 잡을 수 없도록 놓을 수 있는 배치의 경우의 수는?

- 조건 1: 같은 행에는 1개의 퀸만 존재해야 한다.
- 조건 2: 같은 열에는 1개의 퀸만 존재해야 한다.
- 조건 3: 같은 대각선에는 1개의 퀸만 존재해야 한다.



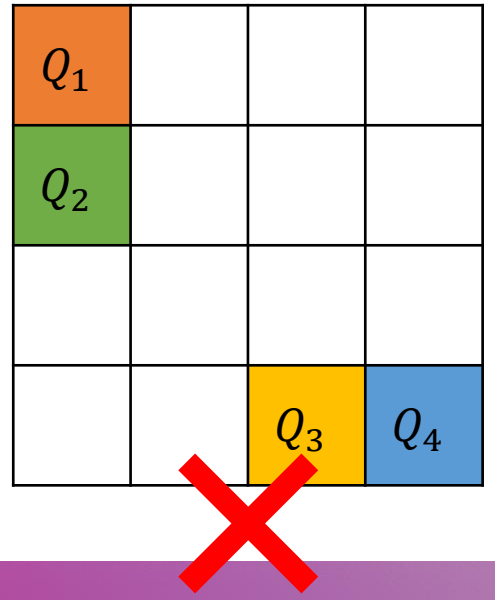
4-Queen 문제의 해답 중 하나를 나타낸 그림

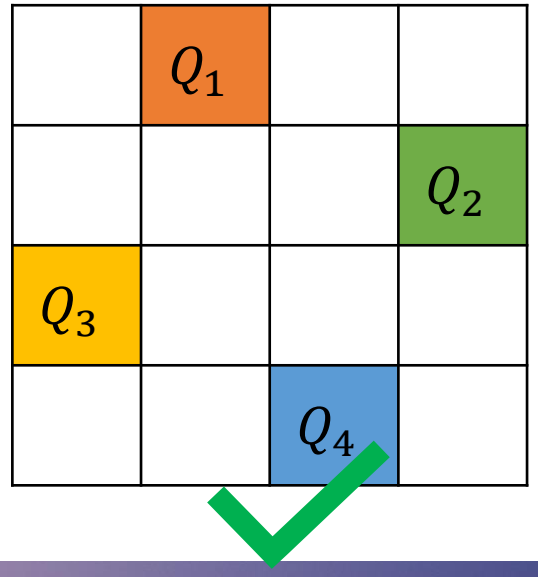
ALOHA
The algorithm club.

4-Queen Problem

조건 1과 조건 2에 의해, 하나의 가로줄/세로줄에는 한 개의 퀸만이 올 수 있어요.

따라서, 모든 칸에 대해 퀸을 놓는 경우를 생각할 필요 없이,
각 **행(또는 열)**에 대해 어느 **y좌표(또는 x좌표)**에 퀸을 둘지 정하면 됩니다!





ALOHA
The algorithm club.

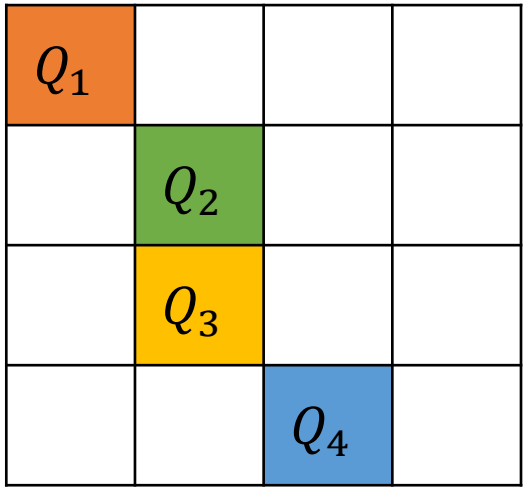
4-Queen Problem

- 후보 해: $\{x_1, x_2, x_3, x_4 | 1 \leq x_i \leq 4\}$

후보 해에서 이미 조건 1을 만족시켰으므로,
최적해에서는 조건 2와 조건 3만 확인해주면 됩니다.

조건 2는 자신보다 앞선 해들이 고르지 않은 수만
고르도록 하면 쉽게 해결됩니다.

가장 중요한 **조건 3**은 어떻게 해결할까요…?



$\{1, 2, 2, 3\}$에 해당하는 체스판의 모습

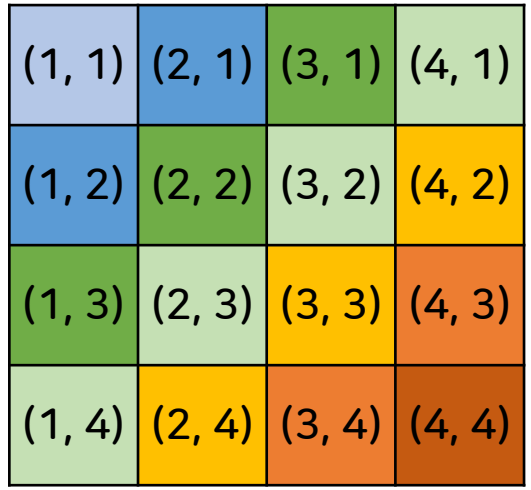
ALOHA
The algorithm club.

4-Queen Problem

x좌표와 y좌표의 합과 차를 이용하는 방법으로 구할 수 있습니다!

(1, 1)	(2, 1)	(3, 1)	(4, 1)
(1, 2)	(2, 2)	(3, 2)	(4, 2)
(1, 3)	(2, 3)	(3, 3)	(4, 3)
(1, 4)	(2, 4)	(3, 4)	(4, 4)

x좌표와 y좌표의 차가 일정한 칸들끼리
같은 색으로 칠한 모습



x좌표와 y좌표의 합이 일정한 칸들끼리
같은 색으로 칠한 모습

4-Queen Problem

```
 1  #define N 4
 2
 3  int ans;
 4  int solution[N];
 5
 6  void Backtracking(int depth) {
 7    if (depth == N) {
 8      if (isOK()) ans++;
 9      return;
10    }
11
12    for (int i = 0; i < N; i++) {
13      solution[depth] = i;
14      Backtracking(depth + 1);
15    }
16  }
```

4-Queen Problem

```
1  bool isOK() {
2    // upper과 lower의 크기는 2배보다 커야 한다!
3    vector<bool> row(N), upperDiagonal(2 * N + 1), lowerDiagonal(2 * N + 1);
4    for (int i = 0; i < N; i++) {
5      if (row[solution[i]]) return false;
6      row[solution[i]] = true;
7
8      // i - solution[i] 의 값이 -N 까지 내려갈 수 있으므로 N을 더해준다
9      if (upperDiagonal[N + i - solution[i]]) return false;
10     upperDiagonal[N + i - solution[i]] = true;
11
12     if (lowerDiagonal[i + solution[i]]) return false;
13     lowerDiagonal[i + solution[i]] = true;
14   }
15   return true;
16 }
```

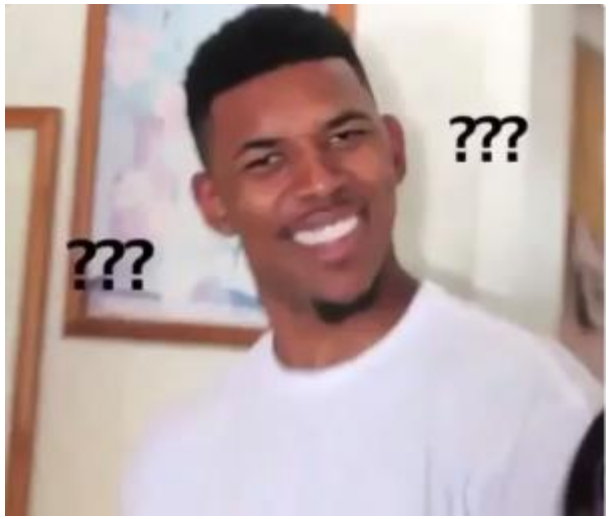
ALOHA
The algorithm club.

연습 문제

BOJ 9663
N-Queen

연습 문제 – N-Queen

연습 문제 – N-Queen



ALOHA
The algorithm club.

연습 문제 – N-Queen

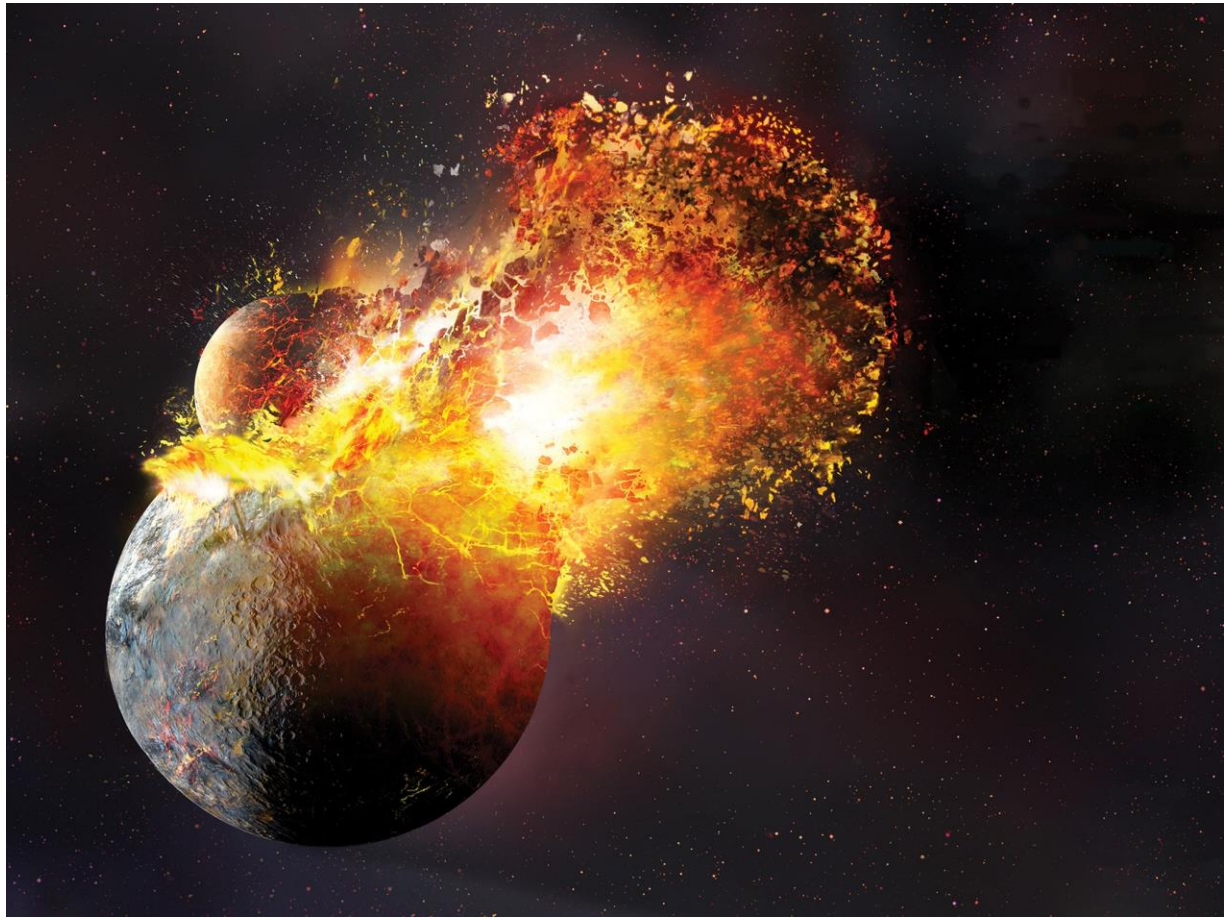
문제에서 주어진 체스판의 최대 크기는 N = 14

14개의 tuple x_i에 대해 1부터 14까지의 숫자를 대입하는 경우의 수

$$14 \times 14 \times \cdots \times 14 = 14^{14} = 11{,}112{,}006{,}825{,}558{,}016$$

ALOHA
The algorithm club.

연습 문제 – N-Queen



ALOHA
The algorithm club.

Bounding Function

- **Bounding Function(한정 함수)**

노드를 생성할 때, 이 노드 아래에 최적해의 존재 여부를 판단해주는 함수

$$B_k(x_1, x_2, \cdots, x_k) = \begin{cases} true, & possible \\ false, & impossible \end{cases}$$

즉, 해를 전부 완성하지 않고도 미리 최적해 가능 여부를 판별하여

불가능한 경우 순회를 중단하고 **이전 노드로 되돌아간다(Backtrack)!**

ALOHA
The algorithm club.

Bounding Function

새로운 퀸을 배치할 때, 이미 해당 x좌표에 다른 퀸이 놓여있을 경우
조건 2에 위배되므로, 해당 depth에서의 x좌표를 정할 때
중복된 숫자가 있는지를 판별하면 좋습니다!

$$14 \times 13 \times \cdots \times 2 \times 1 = 14! = 87{,}178{,}291{,}200$$

와! 정말 많이 줄었어요!

Bounding Function

문제 풀이의 핵심인 **조건 3**은 어떤 식으로 해결할까요?

`IsOK()` 함수에 있던 대각선 **중복 확인 판별을** Bounding Function**으로** 이용하면, 이미 해당 대각선에 퀸이 있을 경우 하위 노드로 내려가지 않을 수 있습니다!

즉, 노드를 생성할 때 퀸을 배치하고, **같은 depth의 다른 노드를 생성할 때 그 퀸을 지우는** 식으로 구현하면 $O(1)$의 시간복잡도로 검사할 수 있습니다!

ALOHA
The algorithm club.

4-Queen Problem w/ Bounding Function

```
1  void Backtracking(int depth) {
2    if (depth == 4) {
3      DoSomething(); // Bounding Function이 IsOK()를 대신하므로 IsOK를 제외해도 된다
4      return;
5    }
6
7    for (int i = 0; i < 4; i++) {
8      // Bounding Function
9      if (row[i] || upperDiagonal[4 + i - depth] ||  lowerDiagonal[i + depth])
10       continue;
11
12     row[i] = upperDiagonal[4 + i - depth] = lowerDiagonal[i + depth] = true;
13     Backtracking(depth + 1);
14     row[i] = upperDiagonal[4 + i - depth] = lowerDiagonal[i + depth] = false;
15   }
16 }
```

ALOHA
The algorithm club.

연습 문제

BOJ 9663
N-Queen

복잡한 Backtracking 문제를 풀 때는…

N-Queen 처럼 단순하게 Backtracking을 하면 시간초과가 나는 문제들은,
가지치기를 적절하게 하여 제한시간 안에 풀게 할 수 있습니다.

즉, 복잡한 Backtracking 문제들을 풀 때의 핵심은

주어진 **조건**과 **수학적 정리** 등을 통해 적절한 Bounding Function을 구현하는 것이라고 할 수 있습니다.

또한, 이러한 Bounding Function은 구현 방법에 따라 여전히 시간 초과가 날 수 있으므로, 한 번 구현한 Bounding Function을 계속 발전시키는 것이 중요합니다!

~~근데 실제 대회에서 이런 문제 나와서 잘 안 풀린다면 그냥 다른 문제 먼저 풀고 다시 봅시다~~

ALOHA
The algorithm club.

더 풀어볼 만한 연습 문제들

- 1987 알파벳
- 2661 좋은 수열
- 1799 비숍
- 2629 양팔저울

- 6716 Collecting Beepers
- 5520 The Clocks

감사합니다!

ALOHA
The algorithm club.