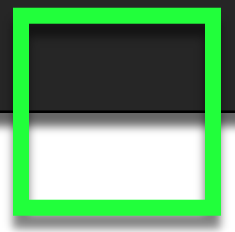
#알고리즘반
#6주차

Backtracking

T. 노주찬
Asst. 박정호, 우한샘



ALOHA
The algorithm club.

제약 충족 문제

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

제약 충족 문제

Example : Sudoku

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

모든 해 : 1부터 9까지의 숫자들.

최적해 : 조건을 충족하면서 남은 빈 칸을 채우는 숫자들.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

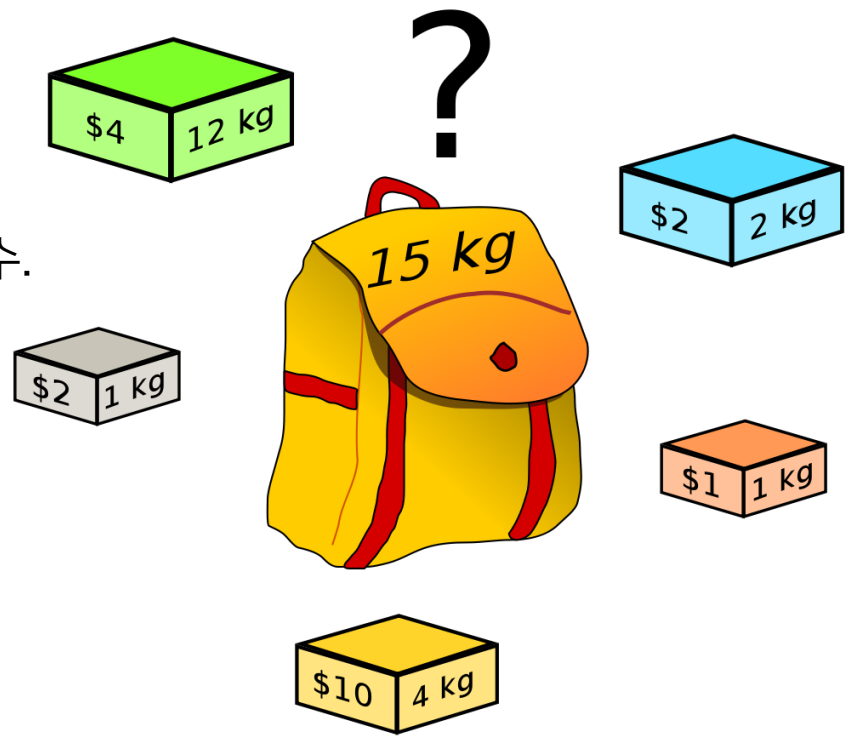
제약 충족 문제

Review Example : 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 문제

Q1. 제약 충족 문제를 해결하는데 최적화된 알고리즘이 존재하나요?

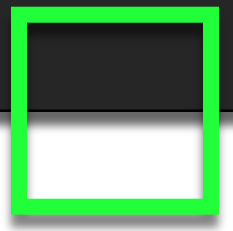
A1. 최적화된 알고리즘이 존재하는 문제도 있고, 그렇지 않은 문제도 있다.
(Review : 0/1 Knapsack은 $O(Nw)$ 시간복잡도에 해결하는 DP 알고리즘이 존재.)

Q2. 그렇다면 최적화된 알고리즘이 없는 문제는 어떻게 해결하나요?

A2. 다음 슬라이드에….

#1

Backtracking



Backtracking

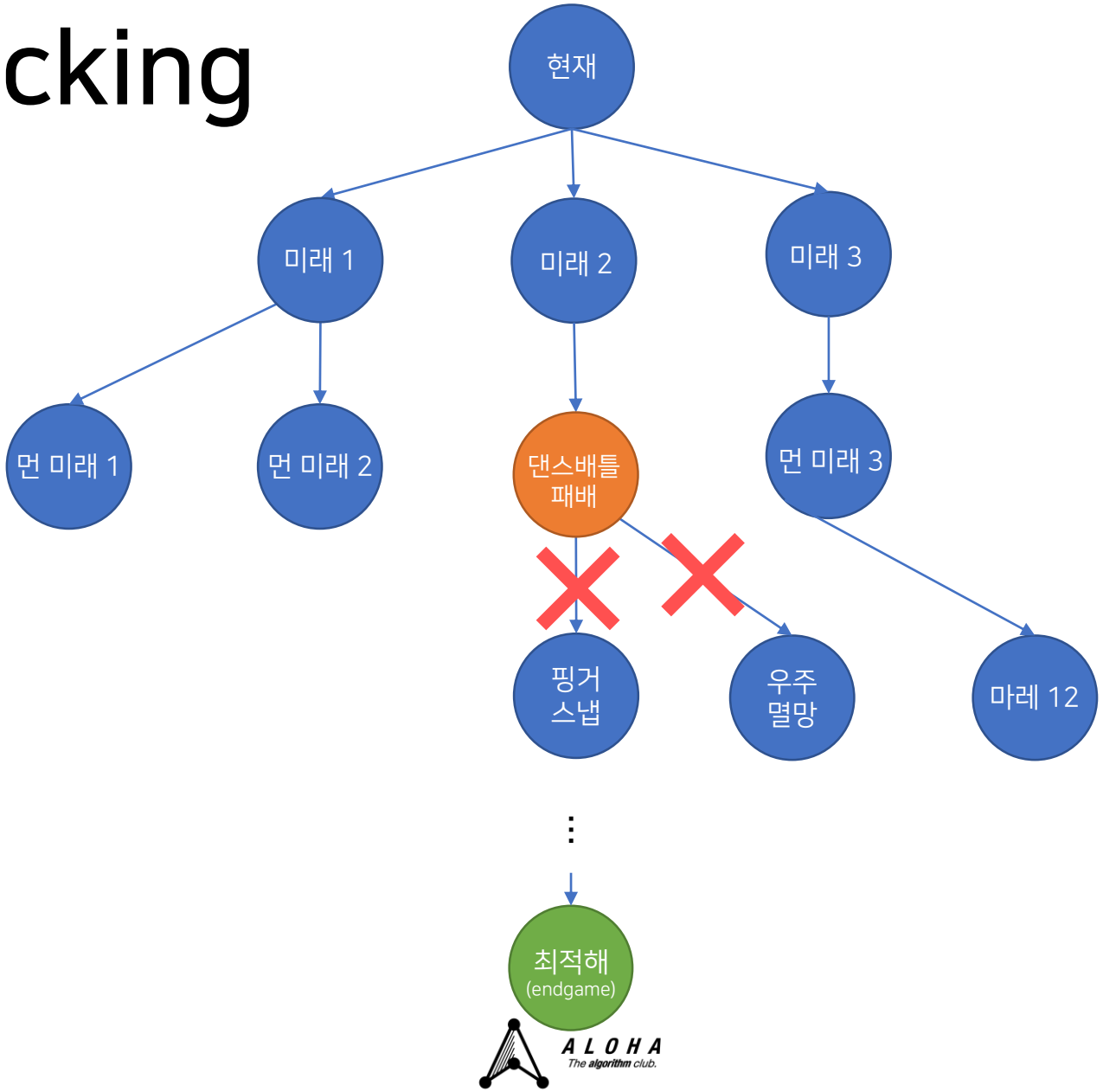
Backtracking (퇴각검색) 은 제약 충족 문제에서 최적해를 찾기 위한 전반적인 알고리즘이다.
최적해를 향해 계속해서 **가지를 뻗어나가고**, 그와 동시에 가능성이 없는 해는 **가지를 제거**하면서 최적해를 찾는다.

- https://en.wikipedia.org/wiki/Backtracking 번역

최적해를 찾는다는 건 알겠는데… 어떻게?

Backtracking

Backtracking



Backtracking

최적해가 될 지도 모르는 해의 집합 (후보 해) 을 **트리**로 나타내어, 이를 **순회**하면서 최적해를 찾는다!

그렇다면 해는 어떻게 표현하나요?

해의 집합은 어떻게 트리로 표현하나요?

순회는 어떻게 이뤄지나요?

해의 표현

일반적으로 제약 충족 문제에서는 해를 n-tuple 로 표현.

Sudoku 해의 표현: 각 빈 칸에 들어갈 숫자를 수열 x_i 에 대입하여 tuple 로!
$\{x_1, x_2, x_3, \ldots, x_n \mid 1 \leq x_i \leq 9\}$

0/1 Knapsack 해의 표현: 각 보석을 넣는다면 1, 넣지 않는다면 0으로 표현, 수열 x_i 에 대입하여 tuple 로!
$\{x_1, x_2, x_3, \ldots, x_n \mid x_i \in \{0, 1\}\}$

해의 집합의 트리 표현

해를 tuple 로 표현했으니 해의 집합을 트리로 표현해보자!

Depth가 0인 지점에서는 아직 아무것도 확정되어있지 않은 tuple을 루트 노드로 나타낸다.

Depth가 1인 지점에서는 루트 노드의 자식으로, x_1 까지 확정된 tuple을 노드로 나타낸다.

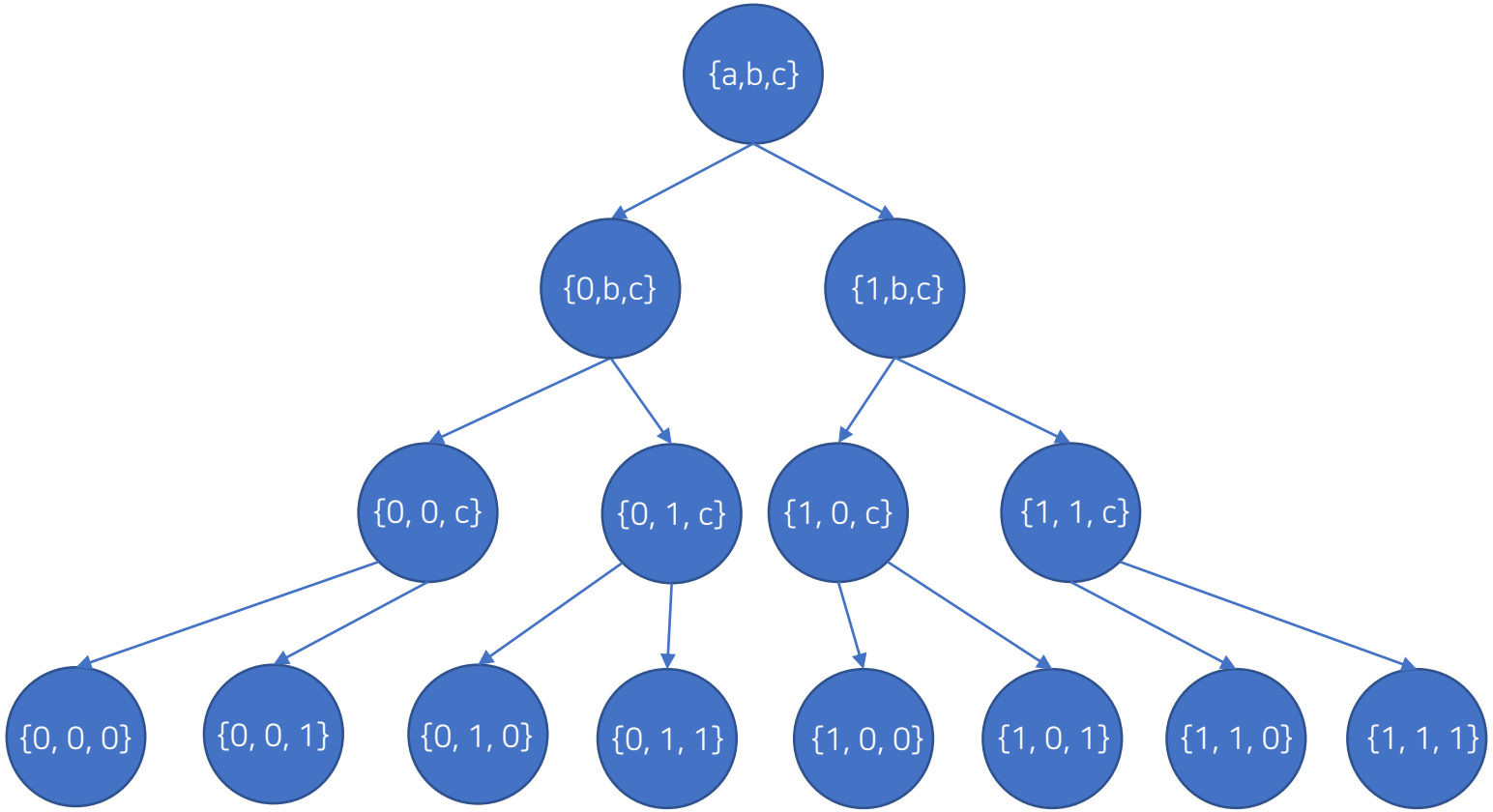
Depth가 2인 지점에서는 각 노드의 자식으로, x_2 까지 확정된 tuple을 노드로 나타낸다.

...

Depth가 n인 지점에서는 각 노드의 자식으로, x_n 까지 확정된 tuple을 노드로 나타낸다.
+ 이 해가 최적해인지 검사한다!

이러한 트리를 상태 공간 트리 (State Space Tree) 라고 한다.

해의 집합의 트리 표현



보석이 3개인 0/1 knapsack 문제의 상태 공간 트리.

트리의 순회

트리는 그래프의 일종.

Review: 그래프를 순회하는 방법에는 BFS, DFS 2가지 방법이 있었다.
BFS: 큐 방식을 사용, Level-order-traversal, 같은 레벨을 가지는 노드들의 개수만큼 메모리 필요
DFS: 스택 방식을 사용, Pre-order-traversal, 트리의 depth 만큼의 메모리 필요

BFS는 x_i의 범위가 2라고 해도, 해를 구성하는 숫자가 많아질수록 공간 복잡도가 빠르게 증가한다. $O(2^N)$
반면, DFS는 x_i의 범위에 상관 없이, 해를 구성하는 숫자가 많아져도 공간 복잡도가 느리게 증가한다. $O(\log_2 N)$

결론: Backtracking에서 트리를 순회하기 위해서는 DFS를 사용한다!

트리의 순회

DFS가 스택에 저장하는 메모리를 적게 사용한다고 해도
만약 후보 해를 모두 노드로 만들고, 그 사이의 간선을 저장한다고 하면 메모리를 많이 사용할 수 밖에 없다.
= **메모리 초과!**

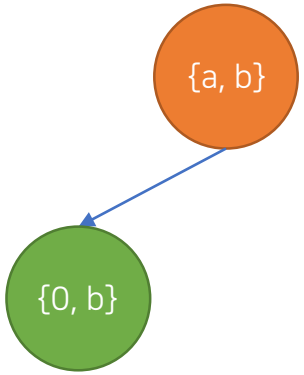
그러나 우리는 해의 범위를 알고 있기 때문에, 한 노드와 그 노드가 가지고 있는 해를 보면
그것을 통해서 그 자식 노드들을 만들어낼 수 있다.

따라서 노드와 간선의 정보는 저장하지 않고 만들면서 순회를 한다!

해의 집합의 트리 표현

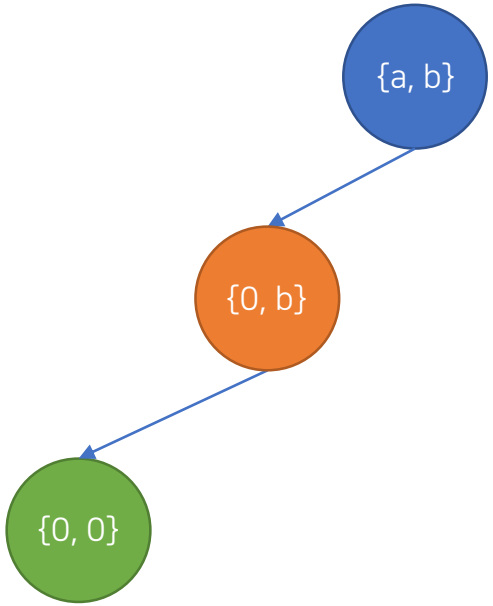
보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

해의 집합의 트리 표현



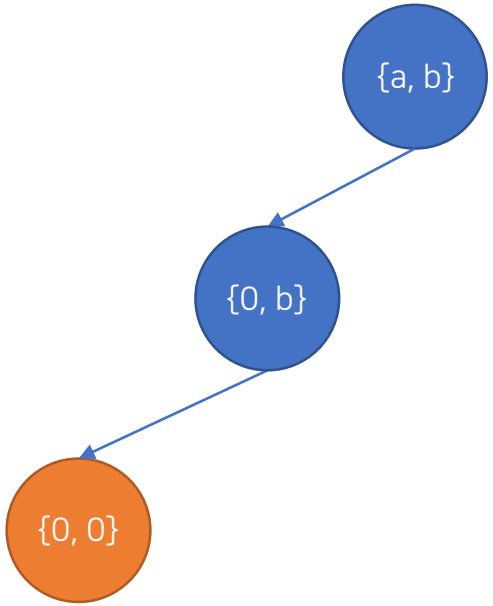
보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

해의 집합의 트리 표현



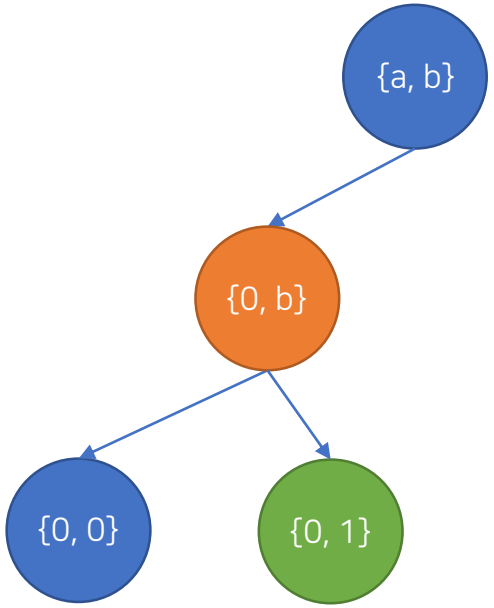
보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

해의 집합의 트리 표현



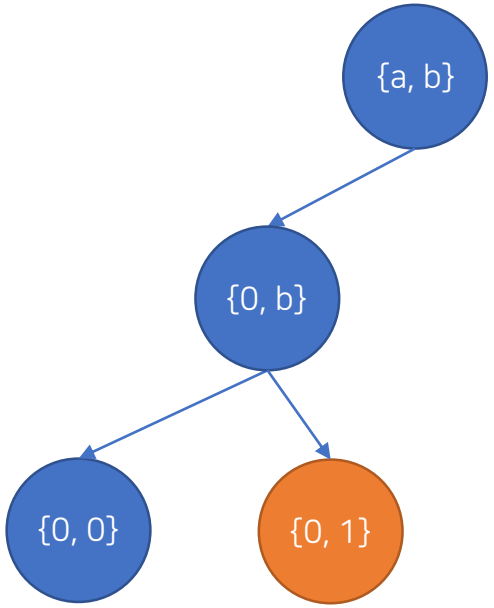
보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

해의 집합의 트리 표현



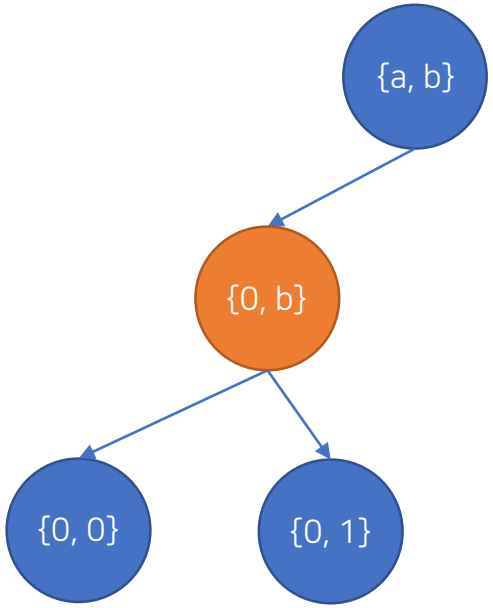
보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

해의 집합의 트리 표현



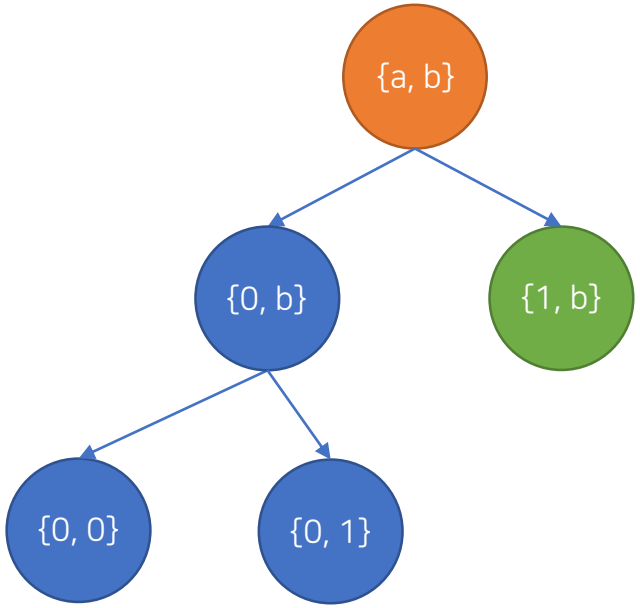
보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

해의 집합의 트리 표현



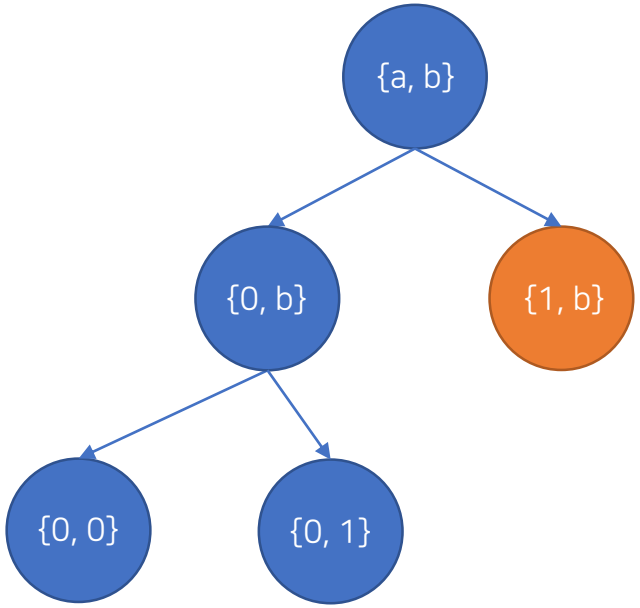
보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

해의 집합의 트리 표현



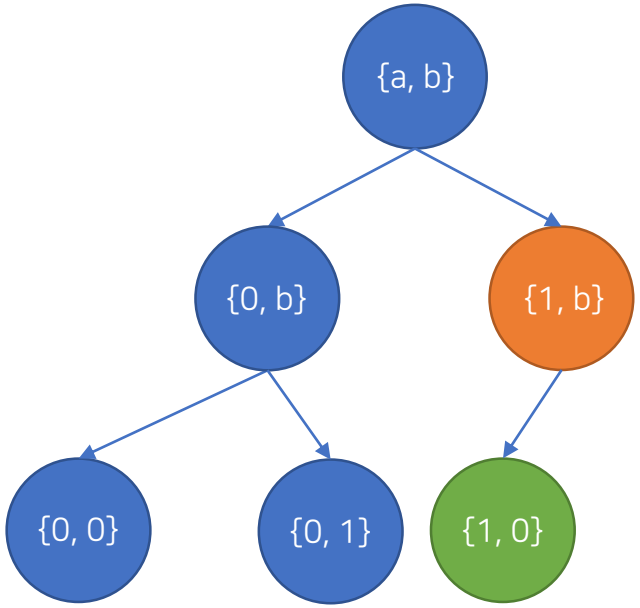
보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

해의 집합의 트리 표현



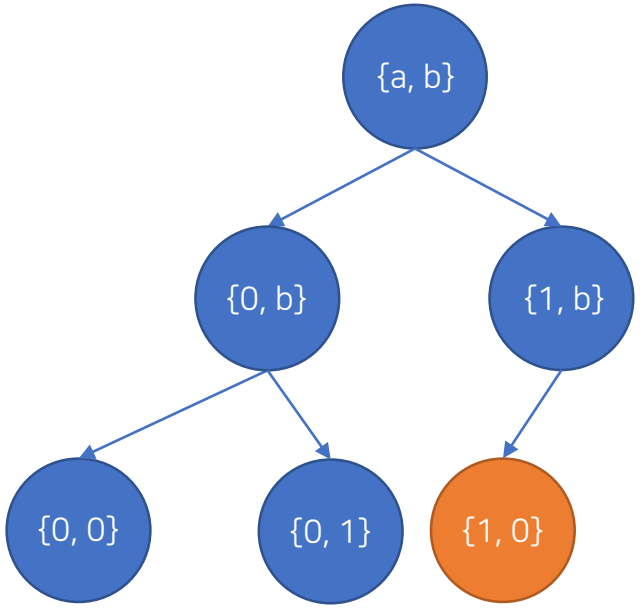
보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

해의 집합의 트리 표현



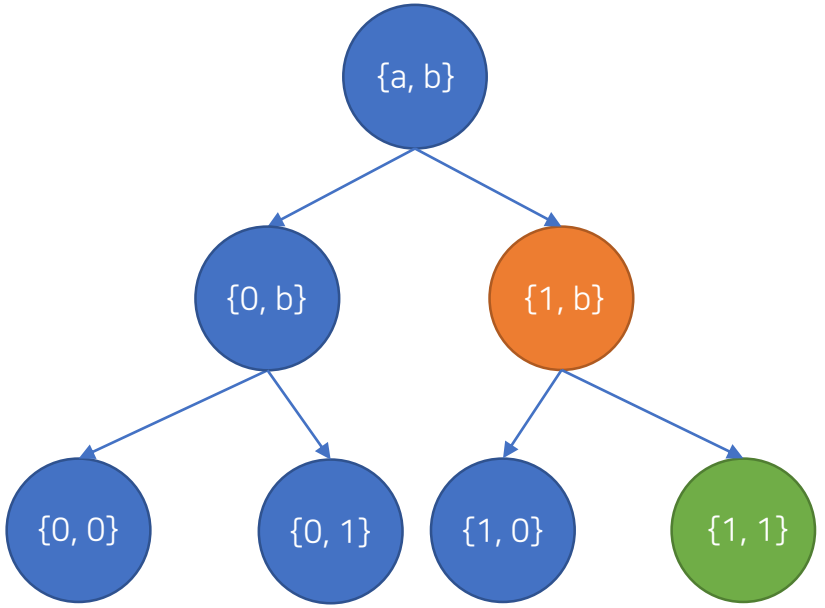
보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

해의 집합의 트리 표현



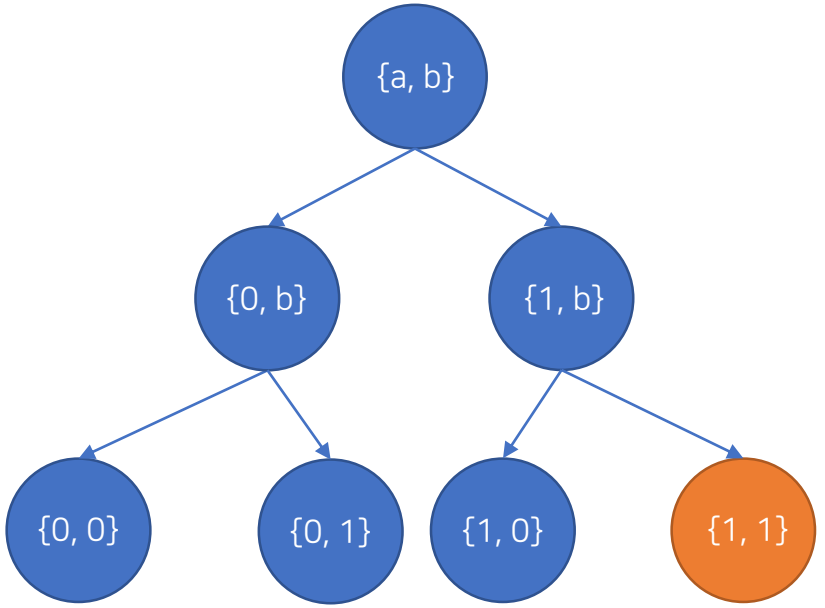
보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

해의 집합의 트리 표현



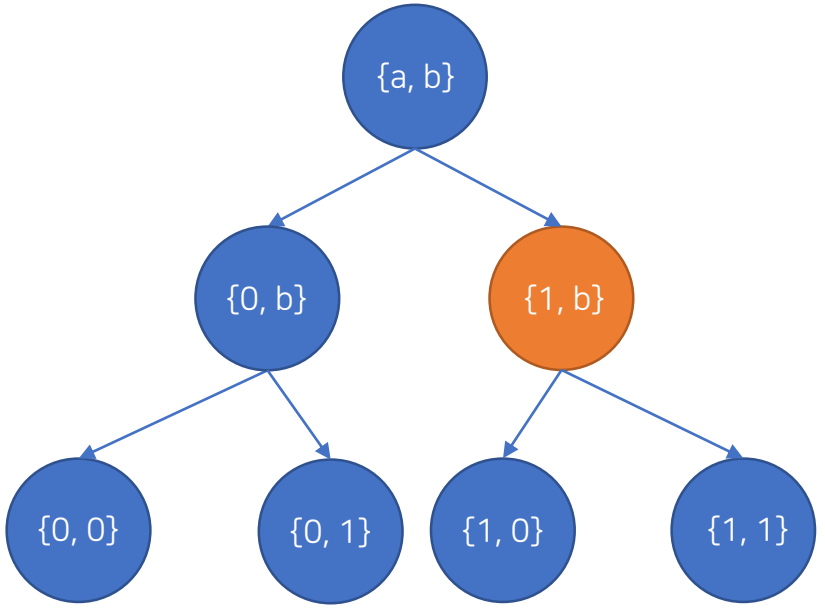
보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

해의 집합의 트리 표현



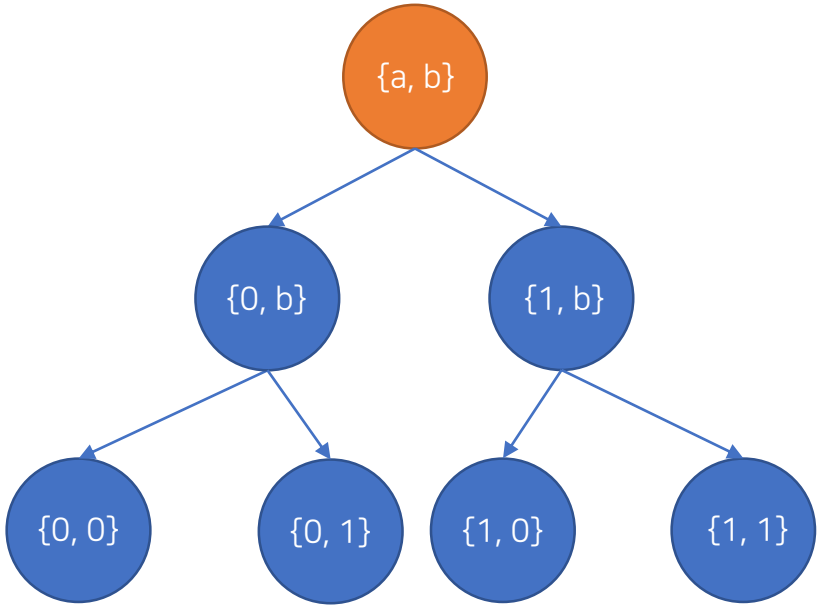
보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

해의 집합의 트리 표현



보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

해의 집합의 트리 표현



보석이 2개인 0/1 knapsack 문제의 상태 공간 트리.

트리의 순회

```
1  int solution[5];
2
3  //Start by calling Backtracking(0);
4  int Backtracking(int depth){
5      if(detph == 5) //해를 완성했을 때
6          if(isOK()) //최적해인지 검사
7              doSomething();
8
9      for(int i=0;i<=1;i++){ //해가 아직 완성되지 않았을 때
10         //해를 구성하는 숫자 중 depth번째 숫자를 i로 설정
11         solution[depth] = i;
12         //depth번째 숫자를 설정하고 자식 노드로 이동
13         Backtracking(depth + 1);
14     }
15 }
```

연습 문제

BOJ 1759
암호 만들기

연습 문제

Point 1. 가능성 있는 암호를 증가하는 순서 (LIS) 로 만들어야 한다.

⇒ 암호에 포함될 수 있는 문자를 사전순으로 정렬한다.

Point 2. 만든 암호를 사전 순으로 출력해야한다.

⇒ 우리는 암호에 사용할 문자를 정렬했다.
⇒ 다시 말해 a를 사용할지 말지 결정하는 depth는, b를 사용할지 말지 결정하는 depth 보다 위에 있다.
⇒ 따라서 포함될 수 있는 문자 중에서 앞에 있는 것을 사용하는 경우를 먼저 순회하고
⇒ 그 다음에 사용하지 않는 경우를 순회해야 사전 순으로 최적해를 만들 수 있다!

암호 만들기

```
int used[20];

int Backtracking(int depth){
    if(detph == C){ //해를 완성했을 때
        if(isOK()) //최적해인지 검사
            doSomething();
        return 0;
    }

    for(int i=1;i>=0;i--){ //해가 아직 완성되지 않았을 때
        //해를 구성하는 숫자 중 depth번째 숫자를 i로 설정
        used[depth] = i;
        //depth번째 숫자를 설정하고 자식 노드로 이동
        Backtracking(depth + 1);
    }
}
```

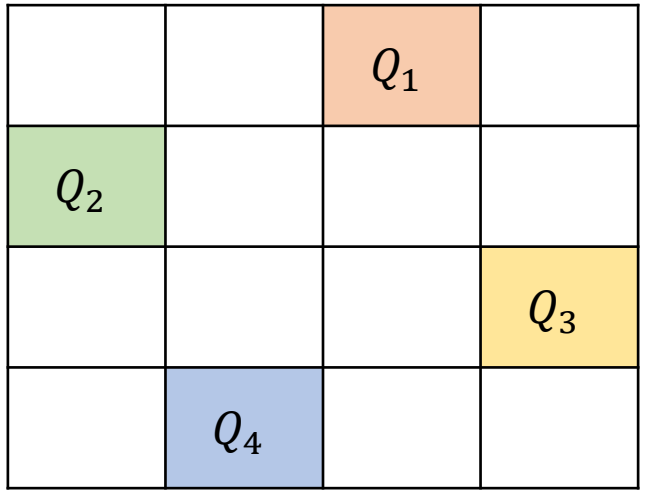
암호 만들기

```
int isOK(){
    int mo = 0, ja = 0, cnt = 0;

    for(int i=0;i<m;i++){
        cnt += used[i];
        if(c[i] == 'a' || c[i] == 'e' || c[i] == 'i' || c[i] == 'o' || c[i] == 'u')
            mo += used[i];
        else if(used[i])
            ja += used[i];
    }

    if(cnt != n) return 0;
    if(mo < 1 || ja < 2) return 0;

    return 1;
}
```

n-Queen Problem

n×n 크기의 체스판 위에서, n개의 여왕을 서로 잡을 수 없도록 놓는 경우의 수는?

조건 1. 같은 행에는 1개의 여왕만이 존재해야한다.
조건 2. 같은 열에는 1개의 여왕만이 존재해야한다.
조건 3. 같은 대각선에는 1개의 여왕만이 존재해야한다.

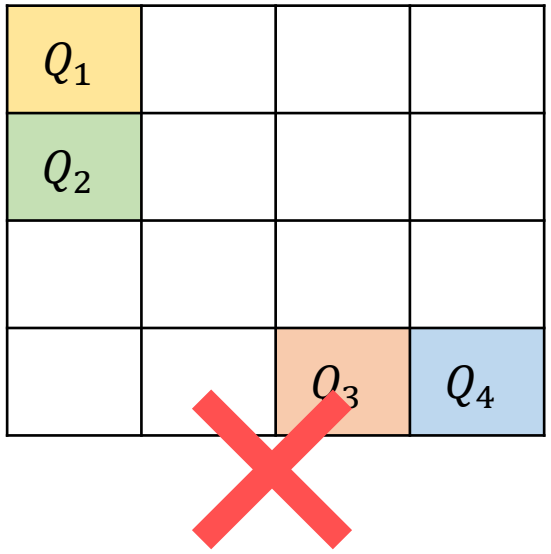


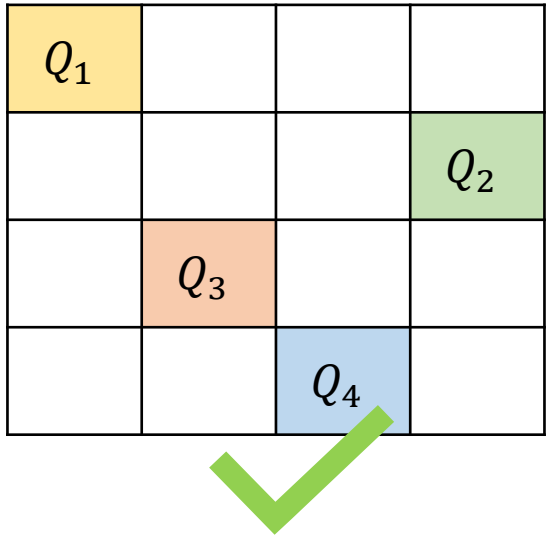
4-Queen Problem

빠른 예제 설명을 위해서 최적해가 있는 가장 작은 경우인 n = 4를 가정하고 풀어보자.
~~n=2, 3인 경우는 왜 안되는지는 직접 생각해보자~~

조건 1 또는 조건 2에 따라서 1개의 가로줄 (또는 세로줄) 에는 1개의 여왕만이 올 수 있다.

따라서 우리는 4×4의 모든 칸에 대해 여왕을 놓는 경우를 계산할 필요가 없고,
각 가로줄 (또는 세로줄) 에 대해서 어느 x좌표 (또는 y좌표) 에 여왕이 올 지를 정하면 된다!





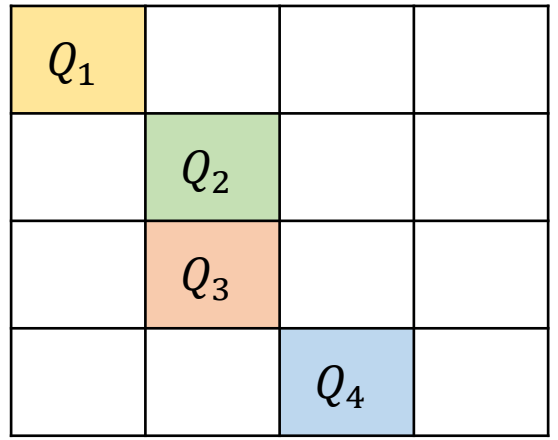
ALOHA
The algorithm club.

4-Queen Problem

그렇다면 후보 해를 좌표를 담고 있는 4-tuple 로 표현할 수 있다.
후보 해 : $\{ x_1, x_2, x_3, x_4 \mid 1 \leq x_i \leq 4\}$

탐색하는 코드는 예제 코드로 주어졌으니, 이제 최적해임을 판단하는 방법을 세워야한다.
이미 후보 해를 세우는 방법을 통해 조건 1을 만족시켰으니, 조건 2와 조건 3을 만족하는 지 확인해야한다.

x좌표가 같으면 같은 세로선 위에 존재하는 것이기 때문에 조건 2에 위배된다.
따라서 후보 해를 구성하는 숫자 중에서 중복된 숫자가 있는지 없는지 확인하면 된다.



$\{1, 2, 2, 3\}$ 에 해당하는 체스판의 모습

4-Queen Problem

대각선을 판별하는 방법은 의외로 간단하다!
바로 x좌표와 y좌표의 합과 차를 이용하는 것.

(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)

좌측 상단 -> 우측 하단 대각선을 칠한 모습.
x좌표, y좌표 차가 일정.

(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)

좌측 하단 -> 우측 상단 대각선을 칠한 모습.
x좌표, y좌표 합이 일정.

각각 대각선에 대해 여왕이 1개만 있는지 확인하면 오케이.

트리의 순회

```
int solution[5];

//Start by calling Backtracking(0);
int Backtracking(int depth){
    if(depth == 4){ //해를 완성했을 때
        if(isOK()) //최적해인지 검사
            doSomething();
        return 0;
    }

    for(int i=1;i<=4;i++){ //해가 아직 완성되지 않았을 때
        //해를 구성하는 숫자 중 depth번째 숫자를 i로 설정
        solution[depth] = i;
        //depth번째 숫자를 설정하고 자식 노드로 이동
        Backtracking(depth + 1);
    }
}
```

트리의 순회

```
bool sero[5], down[10], up[10]; //down, up의 크기는 n의 2배 이상!
int isOK(){
    for(int i=0;i<4;i++){
        if(sero[solution[i]]) return 0;
        sero[solution[i]] = 1;

        //i - solution[i] 가 -n 까지 내려갈 수 있으니 n을 더해준다.
        if(down[4 + i - solution[i]]) return 0;
        down[4 + i - solution[i]] = 1;

        if(up[i + solution[i]]) return 0;
        up[i + solution[i]] = 1;
    }
    return 1;
}
```

연습 문제

BOJ 9663
N-Queen

연습 문제

결과
시간 초과

연습 문제

연습 문제

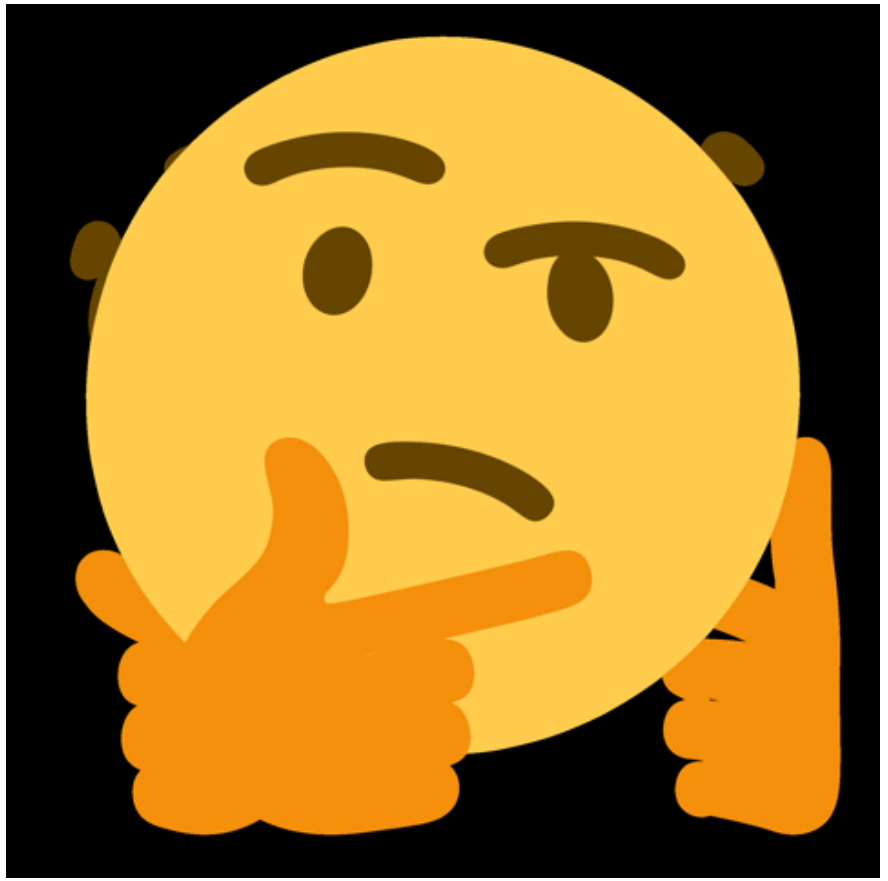
자세히 읽어보니 문제의 N 제한은 14이다.

그런데 14개의 x_i 에 대해 1부터 14까지의 숫자를 대입하는 경우의 수는?

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

1경 1,112조 68억 2,555만 8,016 가지.
PS에서 일반적으로 1초에 처리할 수 있는 1억번 가량의 반복을 처리할 수 있다고 보니까
1경 번의 반복을 돌기 위해서는 무려 1157일이 필요하다!

반복횟수를 줄일 방법이 없을까?



Bounding Function

Bounding Function, 한정함수:
노드를 생성할 때, 이 노드 아래의 **최적해의 존재여부를 판단**해주는 함수.

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

즉, 해를 중간까지 생성했을 때, 이것으로 최적해를 만들 수 있는가를 판별하여
가능성이 있는 경우 계속하여 순회를 진행
그렇지 않은 경우 그곳에서 더 깊게 파고 드는 것을 중단하고 **이전 노드로 되돌아간다** (Backtrack).

Bounding Function

x좌표가 같으면 같은 세로선 위에 존재하는 것이기 때문에 조건 2에 위배된다.
따라서 후보 해를 구성하는 숫자 중에서 중복된 숫자가 있는지 없는지 확인하면 된다.

⇒ 노드를 생성할 때, 숫자를 중복해서 사용하지 않도록 한다.

경우의 수가 14! 으로 감소.
14! = 87,178,291,200 = 871억 7829만 1200번

15분 가량으로 대폭 줄었음을 확인할 수 있다!

Bounding Function

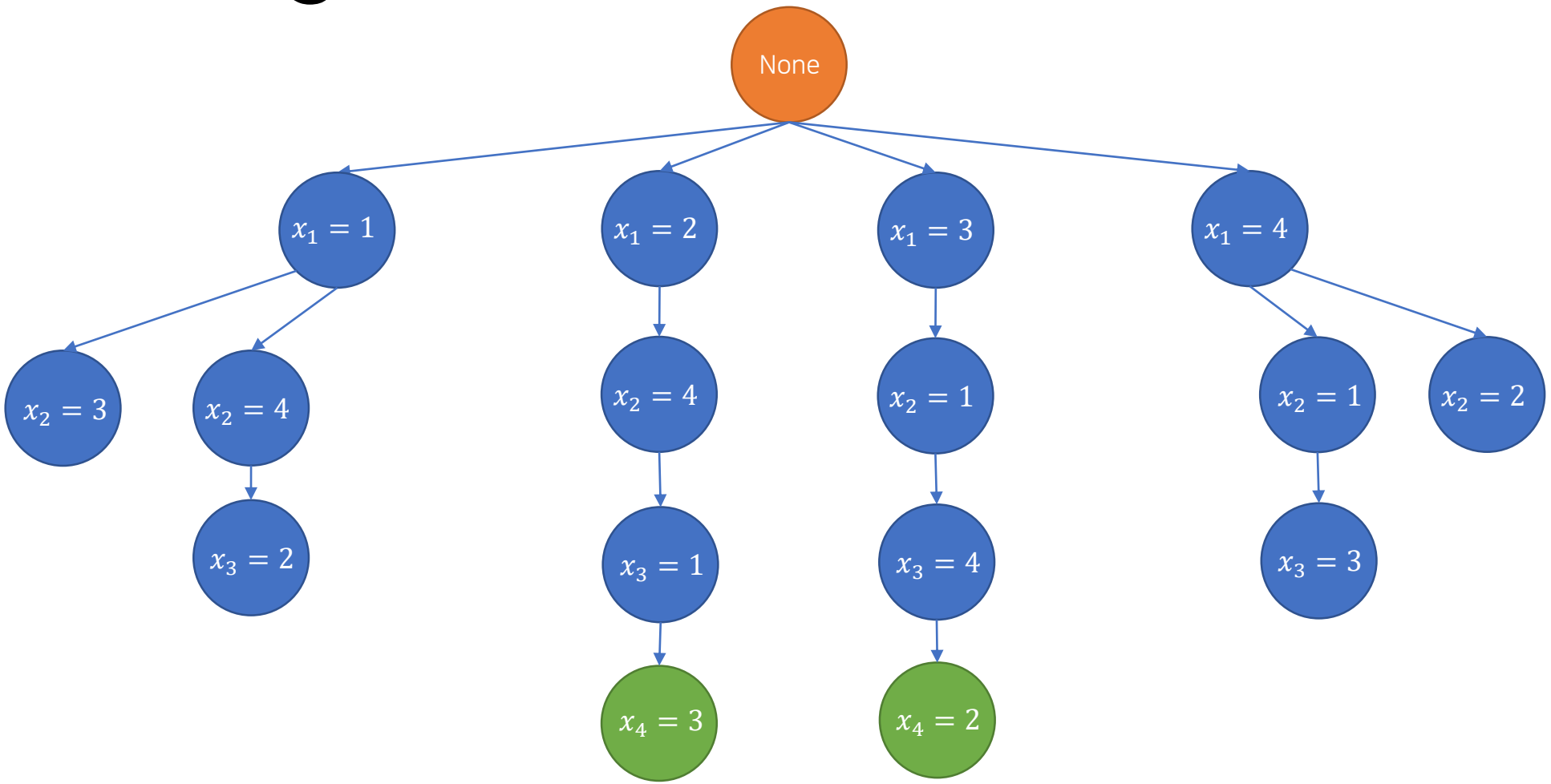
각각 대각선에 대해 여왕이 1개만 있는지 확인하면 오케이.

⇒ 노드를 생성할 때, 대각선 위에 여왕이 1개만 있는지 검사한다.

단, 검사를 $O(N)$ 으로 하게 되면 오히려 줄인 반복 횟수보다 검사를 하는 반복횟수가 많아진다!

따라서 노드를 생성할 때마다 여왕의 위치를 기록하고
순회를 한 다음에 다시 지우는 방식으로 구현하면 $O(1)$로 검사할 수 있다.

Bounding Function



노드 개수가 $\sum_{i=0}^{4} 4^i = 341$ 에서 17개로 대폭 감소했다!

ALOHA
The algorithm club.

4-Queen Problem

```
int Backtracking(int depth){
    if(depth == 4){ //해를 완성했을 때
        doSomething(); //Boundng Function이 isOK를 대신하므로 필요 X
        return 0;
    }

    for(int i=1;i<=4;i++){ //해가 아직 완성되지 않았을 때
        //Bounding Function
        if(sero[i] | down[4 + depth - i] | up[depth + i]) continue;

        solution[depth] = i;
        sero[i] = down[4 + depth - i] = up[depth + i] = true;
        //depth번째 숫자를 설정하고 자식 노드로 이동
        Backtracking(depth + 1);
        sero[i] = down[4 + depth - i] = up[depth + i] = false;
    }
}
```

연습 문제

BOJ 9663
N-Queen

Tip

단순하게 Backtracking 을 돌리면 시간초과가 나는 문제도
적절한 가지치기를 거치면 시간 제한 안에 풀리는 문제도 존재한다!

다시 말하면 어느정도 난이도 있는 Backtracking 문제의 핵심은 Bounding Function 에 있다.
주어지는 조건과 수학적 정리 등을 통해서 적절한 Bounding Function 을 구현할 수 있도록 하자.

또한 Bounding Function 을 통해서 줄여지는 순회 횟수는 수학적으로 쉽게 계산할 수도 있고, 아닐 수도 있다.
따라서 시간 초과가 났다고 하더라도 계속해서 Bounding Function 을 발전시키는 것이 중요하다.

~~물론 실제 대회에선 잘 안 풀리면 다른 문제 먼저 풀고난 후에 다시 보는게 페널티 관리와 정신 건강에 도움이 됩니다.~~

연습 문제

BOJ 1987
알파벳

연습 문제

BOJ 2661
좋은 수열

연습 문제

BOJ 1799
비숍

연습 문제

BOJ 2629
양팔저울

연습 문제

BOJ 6716
Collecting Beepers

Clarification.

Beeper의 최대 개수는 10개
월드의 최대 크기는 20
x,y 좌표의 범위는 1 ~ 월드의 크기

ALOHA
The algorithm club.

연습 문제

BOJ 5520
The Clocks

꿋