알튜비튜
백트래킹

오늘은 코딩테스트에 많이 나오는 알고리즘 중 하나인 백트래킹에 대해 배웁니다.
전번에 배운 완전탐색(브루트포스)에서 조금 더 발전해서, 더 이상 가망 없는 후보를 제외하고 탐색하는 알고리즘이죠.

소개

백트래킹

- 완전탐색처럼 모든 경우를 탐색하나, 중간 과정에서 조건에 맞지 않는 케이스를 가지치기하여 탐색 시간을 줄이는 기법
- 모든 경우의 수를 탐색하지 않기 때문에 완전탐색보다 시간적으로 효율적임
- 탐색 중 조건에 맞지 않는 경우 이전 과정으로 돌아가야 하기 때문에, 재귀를 사용하는 경우 많음
- 조건을 어떻게 설정하고, 틀렸을 시 어떤 시점으로 돌아가야 할지 설계를 잘 하는 것이 중요

백트래킹

과정

- 어떤 노드의 유망성(promising)을 점검
  → 조건에 맞는지 안맞는지
  → 답이 될 수 있는지 없는지
- 유망하지 않다면(non-promising) 배제함 (가지치기)

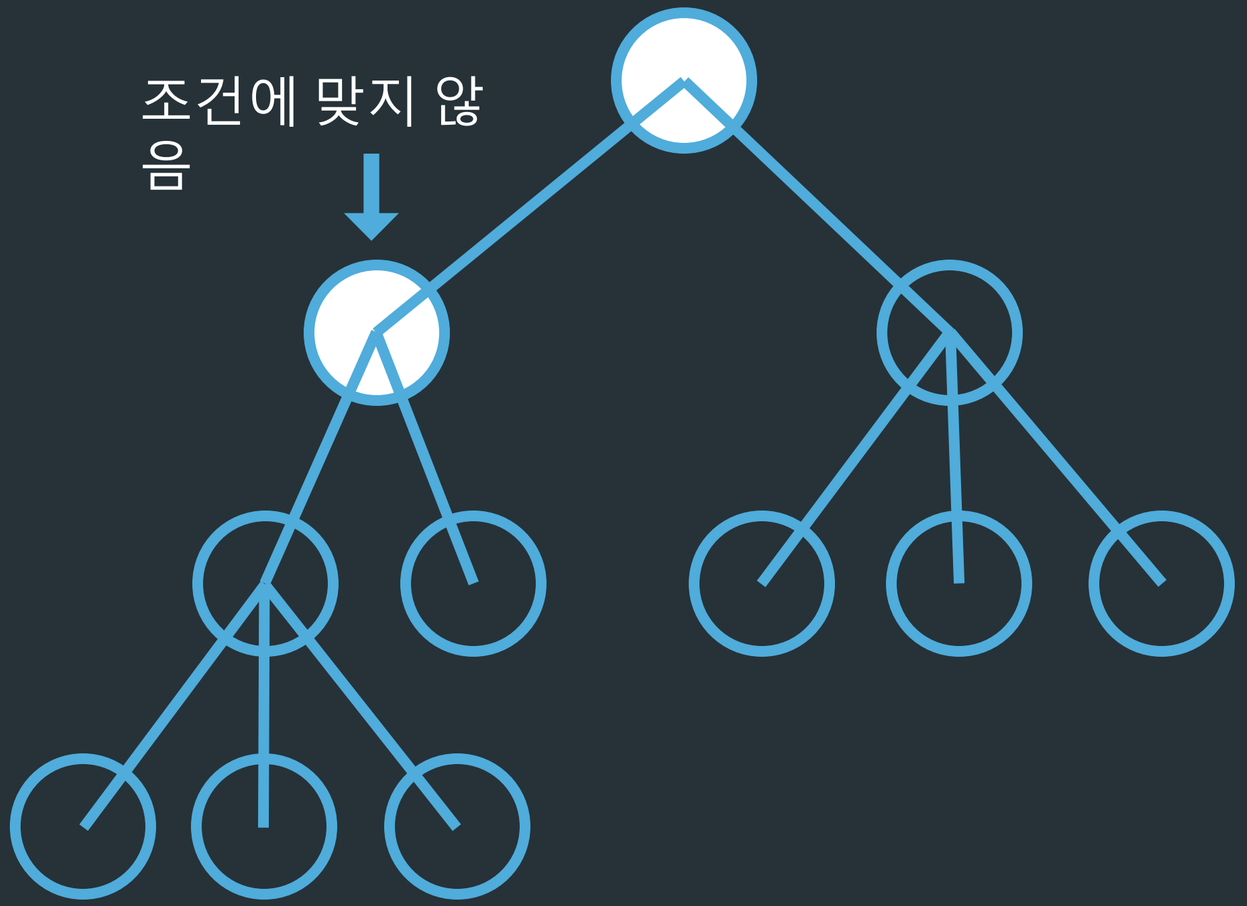
가지치기란?

#알튜비튜

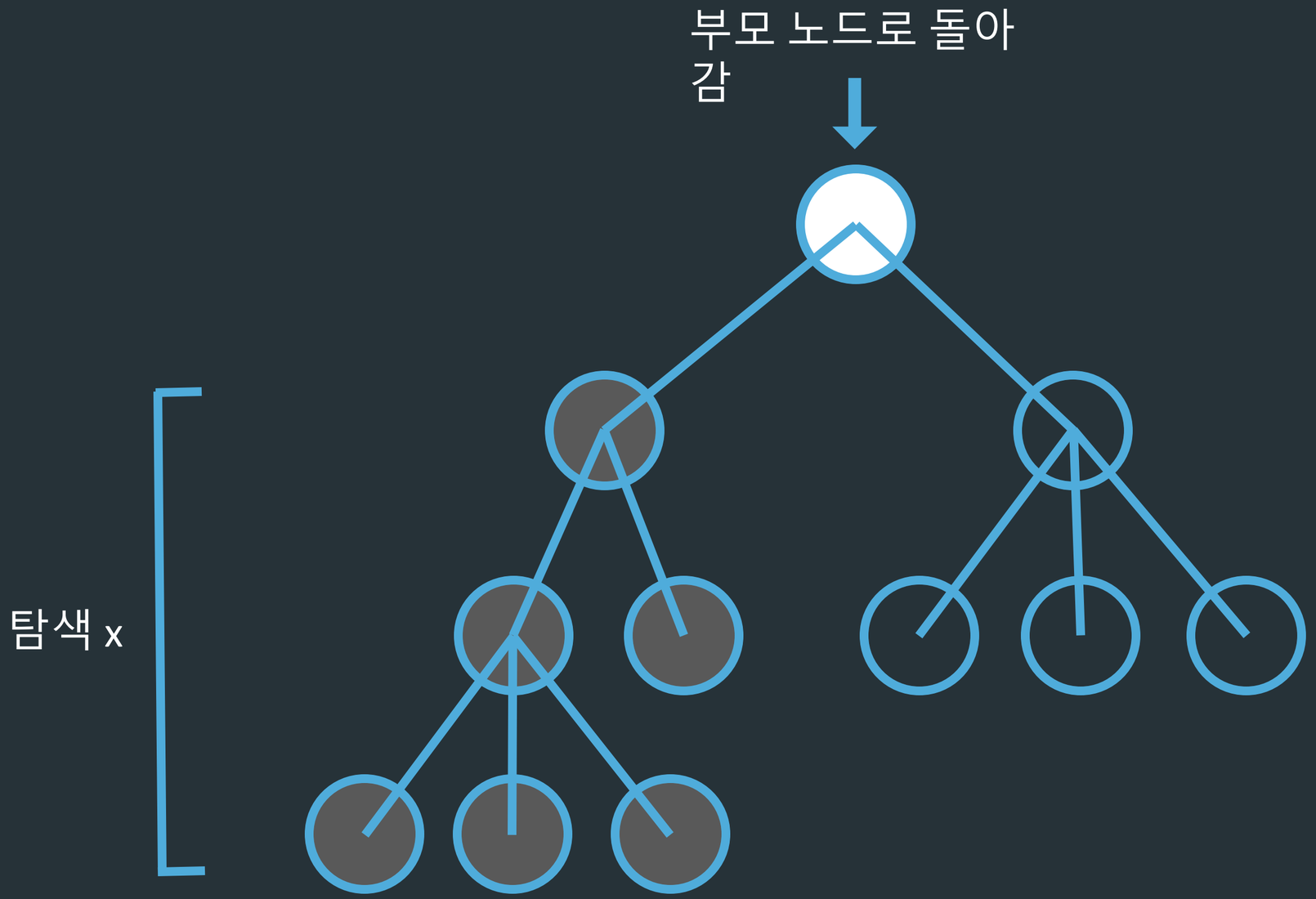
가지치기

- 지금의 경로가 해가 될 것 같지 않으면(non-promising)
  그 전으로 되돌아 가는 것(back)
- 즉, 불필요한 부분을 쳐내는 것
- 되돌아간 후 다시 다른 경로 검사
- 가지치기를 얼마나 잘하느냐에 따라 효율성이 결정됨

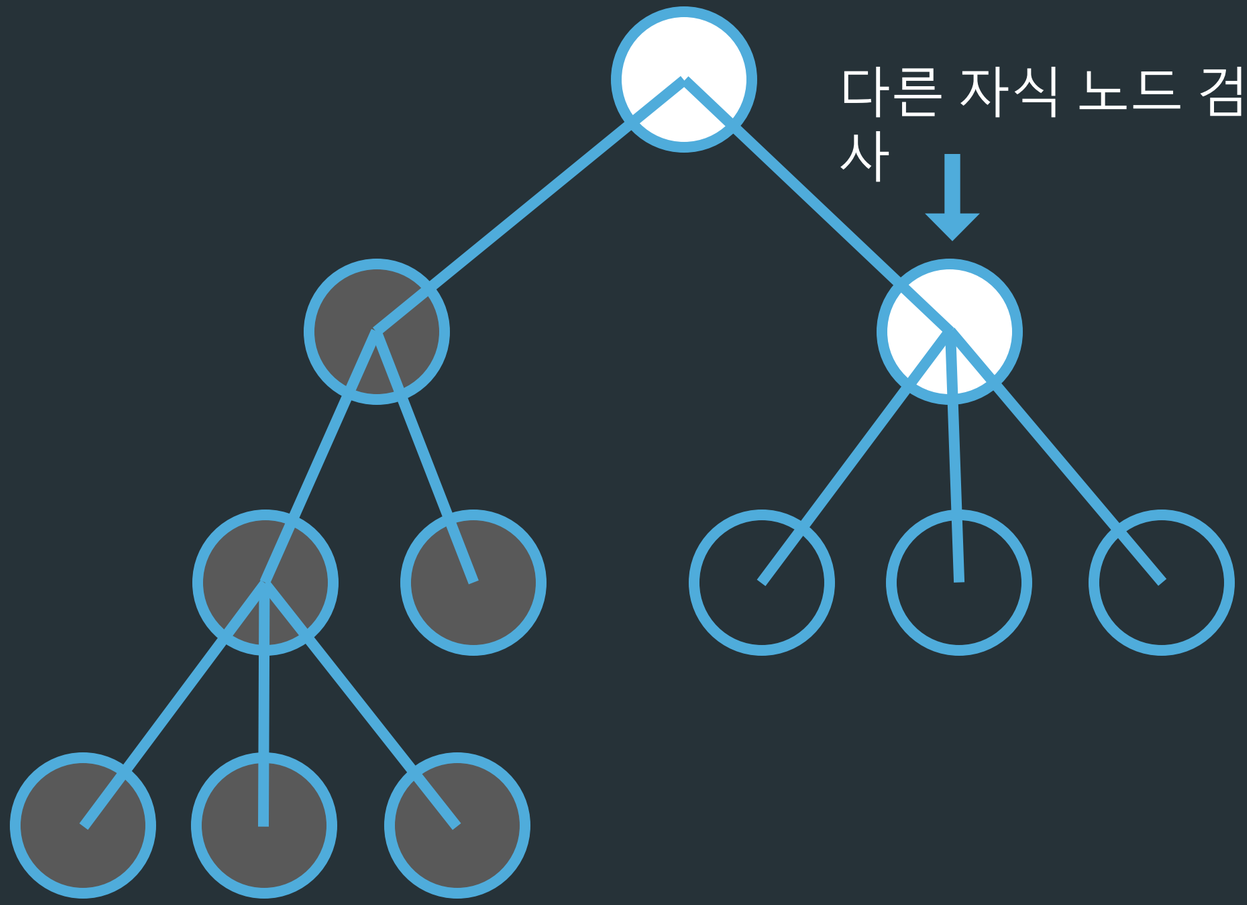
가지치기 예시



가지치기 예시



가지치기 예시



백트래킹

“즉, 답이 될 만한지(promising) 판단하고, 그렇지 않다면 탐색하지 않고 다시 전으로 넘어가서(back) 탐색을 계속 하는 것 "

재귀함수란?

```
void f(int mine){
  f(mine + 1);
}
```

← 문제점은?

- 자기 자신을 호출하는 함수

재귀함수란?

#알튜비튜

```
void f(int mine){
  if (mine == 5)
    return;
  f(mine + 1);
  cout << mine << '\n';
}
```

기저 조건

- 자기 자신을 호출하는 함수
- 가장 중요한 점은 기저 조건(탈출 조건)을 잘 세우는 것!

어떨 때 백트래킹을 적용하지?

#알튜비튜

백트래킹

- N의 크기가 작을 때 (주로 재귀함수로 구현하기 때문)
  → 보통 20 이하
- 그 전 과정으로 돌아가면서 하는 탐색이 필요한 경우

15649번 : N과 M(1) – Silver 3

문제

- 자연수 n, m이 주어짐
- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제

제한 사항

- 입력 범위는 1 <= m <= n <= 8

예제

#알튜비튜

예제 입력

```
4 2
```

예제 출력

```
1 2
1 3
1 4
2 1
2 3
2 4
3 1
3 2
3 4
4 1
4 2
4 3
```

#알튜비튜

15649번 : N과 M(1) – Silver 3

문제

- 자연수 n, m이 주어짐
- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제

제한 사항

- 입력 범위는 1 <= m <= n <= 8

접근

- 제약 조건을 살펴보자
  → 중복 x, 수열 길이가 m

- 재귀함수를 설계해보자
  → 각 수를 넣을 때, 이미 수열 내에 있으면 넘어감 (체크해줄 무언가가 필요)
  → 기저조건은 길이가 m일 때!

체크해줄 무언가?

#알튜비튜

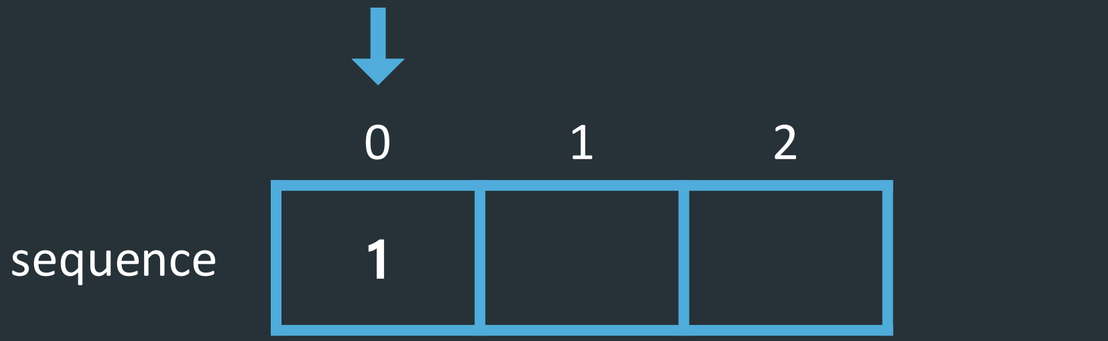
접근

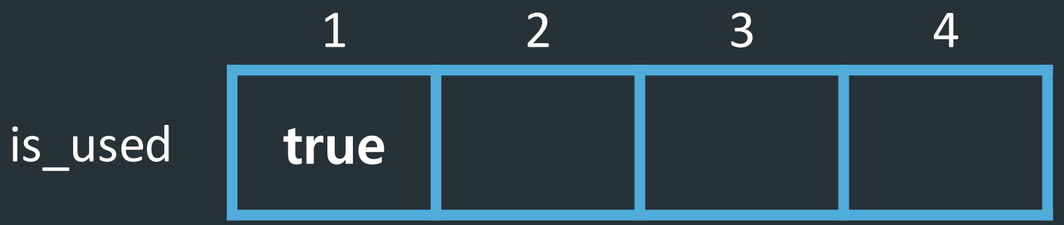
- 해당 수가 현재 수열에서 사용이 되었는지 안되었는지 체크하는 배열을 만들자!
- 즉, 수를 인덱스로 가지는 체크배열
- 가지치기의 판단 기준이 됨

N과 M(1)

#알튜비튜

- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- n = 4, m = 2

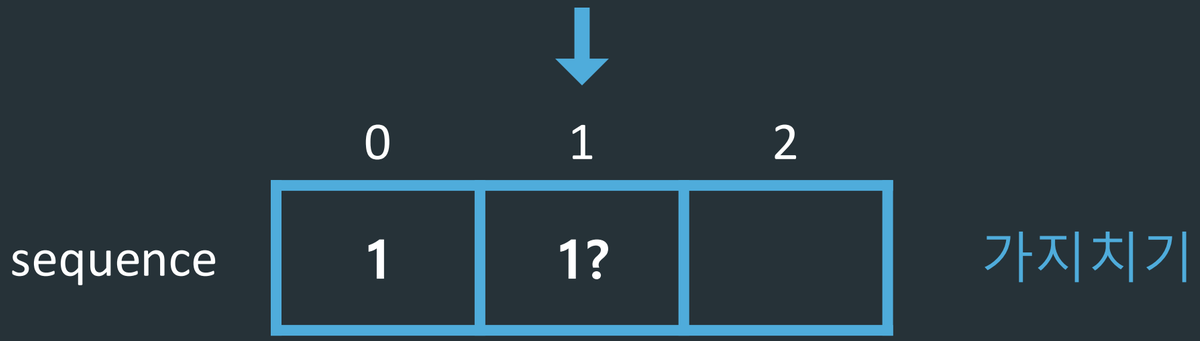


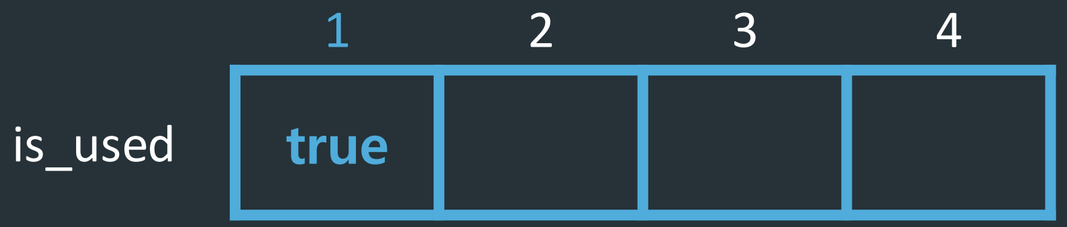


N과 M(1)

#알튜비튜

- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- n = 4, m = 2

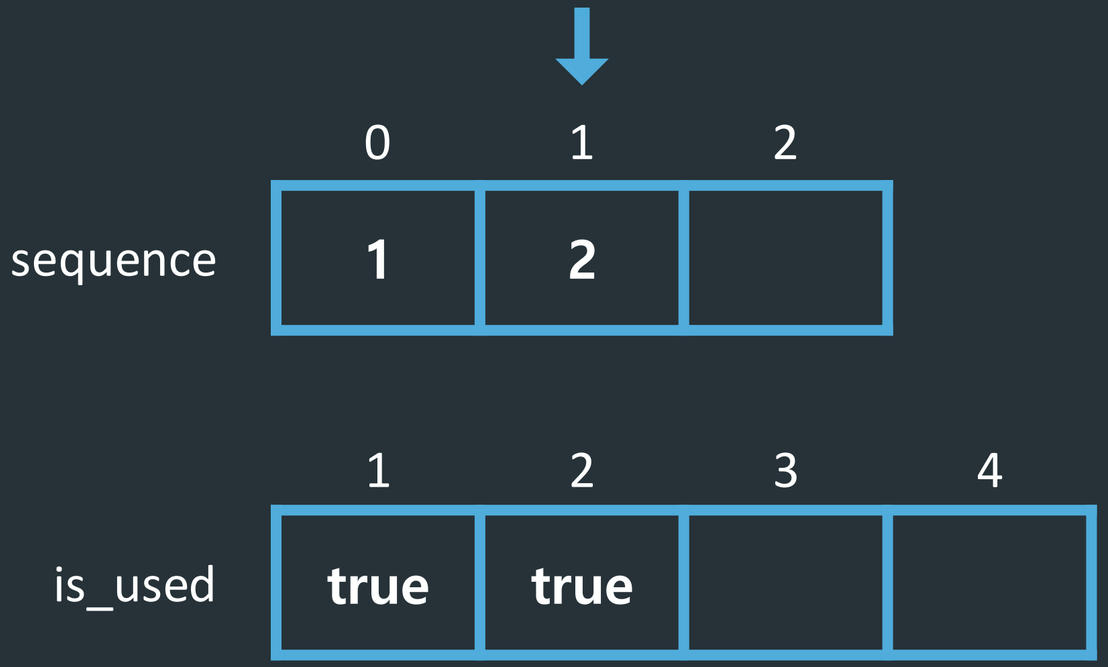




N과 M(1)

#알튜비튜

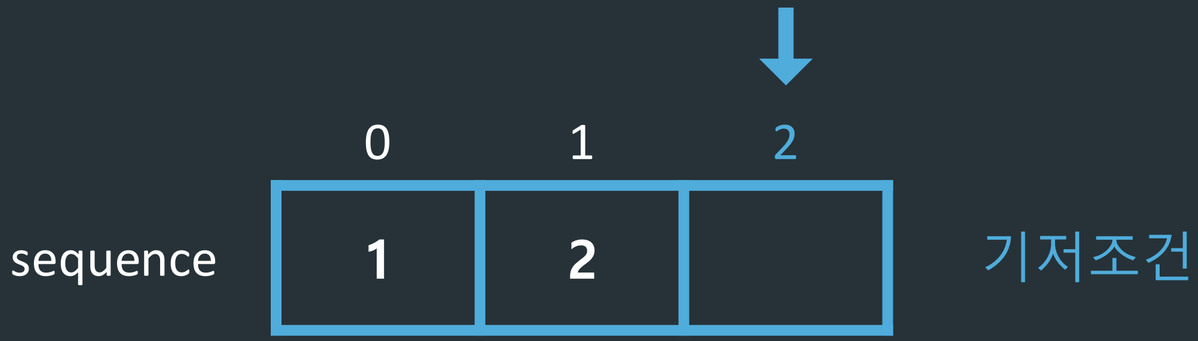
- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- n = 4, m = 2



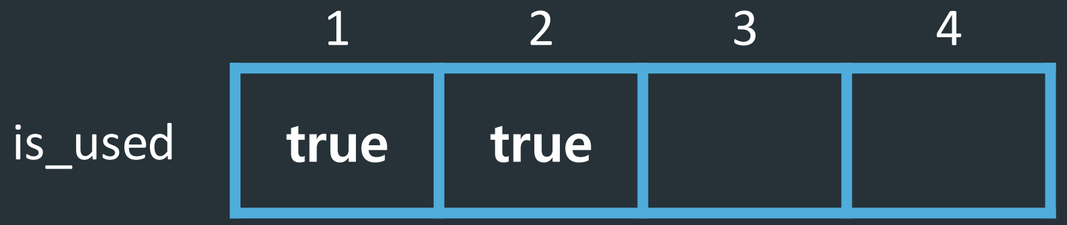
N과 M(1)

#알튜비튜

- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- n = 4, m = 2



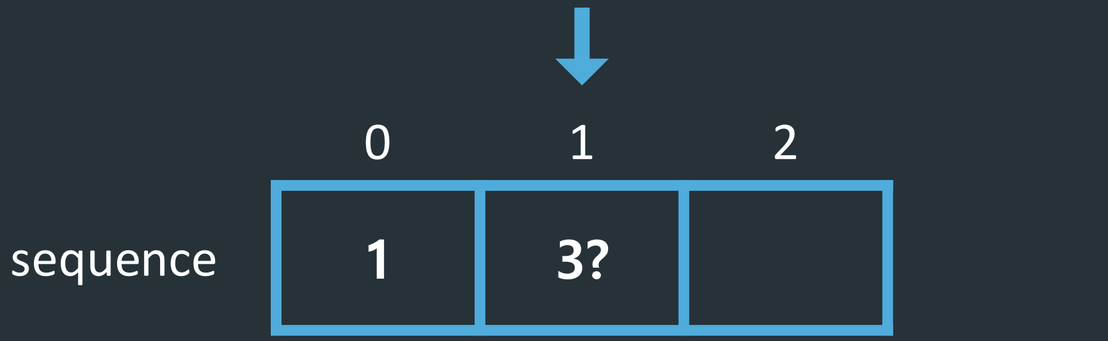
기저조건

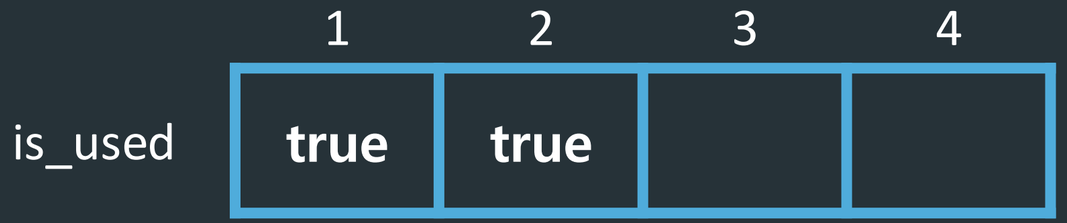


N과 M(1)

#알튜비튜

- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- n = 4, m = 2

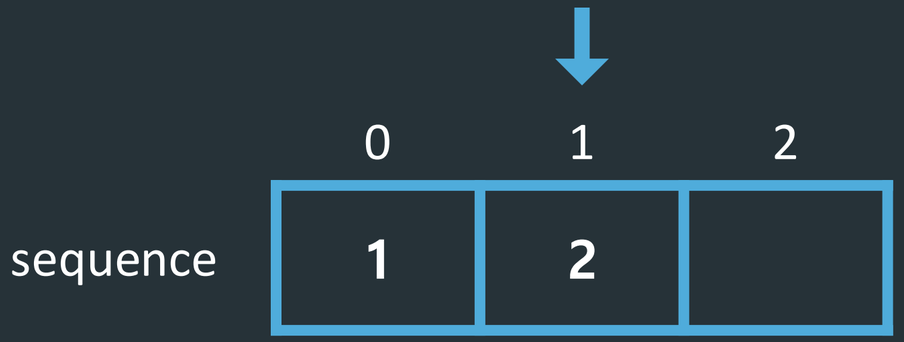


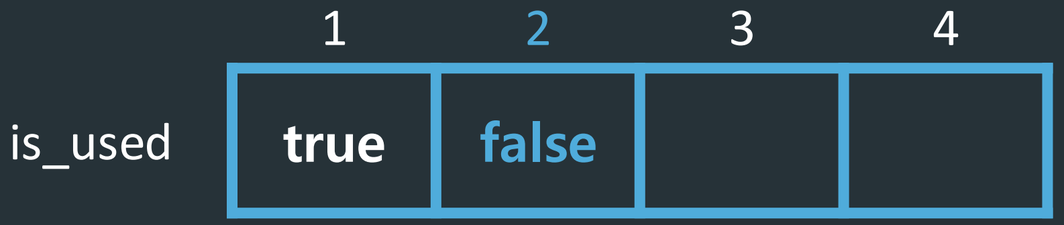


N과 M(1)

#알튜비튜

- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- n = 4, m = 2

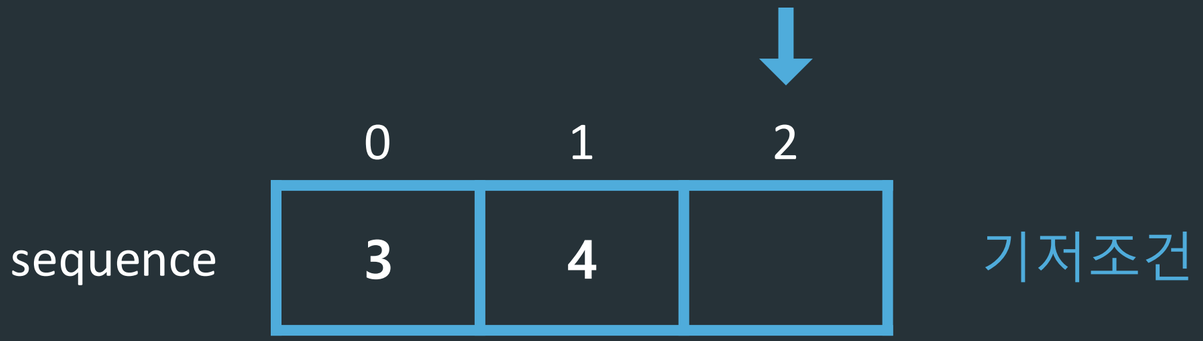


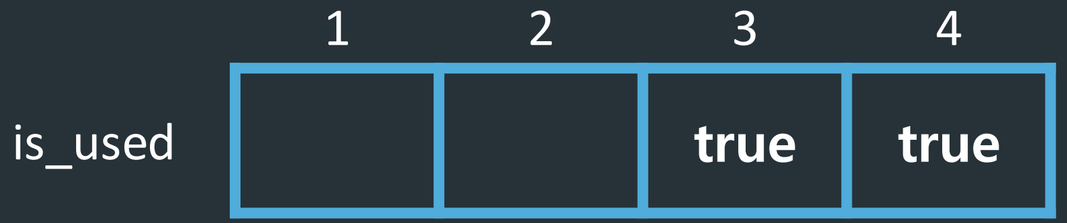


바로 탐색 이어서하지 않고, 꼭 원래 상태로 돌려놓아야 함
그래야 나중에 해당 요소 재탐색 가능

만약 원래대로 돌려놓지 않는다면...

- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- n = 4, m = 2

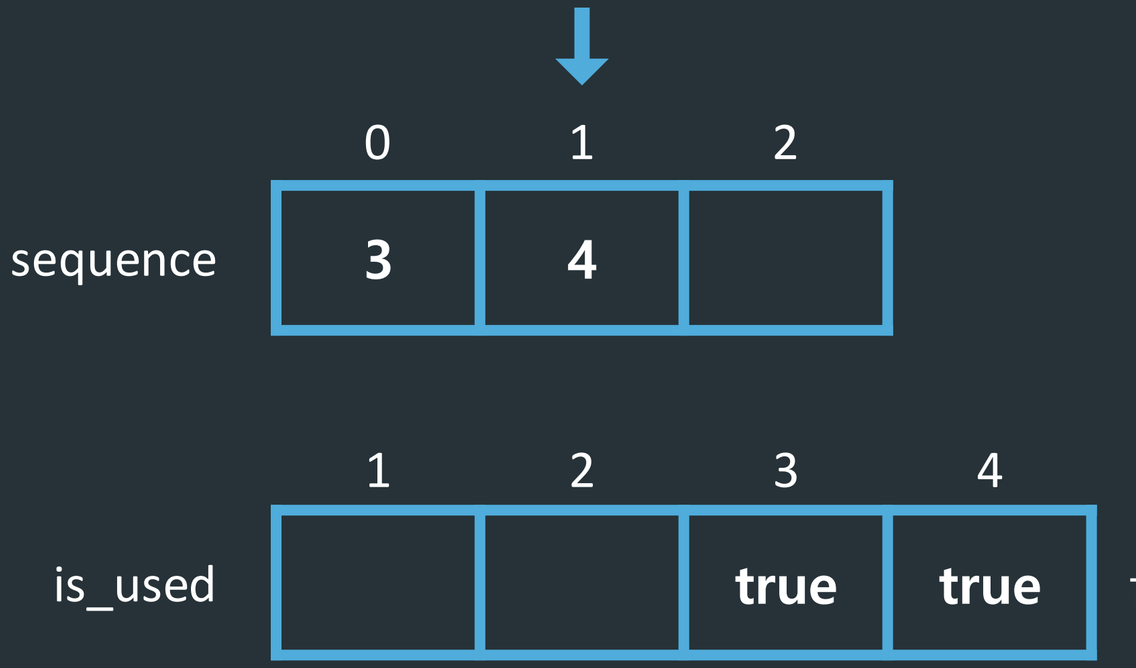




만약 원래대로 돌려놓지 않는다면...

#알튜비튜

- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- n = 4, m = 2

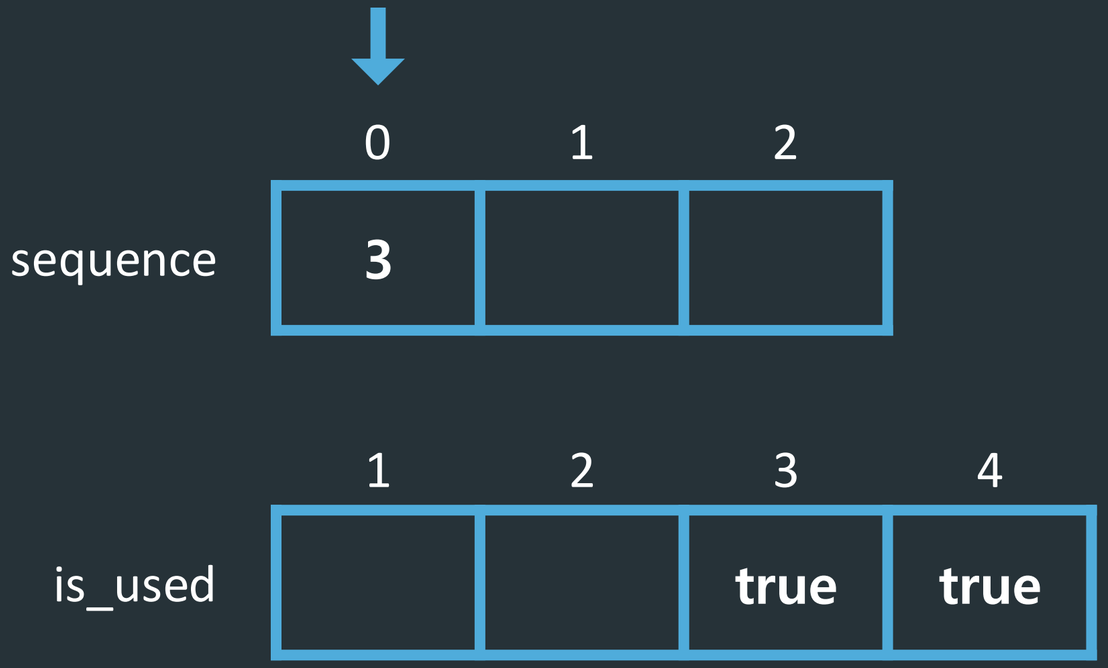


돌아왔을 때, 만약 원래 상태로 되돌려 놓지 않는다면?

만약 원래대로 돌려놓지 않는다면...

#알튜비튜

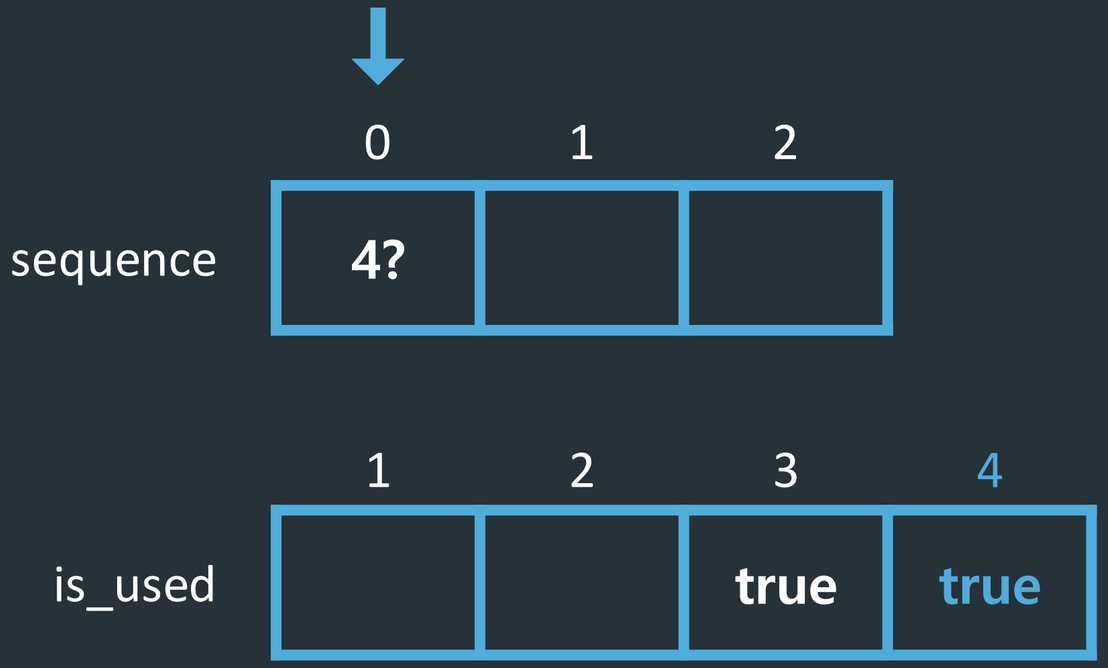
- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- n = 4, m = 2



만약 원래대로 돌려놓지 않는다면...

#알튜비튜

- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- n = 4, m = 2

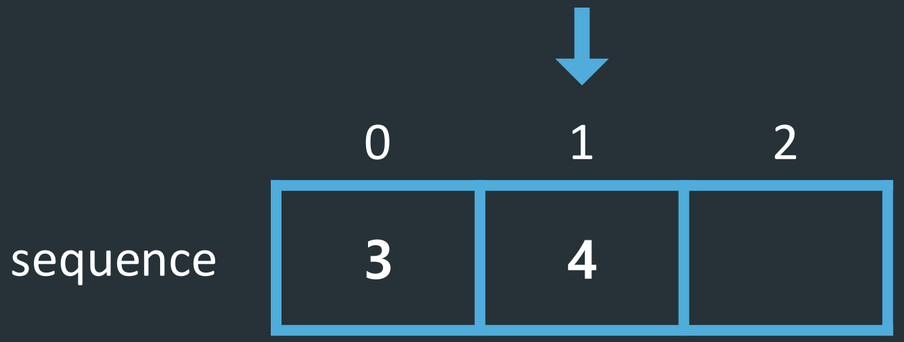


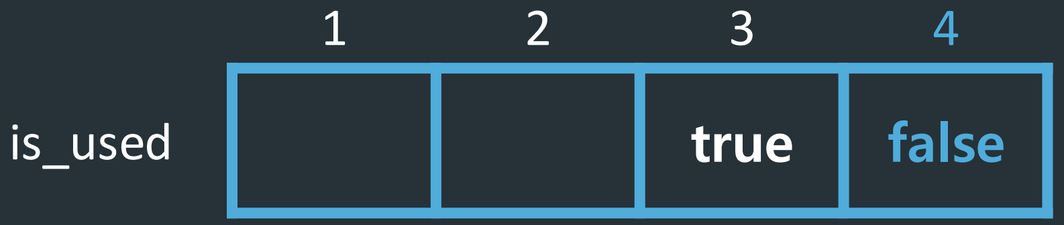
해당 요소 탐색이 불가!

만약 원래대로 돌려놓지 않는다면...

#알튜비튜

- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- n = 4, m = 2

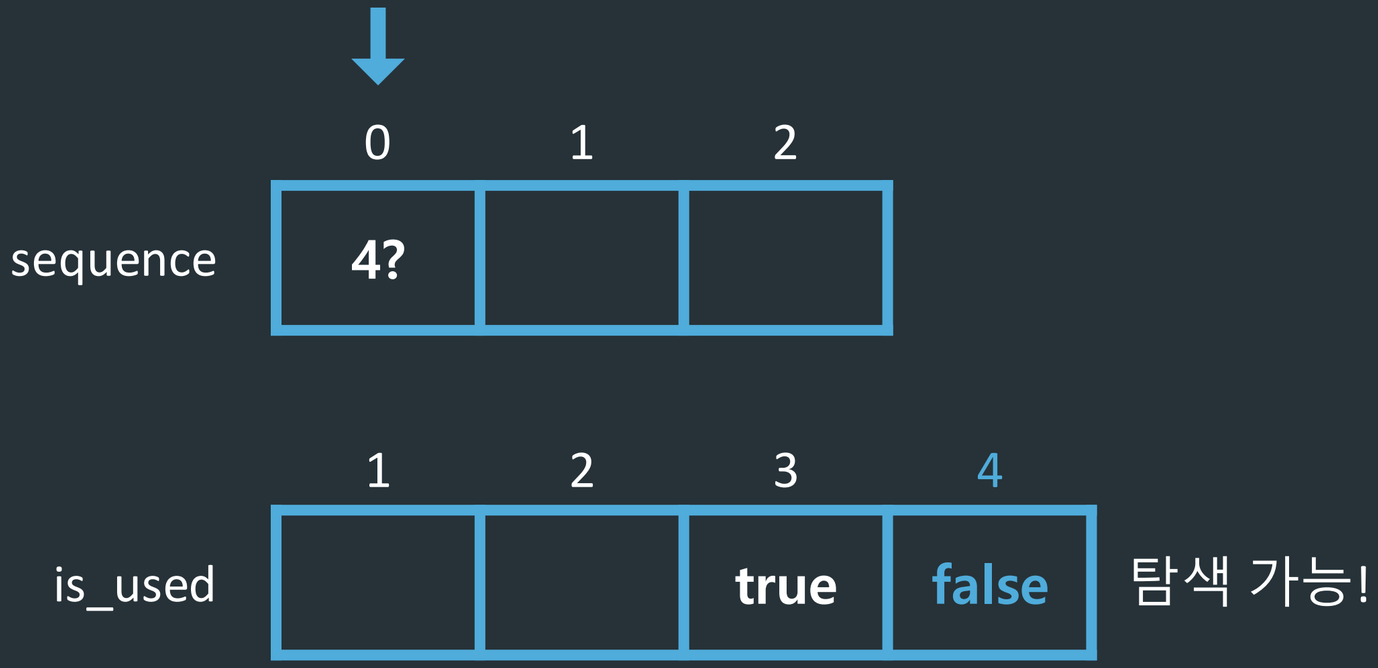




따라서 탐색을 하기 전,
꼭 원래 상태로 돌려놓는 것이 중요

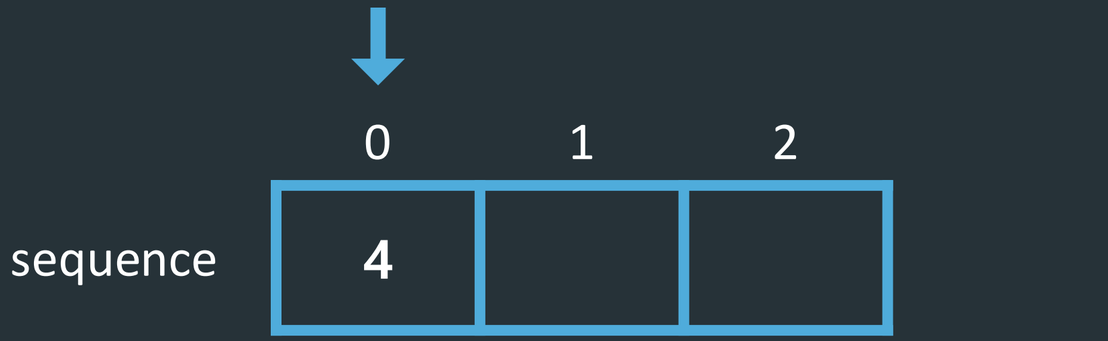
만약 원래대로 돌려놓지 않는다면...

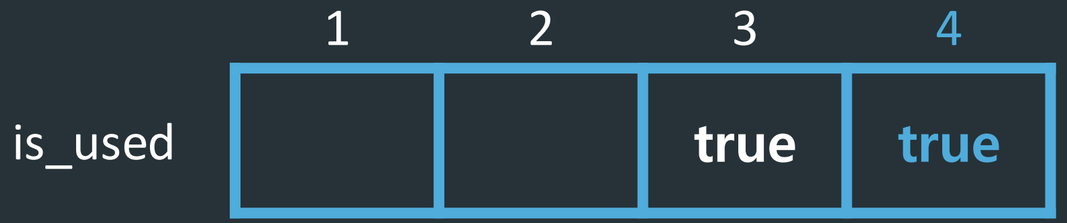
- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- n = 4, m = 2



만약 원래대로 돌려놓지 않는다면...

- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- n = 4, m = 2





2529번 : 부등호 – Silver 1

문제

- 부등호 기호 `<`, `>`가 k개 나열된 순서열 A가 주어짐
  (ex: A ⇒ < < < > < < > < >)
- 부등호 기호 사이에 서로 다른 한 자릿수 숫자(0~9)를 넣어 모든 부등호 관계를 만족시키려고 함
  (ex: 3 < 4 < 5 < 6 > 1 < 2 < 8 > 7 < 9 > 0)
- 부등호 기호를 제거한 뒤, 숫자를 모두 붙여 하나의 수를 만듦
  (ex: 3456128790)
- 부등호 순서를 만족하는 (k+1)자리의 정수 중에서 최댓값과 최솟값을 찾아야 함
- 단, **선택된 숫자는 모두 달라야 함**

예제

#알튜비튜

예제 입력	예제 출력
2 < >	897 021
9 > < < < > > > < <	9567843012 1023765489

부등호

#알튜비튜

접근

- 제약 조건을 살펴보자
  - k개의 부등호 기호 사이에 서로 다른 한 자릿수 숫자를 넣음 → k+1개 숫자 필요
  - 단, 선택된 숫자는 모두 달라야 함 → 중복 X

→ 중복을 허용하지 않는 k+1 길이의 수열 뽑기

부등호

#알튜비튜

접근

- 부등호 관계를 만족시켜야 함
  - < : 이전 숫자보다 큰 숫자를 뽑아야 함
  - > : 이전 숫자보다 작은 숫자를 뽑아야 함
  - 처음 숫자를 뽑는 경우는 따로 처리가 필요 (이전 숫자가 없으므로)

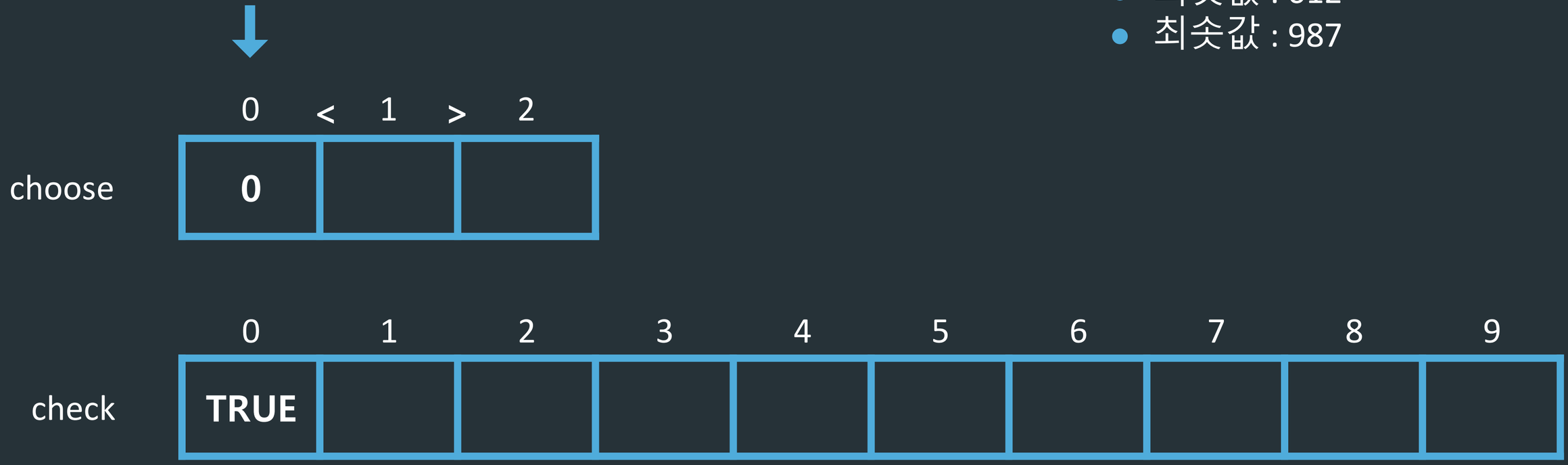
- 수열을 다 뽑았으면 최댓값 & 최솟값 갱신

부등호

- < > 를 만족시키는 수열 뽑기

최댓값과 최솟값 미리 초기화

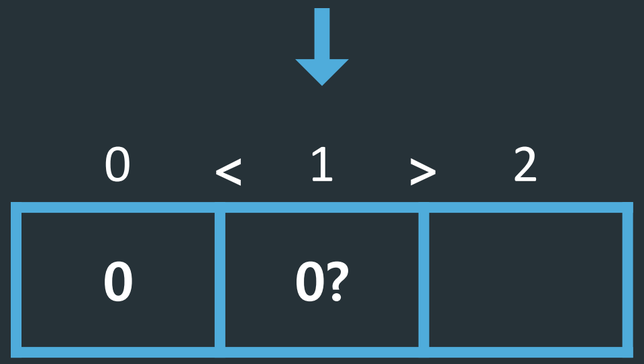
- 최댓값 : 012
- 최솟값 : 987



부등호

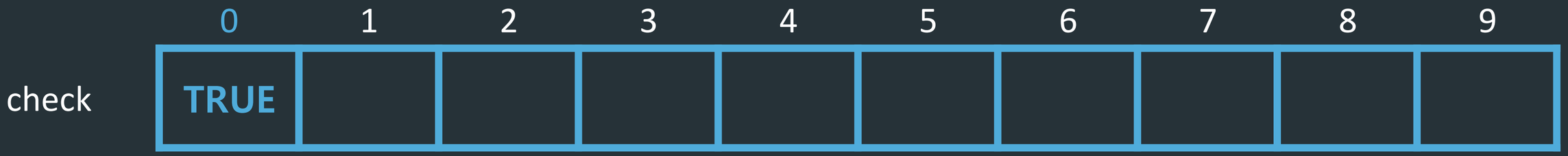
#알튜비튜

- < > 를 만족시키는 수열 뽑기



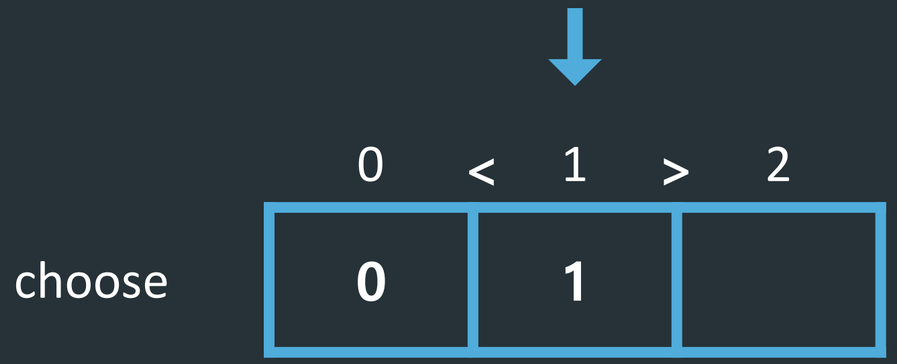
이미 선택한 숫자

- 최댓값 : 012
- 최솟값 : 987

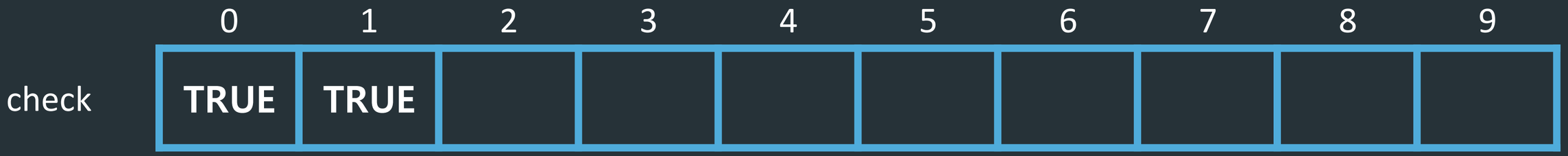


부등호

- < > 를 만족시키는 수열 뽑기



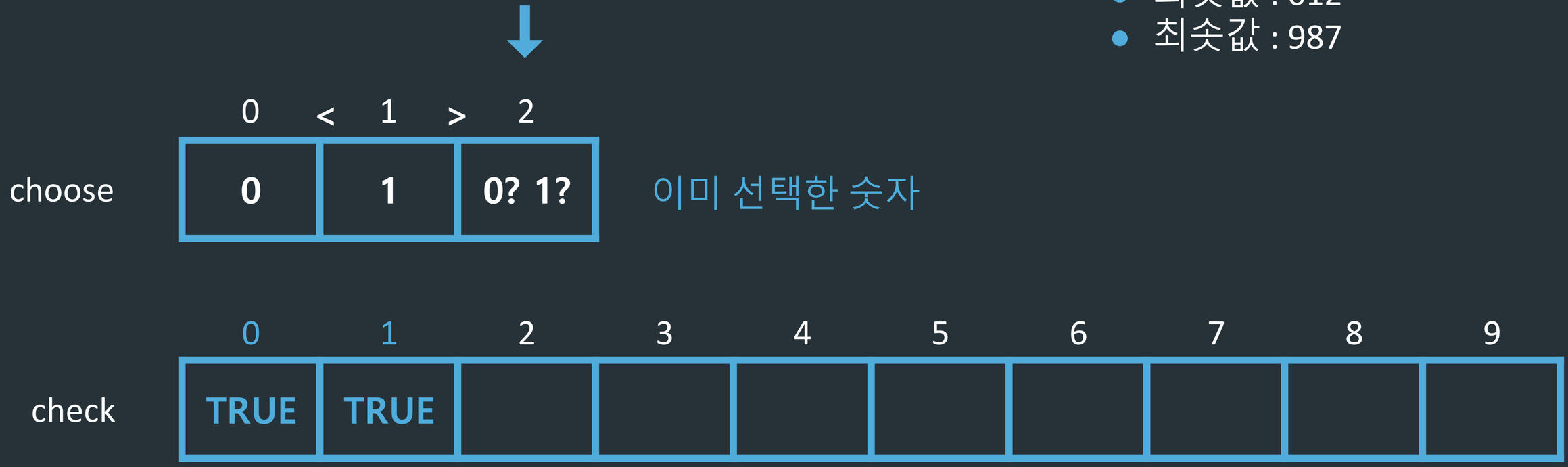
- 최댓값 : 012
- 최솟값 : 987



부등호

- < > 를 만족시키는 수열 뽑기

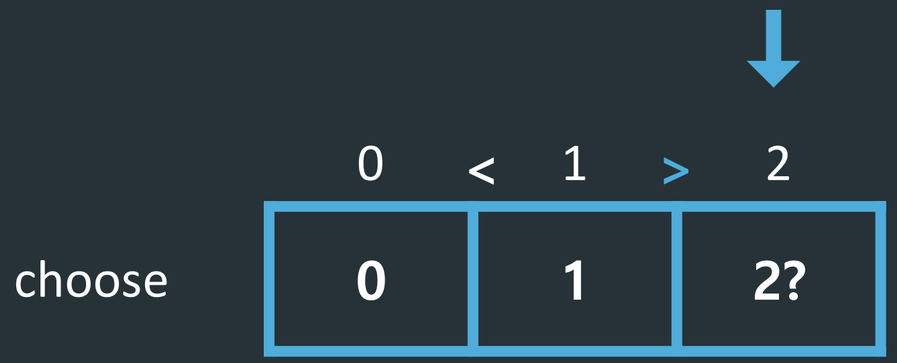
- 최댓값 : 012
- 최솟값 : 987



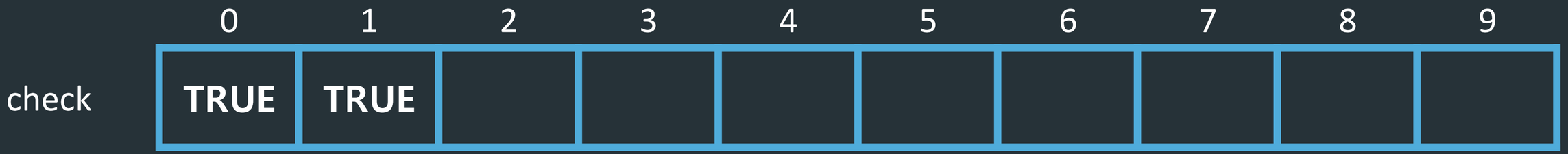
부등호

- < > 를 만족시키는 수열 뽑기

- 최댓값 : 012
- 최솟값 : 987

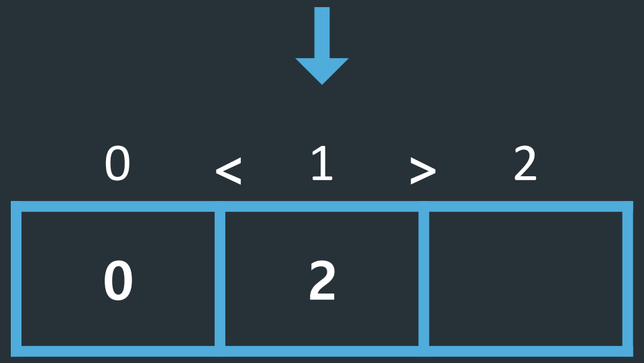


이전 숫자보다 크기 때문에 선택 불가
(나머지 3~9도 마찬가지)

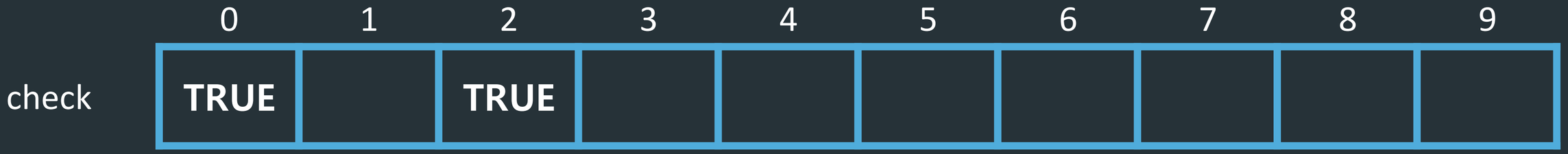


부등호

- < > 를 만족시키는 수열 뽑기



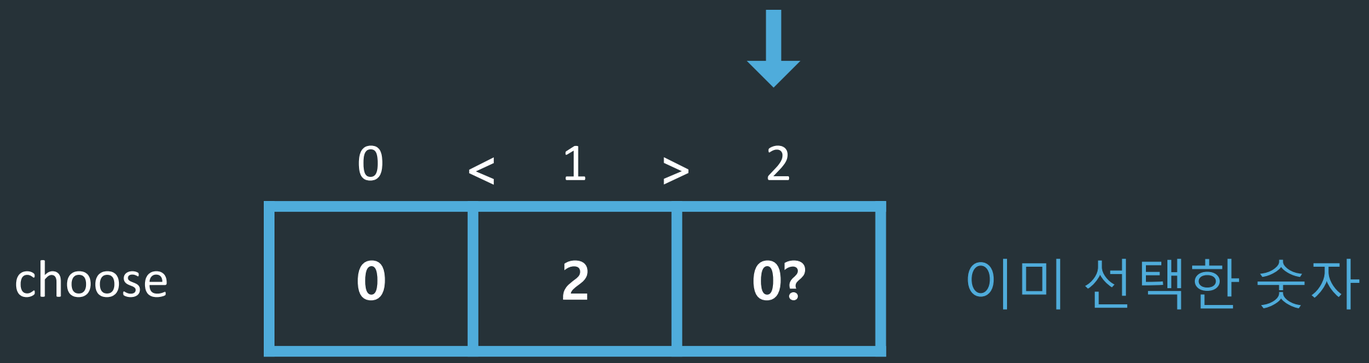
- 최댓값 : 012
- 최솟값 : 987

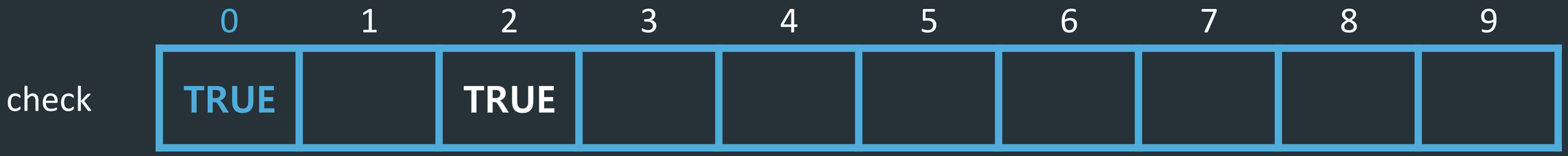


부등호

- < > 를 만족시키는 수열 뽑기

- 최댓값 : 012
- 최솟값 : 987

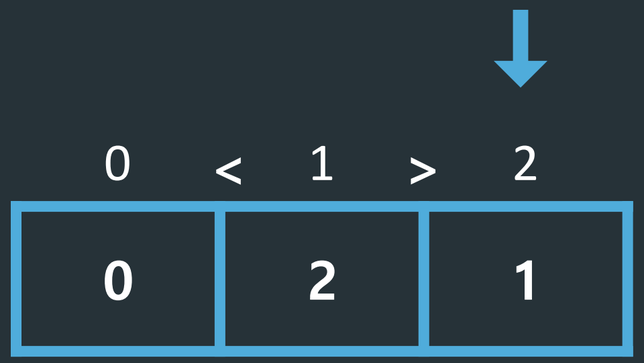




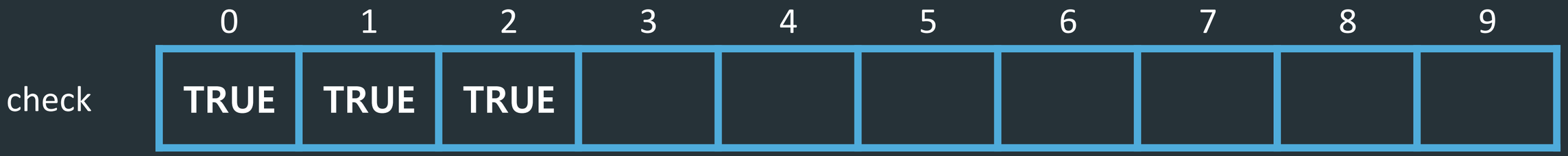
부등호

- < > 를 만족시키는 수열 뽑기

- 최댓값 : 012
- 최솟값 : 987 → **021**



조건 만족!! → 갱신..반복



9663번 : N-Queen – Gold 4

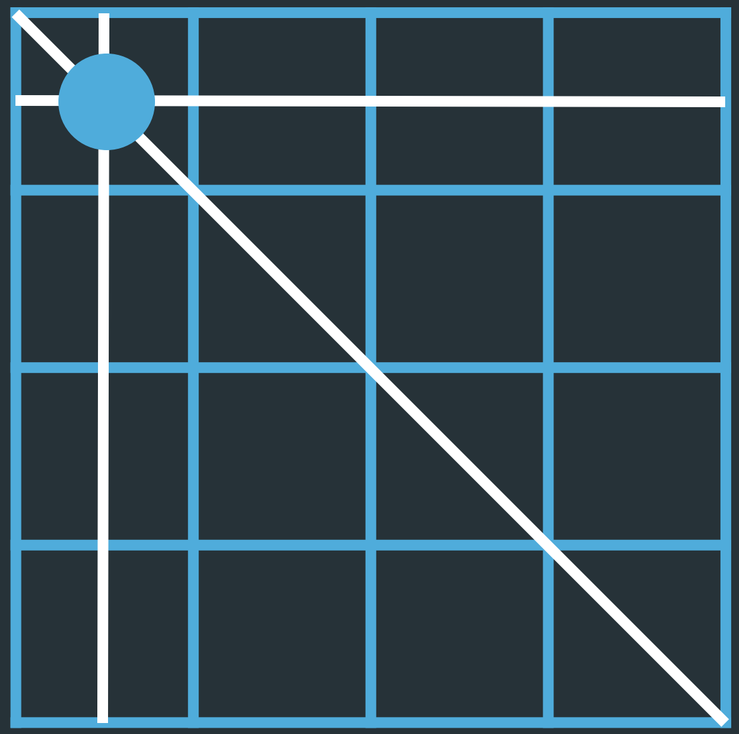
문제

- N X N인 체스판 위에 퀸 N개를 서로 공격할 수 없게 놓는 경우의 수 구하는 문제

제한 사항

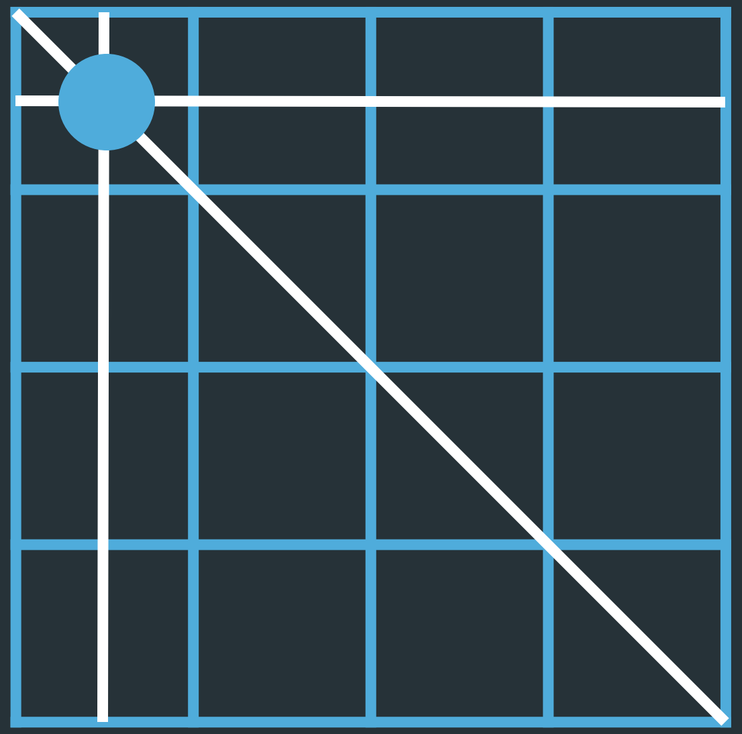
- 입력 범위는 1 <= N < 15

N-Queen



- 퀸이 놓이면, 가로, 세로, 대각선에 위치한 곳엔 다른 퀸을 둘 수 없다.

N-Queen



- 퀸이 놓이면, 가로, 세로, 대각선에 위치한 곳엔 다른 퀸을 둘 수 없다.

제한 사항

- 입력 범위는 1 <= N < 15

접근

- 완전탐색?

→ C(225, 15) > 10^30, 절대 불가능!

- 가지치기를 어떻게 할 수 있을까?

몰래 보세요

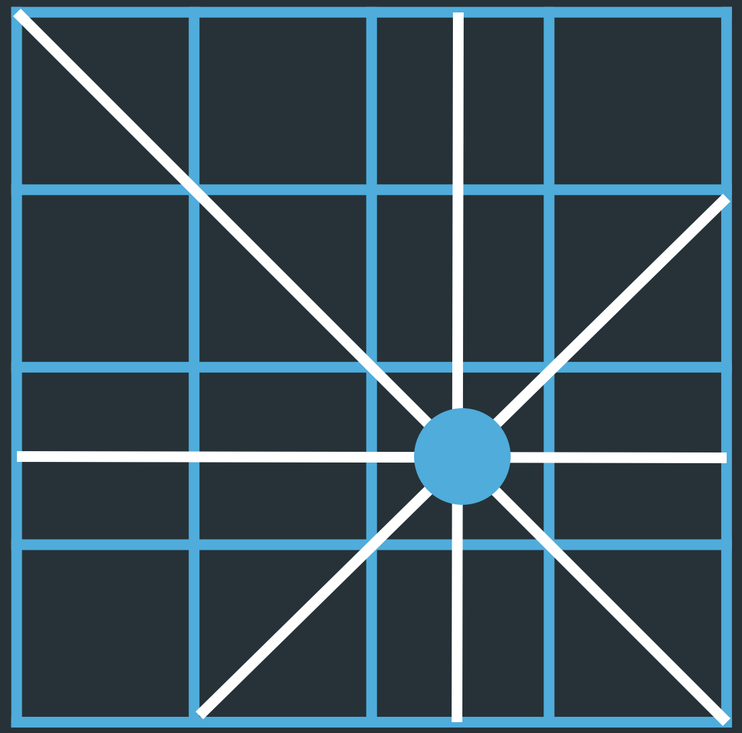
#알튜비튜

Hint

1. N과 M문제에서 가지치기를 위해 무엇을 사용했었죠?
2. 그런데 가지치기할 곳이 1 군데는 아닌 것 같아요. 대각선도 방향이 하나가 아니죠.

N-Queen

- N = 4



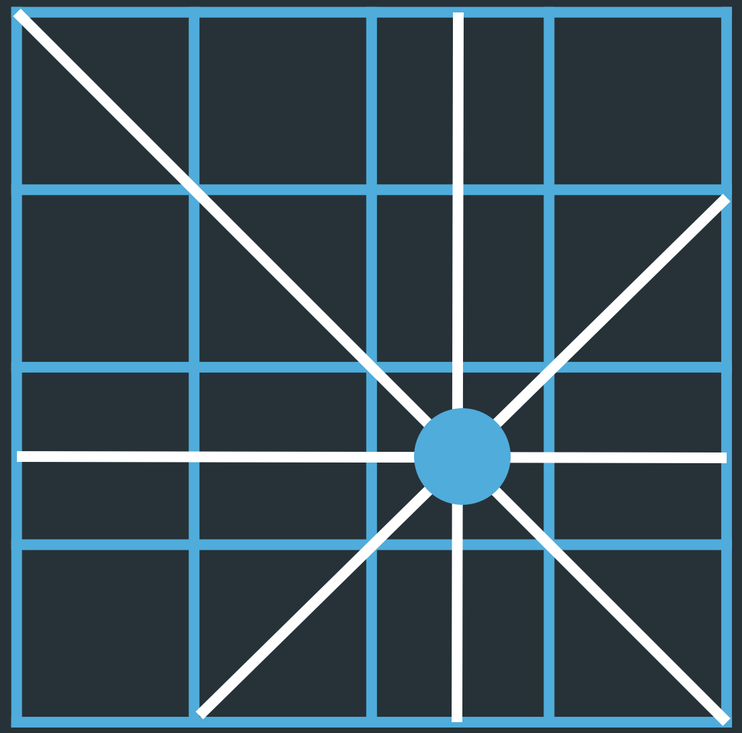
N X N 체스판!

- 우선 세로의 중복을 피하려면 각 열마다 1개 & 가로의 중복을 피하려면 각 행마다 1개
- 즉, 가로와 세로에는 각각 1개만 들어감!

→ 각 가로(행)에 대해, 퀸이 놓일 세로(열) 위치 결정

N-Queen

- N = 4



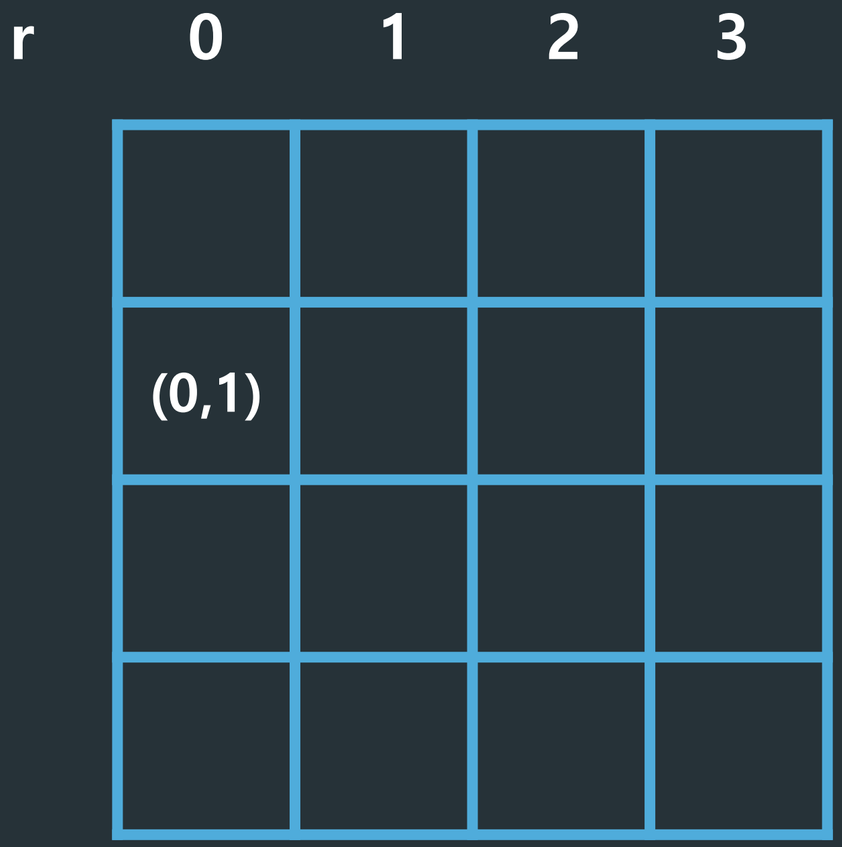
그럼 남은 대각선 방향은 어떻게??

- 대각선 == 기울기 -1 or 1
  → 기울기 절대값 1
- 즉, 기울기의 절대값이 1인 경우는 체스를 놓을 수 없다!
- 기울기 = (세로 길이) / (가로 길이)

N-Queen

#알튜비튜

● N = 4



1. 각 열 1개 필수
2. 각 행 1개 필수
3. |기울기| != 1

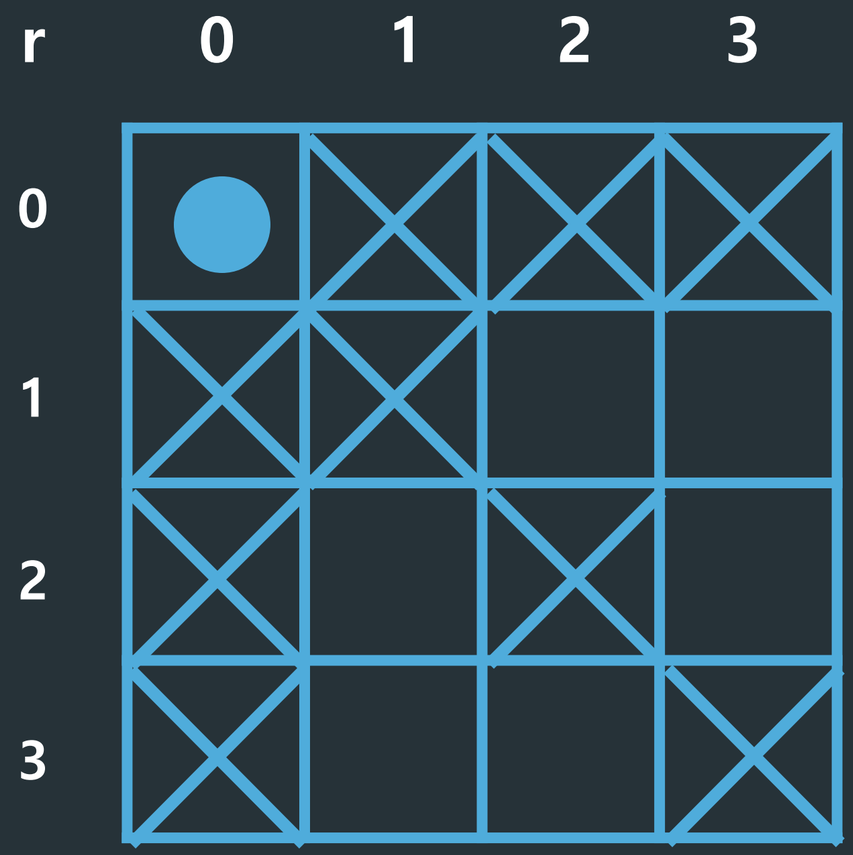
→ 배열 index = 행 위치 & 배열 값 = 열 위치 저장!

● queen_loc[r]

ex) queen_loc[0] = 1 → 좌표 (0,1)

N-Queen

● N = 4

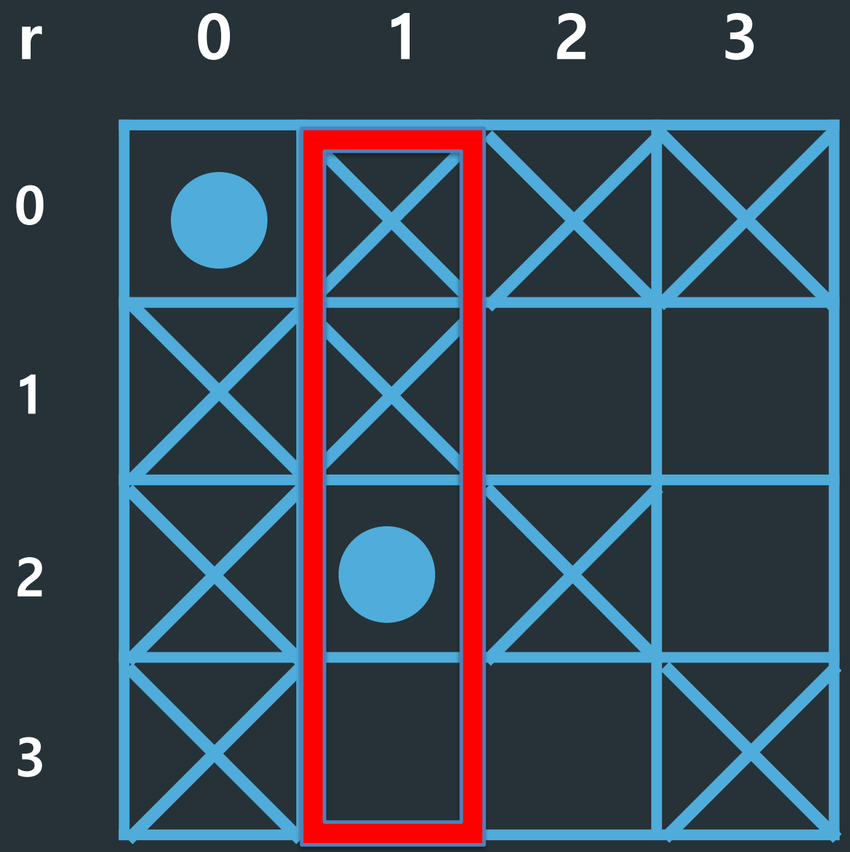


r = 0 부터 채우기 시작!

1. queen_loc[0] = 0

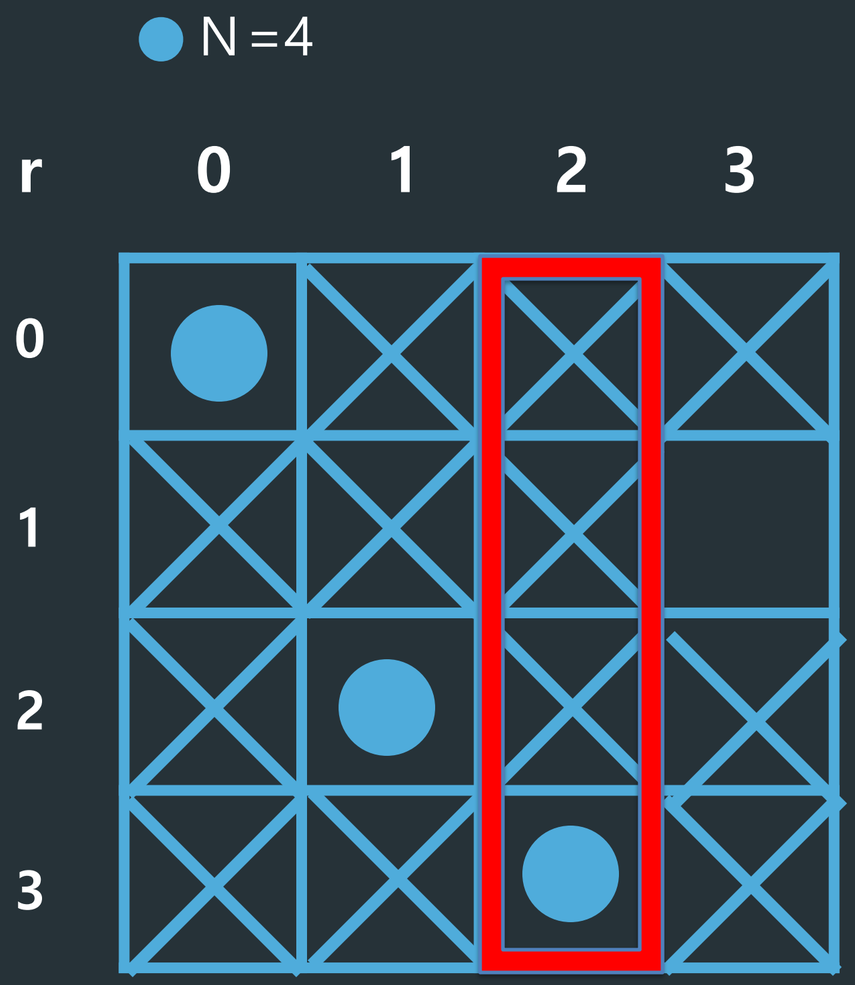
N-Queen

● N = 4



1. queen_loc[0] = 0
2. queen_loc[1] ??
   a. 0? → X 같은 열 존재
   b. 1? → X queen_loc[0]과 대각선 기울기 1
   c. 2? → O

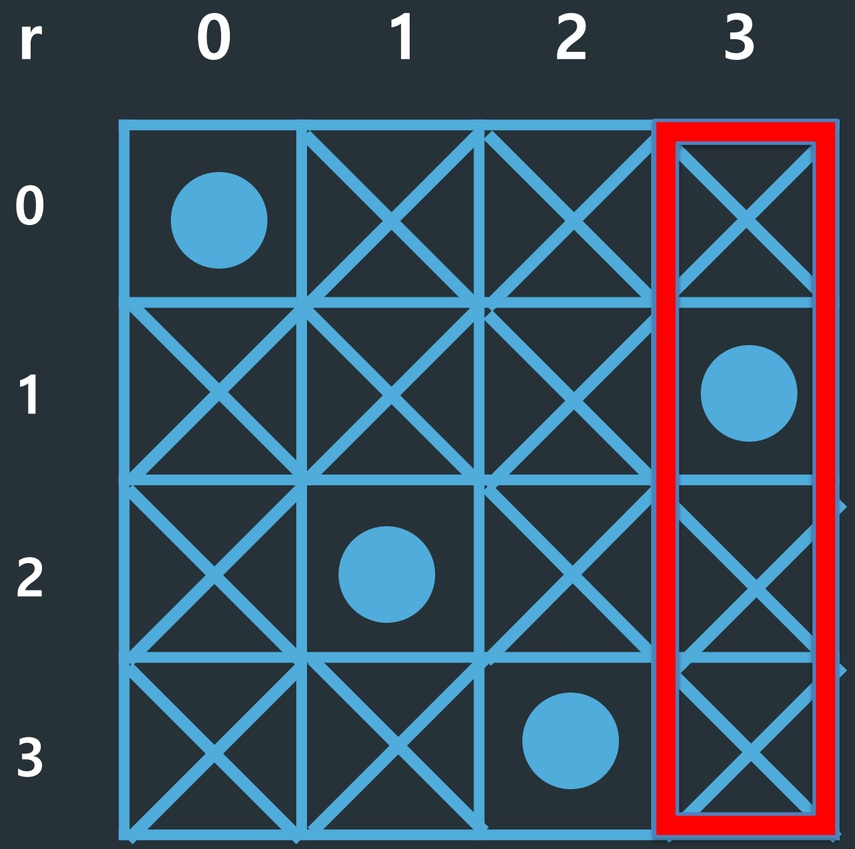
N-Queen



3. queen_loc[2] ??
    a. 0? → X 같은 열 존재
    b. 1? → X queen_loc[1]과 대각선 기울기 1
    c. 2? → X 같은 열 존재

N-Queen

● N = 4

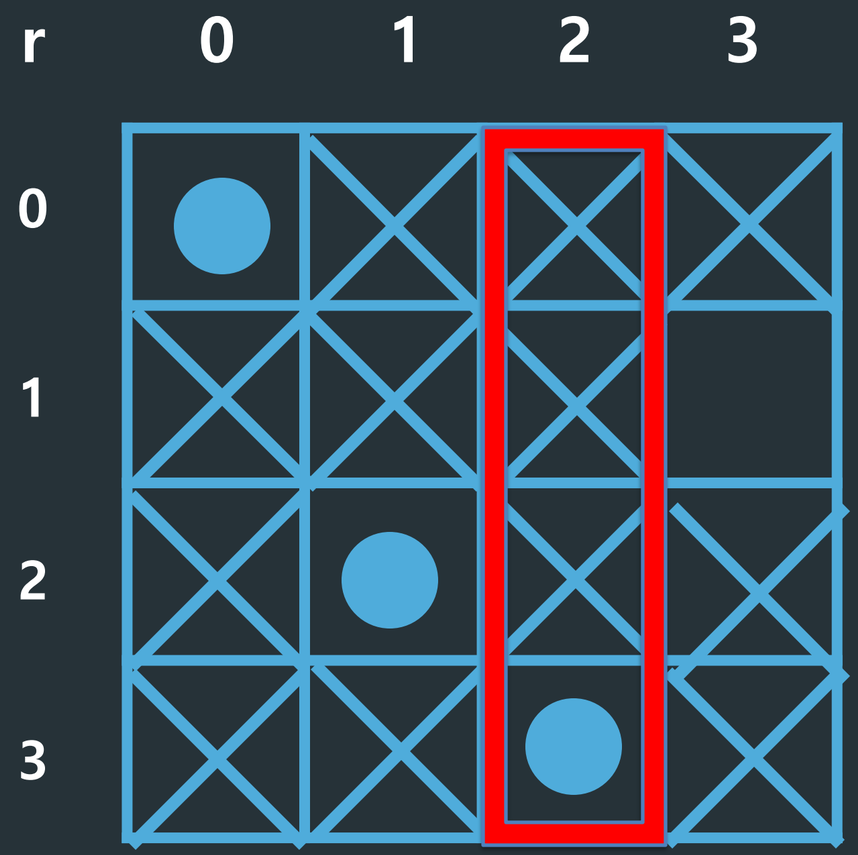


4. queen_loc[3] ??
   a. 0? → X 같은 열 존재
   b. 1? → O

r = 3 까지 탐색 완료!
ans++ 후 백트래킹!

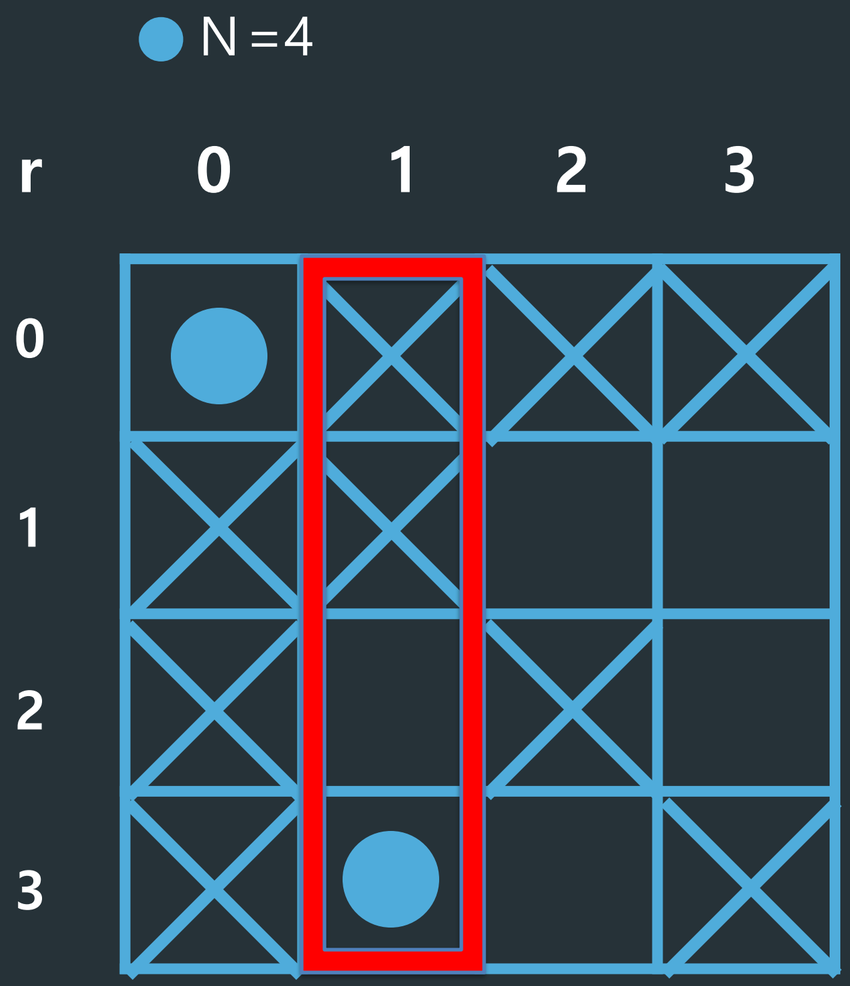
N-Queen

● N = 4



r = 2에서
0, 1, 2 는 이미 불가능 하다고 판단되었으므로
가지치기되었음
탐색할 필요 X

N-Queen



r = 1에서 2까지 탐색 완료
이번엔 queen_loc[1] = 3인 경우부터 탐색 시작!

백트래킹 문제는 다른 풀이(완전 탐색)이 가능한 경우가 많아요

#알튜비튜

비트마스킹

- 비트(0 / 1)의 연산을 활용한 알고리즘
- 배열의 원소 상태가 2가지로 나뉠 수 있는 백트래킹 문제에서 활용 가능

permutation

- 다음 순열(next_permutation) 혹은 이전 순열(prev_permutation)을 구하는 알고리즘
- 원소의 순서를 정해야 하는 백트래킹 문제에서 활용 가능
- 그러나! 백트래킹이 아닌 완전탐색이므로 속도는 백트래킹보다 느림

마무리

정리

- 완전탐색에서 나아가 특정한 조건을 만족하는 경우만 탐색하는 백트래킹
- 입력 범위가 작고(보통 20을 넘지 않는다), 재귀함수로 주로 구현!
- 백트래킹을 쉽게 구현하다보니, 전역변수와 void 함수를 사용하지만 다른 알고리즘에선 가능한 쓰지 않는게 좋다
- 가지치기를 어디서 해야 하는지와 어떻게 할지 파악하는게 중요
- 가지치기를 얼마나 잘하냐에 따라 시간의 효율도 높아짐

이것도 알아보세요!

- 문제를 풀고 제출한 후, 다른 사람들의 풀이와 내 풀이의 시간을 비교해보아요. (가지치기 효율)
- next_permutation과 비트마스킹으로 백트래킹 문제를 풀어보고 시간을 비교해보아요.

과제

필수

- 2477번: 참외밭 - Silver 2
- 15665번 : N과 M (11) - Silver 2
- 14888번 : 연산자 끼워넣기 – Silver 1

도전

- 프로그래머스: 소수 찾기 – Lv.2
- 2580번 : 스도쿠 – Gold 4

과제 마감일

과제제출 마감 ~ 10월 15일 화요일 18:59

추가제출 마감 ~ 10월 17일 목요일 23:59