실전 알고리즘 0x07강

백트래킹, 시뮬레이션

BaaaaaaaaaaaaaaaarkingDog

목차

0x00 백트래킹(Backtracking) - 정의

- 미연시를 해본 적 있나요? 아니면 검은방/역전재판과 같은 추리게임에서의 언쟁을 생각하셔도 좋습니다. ※ 저는 미연시를 하지 않습니다.. ㄹㅇ루다가
- 게임을 클리어하는(혹은 트루엔딩을 보는) 최적의 경로를 알고 있으면 그 경로대로 진행을 하면 되지만, 그렇지 않다면 모든 경우의 수를 전부 다 해봐야합니다.
- 모든 경우의 수를 다 해보기 위해 대충 이런 식으로 플레이를 하게 될 것입니다.

0x00 백트래킹(Backtracking) - 정의

범인을 추리해보자

범인은?

하무열

베드엔딩

서현진

안승범

근거는?

지나치게 소극적인 성격

베드엔딩

범인과의 원한관계

베드엔딩

백선교와 관계된 그의 과거

살인 트릭은?

갈고리와 천을 이용해 목을 조름

트루엔딩

주사기를 몰래 꼽아 독살

베드엔딩

0x00 백트래킹(Backtracking) - 정의

범인을 추리해보자

범인은?

- 하무열 ← 베드엔딩
- 서현진
- 안승범

근거는?

- 지나치게 소극적인 성격 ← 베드엔딩
- 범인과의 원한관계 ← 베드엔딩
- 백선교와 관계된 그의 과거

살인 트릭은?

- 갈고리와 천을 이용해 목을 조름 ← 트루엔딩
- 주사기를 몰래 꼽아 독살 ← 베드엔딩

0x00 백트래킹(Backtracking) - 정의

범인을 추리해보자

범인은?

하무열 ← 베드엔딩

서현진

안승범

근거는?

지나치게 소극적인 성격 ← 베드엔딩

범인과의 원한관계 ← 베드엔딩

백선교와 관계된 그의 과거

살인 트릭은?

갈고리와 천을 이용해 목을 조름 ← 트루엔딩

주사기를 몰래 꼽아 독살 ← 베드엔딩

0x00 백트래킹(Backtracking) - 정의

범인을 추리해보자

범인은?

하무열

베드엔딩

서현진

안승범

근거는?

지나치게 소극적인 성격

베드엔딩

범인과의 원한관계

베드엔딩

백선교와 관계된 그의 과거

살인 트릭은?

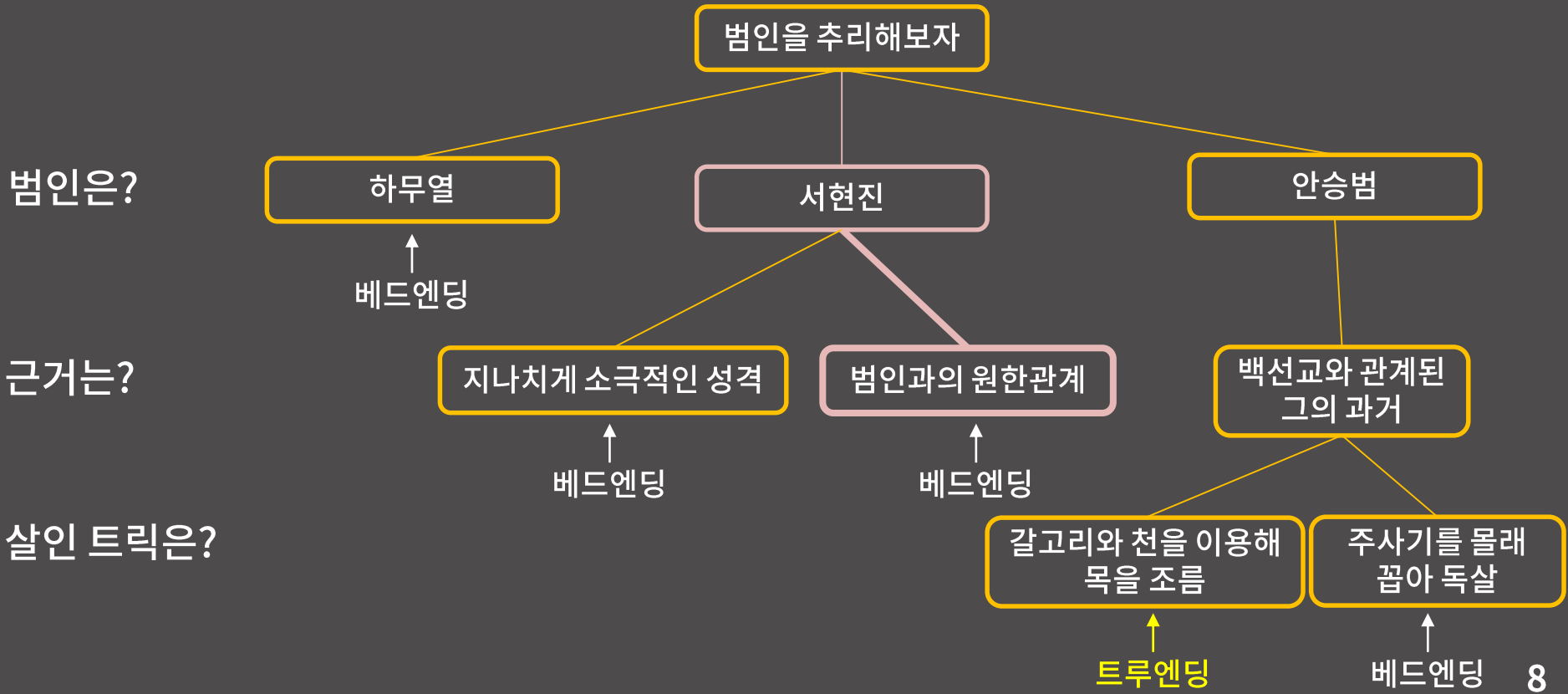
갈고리와 천을 이용해 목을 조름

트루엔딩

주사기를 몰래 꼽아 독살

베드엔딩

0x00 백트래킹(Backtracking) - 정의



0x00 백트래킹(Backtracking) - 정의

범인을 추리해보자

범인은?

하무열 — 베드엔딩

서현진

안승범

근거는?

지나치게 소극적인 성격 — 베드엔딩

범인과의 원한관계 — 베드엔딩

백선교와 관계된 그의 과거

살인 트릭은?

갈고리와 천을 이용해 목을 조름 — 트루엔딩

주사기를 몰래 꼽아 독살 — 베드엔딩

0x00 백트래킹(Backtracking) - 정의

범인을 추리해보자

범인은?

하무열 ← 베드엔딩

서현진

안승범

근거는?

지나치게 소극적인 성격 ← 베드엔딩

범인과의 원한관계 ← 베드엔딩

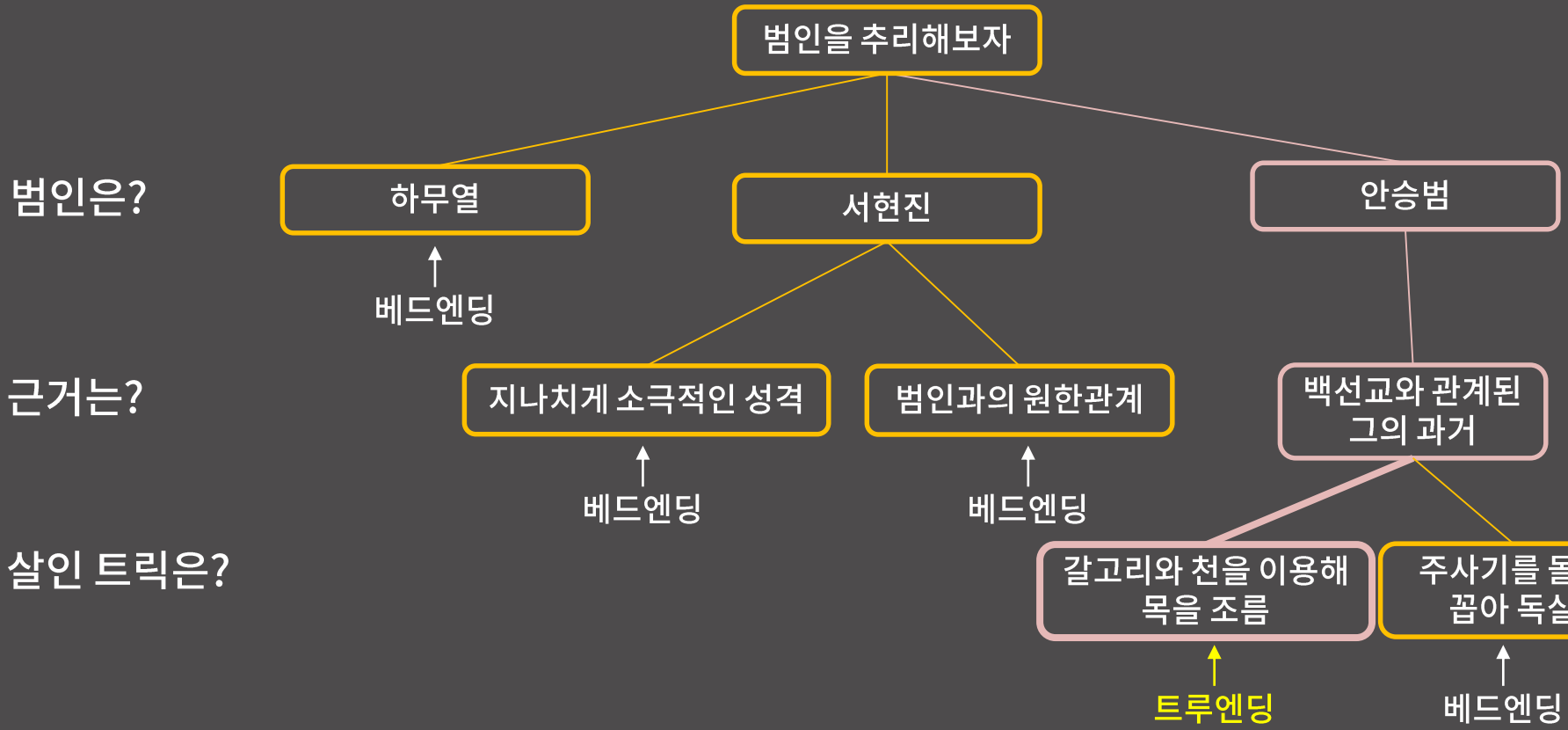
백선교와 관계된 그의 과거

살인 트릭은?

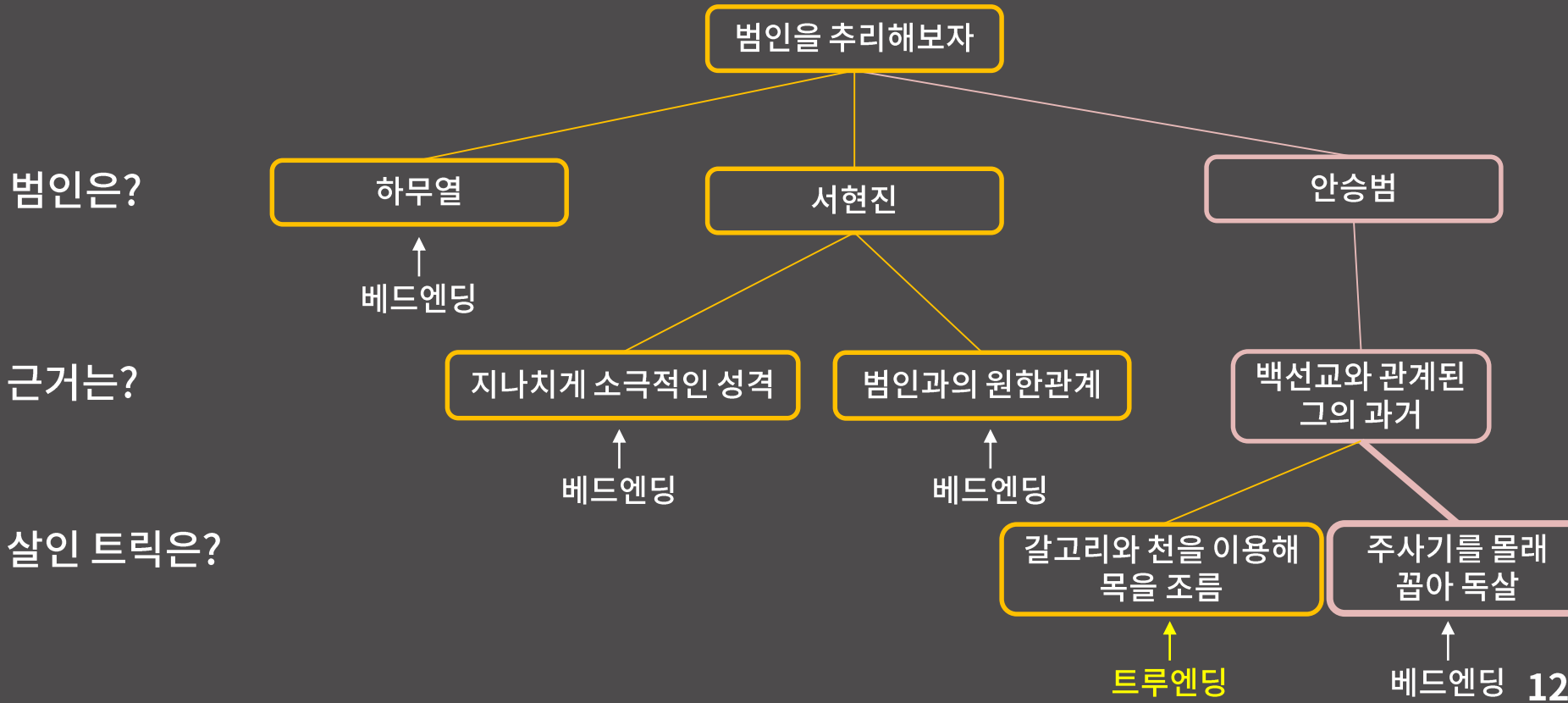
갈고리와 천을 이용해 목을 조름 ← 트루엔딩

주사기를 몰래 꼽아 독살 ← 베드엔딩

0x00 백트래킹(Backtracking) - 정의



0x00 백트래킹(Backtracking) - 정의



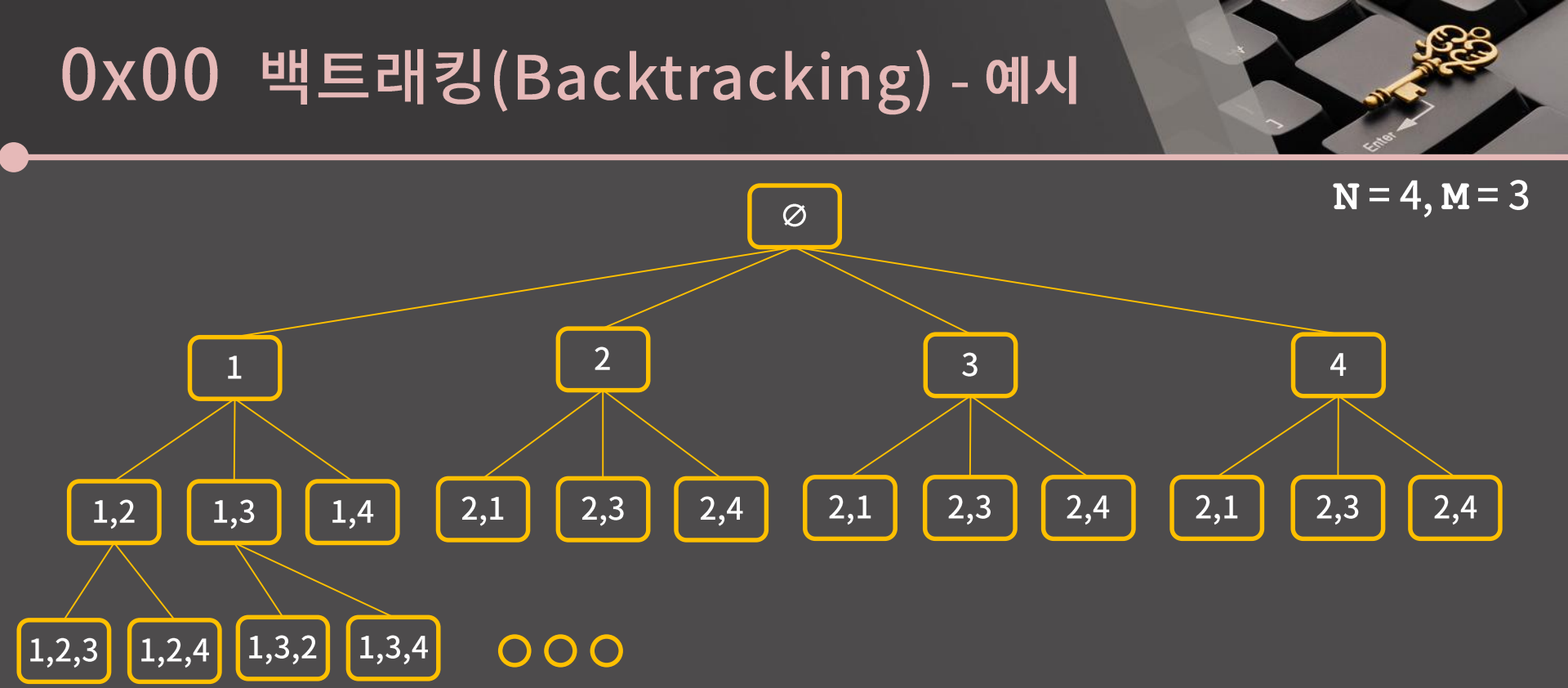
0x00 백트래킹(Backtracking) - 정의

- 이와 같이 주어진 문제의 답을 구하기 위해 현재 상태에서 가능한 모든 후보군을 따라 들어가며 탐색하는 알고리즘을 백트래킹이라고 합니다.
- 상당한 구현력을 필요로 하고, 실수하기도 쉬운데다가 재귀의 특성상 틀리더라도 실수한 부분을 찾기도 힘들어서 굉장히 많은 시간을 할애해 연습하지 않으면 풀이는 아는데 코드로 옮겨내지를 못해 문제를 풀지 못하는 경우가 생길 수 있습니다.

0x00 백트래킹(Backtracking) - 예시

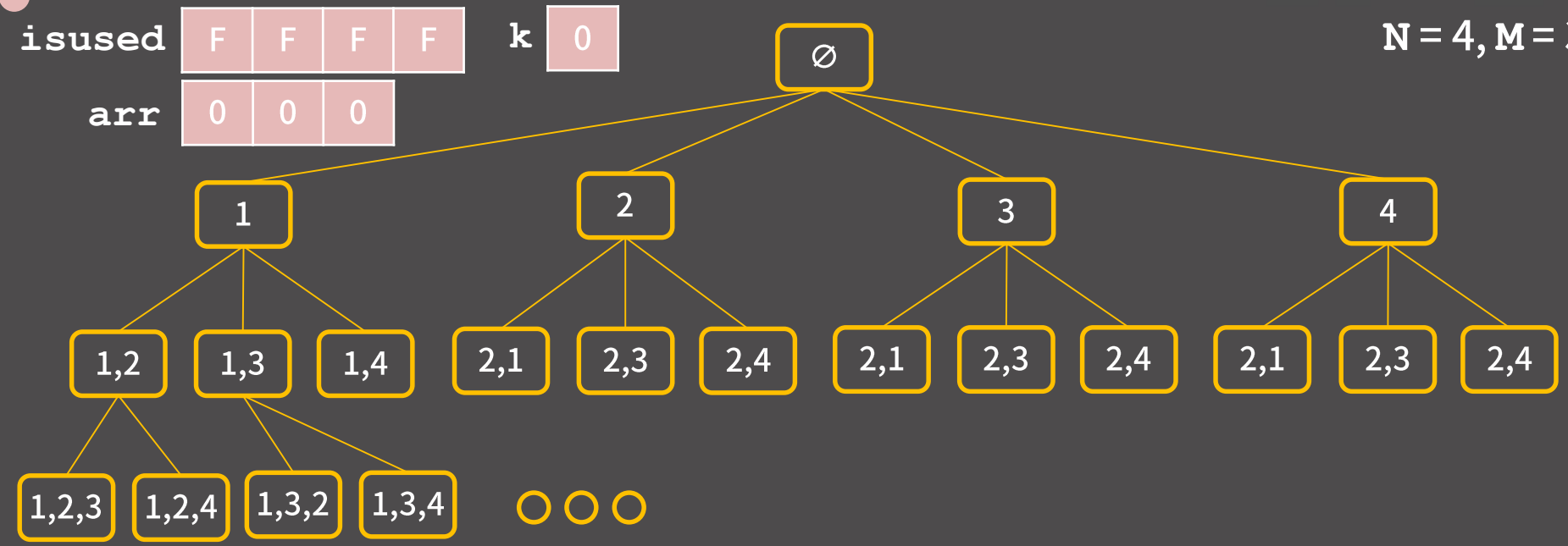
- BOJ의 N과 M 시리즈가 백트래킹을 연습하기에 아주 적합합니다. 총 12문제 중에서 몇 가지를 같이 풀어보겠습니다.
- BOJ 15649번 : N과 M (1)
- 비어있는 리스트에서 시작해 수를 하나씩 추가하면서 길이가 M인 수열을 만들면 해당 수열을 출력하면 됩니다.
- 현재의 상태에서 수를 추가하기 위해서는 어떤 수가 수열에 이미 쓰였고 어떤 수가 아직 쓰이지 않았는지를 따로 저장하고 있어야 합니다.
- 우선 각 상태가 어떤 모양을 이루고 있는지, 트리를 한번 살펴봅시다.

0x00 백트래킹(Backtracking) - 예시



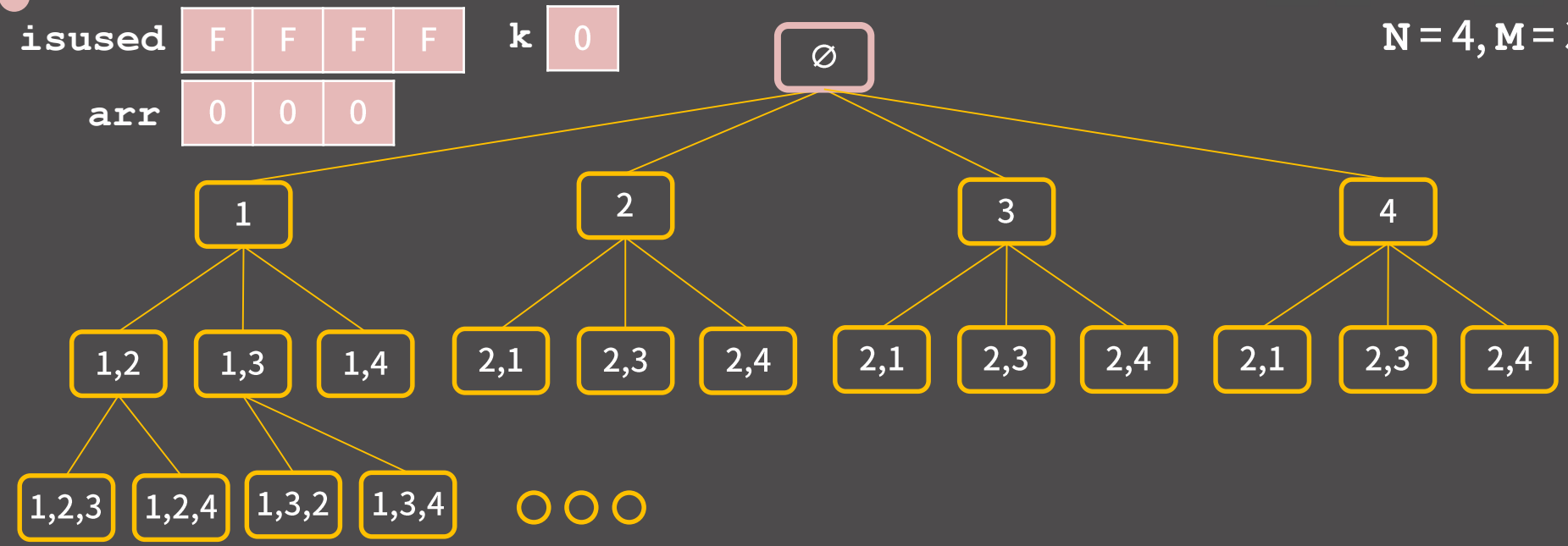
- 적절하게 이러한 트리대로 수열을 만들어가다가 M개의 항이 쌓이면 출력을 하면 됩니다.

0x00 백트래킹(Backtracking) - 예시



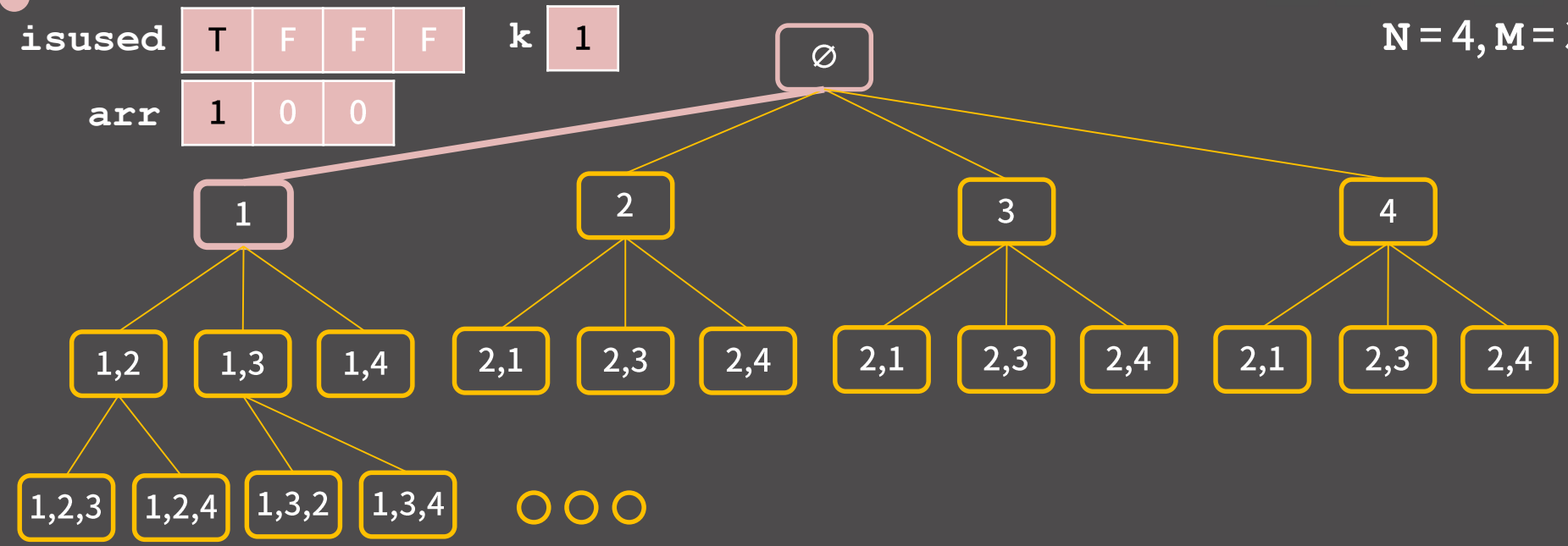
- `isused`는 1부터 4까지의 각 수가 쓰였는지 확인하는 변수이고 `k`는 리스트에 들어있는 수의 갯수를 의미합니다. `arr`는 리스트에 들어있는 수를 의미하고 `arr[1~k]` 까지만 의미를 가집니다.(서술의 편의상 `isused`, `arr` 모두 1-indexed로 생각하겠습니다.)

0x00 백트래킹(Backtracking) - 예시



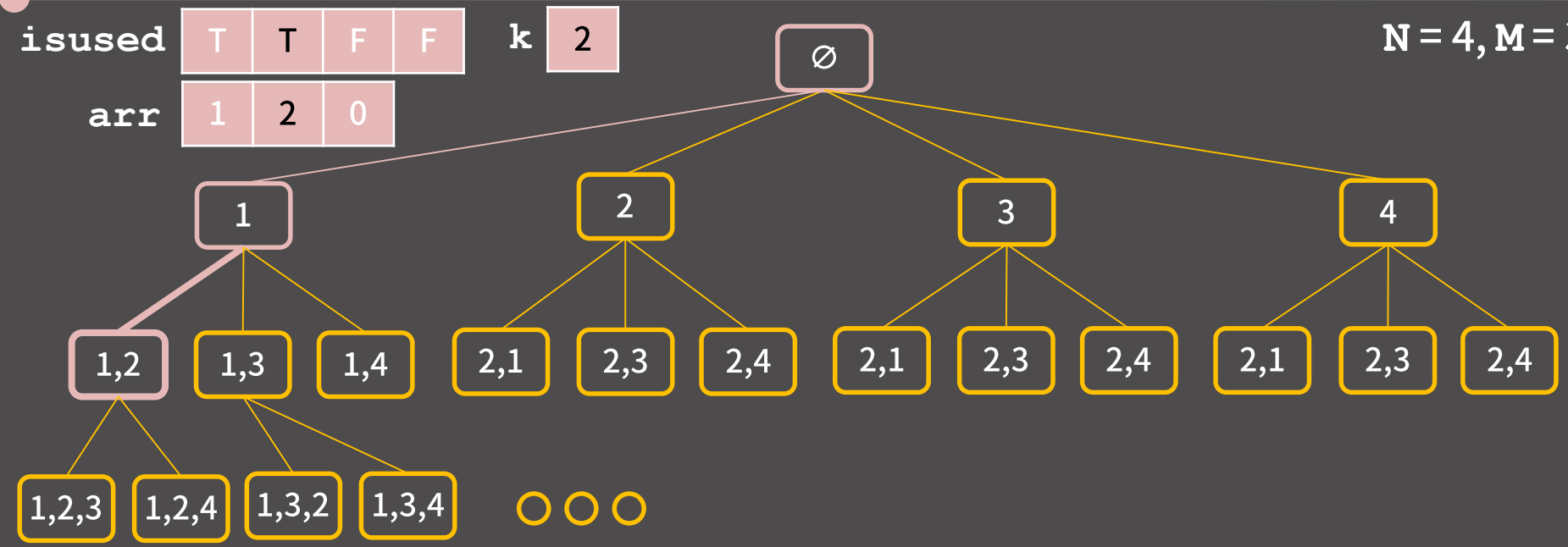
- 백트래킹을 시작합니다. 리스트는 현재 비어있습니다.

0x00 백트래킹(Backtracking) - 예시



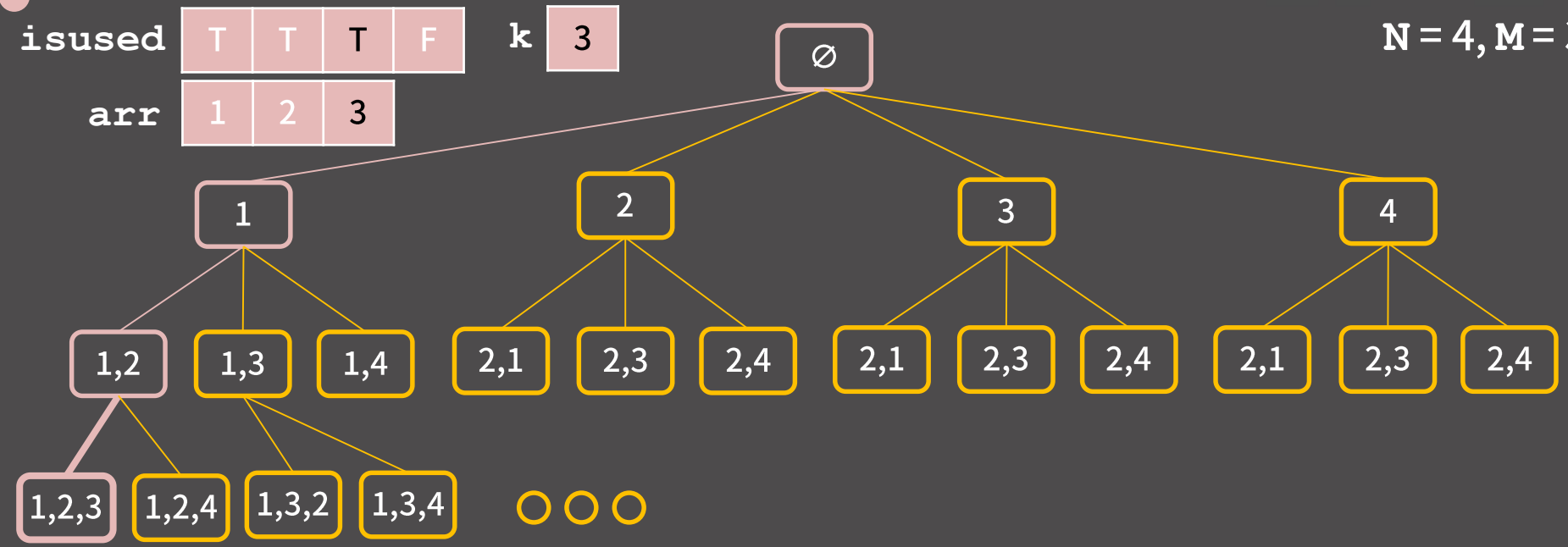
- 리스트의 1번째 원소에 넣을 수를 정합시다. 아직 1이 사용되지 않았으므로 우선 1을 넣습니다.

0x00 백트래킹(Backtracking) - 예시



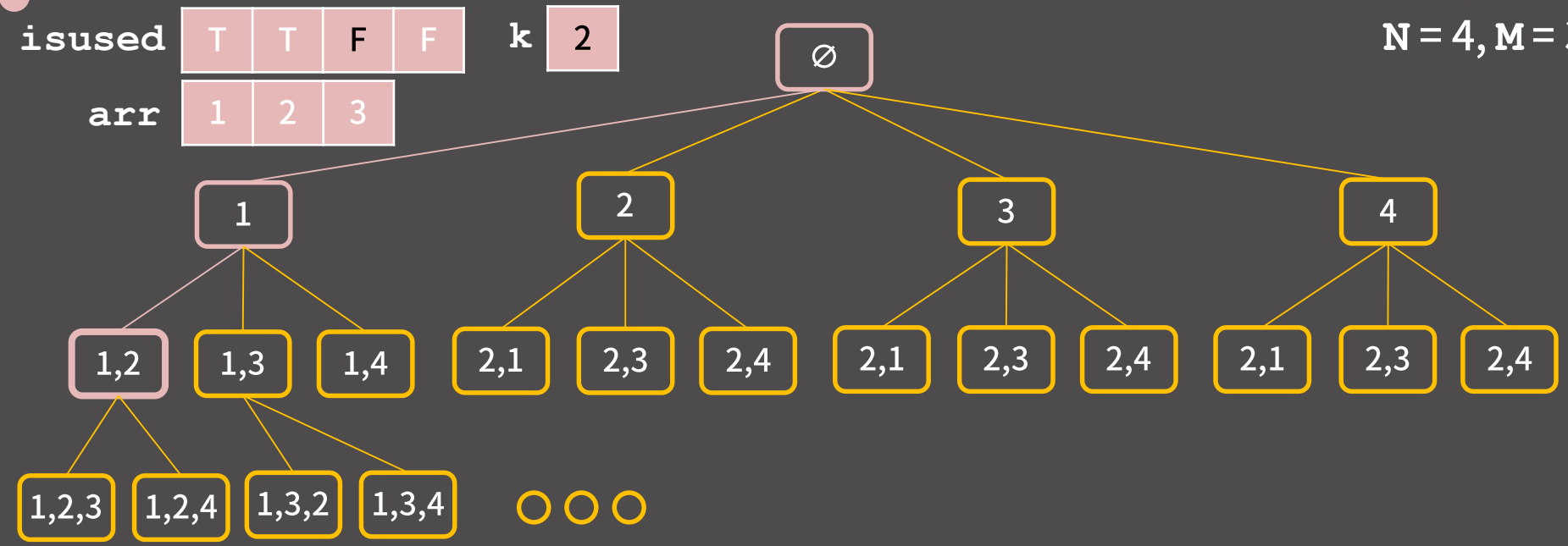
- 리스트의 2번째 원소에 넣을 수를 정합시다. 1은 사용중이므로 불가능하고, 2는 사용되지 않았으므로 2를 넣습니다.

0x00 백트래킹(Backtracking) - 예시



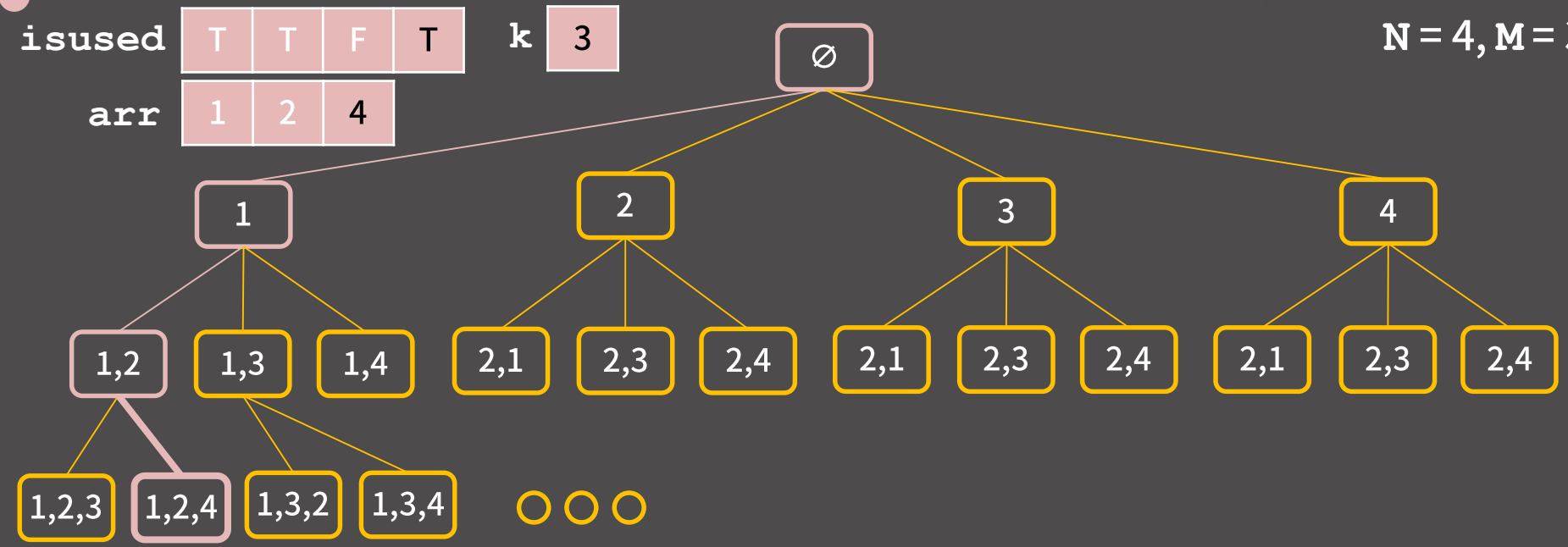
- 리스트의 3번째 원소에 넣을 수를 정합시다. 1과 2는 사용중이므로 불가능하고, 3은 사용되지 않았으므로 3을 넣습니다. 리스트의 크기가 3이 되었으므로 (1, 2, 3)을 출력합니다.

0x00 백트래킹(Backtracking) - 예시



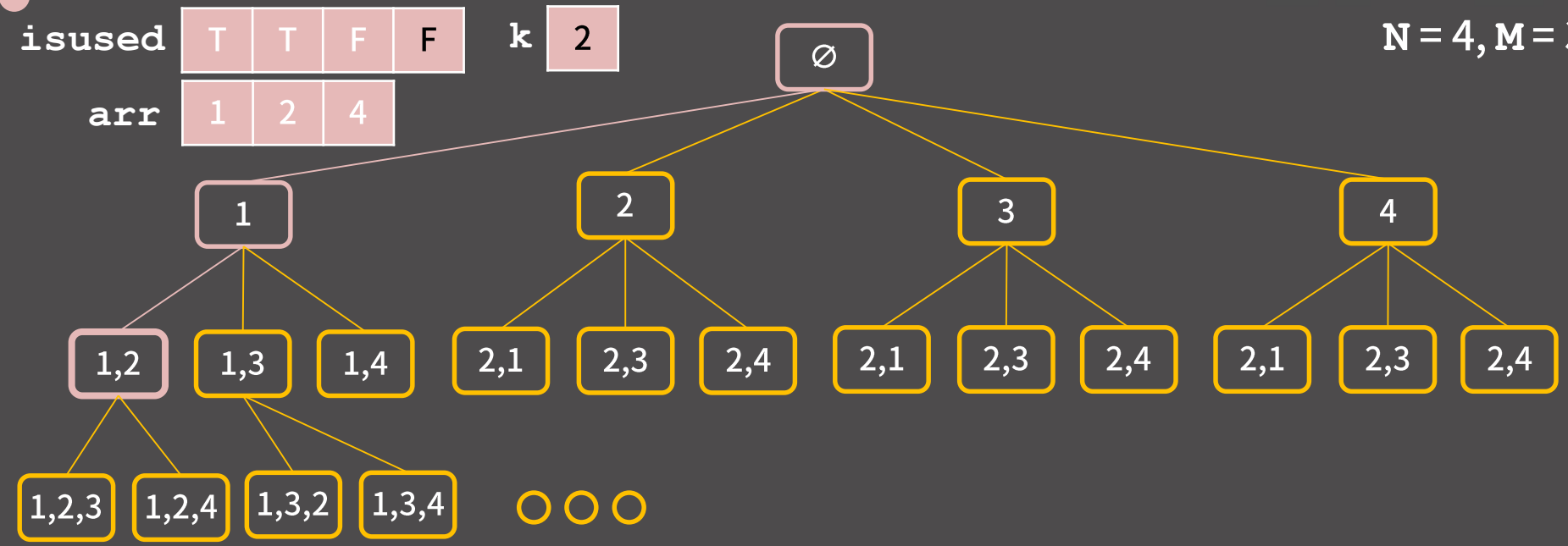
- 한 칸 뒤로 갑니다.(=3번째 원소를 뺍니다.) 이 때 3이 빠졌으므로 반드시 `isused[3]`을 `False`로 만들어주고 `k`를 1 감소시켜야 합니다. 단 `arr[3]`은 어차피 덮어써질 값이므로 신경쓰지 않아도 됩니다.

0x00 백트래킹(Backtracking) - 예시



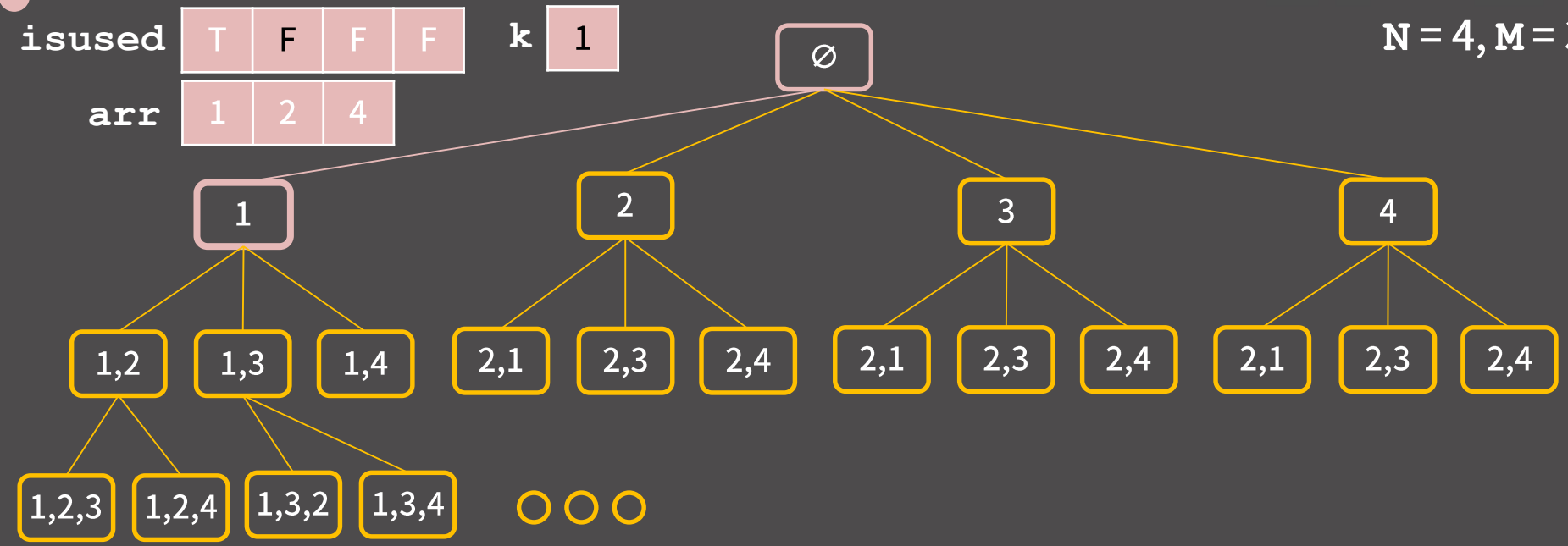
- 리스트의 3번째 원소에 넣을 수를 정합시다. 1, 2는 사용중이고, 3은 이전에 넣어봤으므로 4를 넣습니다. 리스트의 크기가 3이 되었으므로 (1, 2, 4)를 출력합니다.

0x00 백트래킹(Backtracking) - 예시



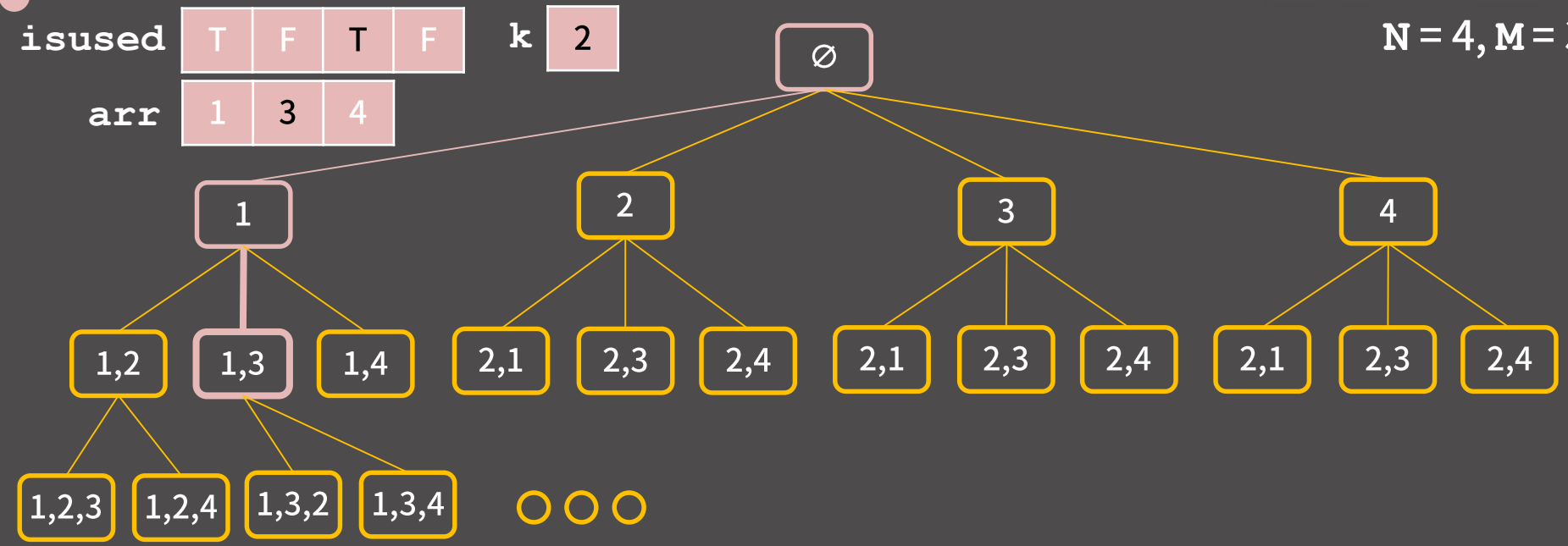
- 한 칸 뒤로 갑니다. 리스트의 3번째 원소에 넣을 수를 정하고 싶은데 1, 2는 사용중이고, 3, 4는 이전에 넣어봤으므로 더 이상 넣을 수가 없습니다.

0x00 백트래킹(Backtracking) - 예시



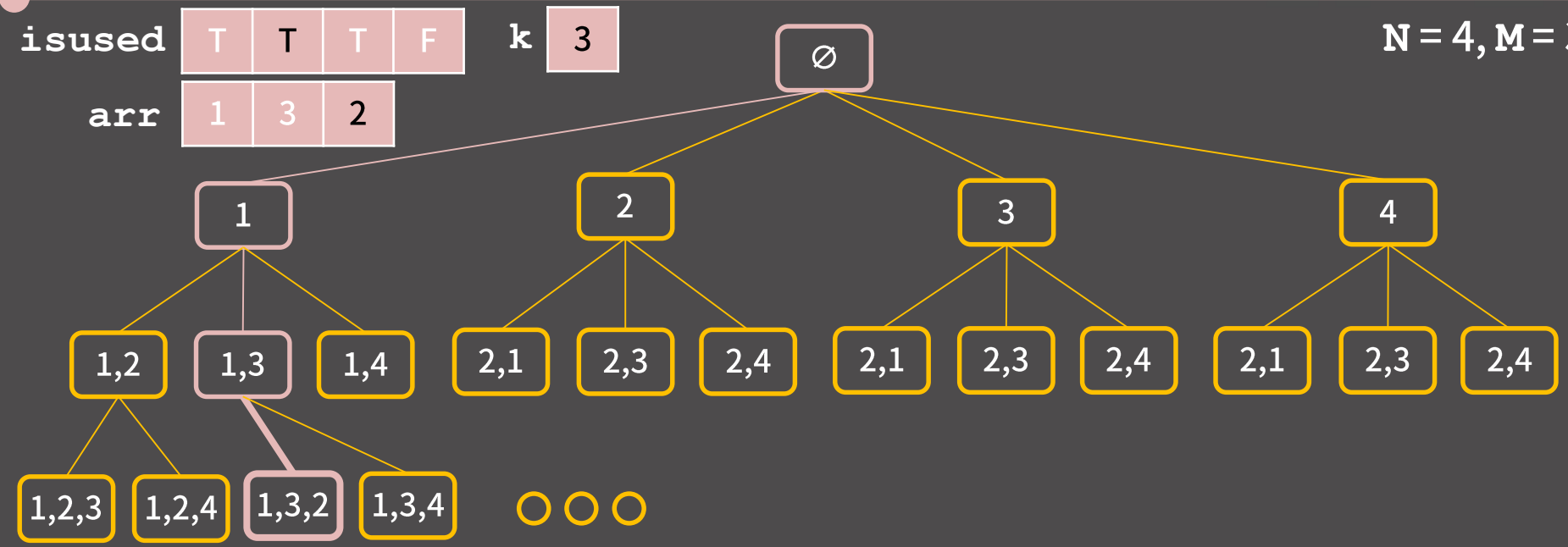
- 다시 한 칸 뒤로 갑니다.

0x00 백트래킹(Backtracking) - 예시



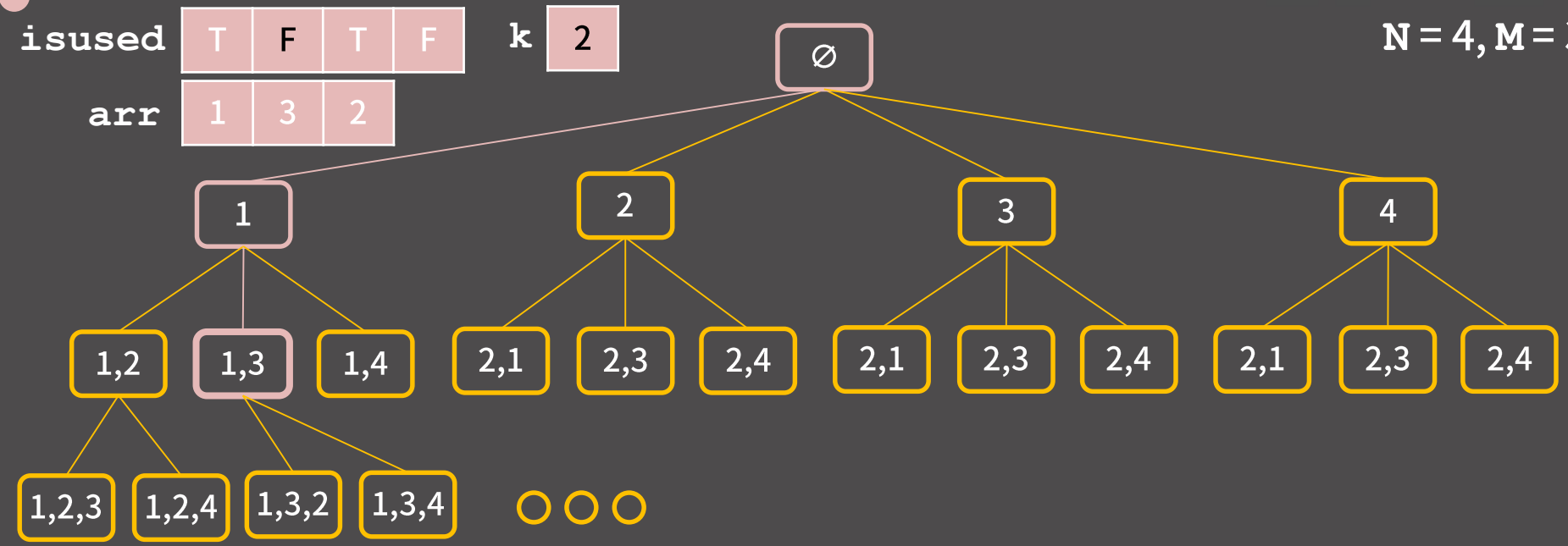
- 리스트의 2번째 원소에 넣을 수를 정합시다. 1은 사용중이고, 2는 이전에 넣어봤으므로 3를 넣습니다.

0x00 백트래킹(Backtracking) - 예시



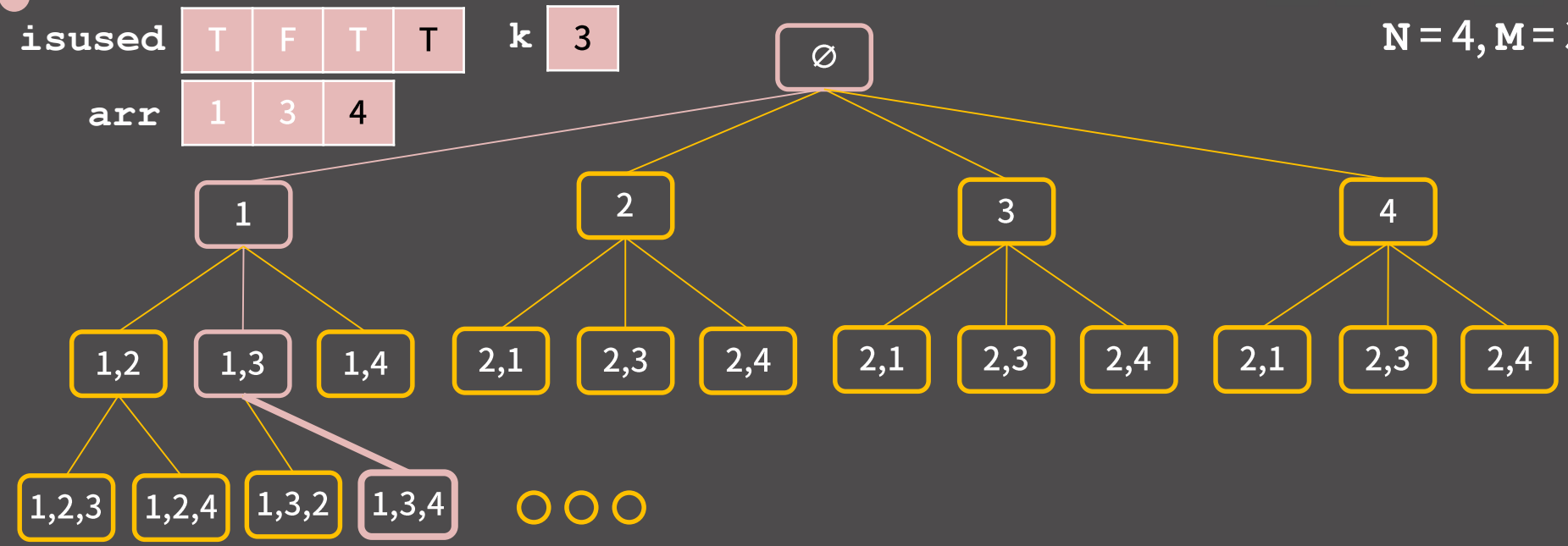
- 리스트의 3번째 원소에 넣을 수를 정합시다. 1은 사용중이므로 2를 넣습니다. 리스트의 크기가 3이 되었으므로 (1, 3, 2)를 출력합니다.

0x00 백트래킹(Backtracking) - 예시



- 한 칸 뒤로 갑니다.

0x00 백트래킹(Backtracking) - 예시



- 리스트의 3번째 원소에 넣을 수를 정합시다. 1, 3은 사용중이고 2는 이전에 넣어봤으므로 4를 넣습니다. 리스트의 크기가 3이 되었으므로 (1, 3, 4)를 출력합니다.

0x00 백트래킹(Backtracking) - 예시

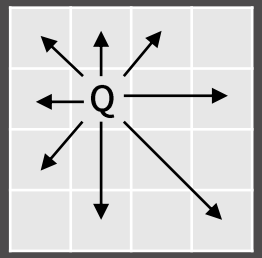
- 백트래킹을 이번에 처음 공부할 경우, 동작하는 원리를 시각적으로 이해한 것과 별개로 실제로 코드를 구현하는건 굉장히 까다롭고, 예시 코드를 보더라도 이해하는게 굉장히 힘들 것입니다.
- 처음 봤을 때 낯설게 느껴지고 이해가 잘 안가는 것이 정상입니다. 그러니 코드가 어렵더라도 너무 상심하지 말고 조급해하지도 말고 천천히 이해를 해보려고 시도해봅시다.
- 정답 코드 : http://boj.kr/afafdc9f833b49368a1b5634161886ef
- `arr`와 `isused`를 vector로 두어도 상관없고, 또 함수 인자로 주고 받는 대신 전역변수로 선언해도 상관 없습니다.

0x00 백트래킹(Backtracking) - 예시

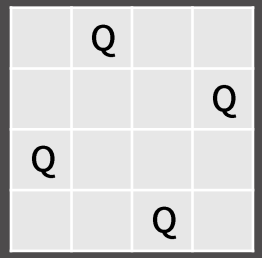
- BOJ 15652번 : N과 M (4) 에서는 수가 중복될 수 있고 고른 수열은 비내림차순이라는 조건이 추가되었습니다. 그렇기 때문에 `isused`가 필요없고 재귀 함수 내의 for문이 0부터 시작하는 대신 `arr[k-1]`부터 시작하면 됩니다.
- 정답 코드 : http://boj.kr/c76662cdbf0a4b45a07f9fee415c4b58
- N과 M 시리즈 12문제를 다 풀어보면 좋으나 9, 10, 11, 12는 어려울 수도 있습니다. 기본기를 다진다고 생각하고 되는데까지 쭉 풀어보세요.

0x00 백트래킹(Backtracking) - 예시

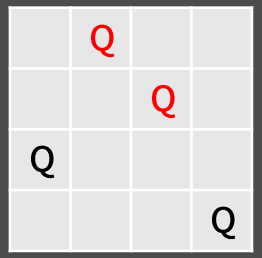
- 이제 본격적으로 백트래킹을 이용해서 해결해야하는 문제를 봅시다.
- BOJ 9663번 : N-Queen 문제입니다. $N \times N$ 체스판에 퀸 N개를 서로 공격하지 못하는 위치에 놓는 경우의 수를 구하는 문제입니다.



퀸의 공격 방향



가능한 배치

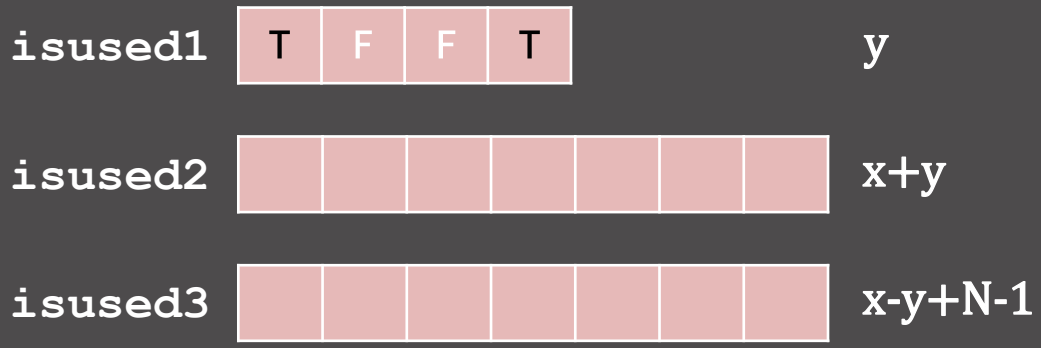


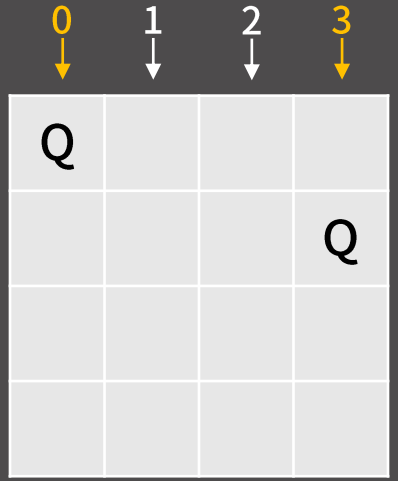
불가능한 배치

- 각 행에 퀸이 정확히 1개씩 있음은 자명합니다. N이 최대 14로 그다지 크지 않기 때문에 각 행에 대해 퀸을 어느 열에 둘지를 가지고 백트래킹을 하면 됩니다.

0x00 백트래킹(Backtracking) - 예시

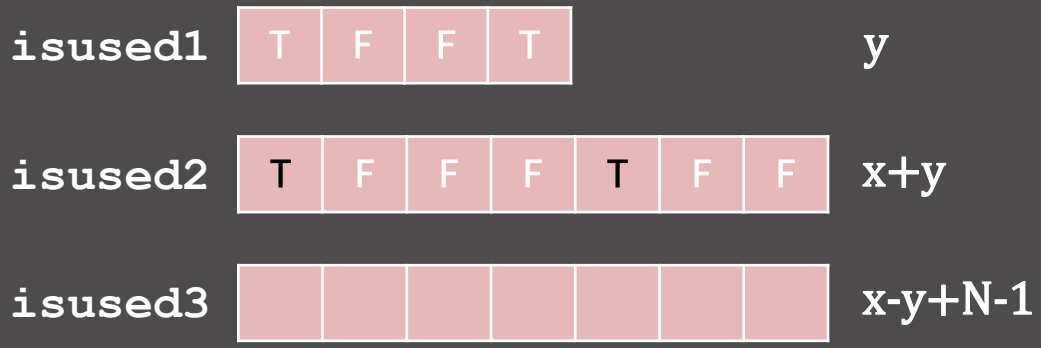
- 백트래킹 과정 중에서 가지치기를 제대로 하지 않으면 O(N^N)이 되어 시간 초과가 발생합니다.
- 두 방향의 대각선과 열에 대해, 다른 퀸이 존재하는지 여부를 따로 저장하고 있어야 합니다.

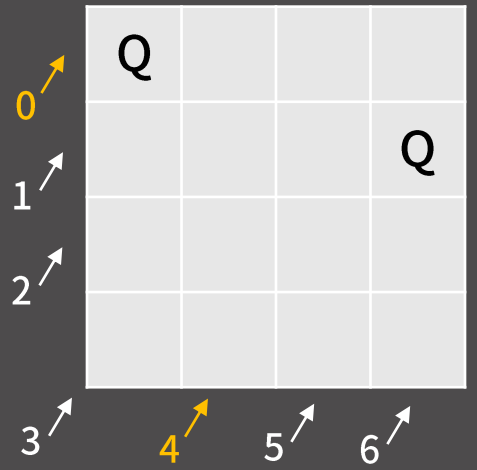




0x00 백트래킹(Backtracking) - 예시

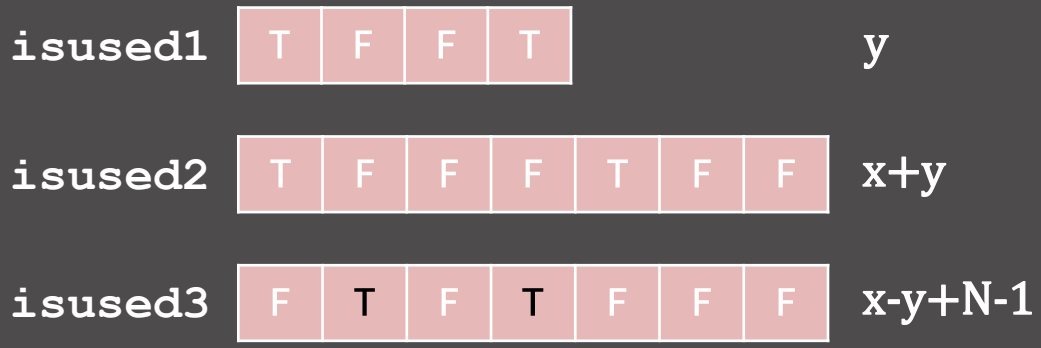
- 백트래킹 과정 중에서 가지치기를 제대로 하지 않으면 O(N^N)이 되어 시간 초과가 발생합니다.
- 두 방향의 대각선과 열에 대해, 다른 퀸이 존재하는지 여부를 따로 저장하고 있어야 합니다.

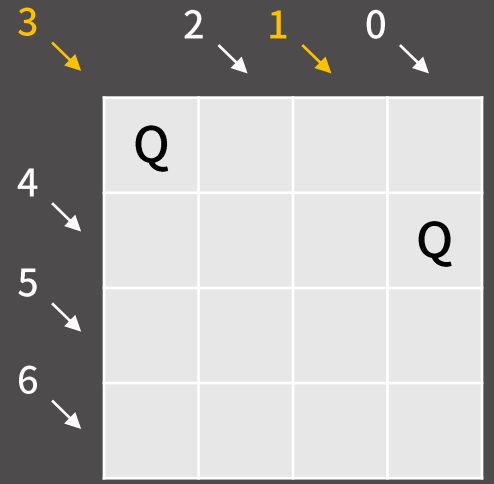




0x00 백트래킹(Backtracking) - 예시

- 백트래킹 과정 중에서 가지치기를 제대로 하지 않으면 $O(N^N)$이 되어 시간 초과가 발생합니다.
- 두 방향의 대각선과 열에 대해, 다른 퀸이 존재하는지 여부를 따로 저장하고 있어야 합니다.





0x00 백트래킹(Backtracking) - 예시

- 정답 코드 : http://boj.kr/46028c7244ad4fe7ae89bcd621cd546c
- 이번에는 `isused1,2,3` 을 전역변수로 두었습니다.

0x00 백트래킹(Backtracking) – 자주 실수하는 요소

- 1. 가지치기를 많이 하지 않아서 시간초과가 발생한다.
- 2. 한단계 더 들어갈 때 가능한 값인지 불가능한 값인지를 비효율적으로 판단해 시간초과가 발생한다.(예를 들어 N-Queen 문제에서 `isused`를 따로 두지 않으면 퀸을 특정 자리에 둘 수 있는지 판단하는데 O(1) 대신 O(N)이 걸린다.)
- 3. 재귀를 들어갔다가 탈출할 때 `isused`와 같은 값을 제대로 바꿔놓지 않아서 잘못된 답을 출력한다.
- 4. 배열 대신 vector로 `isused`와 같은 값을 관리할 때 함수의 인자로 참조자를 넘기는게 아니라 직접 vector를 넘겨 시간초과가 발생한다.

```
void func(vector<int> isused){
  //
}

void func(vector<int>& isused){
  //
}
```

0x01 시뮬레이션

- 시뮬레이션은 말 그대로 주어진 문제의 상황을 그대로 따라가며 풀이를 해야 하는 문제입니다.
- 윷놀이를 실제로 구현하기 / 파싱 / 뿌요뿌요에서 연쇄가 몇 단계에 걸쳐 일어나는지 계산하기 등등 워낙 종류가 다양해서 다양한 문제를 풀어보며 구현력을 높이는 방법 말고는 마땅한 대비법이 없습니다.
- 기능 단위로 함수를 적절히 쪼개고 중간 결과를 계속 출력하게끔 하면서 완성을 하면 되긴 하는데 각자 자기에게 맞는 방법대로 구현을 하면 됩니다.

0x01 시뮬레이션

- 문제를 시뮬레이션으로 풀기 전에 시간복잡도를 계산할 수 있어야 합니다. 그렇지 않으면 몇 시간에 걸쳐 코드를 다 짠 후 제출하고 나서야 시뮬레이션으로 풀면 시간초과가 발생함을 알아버리는 상황이 생길 수도 있습니다.
- 시간복잡도를 계산하기 위해 순열과 조합에 대한 개념을 가지고 있어야 합니다.

- Q1. 1부터 N까지의 수로 만들 수 있는 길이 M의 수열의 갯수는? N^M
- Q2. 1부터 N까지의 수로 만들 수 있는 길이 M의 내림차순 수열의 갯수는? ${}_NC_M$
- Q3. 1부터 N까지의 수로 만들 수 있는 길이 M의 비오름차순 수열의 갯수는? ${}_NH_M$
- Q4. 1부터 N까지의 수를 중복없이 사용해 만들 수 있는 길이 M의 수열의 갯수는? ${}_NP_M$

- 모르는 개념이 있다면 따로 공부를 하시는게 좋습니다.

0x01 시뮬레이션 – 팁

- N과 M 문제를 풀었다면 1, 2, 3, 4를 배치해 만들 수 있는 모든 길이 4짜리 수열을 출력하는 문제를 해결할 수 있을 것입니다.
- 그런데 STL에는 next_permutation, prev_permutation 이라는 아주 강력한 함수가 있습니다.
- next_permutation은 인자로 받은 범위의 수열에 대해 사전 순으로 다음 순열을 만들고 true를 반환합니다. 만약 다음 순열이 존재하지 않으면 false를 반환합니다. prev_permutation은 사전 순으로 이전 순열을 만들고 true를 반환합니다. 만약 이전 순열이 존재하지 않으면 false를 반환합니다.
- next_permutation : http://www.cplusplus.com/reference/algorithm/next_permutation/
- prev_permutation : http://www.cplusplus.com/reference/algorithm/prev_permutation/

0x01 시뮬레이션 – 팁

- 1, 2, 3, 4를 배치해 만들 수 있는 모든 길이 4짜리 수열을 출력하는 문제를 next_permutation을 이용해 아래와 같이 정말 간단하게 해결할 수 있습니다.

```
// array
int a[4] = {1,2,3,4};
do{
  for(int i = 0; i < 4; i++) cout << a[i] << ' ';
  cout << '\n';
}while(next_permutation(a,a+4));

// vector
vector<int> b = {1,2,3,4};
do{
  for(int i = 0; i < 4; i++) cout << b[i] << ' ';
  cout << '\n';
}while(next_permutation(b.begin(),b.end()));
```

0x01 시뮬레이션 – 팁

- next_permutation을 응용하면 1부터 n중에 m개의 원소를 뽑을 때에도 이용할 수 있습니다.
- N과 M 시리즈를 next_permutation을 이용해서도 풀어보세요.

```
// 6개의 수에서 순서를 무시하고 4개씩 뽑고싶은 상황
int a[6] = {6,21,34,37,51,57};
int select[6] = {0,0,1,1,1,1};
do{
  for(int i = 0; i < 6; i++){
    if(select[i]) cout << a[i] << ' ';
  }
  cout << '\n';
}while(next_permutation(select,select+6));
```

```
// 6개의 수에서 순서를 고려해 3개씩 뽑고싶은 상황.
// (6 21 34와 21 6 34를 다른 수열로 취급)
int a[6] = {6,21,34,37,51,57};
int select[6] = {0,0,0,1,2,3};
do{
  int seq[3] = {};
  for(int i = 0; i < 6; i++){
    if(select[i]) seq[select[i]-1] = a[i];
  }
  cout << seq[0] << ' ' << seq[1] << ' ' << seq[2] << '\n';
}while(next_permutation(select,select+6));
```

0x01 시뮬레이션 – 예시

- BOJ 1182번 : 부분집합의 합 문제를 풀어봅시다.
- 원소가 N개인 집합에서 부분집합의 원소는 $O(2^N)$개이므로 모든 부분집합에 대해 합을 계산해보면 됩니다.
- 모든 부분집합을 구하기 위해 이진수를 이용하는 테크닉을 쓰면 20중 for문을 짤 필요 없이 해결할 수 있습니다.
- 정답 코드 : http://boj.kr/842af94c5aa246f6b8eff604053cfb0f
- 테크닉에 대한 설명은 다음 장에 있습니다.

0x01 시뮬레이션 – 예시

- N = 3일 경우

A[2]	A[1]	A[0]		
0	0	0	0	→ 공집합
1	0	0	1	→ A[0]
2	0	1	0	→ A[1]
3	0	1	1	→ A[0],A[1]
4	1	0	0	→ A[2]
5	1	0	1	→ A[0],A[2]
6	1	1	0	→ A[1],A[2]
7	1	1	1	→ A[0],A[1],A[2]

```cpp
int a[3] = {2, 7, 9};
for(int i = 0; i < 8; i++){
  cout << "{ ";
  int tmp = i;
  for(int j = 0; j < 3; j++){
    if(tmp % 2 == 1) cout << a[j] << ' ';
    tmp /= 2;
  }
  cout << "}\n";
}
```

```
{ }
{ 2 }
{ 7 }
{ 2 7 }
{ 9 }
{ 2 9 }
{ 7 9 }
{ 2 7 9 }
```

강의 정리

- 백트래킹, 시뮬레이션에 대해 공부했습니다.
- 지금까지 배운 내용 중에 가장 큰 고비이니 잘 이겨내시고 문제를 많이 풀어보셔서 숙달하시길 바랍니다.
- 특히 삼성 역량테스트 A형 통과를 목표로 하시는 분들은 거의 모든 문제가 BFS/DFS/백트래킹/시뮬레이션/전수조사 범주를 벗어나지 않기 때문에 0x05강과 이번 0x07강을 중점적으로 보고 그룹 문제집에 들어있는 문제를 전부 다 풀어보면 충분한 대비가 될 것입니다.