



2주차 알고리즘 세미나

PoolC 2021 1학기

문제 요약 :

- 1~1,000,000자리 양의 정수 X가 주어진다
- X가 한자리 수가 될 때까지 변환을 한다
- 변환한 수가 만약 한자리일때? 3,6,9이면 X는 3의 배수
- 첫 줄에는 변환 횟수를, 두번째 줄에는 3의 배수인지 아닌지 출력

고려해야 할 사항 :

- X는 자연수이지만 아주 큰 수가 될 수도 있으니 정수형이 아닌 **문자열**로 입력을 받아야 함
- 재귀함수의 **종료조건**이 무엇일까?
- 변환은 X의 각 자리 숫자들을 다 더한숫자 Y로 변환하는 것이다!

● ● 지난시간 도전문제 - 3의 배수




출력 :

3
NO

3 -> 변환 횟수, 다시 말해서 재귀의 깊이

NO -> 한자리수가 3,6,9 셋 중 하나인지

 -> 재귀함수의 종료조건, X의 Length가 1

```
void func (string X, int depth)
```

```
    종료 조건(X.length == 1)
```

```
    변환
```

```
    func(Y, depth+1)
```

문제 요약 :

- $N \times N$ 크기의 행렬이 주어진다. (N 은 3^7 보다 작거나 같은 3^k 꼴 자연수)
- 1. 종이에 모두 같은 수가 적혀 있으면 종이 그대로 사용
- 2. 아니면 종이를 균일하게 9등분한다 .
- 3. 1,2 반복

고려해야 할 사항 :

- 9등분을 어떻게 구현해야 할까?
- 재귀함수의 종료조건이 무엇일까?
- 숫자별로 종이의 개수를 저장할 변수가 필요하다!

지난시간 도전문제 - 종이의 개수

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



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

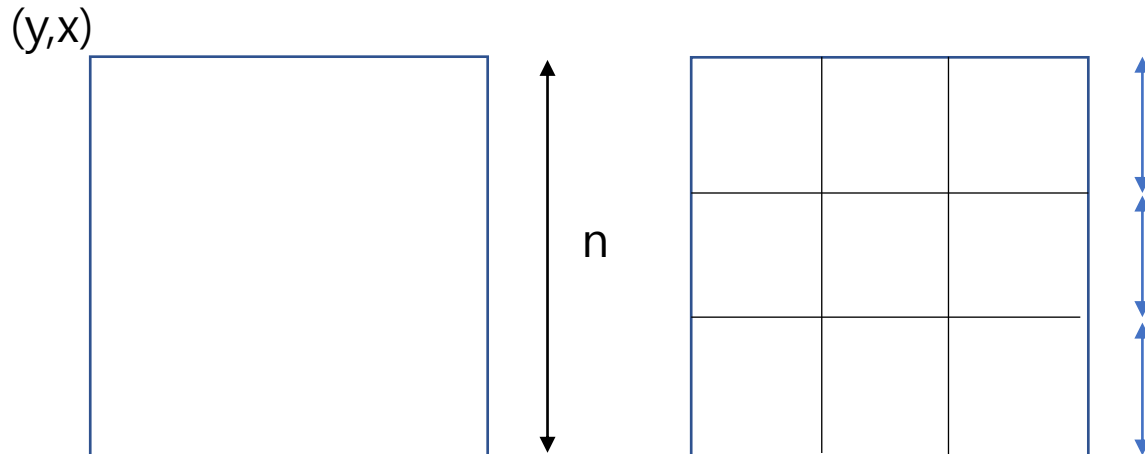
출력 :

10
12
11

지난시간 도전문제 - 종이의 개수

재귀함수를 작성하기에 앞서

- 재귀함수가 다루는 종이의 정보전달
- 재귀함수의 자료형
- 종료조건
- 현재 종이가 모두 같은 숫자로 이루어져 있는지? check



```
void func (int y, int x, int n)
```

종료 조건($n == 1$)

check

`func(y, x, n/3)`

`func(y, x+n/3, n/3)`

...

`func(y+n/3*2, x+n/3*2, n/3)`

9등분



지난시간 도전문제 – 색종이 만들기

문제 요약 :

- N, r, c 가 주어지며 $1 \leq N \leq 15, 0 \leq r < 2^N, 0 \leq c < 2^N$
- Z순으로 방문할 때 r 행의 c 열은 언제 방문될까?
- 2,1,3,4 분면 순으로 방문한다.
- $N > 1$ 이면 4등분 한다.

고려해야 할 사항 :

- 실제로 r 행 c 열까지 방문하면서 풀 수 있을까?
- 재귀함수의 종료조건이 무엇일까?

지난시간 도전문제 - Z

- 0행 0열부터 r행 c열까지 Z순서대로 쪽 탐색?

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

- 최악의 경우 : $O(2^{30}) > O(10^8)$

- 0.5초 시간제한에 x

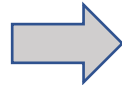
- 모든 칸을 탐색 하지 않아도 되는 방법?

- 정사각형의 길이가 2^N 꼴일 때
모든 칸을 꼭 한번씩 탐색하게 된다!

지난시간 도전문제 - Z

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

4등분



전체 숫자의 개수 : 2^8

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

2^4

32	33	36	37
34	35	38	39
40	41	44	45
42	43	46	47

2^4

16	17	20	21
18	19	22	23
24	25	28	29
26	27	30	31

2^4

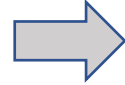
48	49	52	53
50	51	54	55
56	57	60	61
58	59	62	63

2^4

지난시간 도전문제 - Z

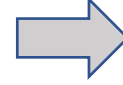
32	33	36	37
34	35	38	39
40	41	44	45
42	43	46	47

4등분



32	33
34	35
2^2	
40	41
42	43
2^2	

36	37
38	39
2^2	
44	45
46	47
2^2	

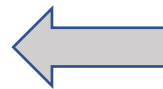


40	41
42	43

방문한 숫자의 개수 : $2^4 + 2^4 + 2^2 + 2^2$



40	41
2^0	2^0
42	43
2^0	2^0

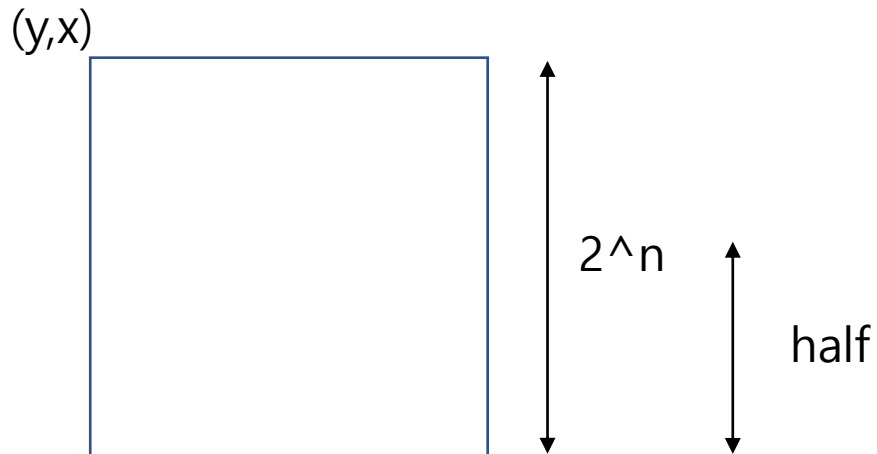


r행 c열까지 오는데 방문한 숫자의 개수 :

$$2^4 + 2^4 + 2^2 + 2^2 + 2^0 =$$

재귀함수를 작성하기에 앞서

- 재귀함수가 다루는 영역의 정보전달
- 재귀함수의 자료형
- 종료조건
- 건너뛰는 숫자들의 개수 계산



```
int func (int y, int x, int n)
```

종료 조건($n == 0$)

```
return func(y, x, n-1)
return func(y, x+half, n-1) + skip
return func(y+half, x, n-1) + skip * 2
return func(y+half, x+half, n-1) + skip * 3
```

완전탐색이란?

- 문제상황에서 가능한 경우의 수를 모두 탐색하는 방법
- 컴퓨터의 빠른 계산 능력을 이용하자

언제 써야 할까?

- 모든 경우의 수를 훑는 연산이 시간제한내에 충분히 이루어 질 수 있을 때
- 문제 이해가 잘 되지 않을 때

특징?

- 모든 경우의 수를 다 고려하기 때문에 틀릴 수 없다. 하지만 시간은?
- 완전탐색만으로 문제를 풀 수 없는 경우가 많다.

완전탐색(*Brute Force*)

예제 : 1~1000까지 숫자중에서 x를 찾는 Up&Down 게임 진행.

1	2	...	x	...	999	1000
---	---	-----	---	-----	-----	------

여러가지 전략 : 절반씩 끊어서 찾기, 짝수만 찾기, 100단위로 끊어서 찾기, ...

무식하게 풀기(*Brute Force*) : 1부터 오름차순으로 1000까지 일일이 다 찾기

x는 무조건 찾을 텐데, 1000까지가 아닌 100억 까지라면?

완전탐색(Brute Force)

1~1000까지 숫자중에서 x를 찾는 Up&Down 게임 진행.

1	2	...	x	...	999	1000
---	---	-----	---	-----	-----	------

절반씩 끊어서 찾기 VS 1~1000까지 일일이 찾기

시간	$O(\log n)$	$O(n)$
구현	까다롭다	단순 for문
발상	어렵다	자연스럽다

쉽게 풀 수 있으면
쉽게 푸는게 장땡

완전탐색을 사용할 때 생각해보면 좋은 것들

1. 코드 최적화를 통해 시간을 좀 더 줄일 수 있을까?

- 가지치기, 거꾸로 생각하기, 전처리 등...

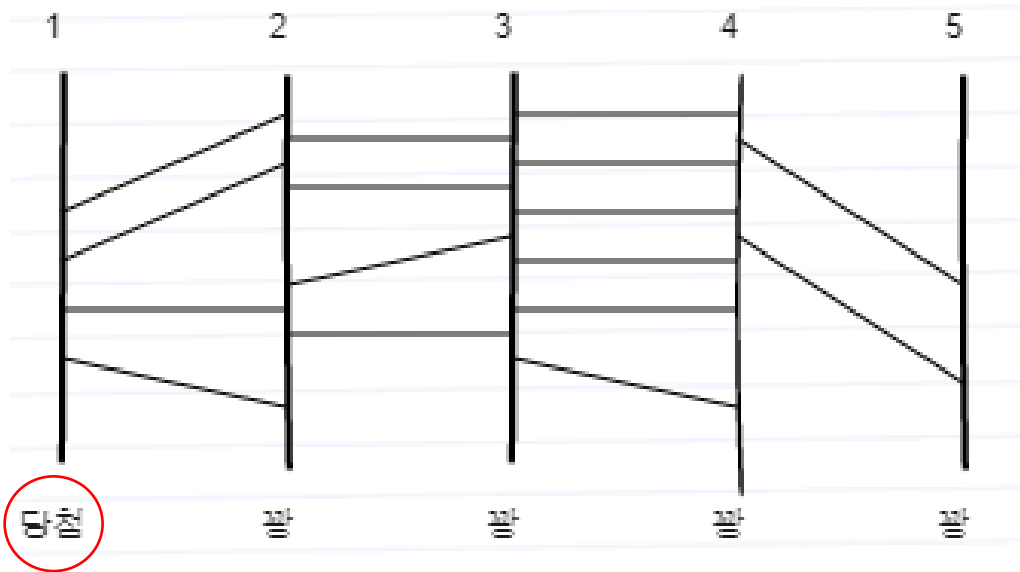
2. 정말 내 코드가 완전하게 모든 경우의 수를 탐색하는 것일까?

- 놓치기 쉬운 부분, 구현의 중요성

완전탐색(Brute Force) - 최적화

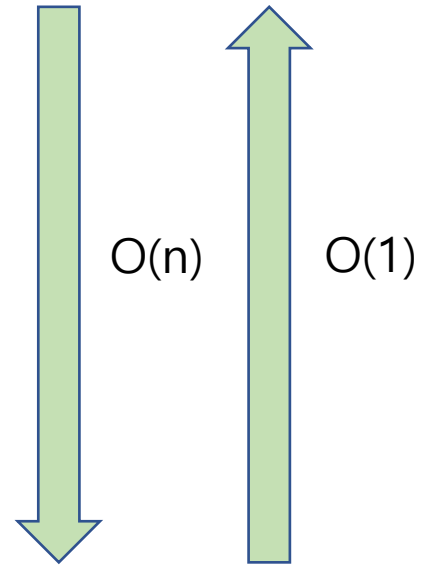
1. 거꾸로 생각하기

- 누가 당첨일까?



2. 전처리

- 누적합
- Z문제
- 정렬



2. 전처리

- 누적합

- Z문제, 팩토리얼 계산 : 계산에 필요한 값을 미리 저장해 놓기

$$5! = 120$$

$$6! = 6 \times 5!$$

완전탐색(Brute Force) - 최적화

2. 전처리

- 정렬 각 배열에서 하나의 원소를 골라 그 둘의 합의 최대값?

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

2	10	5	8	7	8	3	4
---	----	---	---	---	---	---	---

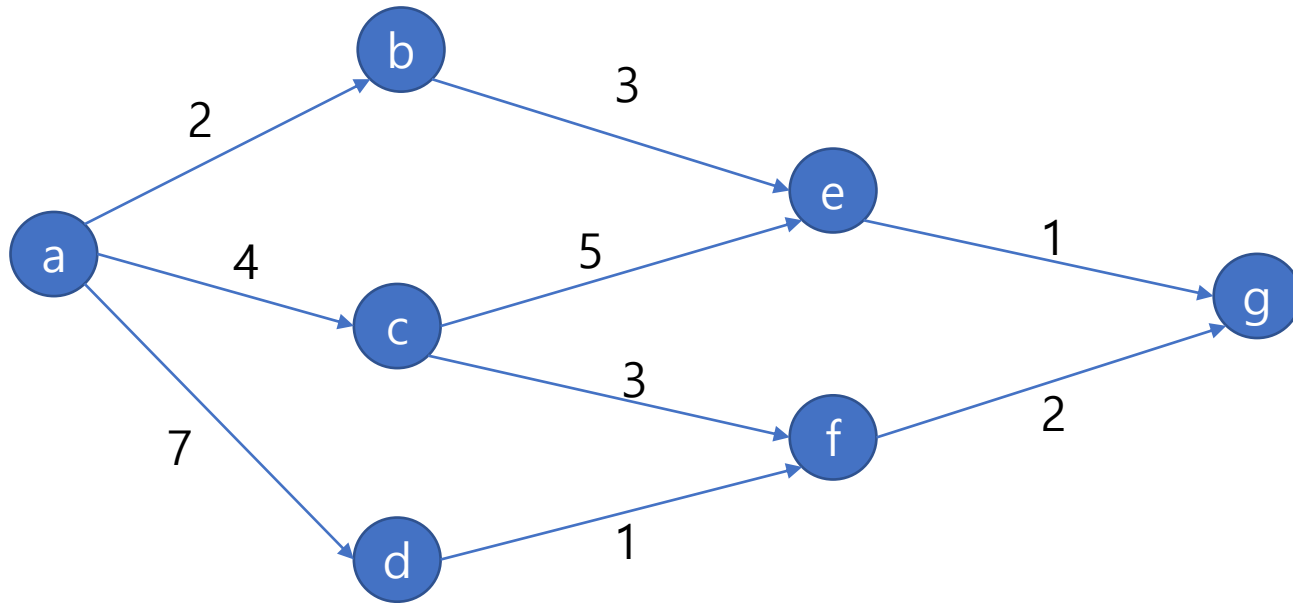
$O(n^2)$

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

2	3	4	5	7	8	8	10
---	---	---	---	---	---	---	----

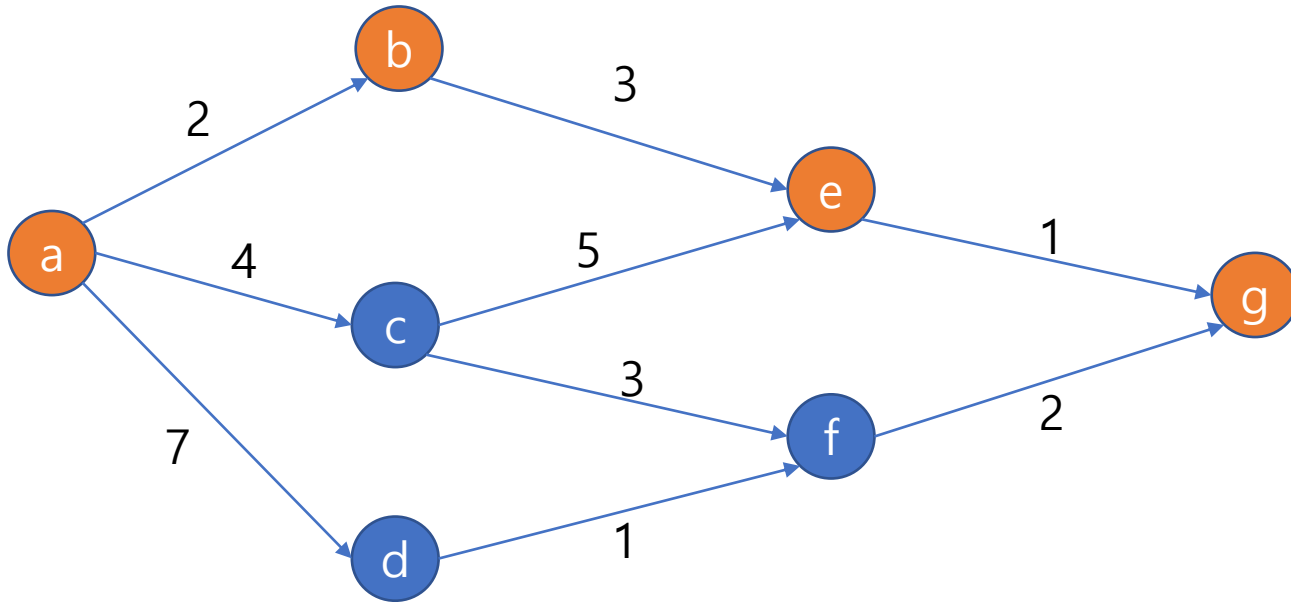
$O(n)$

3. 가지치기



- a부터 g까지 가는 최단경로를 구하기

3. 가지치기

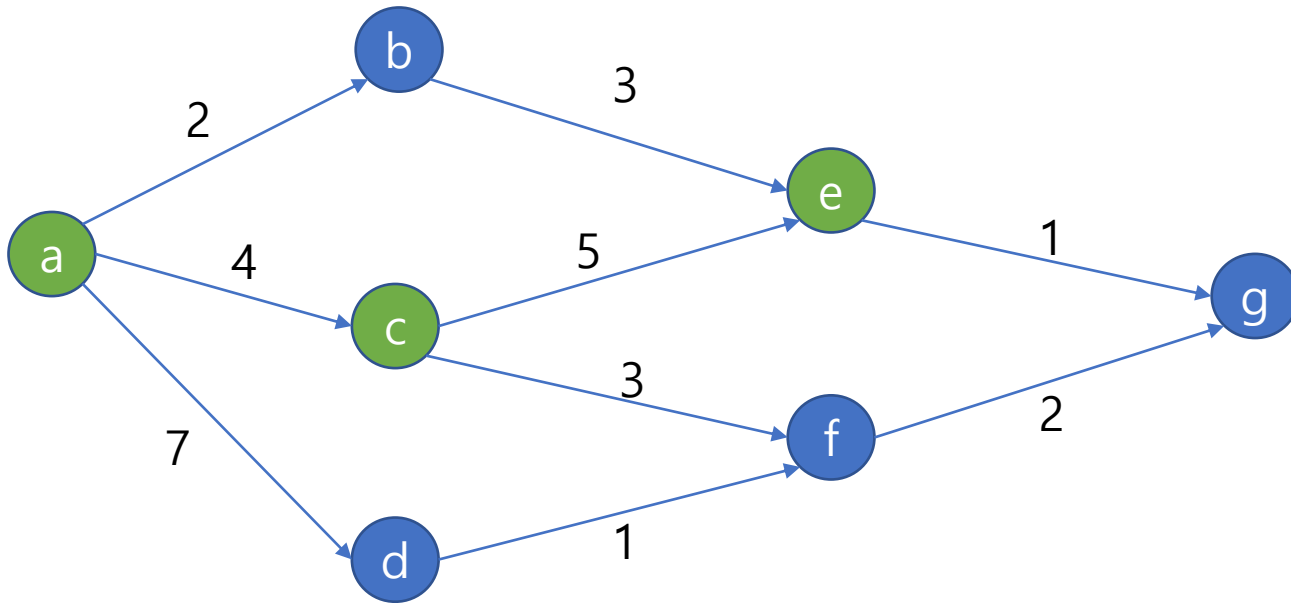


- a부터 g까지 가는 최단경로를 구하기

$$2+3+1 = 7$$

전역변수 int short_path : 7

3. 가지치기



- a부터 g까지 가는 최단경로를 구하기

- e까지 경로의 길이는? **9** > **7**

- e->g 경로를 볼 필요도 없이 탐색 종료

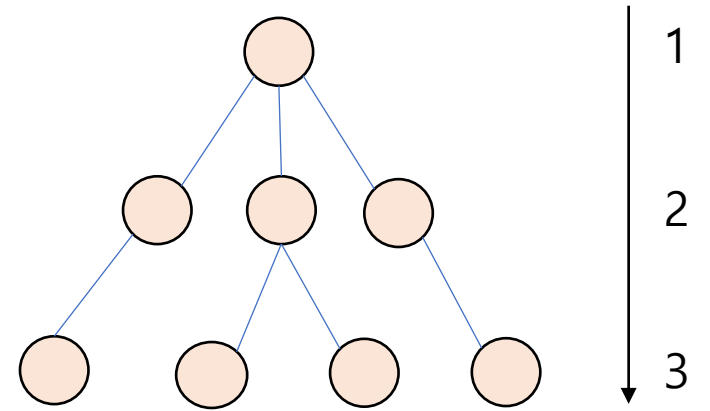
전역변수의 유용함

효율적인 시간 향상

완전탐색은 대충 알겠는데 백트래킹이 뭘까?

1. 완전탐색을 효율적으로 구현 가능하게 해준다.
2. 트리 자료구조에서 유용하다.

트리란?



계층구조로 나타낼 수 있는 자료구조

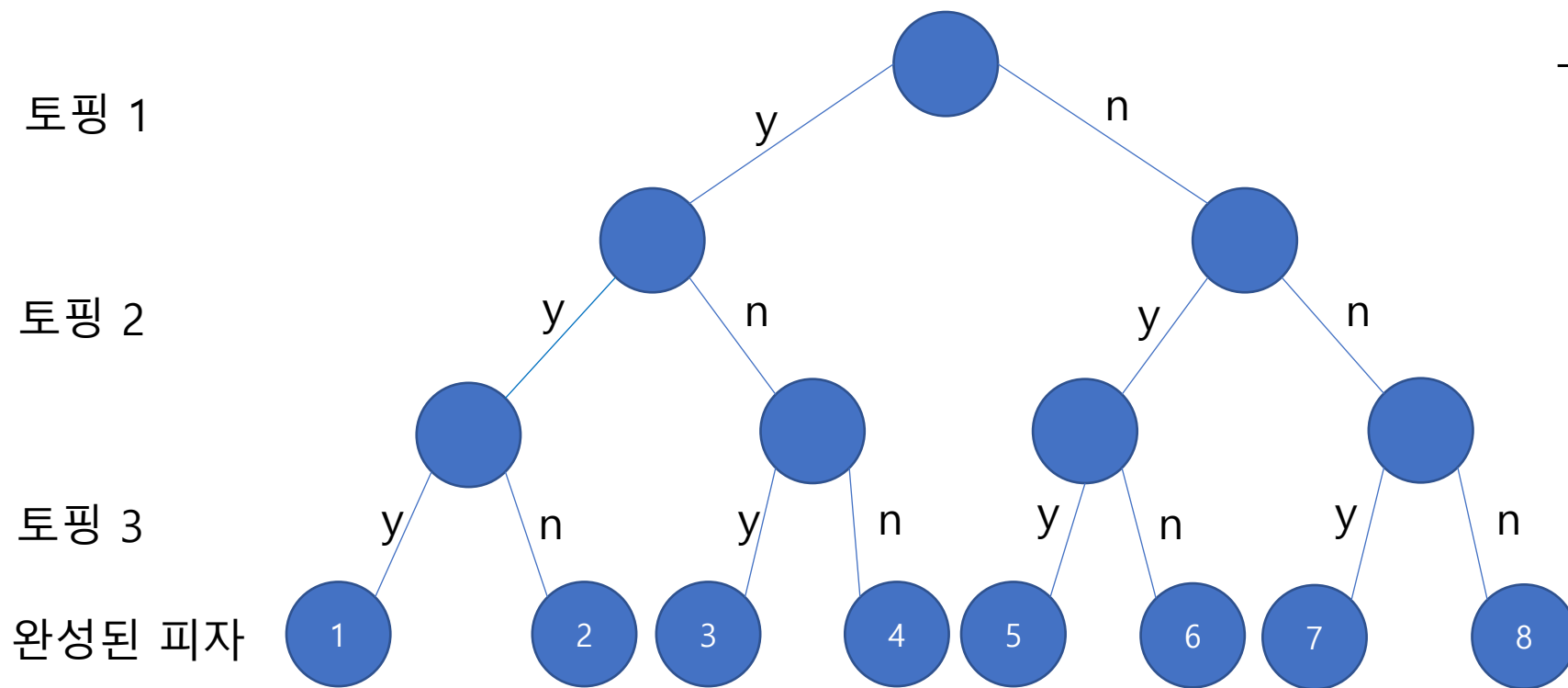
나무를 거꾸로 뒤집은 모양이라 해서 트리

백트래킹(Backtracking)

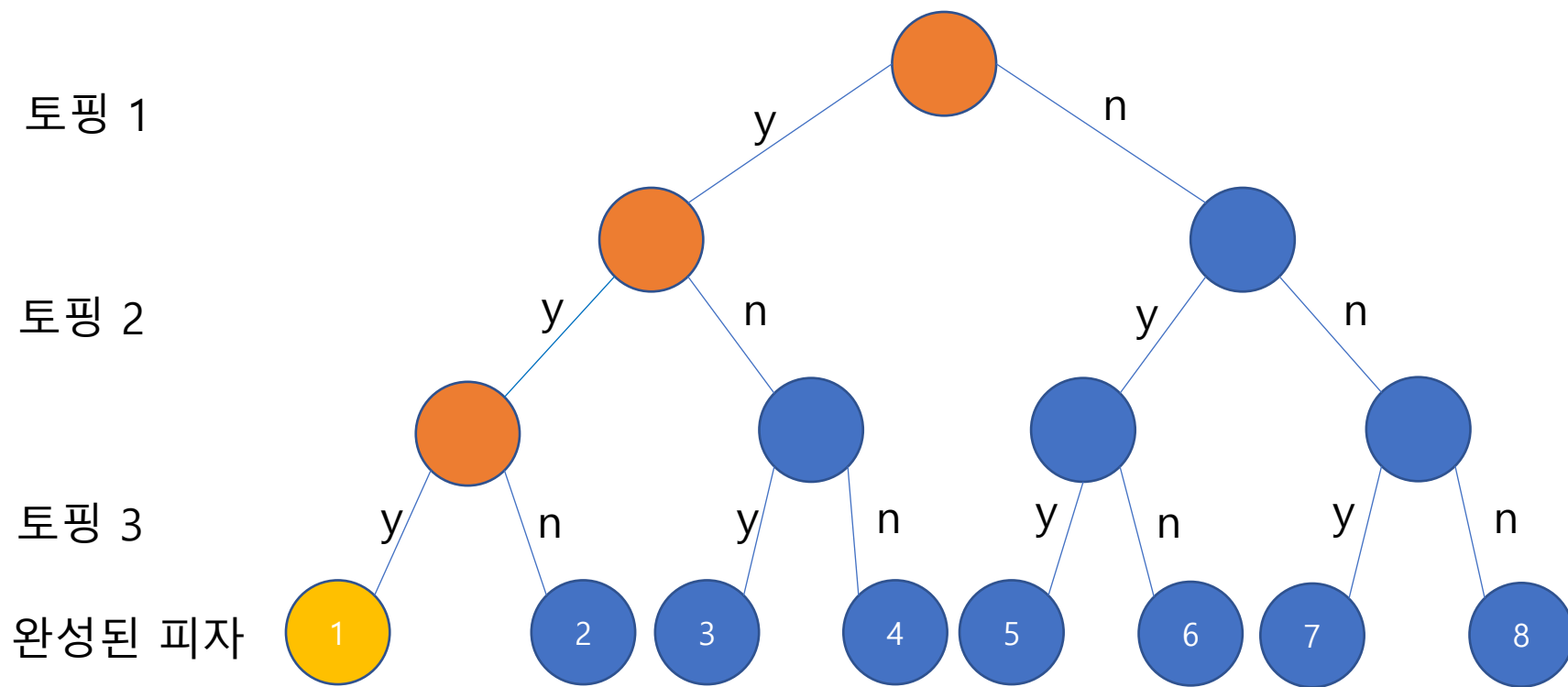
예제 : 피자집 사장님이 제일 맛있는 피자를 발명하게 도와주자!

- 3가지 토핑이 존재

- 1~8번 피자를 다 만들어 본다

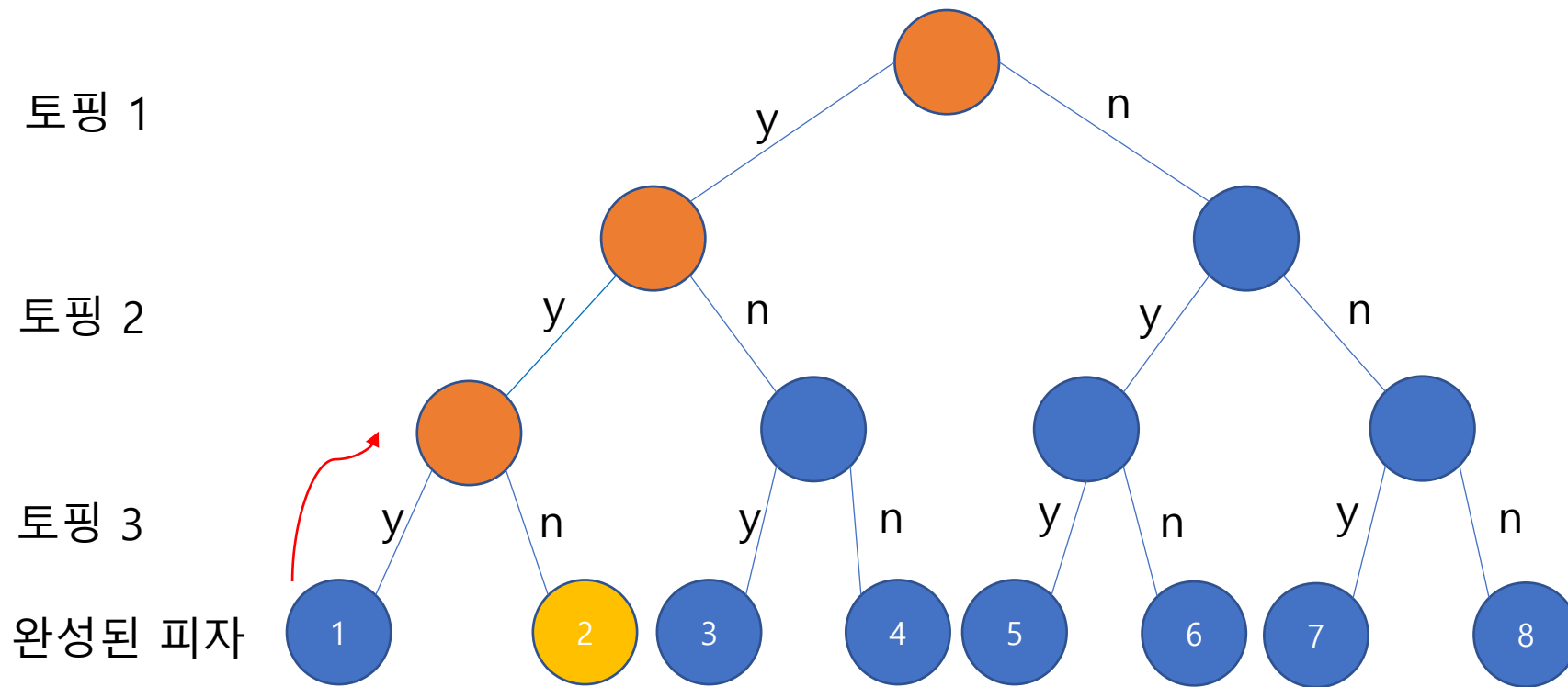


백트래킹(Backtracking)



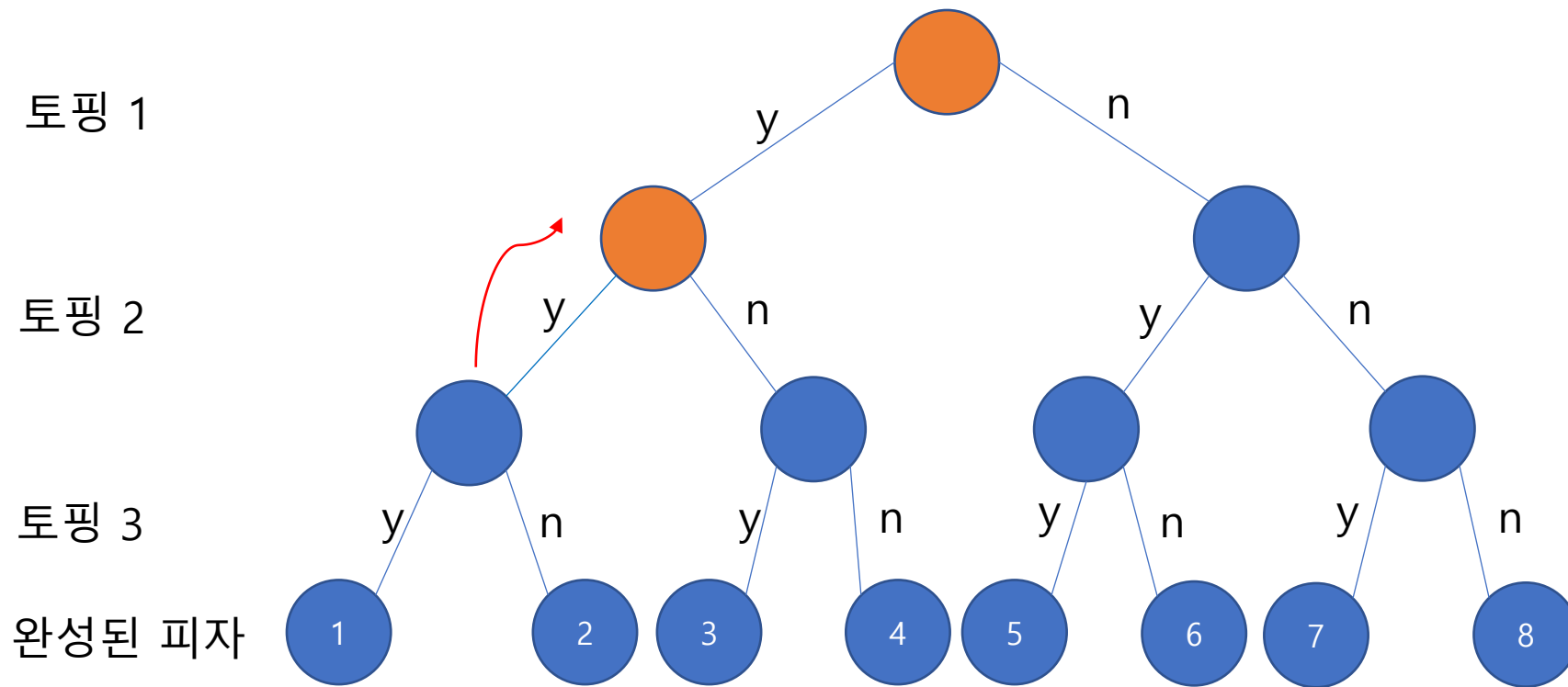
백트래킹(Backtracking)

빨간 선 : 백트래킹

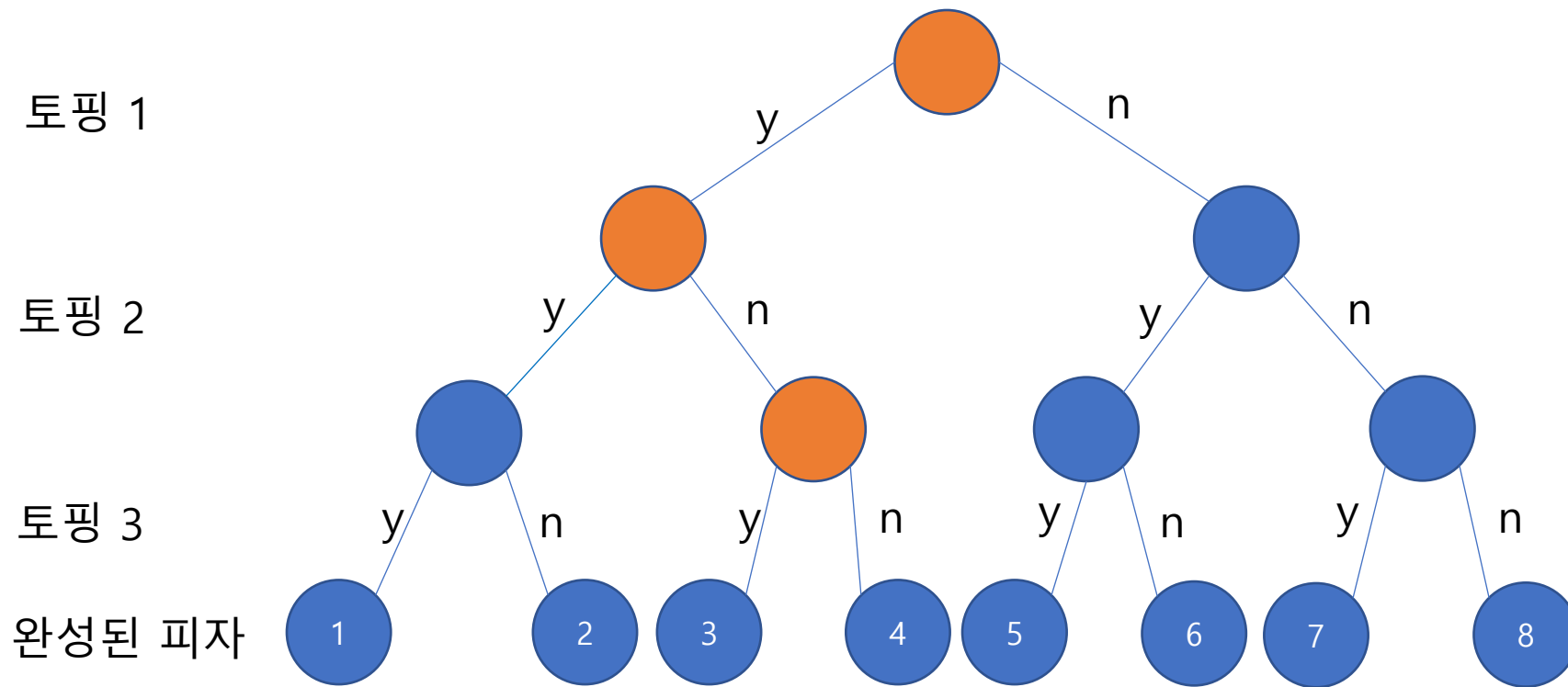


백트래킹(Backtracking)

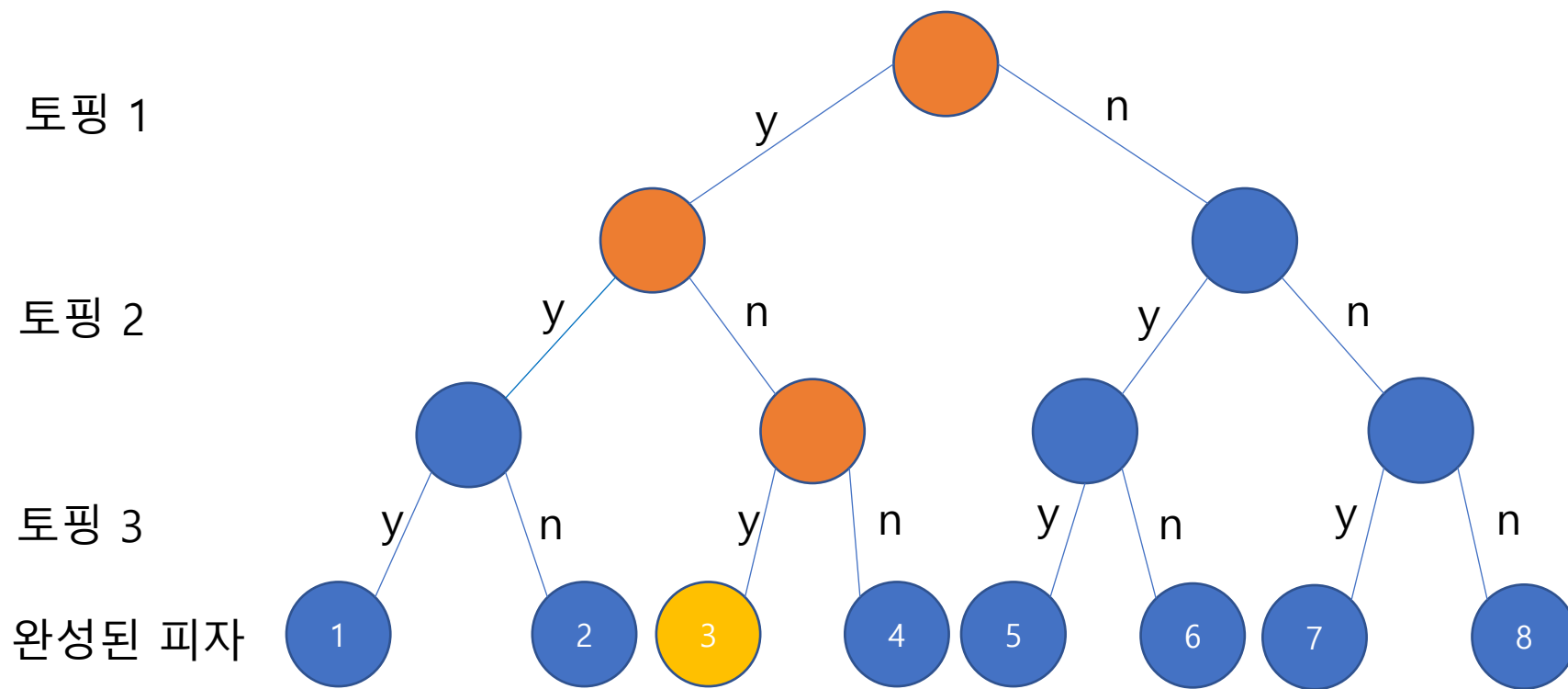
빨간 선 : 백트래킹



백트래킹(Backtracking)



백트래킹(Backtracking)

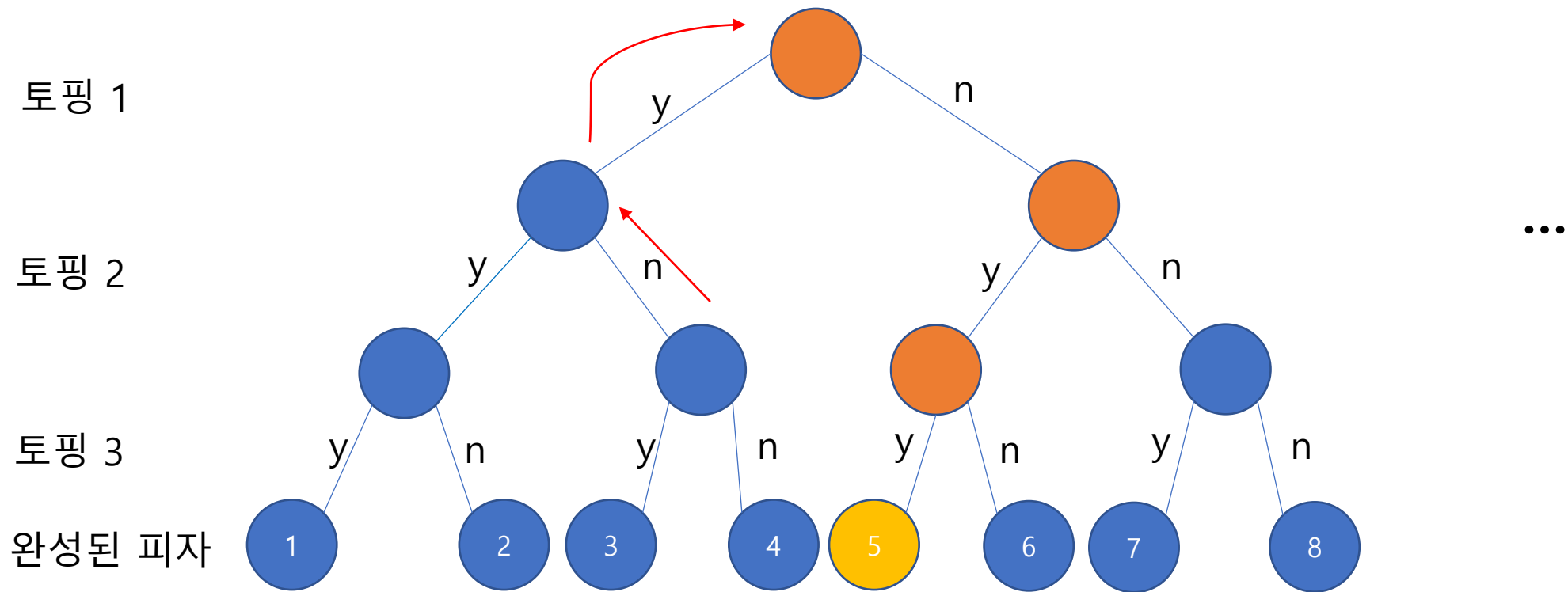


빨간 선 : 백트래킹



백트래킹(Backtracking)

빨간 선 : 백트래킹



백트래킹(Backtracking)

백트래킹 : 상태의 변화를 주고 재귀호출 했을 때, 변화를 주기 전의 상태로 되돌리는 것

이전 단계로 돌아가려면?

`list.push_back(n), toppings[n] = true, sum += yummy[n]`

`list.pop_back(), toppings[n] = false, sum -= yummy[n]`

```
void back_tracking(int n)
```

종료 조건(`n == 4`)

n번째 토핑 추가

```
back_tracking(n+1)
```

n번째 토핑 제거

```
back_tracking(n+1)
```


백트래킹(Backtracking)

백트래킹 : 상태의 변화를 주고 재귀호출 했을 때, 변화를 주기 전의 상태로 되돌리는 것

이전 단계로 돌아가려면?

```
#include <iostream>
#include <vector>

using namespace std;

vector<int> list;

void pick()
{
    if (list.size() == 5)
    {
        for (int i = 0; i < list.size(); i++)
        {
            cout << list[i] << ' ';
        }
        cout << endl;
        return ;
    }
    for (int i = 0; i <= 8; i++)
    {
        list.push_back(i);
        pick();
        list.pop_back();
    }
}

int main()
{
    pick();
    return 0;
}
```

종료조건

void pick()

종료 조건(list.size == 5)

list의 끝에 숫자 추가

pick()

list의 끝의 숫자 제거

● ● 완전탐색을 이용하는 문제 소개

1. 덩치 : <https://www.acmicpc.net/problem/7568>
2. N과M : <https://www.acmicpc.net/problem/15649>
3. 스타트와 링크 : <https://www.acmicpc.net/problem/14889>
4. 외판원 순회2 : <https://www.acmicpc.net/problem/10971>

문제 요약 :

<https://www.acmicpc.net/problem/7568>

- $2 \leq N \leq 50$, $10 \leq x, y \leq 200$
- A의 덩치 (x, y) , B의 덩치 (p, q) 라 했을 때, $x > p \ \&\& \ y > q$ 이면 A가 B보다 덩치가 크다
- 덩치 등수 k 는 그 사람보다 덩치가 큰 사람 수 + 1
- 여러 사람이 같은 덩치 등수를 가질 수 있다.

고려해야 할 사항 :

- $O(N^2)$ 의 시간복잡도로 통과 가능한가?
- 자기보다 덩치가 큰 사람의 수를 저장할 변수가 필요
- 키와 몸무게 2개의 변수를 효율적으로 저장하는 방법 : pair

완전탐색을 이용하는 문제 - 덩치

이름	(몸무게, 키)	덩치 등수
A	(55, 185)	2
B	(58, 183)	2
C	(88, 186)	1
D	(60, 175)	2
E	(46, 155)	5

비교할 수 있는 모든 경우의 수 : N^2 가지

- 따라서 완전탐색으로 모두 비교!
- 구현이 까다로운 재귀보다는 단순한 for문

```
for(int i=1; i<=N; i++)  
    for(int j=1; j<=N; j++)
```

j덩치를 i덩치와 비교

i의 등수 조정

문제 요약 :

<https://www.acmicpc.net/problem/15649>

- $1 \leq M \leq N \leq 8$
- 1부터 N 까지 자연수 중에서 중복 없이 M 개를 고른 수열을 오름차순으로 출력
- 각 수열은 공백으로 구분해서 출력

고려해야 할 사항 :

- 중복 없이 M 개를 고르려면?

완전탐색을 이용하는 문제 - N과 M

- 중복 없이 M개를 고르려면?
 - x라는 숫자가 list에 있는지 확인

list를 일일이 순회하면서 x가 있나 찾아보기

selected[x]의 값?

```
void pick()
```

종료조건 (list.size == M)

```
for(int i=1; i<=N; i++)
```

!selected[i]이면

selected[i] = true

list에 i추가

```
pick()
```

selected[i] = false

list에서 i삭제

```
#include <iostream>
#include <vector>

using namespace std;

vector<int> list;

void pick()
{
    if (list.size() == 5)
    {
        for (int i = 0; i < list.size(); i++)
        {
            cout << list[i] << ' ';
        }
        cout << endl;
        return ;
    }
    for (int i = 0; i <= 8; i++)
    {
        list.push_back(i);
        pick();
        list.pop_back();
    }
}

int main()
{
    pick();
    return 0;
}
```

● ● 완전탐색을 이용하는 문제 - 스타트와 링크(삼성 SW테스트 기출)

문제 요약 :

<https://www.acmicpc.net/problem/14889>

- $4 \leq N \leq 20$, N 은 짝수 , $1 \leq S_{ij} \leq 100$
- 각각 $N/2$ 명으로 2팀을 만들어서 팀 간의 능력치 차이를 최소값을 구해라
- 팀의 능력치는 S_{ij} 의 합으로 결정

고려해야 할 사항 :

- 전체 시간 복잡도
- 한 팀을 뽑았을 때 나머지 팀 뽑기 구현
- 최소값을 나타내는 전역변수 설정

● ● 완전탐색을 이용하는 문제 - 스타트와 링크(삼성 SW테스트 기출)

- N명의 사람중에서 N/2명 팀을 꾸리는 경우의 수 : ${}_N C_{N/2} \leq {}_{20} C_{10} \leq 200,000$
- 스타트팀을 기준으로 완전탐색후 비교!
- 1~N 자연수 중에서 중복없이 N/2개를 뽑는것과 동일하다!

x~N번째 선수를 뽑는 재귀 함수

- 링크팀은 어떻게 뽑아야 할까?
 - 스타트팀의 여집합. How?
 - 누가 스타트팀에 뽑혔는지 저장
- 최소값의 초기값은 얼마로 해야 할까?
 - 무한대를 나타내는 값

```
void pick(int x)
```

```
    종료조건 (x>N OR team.size == N/2)
```

```
        팀의 능력치 계산, 최소값 갱신
```

```
        x번째 선수 영입
```

```
        pick(x+1)
```

```
        x번째 선수 방출
```

```
        pick(x+1)
```

백트래킹

● ● 완전탐색을 이용하는 문제 - 스타트와 링크(삼성 SW테스트 기출)

- 스타트팀을 뽑고나서 링크팀은 어떻게 뽑아야 할까?
 - 링크팀은 스타트팀에 대해서 종속적인 팀. 스타트 팀이 정해져야 링크팀도 정해지므로, 백트래킹으로 새로운 스타트팀을 뽑을 때마다 링크팀도 바뀌어야 함.
 - 따라서 링크팀을 완성하고 차이의 최소값을 구한다음 링크팀은 초기화 되어야 한다
- 최소값의 초기값은 얼마로 해야 할까?
 - int : -2,147,483,648 ~ 2,147,483,647
 - min : 0 ~ 10,000,000(?)
 - INF : 987,654,321

문제 요약 :

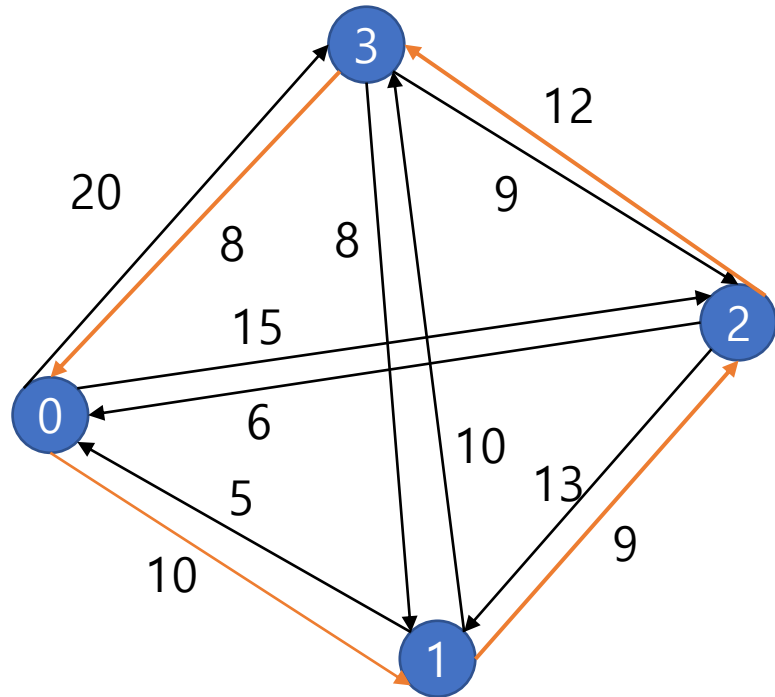
<https://www.acmicpc.net/problem/10971>

- $2 \leq N \leq 10$, $0 \leq W[i][j] \leq 1,000,000$
- 행렬의 값은 도시-도시 간의 비용을 나타낸다. 만약 0이면 갈 수 없는 경우
- 처음 도시에서 쭉 순회 후 다시 처음으로 돌아오는 비용의 최소값을 구해라
- 처음 도시를 제외하고 한번 갔던 도시는 다시 갈 수 없다.

고려해야 할 사항 :

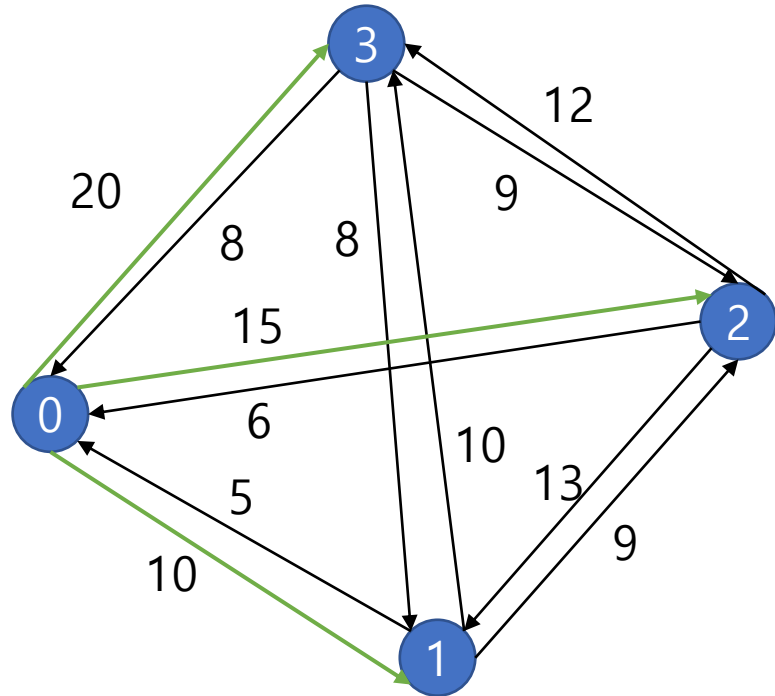
- 시작 도시가 안 주어져 있다.
- 방문한지 여부를 체크하는 변수
- 함수의 자료형

완전탐색을 이용하는 문제 - 외판원 순회2



- 어느 도시에서 시작하나 전체 비용은 동일하다

- 함수의 자료형?



void : 이전 경로의 비용이 현재 재귀함수의 인자로 들어온다.

VS

int : 이전 경로의 비용이 현재 재귀함수의 인자로 들어오지 않고
현재 깊이에서의 최소값만 반환한다.

return min(초록색 선 + func(그 다음 도시))

```
void func(int v, int cnt, int path)
```

```
    종료조건 (cnt == N )
```

```
    MIN = min(MIN, path)
```

```
    for(int i=0; i<N; i++)
```

```
        if (W[v][i] != 0 && !visited[i])
```

```
            visited[i] = true
```

```
            func(i, cnt+1, path+W[v][i])
```

```
            visited[i] = false
```

```
int func(int v, int cnt)
```

```
    종료조건 (cnt == N )
```

```
    for(int i=0; i<N; i++)
```

```
        if (W[v][i] != 0 && !visited[i])
```

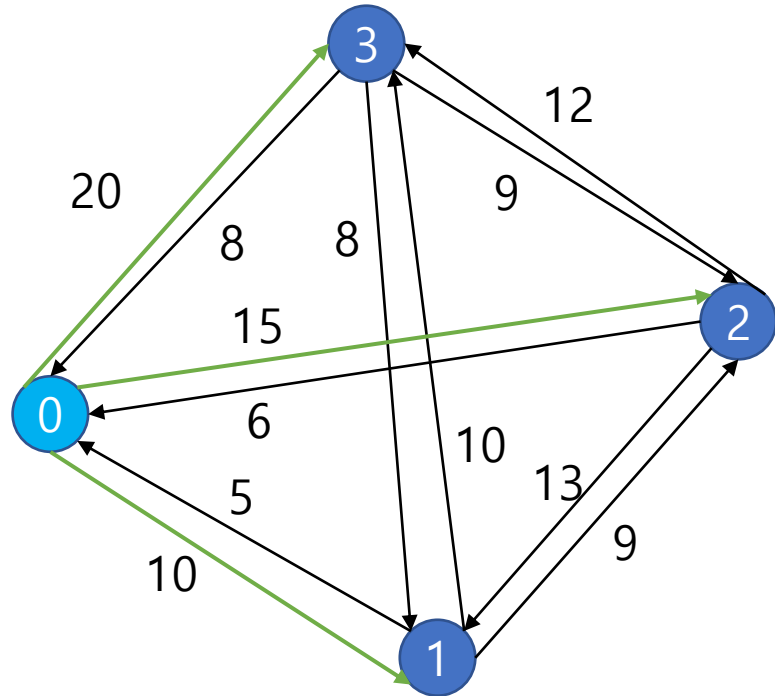
```
            visited[i] = true
```

```
            min(MIN, W[v][i] + func(i, cnt+1))
```

```
            visited[i] = false
```

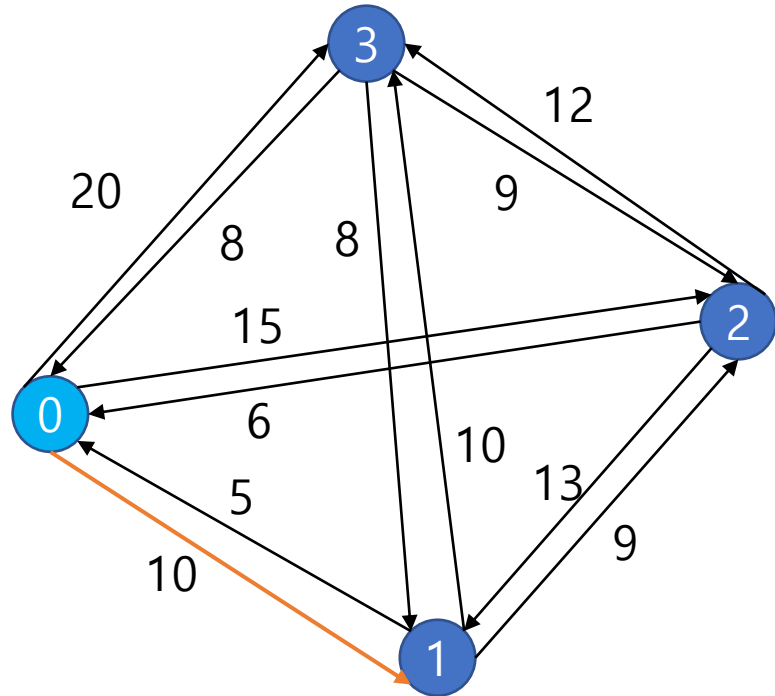
```
    return MIN
```

완전탐색을 이용하는 문제 - 외판원 순회2



- 0을 시작점으로, 시작점은 시작과 동시에 방문처리
- 다음으로 갈 수 있는 도시는 1,2,3 3개의 도시

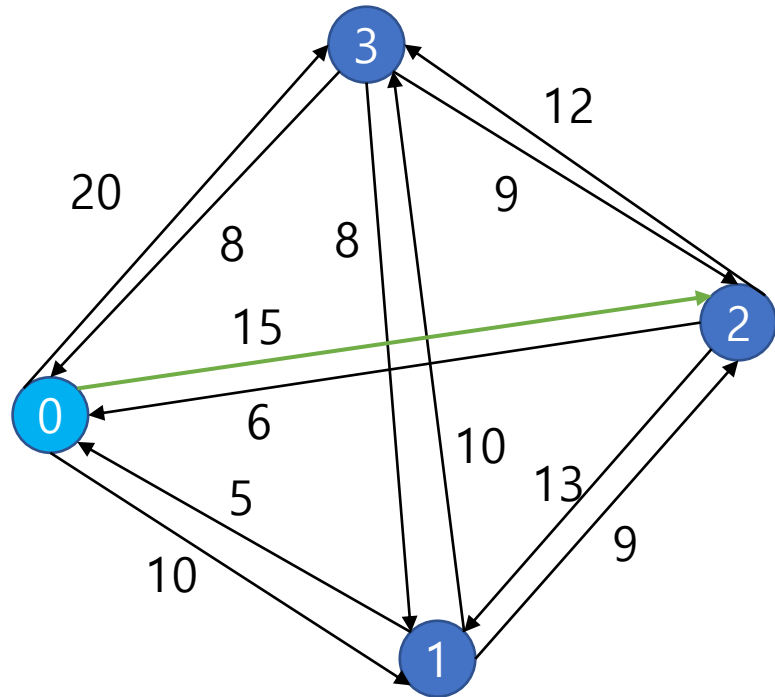
완전탐색을 이용하는 문제 - 외판원 순회2



$$\text{MIN} = \min(\text{MIN}, \text{W}[\text{v}][\text{i}] + \text{func}(\text{i}, \text{cnt}+1))$$

987654321 10 x

완전탐색을 이용하는 문제 - 외판원 순회2



$$\text{MIN} = \min(\text{MIN}, \text{W}[\text{v}][\text{i}] + \text{func}(\text{i}, \text{cnt}+1))$$

987654321

10

x

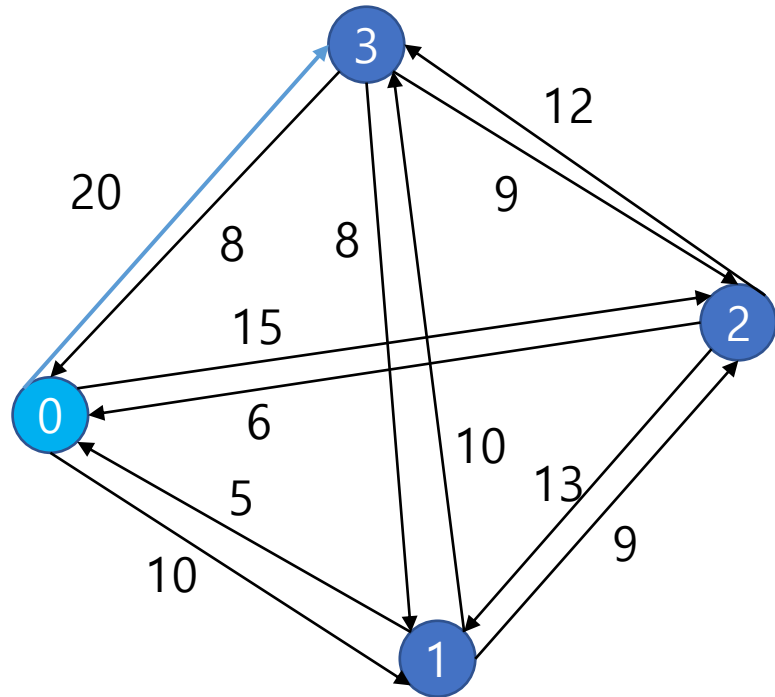
$$\text{MIN} = \min(\text{MIN}, \text{W}[\text{v}][\text{i}] + \text{func}(\text{i}, \text{cnt}+1))$$

10 + x

15

y

완전탐색을 이용하는 문제 - 외판원 순회2



$$\text{MIN} = \min(\text{MIN}, \text{W}[\text{v}][\text{i}] + \text{func}(\text{i}, \text{cnt}+1))$$

987654321
10
x

$$\text{MIN} = \min(\text{MIN}, \text{W}[\text{v}][\text{i}] + \text{func}(\text{i}, \text{cnt}+1))$$

10 + x
15
y

$$\text{MIN} = \min(\text{MIN}, \text{W}[\text{v}][\text{i}] + \text{func}(\text{i}, \text{cnt}+1))$$

min(10+x, 15+y)
20
z

return min(10+x, 15+y, 20+z)



도전 문제!

1. 감시 : <https://www.acmicpc.net/problem/15683>
2. N-Queen : <https://www.acmicpc.net/problem/9663>
3. 치킨 배달 : <https://www.acmicpc.net/problem/15686>
4. 색종이 붙이기 : <https://www.acmicpc.net/problem/17136>

● ● 자유로운 질문 및 토의!