

ICPC Sinchon



2022 Winter Algorithm Camp

6회차. 완전탐색/백트래킹

서강대학교 김성현

2022 Winter Algorithm Camp

6회차. 완전탐색/백트래킹

목차

1. 앞으로의 방향
2. 완전탐색의 개념
3. 완전탐색에 대한 발상
4. 완전탐색 - 예시 문제
5. 백트래킹의 개념
6. 재귀에 대한 간단한 소개
7. 백트래킹 - 예시 문제
8. 부록

2022 Winter Algorithm Camp

앞으로의 방향

* 남은 주제

- 완전탐색 + 백트래킹
- 그래프 탐색
- 최단경로 알고리즘

* 남은 강의를 통틀어서 전하고 싶은 것

- '상태'에 대해 생각하면서 문제를 푸는 발상
- 문제의 상황을 어떤 상태들 간의 전이와 그 상태에 대한 검토로 보는 관점

완전 탐색(Brute Force) - 개론

* 완전 탐색이란?

- 말 그대로 가능한 모든 경우를 탐색해 보는 것
- 모든 경우를 탐색해 가면서 무언가를 세거나 어떤 조건이 가능한 경우가 있는지 탐색하는 등의 문제

완전 탐색 - 개론

* 완전 탐색이란?

- 말 그대로 가능한 모든 경우를 탐색해 보는 것
- 모든 경우를 탐색해 가면서 무언가를 세거나 어떤 조건이 가능한 경우가 있는지 탐색하는 등의 문제

* 어떤 문제가 완전 탐색인가?

- 하나의 경우마다 따져야 할 게 명확한 경우
- 어떤 상태를 구성하고, 그 상태가 특정 조건을 만족하는지 검토해야 하는 등의 문제
- 그리고 나올 수 있는 경우의 수가 모두 탐색할 수 있을 만큼 적은 경우

완전 탐색 - 개론

* 완전 탐색 문제를 푸는 법

- 모든 경우(상태)를 생성할 수 있는 방법을 생각한다
- 각각의 경우에 대해 그 문제의 조건을 만족하는지를 체크한다
- 문제에 따라 모든 경우를 따질 필요가 없거나 각 경우에 대해 체크할 조건이 없는 등의 상황도 있다

완전 탐색 - 발상(예시 : 3273. 두 수의 합)

문제

n 개의 서로 다른 양의 정수 a_1, a_2, \dots, a_n 으로 이루어진 수열이 있다. a_i 의 값은 1보다 크거나 같고, 1000000보다 작거나 같은 자연수이다. 자연수 x 가 주어졌을 때, $a_i + a_j = x$ ($1 \leq i < j \leq n$)을 만족하는 (a_i, a_j) 쌍의 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 수열의 크기 n 이 주어진다. 다음 줄에는 수열에 포함되는 수가 주어진다. 셋째 줄에는 x 가 주어진다. ($1 \leq n \leq 100000, 1 \leq x \leq 2000000$)

완전 탐색 - 발상(예시 : 3273. 두 수의 합)

문제

n 개의 서로 다른 양의 정수 a_1, a_2, \dots, a_n 으로 이루어진 수열이 있다. a_i 의 값은 1보다 크거나 같고, 1000000보다 작거나 같은 자연수이다. 자연수 x 가 주어졌을 때, $a_i + a_j = x$ ($1 \leq i < j \leq n$)을 만족하는 (a_i, a_j) 쌍의 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 수열의 크기 n 이 주어진다. 다음 줄에는 수열에 포함되는 수가 주어진다. 셋째 줄에는 x 가 주어진다. ($1 \leq n \leq 100000, 1 \leq x \leq 2000000$)

수열에서 나올 수 있는 모든 $n(n-1)/2$ 개의 쌍에 대해서 다 시험해 보는 방법을 생각할 수 있다.

완전 탐색 - 발상(예시 : 3273. 두 수의 합)

문제

n 개의 서로 다른 양의 정수 a_1, a_2, \dots, a_n 으로 이루어진 수열이 있다. a_i 의 값은 1보다 크거나 같고, 1000000보다 작거나 같은 자연수이다. 자연수 x 가 주어졌을 때, $a_i + a_j = x$ ($1 \leq i < j \leq n$)을 만족하는 (a_i, a_j) 쌍의 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 수열의 크기 n 이 주어진다. 다음 줄에는 수열에 포함되는 수가 주어진다. 셋째 줄에는 x 가 주어진다. ($1 \leq n \leq 100000, 1 \leq x \leq 2000000$)

수열에서 나올 수 있는 모든 $n(n-1)/2$ 개의 쌍에 대해서 다 시험해 보는 방법을 생각할 수 있다.

그런데 10만의 입력 제한에서는 $O(n^2)$ 시간복잡도를 가지는 코드는 통과할 수 없음

완전 탐색 - 발상

* 완전 탐색은 사실상 모든 문제를 ‘풀 수는’ 있음

- 말 그대로 가능한 모든 경우를 탐색해 보는 것
- 시간복잡도를 생각하지 않으면 모든 문제를 풀 수 있으므로, 처음에 완전탐색 풀이를 생각한 후 차츰 최적화하는 방식도 꽤 많은 문제에서 유용함
- 그런데 만약 모든 경우를 탐색하는 풀이를 생각해 봤는데 $N(\text{입력})$ 제한이 매우 작다면 완전 탐색을 의심

완전 탐색 - 예시 문제 : 7568. 덩치

문제 문제 선정 의도 : 모든 경우를 단순하게 탐색하는 문제를 풀어보자

우리는 사람의 덩치를 키와 몸무게, 이 두 개의 값으로 표현하여 그 등수를 매겨보려고 한다. 어떤 사람의 몸무게가 x kg이고 키가 y cm라면 이 사람의 덩치는 (x, y) 로 표시된다. 두 사람 A와 B의 덩치가 각각 (x, y) , (p, q) 라고 할 때 $x > p$ 그리고 $y > q$ 이라면 우리는 A의 덩치가 B의 덩치보다 "더 크다"고 말한다. 예를 들어 어떤 A, B 두 사람의 덩치가 각각 $(56, 177)$, $(45, 165)$ 라고 한다면 A의 덩치가 B보다 큰 셈이 된다. 그런데 서로 다른 덩치끼리 크기를 정할 수 없는 경우도 있다. 예를 들어 두 사람 C와 D의 덩치가 각각 $(45, 181)$, $(55, 173)$ 이라면 몸무게는 D가 C보다 더 무겁고, 키는 C가 더 크므로, "덩치"로만 볼 때 C와 D는 누구도 상대방보다 더 크다고 말할 수 없다.

N명의 집단에서 각 사람의 덩치 등수는 자신보다 더 "큰 덩치"의 사람의 수로 정해진다. 만일 자신보다 더 큰 덩치의 사람이 k명이라면 그 사람의 덩치 등수는 $k+1$ 이 된다. 이렇게 등수를 결정하면 같은 덩치 등수를 가진 사람은 여러 명도 가능하다. 아래는 5명으로 이루어진 집단에서 각 사람의 덩치와 그 등수가 표시된 표이다.

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

위 표에서 C보다 더 큰 덩치의 사람이 없으므로 C는 1등이 된다. 그리고 A, B, D 각각의 덩치보다 큰 사람은 C뿐이므로 이들은 모두 2등이 된다. 그리고 E보다 큰 덩치는 A, B, C, D 이렇게 4명이므로 E의 덩치는 5등이 된다. 위 경우에 3등과 4등은 존재하지 않는다. 여러분은 학생 N명의 몸무게와 키가 담긴 입력을 읽어서 각 사람의 덩치 등수를 계산하여 출력해야 한다.

완전 탐색 - 예시 문제 : 7568. 덩치

* 무엇을 구해야 하는가?

- 모든 i 에 대하여 ($1 \leq i \leq n$) i 번째 사람보다 덩치가 큰 사람의 수를 각각 구해야 함
- 그 사람을 제외하고 모든 사람에 대해서 검토하면서 그 사람보다 덩치가 큰 사람을 세는 방법 - $O(n^2)$

입력

첫 줄에는 전체 사람의 수 N 이 주어진다. 그리고 이어지는 N 개의 줄에는 각 사람의 몸무게와 키를 나타내는 양의 정수 x 와 y 가 하나의 공백을 두고 각각 나타난다.

출력

여러분은 입력에 나열된 사람의 덩치 등수를 구해서 그 순서대로 첫 줄에 출력해야 한다. 단, 각 덩치 등수는 공백문자로 분리되어야 한다.

제한

- $2 \leq N \leq 50$
- $10 \leq x, y \leq 200$

완전 탐색 - 예시 문제 : 7568. 덩치

* 무엇을 구해야 하는가?

- 모든 i 에 대하여 ($1 \leq i \leq n$) i 번째 사람보다 덩치가 큰 사람의 수를 각각 구해야 함
- 그 사람을 제외하고 모든 사람에 대해서 검토하면서 그 사람보다 덩치가 큰 사람을 세는 방법 - $O(n^2)$
- n 제한이 작으므로 $O(n^2)$ 로도 통과 가능하다

입력

첫 줄에는 전체 사람의 수 N 이 주어진다. 그리고 이어지는 N 개의 줄에는 각 사람의 몸무게와 키를 나타내는 양의 정수 x 와 y 가 하나의 공백을 두고 각각 나타난다.

출력

여러분은 입력에 나열된 사람의 덩치 등수를 구해서 그 순서대로 첫 줄에 출력해야 한다. 단, 각 덩치 등수는 공백문자로 분리되어야 한다.

제한

- $2 \leq N \leq 50$
- $10 \leq x, y \leq 200$

완전 탐색 - 예시 문제 : 7568. 덩치

* 완전 탐색의 아이디어는 매우 단순하다

- 모든 경우에 대해 문제의 조건을 만족하는지 확인하기만 하면 된다
- 다만 각각의 경우들을 어떻게 확인할 것인지, 그리고 어떻게 가능한 모든 경우를 다 따질 것인지를 생각해야 하는 문제들이 존재한다
- 각각의 경우들을 체크 : 보통 각각의 경우를 나타내는 무언가를 인수로 받아서 그 경우에 대한 테스트를 진행하는 함수를 따로 작성한다
- 모든 경우를 따지기 : 추후에 다룰 백트래킹이 하나의 방법이 될 수 있다

완전 탐색 - 예시 문제 : 7568. 덩치

* 완전 탐색의 아이디어는 매우 단순하다

- 모든 경우에 대해 문제의 조건을 만족하는지 확인하기만 하면 된다
- 다만 각각의 경우들을 어떻게 확인할 것인지, 그리고 어떻게 가능한 모든 경우를 다 따질 것인지를 생각해야 하는 문제들이 존재한다
- 각각의 경우들을 체크 : 보통 각각의 경우를 나타내는 무언가를 인수로 받아서 그 경우에 대한 테스트를 진행하는 함수를 따로 작성한다
- 모든 경우를 따지기 : 추후에 다룰 백트래킹이 하나의 방법이 될 수 있다

* '덩치' 문제는 그 중 하나만 생각하면 됨

- 각각의 경우를 생성하는 과정이 필요없다. 따져야 할 모든 경우가 '각각의 사람의 정보'로 이미 주어짐
- 따라서 우리는 각각의 사람보다 덩치가 큰 사람 수를 세어 주기만 하면 됨

완전 탐색 - 예시 문제 : 1747. 소수&팰린드롬

문제 문제 선정 의도 : 모든 경우 각각을 좀더 복잡하게 따져주는 문제를 풀어보자

어떤 수와 그 수의 숫자 순서를 뒤집은 수가 일치하는 수를 팰린드롬이라 부른다. 예를 들어 79,197과 324,423 등이 팰린드롬 수이다.

어떤 수 N ($1 \leq N \leq 1,000,000$)이 주어졌을 때, N 보다 크거나 같고, 소수이면서 팰린드롬인 수 중에서, 가장 작은 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N 이 주어진다.

출력

첫째 줄에 조건을 만족하는 수를 출력한다.

완전 탐색 - 예시 문제 : 1747. 소수&팰린드롬

* 모든 경우를 체크해도 되는가?

- 1,000,000 보다 큰 소수&팰린드롬 중 가장 작은 건 1003001이다
- 어떤 수가 팰린드롬인지 판정하는 것은 그 자릿수만큼의 시간복잡도인데 자릿수는 최대 7자리까지이므로 $O(1)$ 이라고 볼 수 있음
- 어떤 수가 소수인지 판정하는 것은 $O(\sqrt{n})$ 의 시간복잡도로 가능
- 따라서 주어진 숫자부터 1씩 증가시켜 나가면서 소수&팰린드롬인지 체크하는 방식으로도 충분히 통과 가능하다

완전 탐색 - 예시 문제 : 1747. 소수&팰린드롬

* 어떤 수가 소수&팰린드롬인지 체크하기

- 소수 판정은 에라토스테네스의 체를 이용한 전처리로 빠르게 가능
- $O(\sqrt{n})$ 으로 소수 판정하는 코드도 통과한다

```
int notprime[2000005] = { 1,1,0 }; //소수가 아니면 1

void sieve(void) {
    for (int i = 2; i <= 2000000; i++) {
        if (notprime[i] == 0) {
            for (int j = 2 * i; j <= 2000000; j += i) {
                notprime[j] = 1;
            }
        }
    }
}
```

- 주어진 수가 팰린드롬인지 확인하는 함수도 작성

```
int check(int n) {
    //그 수가 팰린드롬인지 체크하는 함수
    vector<int> digit;
    while (n != 0) {
        digit.push_back(n % 10);
        n /= 10;
    }
    int len = digit.size();
    for (int i = 0; i < len / 2; i++) {
        if (digit[i] != digit[len - i - 1]) { return 0; }
    }
    return 1;
}
```

2022 Winter Algorithm Camp

완전 탐색 - 연습 문제

* 연습 문제

- 2309. 일곱 난쟁이
- 2231. 분해합
- 1018. 체스판 다시 칠하기 : 문제에서 나올 수 있는 모든 경우를 체크한다는 아이디어
- (가능한 모든 체스판 영역을 따져 본다)
- 11170. 0의 개수

백트래킹 - 개론

* 어떻게 모든 경우를 탐색할 것인가?

- 나올 수 있는 모든 경우를 탐색해서 조건을 체크해야 하는 완전 탐색 문제를 생각해 보자
- 그럼 나올 수 있는 모든 경우를 어떻게 생성할 것인가?
- 예를 들어 n 개 중 k 개의 원소를 골랐을 때 어떤 조건을 만족하는 k -tuple의 쌍((a_1, a_2, \dots, a_k) 쌍 개수)을 세어야 하는 경우
- 경우를 생성하는 원소의 개수가 k 이면 k 중 **for**문을 작성해서 가능한 모든 쌍을 따지는 것도 가능
- 가령 n 개의 수 중 3개를 골랐을 때 그 세 수의 합에 대해 무언가를 체크해야 한다면 3중 **for**문으로 가능

백트래킹 - 개론

* 어떻게 모든 경우를 탐색할 것인가?

- 나올 수 있는 모든 경우를 탐색해서 조건을 체크해야 하는 완전 탐색 문제를 생각해 보자
- 그럼 나올 수 있는 모든 경우를 어떻게 생성할 것인가?
- 예를 들어 n 개 중 k 개의 원소를 골랐을 때 어떤 조건을 만족하는 k -tuple의 쌍((a_1, a_2, \dots, a_k) 쌍 개수)을 세어야 하는 경우
- 경우를 생성하는 원소의 개수가 k 이면 k 중 for문을 작성해서 가능한 모든 쌍을 따지는 것도 가능
- 가령 n 개의 수 중 3개를 골랐을 때 그 세 수의 합에 대해 무언가를 체크해야 한다면 3중 for문으로 가능
- 그러나 숫자 중 10개를 골라야 한다면 10중 for문을 써야 하는가????
- 불가능한 건 아니지만 너무 끔찍한 방법이다
- 게다가 고르는 숫자의 수 k 도 입력으로 주어진다면 k 에 따라 1중, 2중, k 중... for문을 작성해야 한다

백트래킹 - 개론

* 시간복잡도는 똑같더라도, 좀더 깔끔하게 모든 경우를 생성하자! - 백트래킹

- 문제에서 나올 수 있는 '경우'를 하나의 '상태'로 보는 관점이 여기부터 필요하다
- 현재 상태에서 전이될 수 있는 다음 상태들을 재귀적으로 탐색하다가 특정 조건을 만족하면 문제에서 따져야 하는 무언가가 된다

백트래킹 - 개론

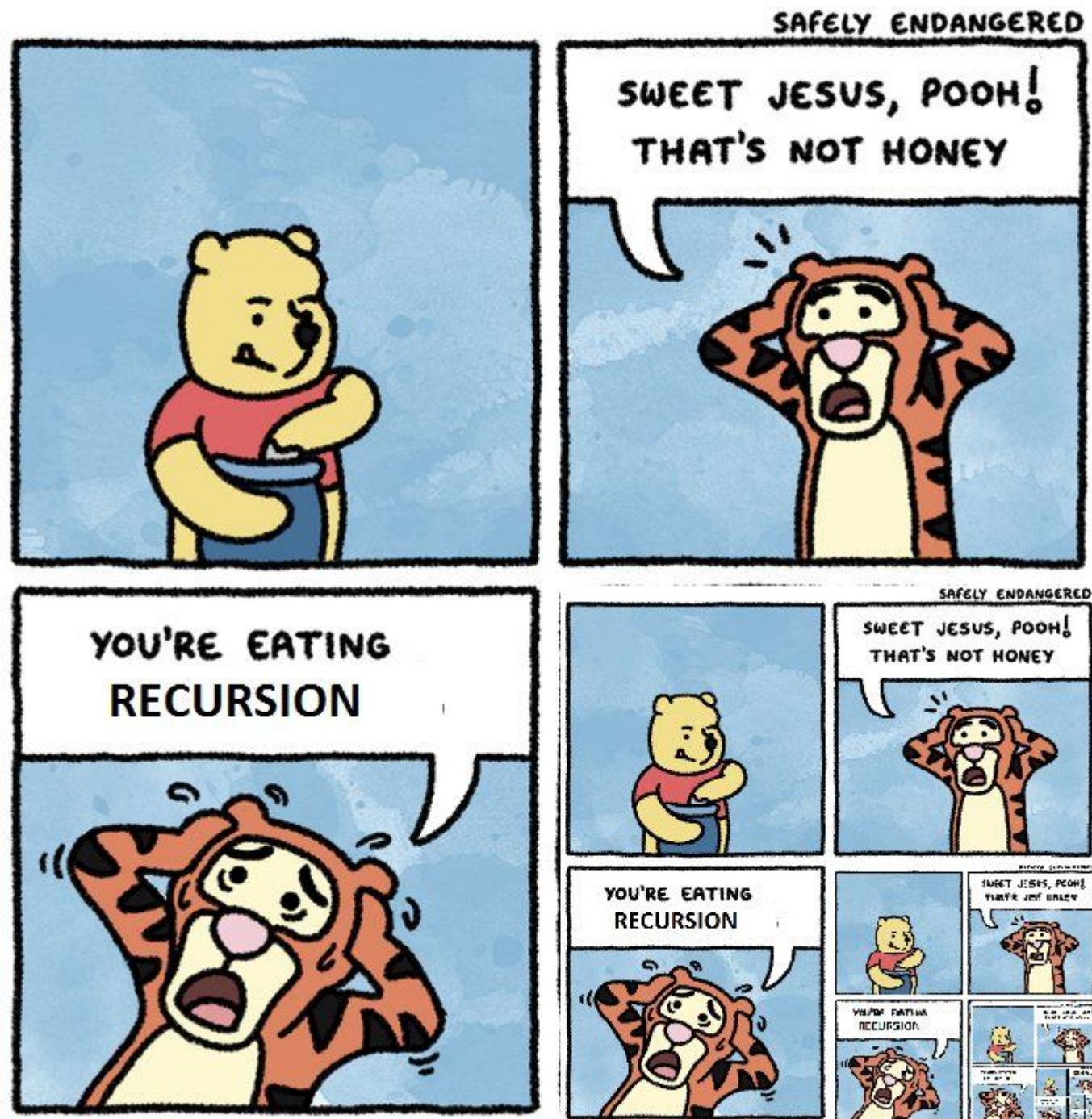
* 시간복잡도는 똑같더라도, 좀더 깔끔하게 모든 경우를 생성하자! - 백트래킹

- 문제에서 나올 수 있는 '경우'를 하나의 '상태'로 보는 관점이 여기부터 필요하다
- 현재 상태에서 전이될 수 있는 다음 상태들을 재귀적으로 탐색하다가 특정 조건을 만족하면 문제에서 빠져야 하는 무언가가 된다

* 하나의 상태에서 전이될 수 있는 다음 상태는 무엇인가?

- n 개 숫자 중 서로 다른 k 개를 고르는 경우를 생각해 보자
- i 번째 숫자를 검토하고 있다고 할 때, 만약 아직 k 개를 다 고른 상태가 아니라면 이전 상태에 더해 i 번째 숫자를 고르는 상태와 고르지 않는 상태가 있을 수 있다
- 재귀에 대한 이해가 필요하다
- 백트래킹도 재귀를 이용한 완전 탐색에 불과

재귀에 대한 소개



어느 한 컴퓨터공학과 학생이 유명한 교수님을 찾아가 물었다.

"재귀함수가 뭔가요?"

"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.

마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.

그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."

----- "재귀함수가 뭔가요?"

----- "잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.

----- 마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.

----- 그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."

----- "재귀함수가 뭔가요?"

----- "잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.

----- 마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.

----- 그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."

----- "재귀함수가 뭔가요?"

----- "잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.

----- 마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.

----- 그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."

----- "재귀함수가 뭔가요?"

----- "재귀함수는 자기 자신을 호출하는 함수라네"

----- 라고 답변하였지.

----- 라고 답변하였지.

----- 라고 답변하였지.

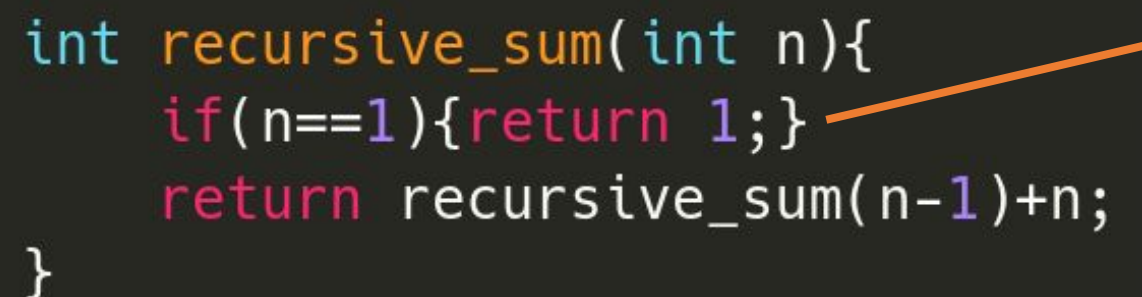
----- 라고 답변하였지.

라고 답변하였지.

재귀에 대한 소개

* 재귀 함수는 자기 자신을 호출하는 함수이다

- 자기 자신을 호출하지만 호출할 때 다른 상태를 전달해서 호출한다
- 재귀 호출을 하면서 상태가 점차 **base case**에 가까워져야 한다
- A_1 (base case)가 정해져 있고 A_i 가 A_{i-1} 또는 그 이전 케이스들에 영향을 받아 정해지는 것을 그대로 구현한 것(점화식이라고도 할 수 있다)
- 재귀 호출 스택을 완벽히 따라갈 필요까지는 없지만 함수들이 어떻게 상태를 주고받는지는 받아들이는 연습을 해야 한다



```
int recursive_sum(int n){  
    if(n==1){return 1;}  
    return recursive_sum(n-1)+n;  
}
```

base case

재귀에 대한 소개

* 재귀 함수는 자기 자신을 호출하는 함수이다

- 자기 자신을 호출하지만 호출할 때 다른 상태를 전달해서 호출한다
- 재귀 호출을 하면서 상태가 점차 **base case**에 가까워져야 한다
- A_1 (base case)가 정해져 있고 A_i 가 A_{i-1} 또는 그 이전 케이스들에 영향을 받아 정해지는 것을 그대로 구현한 것(점화식이라고도 할 수 있다)
- 재귀 호출 스택을 완벽히 따라갈 필요까지는 없지만 함수들이 어떻게 상태를 주고받는지는 받아들이는 연습을 해야 한다
- **base case**는 진짜로 기본이 되는 케이스라기보다는, 재귀 함수도 결국은 종료되고 결과를 내놓아야 하므로 언젠가 도달할 종료 조건을 의미한다. $n==1$ 등의 경우일 필요가 있는 건 아니다

재귀에 대한 소개

* 재귀 함수를 짜면서 생각해야 하는 것은 무엇인가?

- 재귀 함수가 가지고 있어야 하는 상태는 무엇인가? 상태를 무엇이 결정하는가?
- 현재 상태가 정해지기 위해서 어떤 상태들이 필요한가? 현재 상태에 영향을 받지 않는 이전의 상태들만으로 가능한가?
- 함수들이 재귀 호출을 통해 상태를 주고받을 때 사이클이 형성되지 않는가?
ex) A_3 를 결정하기 위해 A_2 가 필요한데 A_2 를 결정하기 위해서는 A_3 이 필요한 경우
- 코드가 만들어내는 상황이 점점 재귀 함수의 **base case**에 가까워지는 방향으로 진행하는가?

예시 문제 - 23304. 아카라카

* 아카라카 팰린드롬 revisited!

AKARAKA(아카라카)는 컴퓨터 과학적 관점으로 바라봤을 때, 튜링도 기립 박수를 치고 갈 가히 최고의 구호라 할 수 있다. **AKARAKA**는 그 자체로도 팰린드롬이고, 접두사이자 접미사인 **AKA**가 또한 팰린드롬이기 때문이다.

신촌에서는 **AKARAKA**같은 특성을 가진 팰린드롬을, **아카라카 팰린드롬**이라 아래와 같이 정의한다.

1. 문자열 S 가 팰린드롬이다. 팰린드롬이란 거꾸로 뒤집어 읽어도 같은 문자열을 뜻한다.
2. 문자열 S 의 길이를 $|S|$ 라 할 때, $\lfloor \frac{|S|}{2} \rfloor$ 길이의 접두사와 접미사가 모두 아카라카 팰린드롬이다. 만약 $|S| = 1$ 이면, S 는 아카라카 팰린드롬이다.

아카라카 팰린드롬이 바로
재귀적인 구조를 띤다
- 같은 구조가 반복됨

임의의 문자열이 주어졌을 때, 그 문자열이 아카라카 팰린드롬인지 알아보자. 만약 알아내지 못하면, 졸업할 때까지 아카라카를 못 갈지도 모른다!

예시 문제 - 23304. 아카라카

```
int solve(string word) {  
    if (word.length() == 1) { return 1; }  
    int len = word.length();  
    string l = word.substr(0, len / 2), r = word.substr((len + 1) / 2, len);  
    if (check(word) == 0) { return 0; }  
    if (solve(l) && solve(r)) { return 1; }  
    return 0;  
}
```

- 재귀를 이용해서 접두사와 접미사가 모두 아카라카 팰린드롬인지 확인
- l(left, prefix)와 r(right, postfix)에 대해 재귀 호출을 진행하고 있다
- base case는 문자열 길이가 1일 때

2022 Winter Algorithm Camp

재귀 - 연습 문제

* 필수 문제

- 17478. 재귀함수가 뭔가요?

백트래킹 - 예시 문제 : 15650. N과 M (2)

문제 문제 선정 의도 : 가능한 모든 경우를 단순히 생성하는 문제를 통해,
재귀로 상태를 전이시켜 가며 문제를 푼다는 것을 이해해 보자

자연수 N과 M이 주어졌을 때, 아래 조건을 만족하는 길이가 M인 수열을 모두 구하는 프로그램을 작성하시오.

- 1부터 N까지 자연수 중에서 중복 없이 M개를 고른 수열
- 고른 수열은 오름차순이어야 한다.

입력

첫째 줄에 자연수 N과 M이 주어진다. ($1 \leq M \leq N \leq 8$)

출력

한 줄에 하나씩 문제의 조건을 만족하는 수열을 출력한다. 중복되는 수열을 여러 번 출력하면 안되며, 각 수열은 공백으로 구분해서 출력해야 한다.

수열은 사전 순으로 증가하는 순서로 출력해야 한다.

백트래킹 - 예시 문제 : 15650. N과 M (2)

예제 입력 2 복사

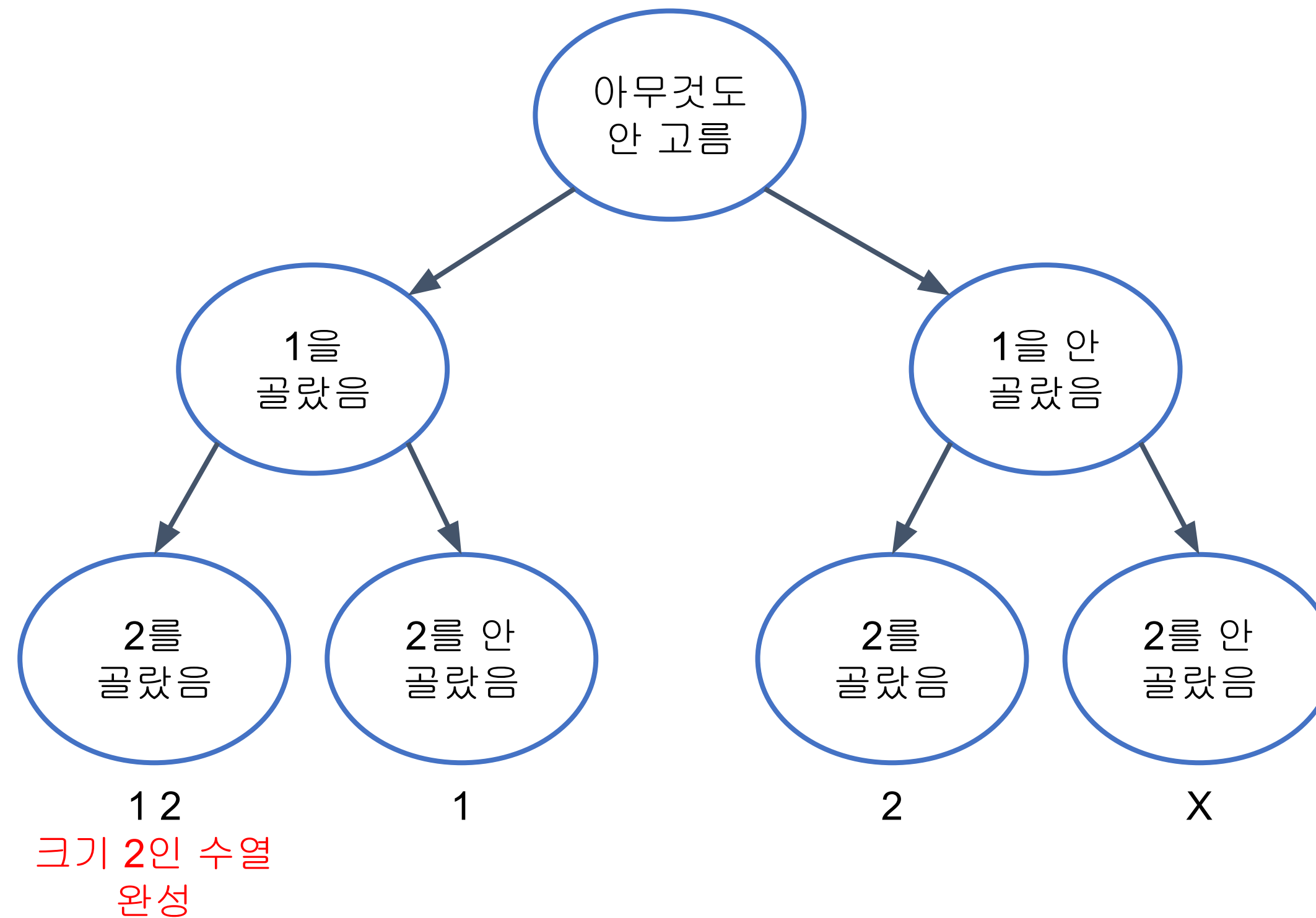
```
4 2
```

예제 출력 2 복사

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

- 아무것도 고르지 않은 상태일 때, 1을 고르는 상태와 1을 안 고르는 상태로 전이될 수 있다
- 1을 고른 상태일 때와 안 고른 상태일 때 각각, 2를 고르는 상태와 2를 안 고르는 상태로 전이될 수 있다
(만들어야 하는 수열은 오름차순이고, 중복을 허용하지 않으므로 1을 고른 다음 1을 또 고른다거나 다른 숫자부터 본다거나 하지 않는다)
- 만약 1과 2를 둘 다 골랐다면 2개를 고른 것이므로 원하는 크기의 수열이 하나 만들어졌다

백트래킹 - 예시 문제 : 15650. N과 M (2)



백트래킹 - 예시 문제 : 15650. N과 M (2)

* 무엇을 따지면서 상태를 전이시켜야 하는가?

- 어떤 상태가 필요한가?
- 현재 넣을 숫자, 현재 숫자를 넣을 인덱스(물론 다른 답이 있을 수 있다)
- 현재 상태에서 다음 상태로 전이되기 위해서 재귀 함수 호출에서 어떤 것을 변화시켜줘야 하는가?
- 우리가 정의한 상태를 적절히 변화시켜 재귀 호출에서 인수로 전달해 줘야 한다
- 상태는 어떻게 관리하는가?
- 전역 변수로 할 수도 있고 포인터나 참조자로 배열을 받을 수도 있지만 보통은 전역 변수 사용

백트래킹 - 예시 문제 : 15650. N과 M (2)

* 무엇을 따지면서 상태를 전이시켜야 하는가?

- 어떤 상태가 필요한가?

- 현재 넣을 숫자, 현재 숫자를 넣을 인덱스(물론 다른 답이 있을 수 있다)

- 현재 상태에서 다음 상태로 전이되기 위해서 재귀 함수 호출에서 어떤 것을 변화시켜줘야 하는가?

- 우리가 정의한 상태를 적절히 변화시켜 재귀 호출에서 인수로 전달해 줘야 한다

- 상태는 어떻게 관리하는가?

- 전역 변수로 할 수도 있고 포인터나 참조자로 배열을 받을 수도 있지만 보통은 전역 변수 사용

- 이전 상태에 대해서도 무언가 따져야 한다거나 어떤 방문 체크를 해줘야 한다거나 전이 시 따로 함수의 인수로 가져가야 하는 어떤 수치가 있다거나 하는 문제도 있다. 상태의 전이 시 무엇이 필요한지 잘 살펴서 상태를 생성하는 함수와 필요한 전역 변수들을 작성하자

백트래킹 - 예시 문제 : 15650. N과 M (2)

* 재귀로 코드를 작성

```
void solve(int cur, int idx){
    //cur는 현재 들어갈 숫자, idx는 현재 들어갈 인덱스
    if(idx==m+1){
        for(int i=1;i<=m;i++){cout<<arr[i]<<" ";}
        cout<<"\n";
        return;
    }
    if(cur==n+1){return;}
    arr[idx]=cur;
    solve(cur+1,idx+1);
    //idx에 cur을 넣을 경우
    solve(cur+1,idx);
    //idx에 cur을 넣지 않을 경우 (idx에는 다른 값이 들어갈 것)
}
```

기본적으로는 문제 풀이 코드를 제공하지 않는 것이 원칙이지만 재귀로 백트래킹을 하는 코드를 혼자서 처음부터 짜기는 쉽지 않다고 여겨져서 핵심 부분이 되는 코드를 제공

앞으로도 강의의 핵심 주제가 되는 알고리즘의 기본 코드는 약간 제공 예정

백트래킹 - 예시 문제 : 15650. N과 M (2)

* 재귀로 코드를 작성

```
void solve(int cur, int idx){
    //cur는 현재 들어갈 숫자, idx는 현재 들어갈 인덱스
    if(idx==m+1){
        for(int i=1;i<=m;i++){cout<<arr[i]<<" ";}
        cout<<"\n";
        return;
    }
    if(cur==n+1){return;}
    arr[idx]=cur;
    solve(cur+1,idx+1);
    //idx에 cur을 넣을 경우
    solve(cur+1,idx);
    //idx에 cur을 넣지 않을 경우 (idx에는 다른 값이 들어갈 것)
}
```

기본적으로는 문제 풀이 코드를 제공하지 않는 것이 원칙이지만 재귀로 백트래킹을 하는 코드를 혼자서 처음부터 짜기는 쉽지 않다고 여겨져서 핵심 부분이 되는 코드를 제공

앞으로도 강의의 핵심 주제가 되는 알고리즘의 기본 코드는 약간 제공 예정

어렵거나 귀찮아하는 사람이 많은 주제지만 그만큼 잘하면 이점이 있다. 코딩테스트에도 빈출...

꼭 자신이 생각하여 코드를 이해하며 다시 짜보도록 하자

백트래킹 - 발상

* 백트래킹의 기본적인 풀이법

- 초기 상태가 무엇인지 생각한다
- 거기서 재귀적으로 전이될 수 있는 상태가 무엇인지 생각한다
- 생성하고 있는 상태가 어떤 사전 조건을 만족하게 된 경우 문제에서 따져야 하는 상태가 되는지 고민한다(예를 들어 n 개 중 m 개를 고른 수열들에 대해서만 무언가 따져야 하는 문제의 경우, m 개를 고르지 않은 상태는 따지면 안 된다) 재귀의 **base case**같은 것
- 생성된 상태를 어떻게 보관하고 어떻게 문제의 조건에 대해 따져 줄지 고민한다
- 만들어진 상태에서 어떻게 이전 상태로 돌아가고, 무엇을 보존할지 고민한다(말 그대로 **Back tracking**)

백트래킹 - 예시 문제 : 1182. 부분수열의 합

문제 문제 선정 의도 : 가능한 모든 상태에 대해 어떤 조건을 따져주는 문제를 풀어보자

N개의 정수로 이루어진 수열이 있을 때, 크기가 양수인 부분수열 중에서 그 수열의 원소를 다 더한 값이 S가 되는 경우의 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 정수의 개수를 나타내는 N과 정수 S가 주어진다. ($1 \leq N \leq 20$, $|S| \leq 1,000,000$) 둘째 줄에 N개의 정수가 빈 칸을 사이에 두고 주어진다. 주어지는 정수의 절댓값은 100,000을 넘지 않는다.

출력

첫째 줄에 합이 S가 되는 부분수열의 개수를 출력한다.

백트래킹 - 예시 문제 : 1182. 부분수열의 합

예제 입력 1 복사

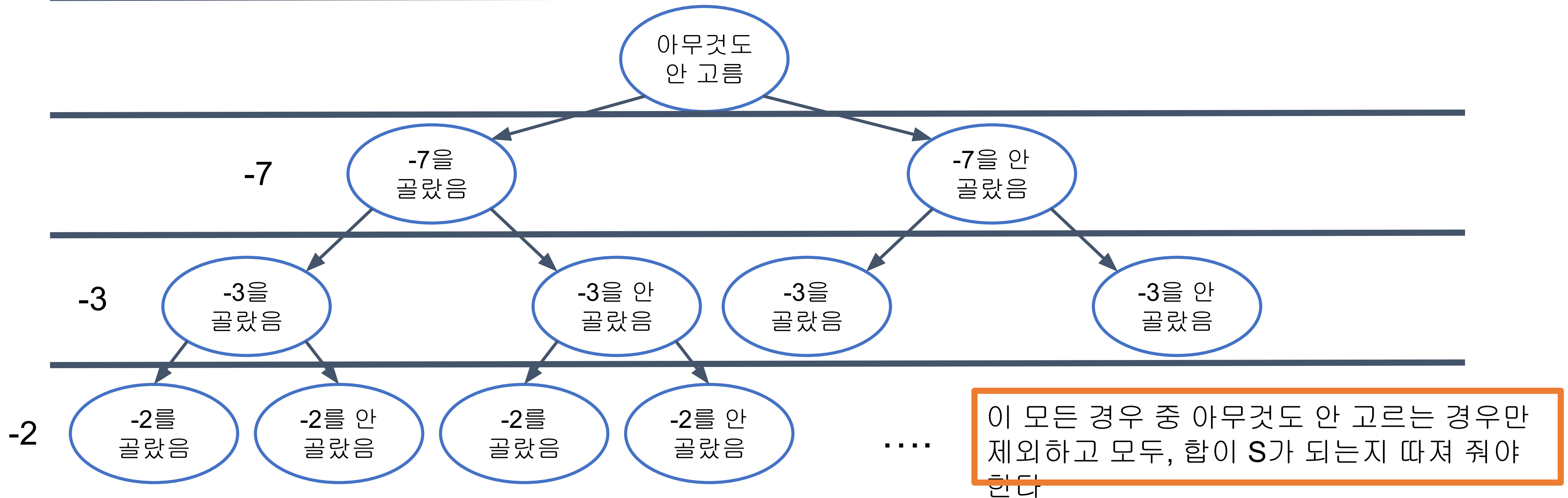
```
5 0
-7 -3 -2 5 8
```

예제 출력 1 복사

```
1
```

- 예제에서 부분 수열 중 그 합이 0이 되는 부분 수열은 (-3,-2,5) 하나뿐이다.
- $N=5$ 일 때 나올 수 있는 크기가 양수인 부분 수열은 (2^5-1) 개이고 이 모든 부분 수열에 대해 합을 따져 줘야 한다
- 어떤 i 번째 원소를 검토하고 있을 때, 그 원소를 넣는지 안 넣는지에 따라서 2가지 상태가 생길 수 있음
- 그러면 이전 상태의 수열 합을 알고 있다면 i 번째 원소를 넣는지 안 넣는지에 따라 그 수열 합을 바로 계산 가능

백트래킹 - 예시 문제 : 1182. 부분수열의 합



백트래킹 - 예시 문제 : 1182. 부분수열의 합

* 이번에도 재귀로 코드를 작성

```
int n,s,arr[25];
int ans=0, sum=0;

void dfs(int idx){
    if(idx==n){return;}
    if(sum+arr[idx]==s){ans++;}
    dfs(idx+1);
    sum+=arr[idx];
    dfs(idx+1);
    sum-=arr[idx];
}
```

idx는 이번에 포함시킬지를 선택하는 원소의 인덱스

인덱스는 $n-1$ 까지이므로 n 인덱스를 포함시키는 경우는 따지면 안 됨

이번 인덱스를 포함시켜 합이 s 가 되는 경우 카운트를 1 증가시킴

idx번째 원소를 합에 포함시키지 않고 다음 인덱스에 대한 상태로 전이함

idx번째 원소를 합에 포함시키고 다음 인덱스에 대한 상태로 전이함

합을 idx번째 원소를 포함시키지 않은 상태로 되돌려 줌

2022 Winter Algorithm Camp

백트래킹 - 연습 문제

* 연습 문제

- 15649~15666 중 N과 M (1)~(12)
- 10819. 차이를 최대로
- 10597. 순열장난
- 1759. 암호 만들기
- 15811. 복면산?!
- (Challenging) 17136. 색종이 붙이기

Appendix : 백트래킹에서 가지치기

* 모든 경우를 따질 필요가 없을 수 있다

- 가능한 경우들 중 하나만 필요하다면 그 하나를 찾는 순간 나머지를 더 탐색할 필요가 없어진다
- 상태를 전이시켜 가는 도중 우리가 목표로 하는 상태에 도달할 수 없다는 것을 미리 알 수 있는 경우가 있다
- 고전적인 머신러닝 이론에서 쓰이기도 한다(alpha-beta pruning 등)

가지치기 - 예시 문제 : 2661. 좋은수열

문제 문제 선정 의도 : 모든 경우에 대해 따져주지 않아도 되는 문제를 풀어보자

숫자 1, 2, 3으로만 이루어지는 수열이 있다. 임의의 길이의 인접한 두 개의 부분 수열이 동일한 것이 있으면, 그 수열을 나쁜 수열이라고 부른다. 그렇지 않은 수열은 좋은 수열이다.

다음은 나쁜 수열의 예이다.

- 33
- 32121323
- 123123213

다음은 좋은 수열의 예이다.

- 2
- 32
- 32123
- 1232123

길이가 N인 좋은 수열들을 N자리의 정수로 보아 그중 가장 작은 수를 나타내는 수열을 구하는 프로그램을 작성하라. 예를 들면, 1213121과 2123212는 모두 좋은 수열이지만 그 중에서 작은 수를 나타내는 수열은 1213121이다.

입력

입력은 숫자 N하나로 이루어진다. N은 1 이상 80 이하이다.

출력

첫 번째 줄에 1, 2, 3으로만 이루어져 있는 길이가 N인 좋은 수열들 중에서 가장 작은 수를 나타내는 수열만 출력한다. 수열을 이루는 1, 2, 3들 사이에는 빈칸을 두지 않는다.

가지치기 - 예시 문제 : 2661. 좋은수열

* 만약 나올 수 있는 모든 길이 N짜리 수열을 완전탐색한다면?

- 당연히, 가장 작은 '좋은수열'을 찾을 수 있을 것이다
- 그러나 나올 수 있는 길이 N 수열은 3^N 개가 있고 N 제한은 80이므로 3^{80} 개의 수열을 모두 좋은수열인지 확인한다면 당연히 시간초과가 발생한다($3^{80} \sim 1.4780883e+38$)

가지치기 - 예시 문제 : 2661. 좋은수열

* 만약 나올 수 있는 모든 길이 N짜리 수열을 완전탐색한다면?

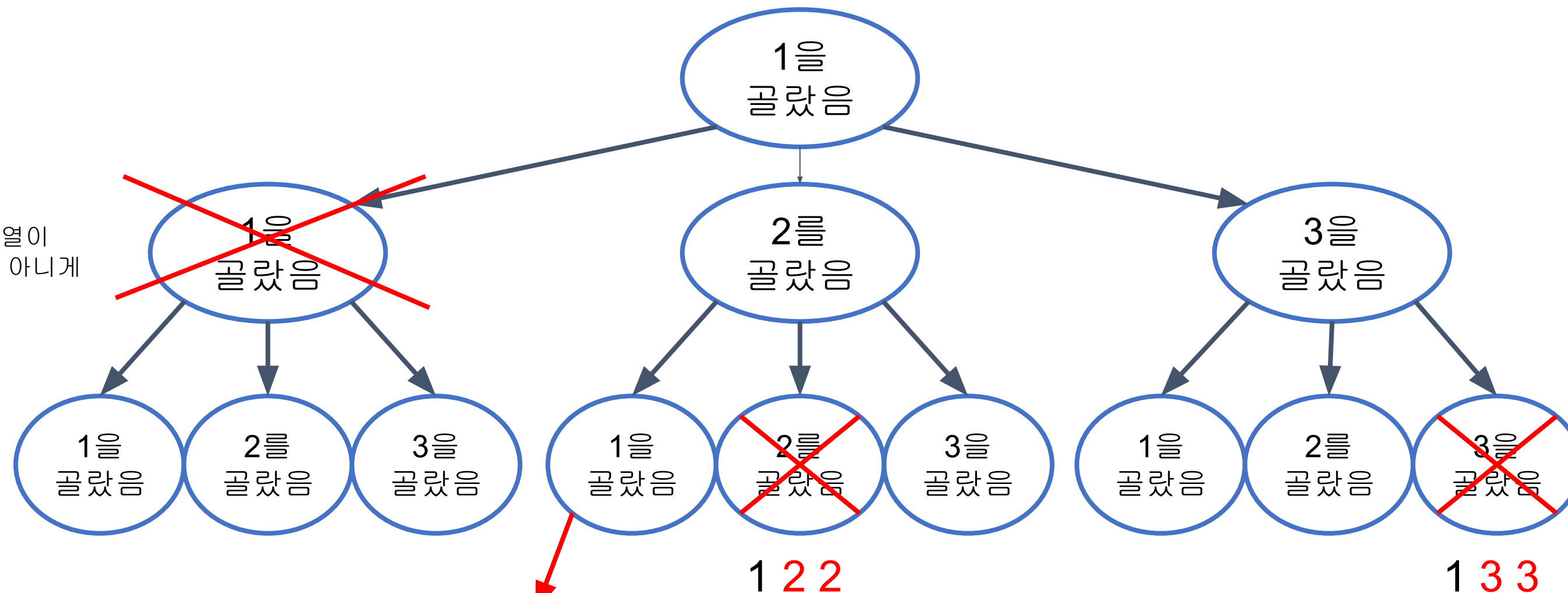
- 당연히, 가장 작은 '좋은수열'을 찾을 수 있을 것이다
- 그러나 나올 수 있는 길이 N 수열은 3^N 개가 있고 N 제한은 80이므로 3^{80} 개의 수열을 모두 좋은수열인지 확인한다면 당연히 시간초과가 발생한다($3^{80} \sim 1.4780883e+38$)

* 모든 경우를 다 따져주지 않아도 된다는 것을 이용하자

- 임의의 길이의 인접 부분 수열이 같으면 안 되므로, 수열의 모든 부분은 좋은수열이어야 한다.
- 따라서 수열의 마지막 부분에 어떤 숫자를 추가로 붙여서 좋은수열을 만들기 위해서는 원래 수열도 좋은수열이어야 한다
- 따라서 만들어지는 좋은수열들에 대해서만 추가적으로 숫자를 붙여가며 좋은수열인지 테스트하면 된다 (나머지 경우들은 가지치기한다 - 이를 **pruning**이라고 하기도 한다)

가지치기 - 예시 문제 : 2661. 좋은수열

1 1이 되면 인접한 수열이
같아져서 좋은수열이 아니게
됨



길이 3인 수열 중 이게 가장 작은
좋은수열 -> 이 방향으로 계속

답책

가지치기 - 예시 문제 : 2661. 좋은수열

* 어떤 경우에 새 수열을 만들어 볼 것인가?

- 지금까지 만든 수열이 좋은수열일 경우 새 숫자를 붙여 좋은수열을 만들어 본다
- 현재까지 만든 수열이 좋은수열인지 확인하는 함수를 통과한 경우 다음 인덱스를 붙인다
- 단, 만약 이미 지금까지 만든 수열 길이가 n 에 해당할 경우 그냥 그 수열을 출력해 준 후 종료하면

```
if (cnt == n + 1) {
    done = 1;
    for (int i = 1; i <= n; i++) {
        cout << arr[i];
    } cout << "\n";
    return;
}
```

먼저, 현재까지 만들어진 좋은수열 길이가 n 일 경우 출력해 주고 함수 종료

```
for (int i = 1; i <= 3; i++) {
    arr[cnt] = i;
    if (check(cnt)) {
        solve(cnt + 1);
    }
    arr[cnt] = 0;
}
```

새로운 숫자 i 를 추가해 보고 그게 좋은수열이 되면 수열의 다음 인덱스를 재귀적으로 만든다

가지치기 - 예시 문제 : 2661. 좋은수열

* 새 수열이 좋은수열인지는 어떻게 확인할 것인가?

- 기존에 좋은수열이었던 수열에만 숫자를 붙여서 새로운 수열을 만들어 본다
- 따라서 마지막에 추가된 숫자가 좋은수열을 나쁜 수열로 만드는데만 살피면 된다



```
int check(int cur) {  
    //마지막으로 숫자 넣은 인덱스 표시  
    //새로 넣은 숫자가 나쁜수열을 만드는가?  
    for (int len = 1; len <= (cur) / 2; len++) {  
        int same = 1; //인접한 수열이 같은지  
        for (int idx = 1; idx <= len; idx++) {  
            if (arr[cur - len + idx] != arr[cur - 2 * len + idx]) { same = 0; }  
        }  
        if (same == 1) { return 0; }  
    }  
    return 1;  
}
```


가지치기 - 예시 문제 : 2661. 좋은수열

* 만약 나올 수 있는 모든 길이 N짜리 수열을 완전탐색한다면?

- 당연히, 가장 작은 '좋은수열'을 찾을 수 있을 것이다
- 그러나 나올 수 있는 길이 N 수열은 3^N 개가 있고 N 제한은 80이므로 3^{80} 개의 수열을 모두 좋은수열인지 확인한다면 당연히 시간초과가 발생한다($3^{80} \sim 1.4780883e+38$)

* 좋은수열의 특성을 이용해 가지치기를 해주면 충분히 시간 내에 통과한다

38098343	dart	 2661	맞았습니다!!	2020 KB	0 ms	C++17 / 수정	1045 B	4분 전
37993634	dart	 2661	맞았습니다!!	2020 KB	0 ms	C++17 / 수정	1040 B	1일 전

만약 정직하게 완전탐색을 하면 시간 내에 통과할 수 없더라도, 제한이 50~100 정도로 매우 작은 편이라면 가지치기를 하는 완전탐색도 한번 생각해 보자

Appendix : 재귀적 구조를 갖는 문제

* 문제 자체가 재귀적인 구조를 갖는 것들이 있다

- N개의 원판을 옮길 때, 맨 밑의 하나를 옮기고 나서 (N-1)개를 옮기는 것과 같은 문제 구조를 갖는 하노이의 탑
- 서브트리에 대해서 재귀적으로 순회하는 트리의 전위/후위/중위 순회
- 트리 자체가 재귀적인 자료구조라서 트리에서 dp를 하는 경우 재귀가 필요할 때가 많다
- 분할 정복 대부분의 문제들