2025 여름방학 어바웃 코딩테스트

#설명과 짧은 리뷰 python, c++, java 왔다갔다 한다.

▼ 1주차

▼ BOJ1018 체스판 다시 칠하기

```
'''666으로 끝나는 숫자들, 앞뒤에 들어가는 숫자들, 666으로 시작하는 숫자들'''

'''import sys
import itertools

def find_nth_movie(n):
    # 666부터 시작해서, 숫자에 '666'이 포함된 것만 걸러내는 제너레이터
    movies = (num for num in itertools.count(666) if '666' in str(num))
    # n번째(0-based가 아니므로 n-1) 요소를 꺼내서 반환
    return next(itertools.islice(movies, n-1, None))

if __name__ == "__main__":
    n = int(sys.stdin.readline())
    print(find_nth_movie(n))'''

'''itertools.count(666)
무한히 666, 667, 668, ... 을 생성하는 이터레이터를 만듭니다.

(num for num in ... if '666' in str(num))
숫자를 문자열로 바꿔 '666'이 포함된 것만 걸러내는 제너레이터 표현식을 사용합

itertools.islice(movies, n-1, None)
그 제너레이터에서 첫 번째(인덱스 0)가 666이므로, n번째를 얻으려면 n-1번째까

next(...)
최종적으로 n번째 “영화”의 번호를 출력합니다.
'''
```

```
'''
전체 흐름

1부터 차례로 num을 늘려 가며 "666" 포함 여부를 보고,

포함될 때마다 count를 하나씩 올립니다.

count가 입력 n과 같아지는 순간 num을 프린트하고 함수를 종료합니다.'''

import sys

def main():
    n = int(sys.stdin.readline())
    count = 0        #나중에 n과 같은 숫자가 될때까지 증가
    num = 0          #계속 1씩 증가

    while True:
        num += 1
        if '666' in str(num):
            count += 1
            if count == n:
                print(num)
                return

if __name__ == "__main__":
    main()
```

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    int N;
    cin >> N;
    int cnt{ 0 };
    string num{ "666" };
```

```
    while (cnt < N) {
        for (int i{ 0 }; i < num.length(); i++) {
            if (num[i] == '6' && num[i + 1] == '6' && num[i + 2] == '6') { cnt+
        }
        num = to_string(stoi(num) + 1);
    }
    cout << stoi(num) - 1 << "\n";
}



/*
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int N;
    cin >> N;

    int count = 0;
    int num = 665;  // 첫 번째로 검사할 때 ++ 연산 후 666부터 시작되도록

    while (count < N) {
        num++;
        // to_string으로 변환 후 "666" 포함 여부 검사
        if (to_string(num).find("666") != string::npos) {
            count++;
        }
    }

    cout << num << "\n";
    return 0;
}

*/
```

- `repaint_count(si, sj, board)` **함수**
  - `(si, sj)`를 왼쪽 위로 하는 8×8 영역을 두 가지 체스판 패턴과 비교.
  - `expected1`, `expected2` 로 각각 “W 시작” 패턴과 “B 시작” 패턴의 기대값을 계산.
  - 실제 색(`actual`)과 비교해 다르면 카운트 증가.
  - 두 패턴 중 더 작은 재칠 횟수를 반환.
- `main()` **함수**
  - `sys.stdin.readline` 으로 빠르게 입력을 읽음.
  - `n, m` 크기의 보드를 리스트로 저장.
  - 모든 가능한 8×8 블록의 시작 좌표 `(i, j)`에 대해 `repaint_count` 호출.
  - 최솟값을 갱신하고 최종적으로 출력.

▼ BOJ1436 영화감독 숌

```
'''666으로 끝나는 숫자들, 앞뒤에 들어가는 숫자들, 666으로 시작하는 숫자들'''

'''import sys
import itertools

def find_nth_movie(n):
    # 666부터 시작해서, 숫자에 '666'이 포함된 것만 걸러내는 제너레이터
    movies = (num for num in itertools.count(666) if '666' in str(num))
    # n번째(0-based가 아니므로 n-1) 요소를 꺼내서 반환
    return next(itertools.islice(movies, n-1, None))

if __name__ == "__main__":
    n = int(sys.stdin.readline())
    print(find_nth_movie(n))'''

'''itertools.count(666)
무한히 666, 667, 668, ... 을 생성하는 이터레이터를 만듭니다.

(num for num in ... if '666' in str(num))
숫자를 문자열로 바꿔 '666'이 포함된 것만 걸러내는 제너레이터 표현식을 사용합
```

```
itertools.islice(movies, n-1, None)
그 제너레이터에서 첫 번째(인덱스 0)가 666이므로, n번째를 얻으려면 n-1번째까

next(...)
최종적으로 n번째 “영화”의 번호를 출력합니다.
'''



'''
전체 흐름

1부터 차례로 num을 늘려 가며 "666" 포함 여부를 보고,

포함될 때마다 count를 하나씩 올립니다.

count가 입력 n과 같아지는 순간 num을 프린트하고 함수를 종료합니다.'''

import sys

def main():
    n = int(sys.stdin.readline())
    count = 0       #나중에 n과 같은 숫자가 될때까지 증가
    num = 0         #계속 1씩 증가

    while True:
        num += 1
        if '666' in str(num):
            count += 1
            if count == n:
                print(num)
                return

if __name__ == "__main__":
    main()
```

```
#include <iostream>
#include <string>
using namespace std;
```

```
int main() {
    int N;
    cin >> N;
    int cnt{ 0 };
    string num{ "666" };
    while (cnt < N) {
        for (int i{ 0 }; i < num.length(); i++) {
            if (num[i] == '6' && num[i + 1] == '6' && num[i + 2] == '6') { cnt+
        }
        num = to_string(stoi(num) + 1);
    }
    cout << stoi(num) - 1 << "\n";
}


/*
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int N;
    cin >> N;

    int count = 0;
    int num = 665;  // 첫 번째로 검사할 때 ++ 연산 후 666부터 시작되도록

    while (count < N) {
        num++;
        // to_string으로 변환 후 "666" 포함 여부 검사
        if (to_string(num).find("666") != string::npos) {
            count++;
        }
    }
```

```
    cout << num << "\n";
    return 0;
}

*/
```

소요 시간: 40분

리뷰: 1. c++로 할 때 모두 문자열로 생각해서 6 && 6 && 6을 검사하는 식으로 풀었다. (6의 연속성)

```
if (num[i] == '6' && num[i + 1] == '6' && num[i + 2] == '6') { cnt++; brea
```

2. python이 아직 익숙치 않은데, 더 간결하기도 하고, 어렵기도 한 것 같다.

   위의 코드를 파이썬으로 바꾸면

   ```
   if '666' in str(num):
           count += 1
   ```

   이게 끝이다. 짧고 간결하다.

   그 다음에 배운 건 파이썬 입출력 속도 높이기

   들어가는 입출력이 많을 때,
   c++에서는 ios_base::sync_with_stdio(false);   cin.tie(0); cout.tie(0);를 적어주는데,
   파이썬 에서도 대용량 입력시 필요한 메소드가 있고 패키지는 sys라고 한다.

   ```
   import sys
    n = int(sys.stdin.readline())   #\n까지 포함하기 때문에 몇줄을 받아올 수
                                    #EOF 시 ""를 반환한다.
   ```

▼ BOJ2839 설탕배달

소요 시간: 1시간

리뷰: c++

- 반복문을 while로 시작했다면 더 가독성있는 코드로 바꿀 수 있지 않았을까 생각한다.
- -1같은 세부적인 정보를 놓쳤다.
- idea

1. 먼저 5kg 봉지를 최대한 사용합니다.
2. 남은 무게가 3kg로 나눠 떨어지면 바로 정답을 출력합니다.
3. 그렇지 않으면, 5kg 봉지 개수를 줄여가며 남은 무게가 3kg로 나눠 떨어지는 경우를 찾습니다.
4. 정확히 나눠 떨어지지 않으면 -1을 출력한다.

▼ 2주차

▼ BOJ7785 회사에 있는 사람

```
import sys
input = sys.stdin.readline

n = int(input())
s = set()

for _ in range(n):
    name, cmd = input().split()
    if cmd == "enter":
        s.add(name)    #set, add()
    else:
        s.discard(name)  #나가면 제거 (없어도 에러 안 남)

for name in sorted(s, reverse=True):
    print(name)
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <unordered_map>
using namespace std;
```

```
unordered_map<string, bool> um;

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    int n;
    cin >> n;

    for(int i { 0 }; i < n;i++){
        string name;
        string ent_or_lea;
        cin >> name >> ent_or_lea;

        if( ent_or_lea == "enter"){
            um[name] = true; continue;
        }
        um[name] = false;
    }

    // 사무실에 남아 있는 사람만 골라서 벡터에 담고
    vector<string> ans;
    ans.reserve(um.size());  //reserve 성능 개선용(많은 할당, 복사 없어짐)
    for (auto& [name, here] : um) {
        if (here)               // 이 조건을 통과해야만 ans에 추가
            ans.push_back(name);
    }
    // Baha는 here가 false이므로 ans에 들어가지 않음

    // 사전 내림차순으로 정렬
    sort(ans.begin(), ans.end(), greater<string>());

    // 출력
    for (auto& s : ans) {
        cout << s << '\n';
    }
```

```
    return 0;
}
```

1. 앞에 입력 속도를 높이기 위해서 **input = sys.stdin.readline** 를 사용할 때에는 `input = sys.stdin.readline`이라는 한 줄의 코드가 `input()` 자체를 빠른 입력 함수로 바꿔버린다. `input = sys.stdin.readline`은 `input()`을 빠른 입력 함수(`readline`)로 **재정의한 것**이며, 이후 모든 `input()` 호출은 실제로는 `sys.stdin.readline()`을 부르는 것과 같다.

```
#안 좋은 예
data = input().split()  # 한 줄만 읽음 → 전체 출입 기록이 아니라 첫 줄만 처리됨
                        #data를 만드는 데 실패한다. 여러줄 입력이 안 됨.
#올바른 예
n = int(input())
for _ in range(n):
    name, action = input().split()  # 한 줄씩 읽고 나눔
```

sys.stdin.read()을 직접 사용하는 방법도 있다.

```
import sys

data = sys.stdin.read().split()
n = int(data[0])  # 첫 번째 값은 출입 기록 개수

s = set()
idx = 1  # data[1]부터 이름과 출입기록이 번갈아 나옴

for _ in range(n):
    name = data[idx]
    action = data[idx + 1]
```

`sys.stdin.read()`를 사용하면 **전체 입력을 한 번에 문자열로 읽은 뒤**, 이를 `split()`해서 처리한다.

2. discard(x) → 제거된 바를 조용히 무시
   remove(x) →KeyError 에러 발생

▼ BOJ2751 수 정렬하기 2

소요 시간: 15분

리뷰:

python의 경우 list.sort() : in-place로 list가 바뀌지 않는다.
또 sorted()의 경우 b=sorted(arr, reverse(True)) 처럼 list를 반환한다.

C++에서는 sort(arr, arr + 8, greater<string>()); 등 변화된 상태지만 반환하지는 않는다

이 외의 정렬도 있었다.
안정 정렬(stability):

Python: list.sort()와 sorted() 모두 **안정 정렬**(stable sort, Timsort)

C++: std::sort은 **불안정**(unstable), 동등 원소 순서 유지 불가

안정 정렬이 필요하면 std::stable_sort 사용

▼ BOJ1181 단어 정렬

소요 시간: 30m

리뷰: sort()의 3번째 인자
1.

```
bool compare(const pair<int, string>& a, const pair<int, string>& b){
    if (a.first != b.first){
        return a.first < b.first;
    }
    return a.second < b.second;
}
```

문제에서 sort 기준을 보면, 문자열 길이가 다르면, 짧은 문자열이 먼저이고,

길이가 같으면, 알파벳순으로 비교한다.

- `if (a.first != b.first)`

  문자열 길이가 다르면:

  - `a.first < b.first`가 `true`면 a가 앞에 와야 하므로 true 반환
  - `false`면 b가 앞에 와야 하므로 false 반환
- 그렇지 않으면 (즉, 길이가 같으면):
  - `a.second < b.second`로 알파벳순 비교

2. <vector>의 .emplace_back()

자주 쓰던 push_back(obj)은 객체를 복사(또는 이동)해서 벡터에 추가하던 방법
.emplace_back(args....) 객체를 백터 내부에서 직접 생성해서 추가

```
#include <vector>
#include <string>
#include <utility>
using namespace std;

vector<pair<int, string>> v;

// 1. push_back 방식
string input = "hello";
v.push_back(make_pair(input.length(), input));  // 임시 객체 생성

// 2. emplace_back 방식
v.emplace_back(input.length(), input);  // 여기서 바로 pair<int, string> 생
```

emplace_back()이 더 효율적인 이유
백터 내부에서 직접 생성, 불필요한 복사 없음으로 인한 속도 증가

3. `v = move(uniq);`

- `move(uniq)`는 **"uniq은 더 이상 안 쓸 테니 자원(메모리 등)을 `v`로 통째로 넘겨"**라는 뜻
- 복사보다 훨씬 빠르고 효율적
- 사용 후 `uniq`은 더 이상 유효한 데이터를 가지고 있다고 보지 않아야 함
- `std::move()`는 복사 대신 **"이동"**을 유도하는 함수이며, `#include <utility>`가 필요함

필요한 이유:

- 대량의 데이터를 다룰 때 **복사보다 훨씬 빠르고 메모리 효율이 좋기 때문**
- 특히 **임시 객체**나 **사용이 끝난 객체**를 옮길 때 매우 유용

4. 한 줄씩 설명:

- `vector<pair<int, string>> uniq;`
  - 중복 제거한 결과를 담을 새로운 벡터
- `uniq.reserve(v.size());`
  - 미리 메모리 할당해서 성능 최적화
- `string prev = "";`
  - 마지막으로 추가한 단어 저장용 (초기값은 빈 문자열)
- `for (const auto& p : v)`
  - 정렬된 `v`를 순회하면서
- `if (p.second != prev)`
  - 이전 단어와 다르면 (=중복이 아님)
- `uniq.push_back(p);`
  - 중복이 아니므로 새 벡터에 추가
- `prev = p.second;`
  - 현재 단어를 다음 비교를 위해 저장
- `v = move(uniq);`
  - 기존 벡터를 중복 제거한 것으로 교체 (복사 X, 이동 O)
- `for (const auto& e : v)`
  - 최종 결과 출력

▼ BOJ10815 숫자 카드

소요 시간: 파이썬으로는 c++보다 엄청 간단해져서 놀랐다. c++로 복잡하게 구현되던 게 (40m) 파이썬으로 는 10분 내지로 끝났다.

리뷰:

```
import sys
input = sys.stdin.read

words = input().split()[1:]  # 첫 줄은 단어 수, 나머지만 사용

unique_words = set(words)  # 중복 제거
```

```
sorted_words = sorted(unique_words, key=lambda x: (len(x), x))

print('\n'.join(sorted_words))
```

```
key=lambda x: (우선 기준, 그다음 기준, 그다음 기준...)     단어 길이순 -> 알파
sorted_words = sorted(unique_words, key=lambda x: (len(x), x))
```

```
sorted(unique_words, key=lambda x: (len(x), x))

/*집합 unique_words의 모든 원소를,
길이를 기준으로 먼저 정렬하고,
**길이가 같을 경우 알파벳순(사전순)**으로 정렬해라.*/
```