C++ 코딩테스트 때 자주 사용하는 STL 및 알고리즘 정리

1. std::vector

가장 많이 사용하는 벡터 자료형

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

int main(){
    vector<int> v;

    auto iter = find(v.begin(), v.end(), 178); // 178의 반복자 찾기
    auto idx = find(v.begin(), v.end(), 178) - v.begin() // 178의 인덱스 찾기

    v.insert(v.begin(), 100) // 100을 제일 처음에 추가
    v.insert(v.begin()+5, 5) // 5를 (처음주소 + 5)에 추가
    v.insert(v.end(), 9) // 9를 제일 마지막에 추가

    v.erase(v.begin()); // 제일 첫번째 원소 제거
    v.erase(v.end()-1); // 마지막에서 한 칸 앞에 원소 제거
    return 0;
}
```

```
#include <algoritm>

// 벡터에서 특정 원소 제거하는 방법
v.erase(v.find(v.begin(), v.end(), 'A'), v.end());

// 벡터에서 중복 제거하는 방법
sort(v.begin(), v.end());
v.erase(unique(v.begin(), v.end()), v.end()); // unique는 '연속된' 중복 원소를
vector의 제일 뒷부분으로 쓰레기 값으로 보내기 때문에 sorting 필요
```

2. std::map

cpp의 딕셔너리와 동일

`map: pair<const Key, T>` 로 이루어진 컨테이너. Key값 기준으로 sorting 되어있음. 이진탐색트리로 구현되면서 sorting하므로 unordered_map보다 value값을 찾는 데에 오래걸릴 수 있음, `#include <map>`에 포함

`unordered_map`: map과는 달리 Key 혹은 Value 기준으로 sorting 되어있지 않은 컨테이너. Key값으로 hash value를 찾는 데에 시간이 적게 걸림.(Average: O(1)) 우리가 흔히 사용하는 hash 자료구조에 해당된다고 보면 됨, `#include <unordered_map>`에 포함

```cpp
#include <map>

using namespace std;

int main(){
    map<string, int> m;
    m["12345678"] = 1234; // 삽입, 문자열 "12345678"에 저장된 값은 정수형 1234

    auto iter = m.find("12345678")
    if (um.find("bigbigpark") != um.end() ) // 찾지 못한다면 end()를 반환

    auto key = iter->first; // key에 접근한다, 문자열 "12345678" 반환
    auto value = iter->second; // value에 접근한다, 정수형 1234 반환

    m.erase("12345678") // 없으면 m.end()를 반환한다

    // vector의 경우 아래와 같이 반복문을 순회하며 값에 접근이 가능하다.

    vector<int> v = {1, 2, 3, 4, 5};
    for(auto n: v) // n으로 원소가 넘어옴
    {
        cout << n << endl;
    }

    // map의 경우 어떨까?
    map<string, int> um;
    um["A"] = 1111;
    um["B"] = 2222;

    for (auto it = m.begin(); it != m.end(); ++it) {
    answer.push_back(it->first);  // 'it' is an iterator, so use '->'
    }

    return 0;
}
```

3. std::string

문자열에 필수적인 라이브러리

```cpp
#include <string>
#include <algorithm>

int main(){
    string s = "bigbigpark";

    if (s.find("big") != std::string::npos) // 찾았을때 반환값은 해당 인덱스
    s.replace(pos, length, "big") // pos 위치에서 length 길이만큼 "big"으로 채운다
    s.erase(pos, len) // pos에서 len길이 만큼 삭제한다
    s.erase(s.begin()+len) // len 길이 만큼 삭제한다
```

```
    s.erase(remove(s.begin(), s.end(), 'A'), s.end()) // 문자열 s에서 'A'가 들어간
부분을 모조리 지운다

    string new_string = s.substr(5); // 5번 인덱스부터 끝까지 가져오기
    string new_string = s.substr(5, 1); // 5번 인덱스부터 1길이 만큼 가져오기

    string str = "bigbigpark";
    char c = str[0];

    int num = 5;
    string str = to_string(num);
    string n_str = "1122";
    int n1 = stoi(n_str); // string에서 int 변환
    double n2 = stod(n_str); // string에서 double 변환

    char ch = '9';
    int n = ch - '0'; // '0'를 빼게 되면 보이는 그대로 숫자가 저장되게 된다.
}
```

4. std::sort

모든 코딩테스트에서 거의 필수적인 정렬 알고리즘 사용법

```
#include <algorithm>
#include <vector>

using namespace std;

bool compare(int i, int j){
    return i < j; // true면 j가 앞으로, 아니라면 i가 앞으로
}

int main(){
    vector<int> v = {4, 2, 3, 5, 1};

    sort(v.begin(), v.end()); // 오름차순 정렬
    sort(v.begin(), v.end(), compare) // 내림차순 정렬

    return 0;
}
```

4. std::stack

```
#include <stack>

using namespace std;

int main(int argc, char** argv){
    stack<int> st;
```

```
    s.push(10); // 스택에 10추가
    cout << s.size(); // 1 출력

    if (s.empty()) // empty면 true
    s.pop(); // top에 있는 값 뽑기 (10을 반환하지 않음)
    s.top(); // top에 있는 값 확인
}
```

5. std::queue

```
#include <queue>

using namespace std;

int main(int argc, char** argv){
    queue<int> q;
    q.push(10); // 10추가
    cout << q.size(); // 1 출력
    q.pop(); // 가장 먼저들어온 10 뽑기 (10을 반환하지 않음)
    q.front(); // 가장 앞에 있는 값 확인
    q.back(); // 가장 뒤에 있는 값 확인
}
```

```
#include <queue>

using namespace std;

int main(int argc, char** argv){
    priority_queue<int> pq; // 힙 생성 (디폴트는 최대 힙!!!, 파이썬과 다름!!!)
    pq.push(10); // 10추가
    pq.pop(); // 루트 값 뽑기
    pq.top(); // 루트 값 출력
}
```

6. std::deque

```
#include <deque>

using namespace std;

int main(int argc, char** argv){
    deque<int> dq;
    dq.push_front(10); // 10
    dq.push_back(50); // 10 50
    dq.push_front(24); // 24 10 50
```

```
    for(auto x: dq) cout << x << ' '; // 24 10 50

    cout << dq.size() << endl; // 3
    cout << dq.empty() << endl; // False

    dq.pop_front(); // 10 50
    dq.pop_back(); // 10

    cout << dq.back() << endl; // 10
    dq.push_back(72); // 10 72
    cout << dq.front() << endl; // 10

    dq[2] = 17; // 10 72 17

    dq.insert(dq.begin()+1, 33); // 10 33 72 17
    dq.erase(dq.begin() + 3); // 10 33 72
    dq.clear(); // clear
}
```

실전문제 [백준, 프로그래스, 코드트리]

Hash

달리기 경주 - 프로그래머스

```
def solution(players, callings):
    race = players[:]

    for c in callings:
        for i in range(len(race)):
            if race[i] == c:
                race[i-1], race[i] = race[i], race[i-1]

    return race
```

```
def solution(players, callings):
    position = {player:idx for idx, player in enumerate(players)}

    for c in callings:
        rank = position[c] # 호명당한 선수의 현재 순위

        position[players[rank]] -= 1 # 호명당한 선수의 순위를 앞당김
        position[players[rank-1]] += 1 # 호명당한 선수의 앞에 있는 선수
        players[rank-1], players[rank] = players[rank], players[rank-1] # 두 선수
의 위치를 바꿈

    return players
```

> 반복문 최대한 안쓴다고 첫번째 코드를 작성했는데 이것 마저도 시간초과가 발생한다, dictionary를 잘 사용해보자

> dictionary의 key로 현재 순위에 접근, 현재 순위 앞에 있는 선수의 인덱스를 이용하여 이름을 알아내고 다시 딕셔너리에 접근, 두 선수를 교환

> 감탄할만한 풀이다

대충 만든 자판 - 프로그래머스

keymap을 모두 순회하면서 해당 알파벳의 최소 인덱스가 무엇인지 찾으면 되는 문제

```python
def solution(keymap, targets):
    answer = []
    memory = {}

    for k in keymap: # keymap을 완전 탐색하면서 최소 인덱스가 무엇인지 확인하기, memory 업데이트
        for i in range(len(k)):
            memory[k[i]] = (i + 1) if k[i] not in memory else min(memory[k[i]], i+1)

    for t in targets:
        cnt = 0
        for i in range(len(t)):
            if t[i] in memory:
                cnt += memory[t[i]]
            else:
                cnt = -1
                break
        answer.append(cnt)

    return answer
```

```cpp
#include <string>
#include <vector>
#include <unordered_map>

using namespace std;

vector<int> solution(vector<string> keymap, vector<string> targets) {
    vector<int> answer;
    unordered_map<char, int> um;

    for(string k: keymap){
        for(int i=0; i < k.length(); i++){
            if(um.find(k[i]) == um.end()){
                um[k[i]] = i+1;
```

```
            }
            else{
                um[k[i]] = min(i+1, um[k[i]]);
            }
        }
    }

    for(string t: targets){
        int cnt = 0;
        for(char c: t){
            if(um.find(c) != um.end()){
                cnt += um[c];
            } else{
                cnt = -1;
                break;
            }
        }
        answer.emplace_back(cnt);
    }
    return answer;
}
```

> 풀이자체는 직관적으로 떠올라서 분석할 내용은 없다

가희와 키워드 - 백준

중복된 키워드가 있으면 n에서 얼마를 뺄 것인가를 결정하는 문제

```
import sys
n, m = map(int, sys.stdin.readline().split())
note = {sys.stdin.readline().rstrip(): 1 for _ in range(n)}

for _ in range(m):
    keywords = sys.stdin.readline().rstrip().split(",")

    for k in keywords:
        if k in note:
            note[k] -= 1

            if not note[k] < 0:
                n -= 1
    print(n)
```

> 일단 키워드가 무조건 한번씩 등장하므로 hash 이용해서 노트를 구성

> 중복되는 키워드에 대해서 이미 1이 빼진 경우 n에서 카운트 빼는 것을 수행하지 않는다!

영단어 암기는 괴로워 - 백준

python의 강력한 sorted()를 이용하면 되는 문제

```
# 우선순위
# 1.자주 나오는 단어일수록 앞에 배치한다.
# 2.해당 단어의 길이가 길수록 앞에 배치한다.
# 3.알파벳 사전 순으로 앞에 있는 단어일수록 앞에 배치한다 (이건 건드릴 수가 없음)
import sys

n, m = map(int, sys.stdin.readline().split()) # 단어의 갯수, 외울 단어의 길이 기준
dictionary = {}

for _ in range(n):
    word = sys.stdin.readline().strip()
    if len(word) >= m: # 길이가 m 이상이어야 함
        if word not in dictionary:  # 'None' 체크 대신 'in' 사용
            dictionary[word] = 0

        else:
            dictionary[word] += 1

# 가장 많이 나오는 단어가 가장 앞에 배치됨 (제일 작은 -값)
# 단어가 길수록 앞에 배치됨 (제일 작은 -값)
# 알파벳 사전 순으로 정렬됨
dictionary = sorted(dictionary, key=lambda x: (-dictionary[x], -len(x), x))
print(*dictionary, sep="\n")
```

> if word not in dictionary의 존재를 알 수 있었다. 꽤 어려웠던 문제!

> sorted 부분이 문제였는데 우선순위 1, 2는 내가 정할 수 있어도 우선순위 3은 건들수가 없었다. 생각해 본 결과 우선순위 1, 2에 -값을 주면 원하는대로 정렬이 나온다는 것을 깨달아야 했던 문제

> 정렬 문제는 이제 좀 감을 잡은 것 같다

임스와 함께하는 미니게임 - 프로그래머스

```
# 임스와 여러 번 미니게임을 플레이하고자 하는 사람이 있으나,
# 임스는 한 번 같이 플레이한 사람과는 다시 플레이하지 않습니다.

import sys

n, game = sys.stdin.readline().split()
mini_game = {"Y": 2-1, "F": 3-1, "O": 4-1} # 임스를 포함해야 하므로 1씩 빼기
players = [sys.stdin.readline().strip() for _ in range(int(n))]
players = set(players)

answer = len(players) // mini_game[game]
print(answer)
```

> 가장 먼저 임스를 포함해야 하므로 -1씩 빼주는 것을 누락했다 이 부분을 추가

> 사람이 입장할때마다 리스트에 추가하고 조건을 검사할 필요가 없다. 한번에 입력을 다 받은 다음에 set으로 중복제거를 하고, 각 게임에 필요한 인원 수 만큼 나누어주면 바로 해결된다!

두개 뽑아서 더하기 - 프로그래머스

```
#include <string>
#include <vector>
#include <algorithm>
#include <map>

using namespace std;

vector<int> solution(vector<int> numbers) {
    vector<int> answer;
    map<int, int> m; // ordered map (오름차순 정렬)

    for(int i = 0; i < numbers.size() - 1; i++){
        for(int j = i + 1; j < numbers.size(); j++){
            int comb = numbers[i] + numbers[j];
            m[comb] = 0;
        }
    }

    for (auto it = m.begin(); it != m.end(); ++it) { // iterator도 연속된 공간에 위치하므로 1씩 차이난다
        answer.push_back(it->first);
    }

    /*
    for (auto it : m) {
        answer.push_back(it.first);
    }
    */


    return answer;
}
```

> 중복을 없애기 위해 map사용 (ordered_map 이므로 이미 오름차순 정렬이 되어있다)

> iterator와 원소 값의 차이를 명확히 알고 있을 것

폰켓몬 - 프로그래머스

```
#include <vector>
#include <map>
```

```
#include <algorithm>

using namespace std;

int solution(vector<int> nums)
{
    if (nums.size() == 1){
        return 1;
    }

    else{
        int targetNum = nums.size() / 2;

        map<int, int> bag;
        for(auto n: nums){
            bag[n]++;  // Simply increment the count of each number
        }

        int monsterType = bag.size(); // 몬스터 종류갯수

        if(monsterType > targetNum){ // 몬스터 종류 갯수가 더 많으면 N/2 가져가면 된다.
            return targetNum;
        }

        else{ // 아니라면 몬스터 종류만큼 가져가면 된다.
            return monsterType;
        }
    }
}
```

> Hash를 이용해서 중복을 처리하는 기법을 외워놓자

> 어찌보면 이 문제는 해시에 속하긴 하지만 그리디에 가까운 문제라고 생각한다. 문제를 푸는 전략 자체가, 총 몬스터의 갯수 N/2 만큼 가져가야 하므로

> 1. 몬스터의 종류가 N/2보다 많다면 몬스터의 종류만큼 가져가면 된다.
> 2. 몬스터의 종류가 N/2보다 적다면 N/2만큼 가져간다.

완주하지 못한 선수 - 프로그래머스

```
#include <string>
#include <vector>
#include <algorithm>
#include <map>

using namespace std;

string solution(vector<string> participant, vector<string> completion) {
    map<string, int> people;
```

```
    for (auto p: participant){
        people[p]++;
    }

    for (auto p: completion){
        people[p]--;
    }

    for(auto person: people){ // map을 for문으로 순회할때는 pair<const Key, T>, auto의 장점
        if (person.second != 0){
            return person.first;
        }
    }
}
```

> map에 대한 사용법을 좀더 확실하게 알 수 있는 문제 였다.

> 풀이 자체는 직관적으로 떠올라서 분석내용은 딱히 없다.

전화번호 목록 - 프로그래머스

```
#include <string>
#include <vector>

using namespace std;

bool solution(vector<string> phone_book) {
    for(int i=0; i < phone_book.size()-1; i++){
        string prefix = phone_book[i];
        for(int j=i+1; j < phone_book.size(); j++){
            string check = phone_book[j].substr(0, prefix.length());
            if(check == prefix){
                return false;
            }
        }
    }
    return true;
}
```

> 처음에 풀었던 방식, 정확성: 70.8, 반례를 찾자

```
#include <string>
#include <vector>
#include <algorithm>

using namespace std;
```

```
bool solution(vector<string> phone_book) {
    // 아스키 코드 값을 기준으로 정렬됨
    // 첫 번째 문자가 동일할 경우, 두 번째 문자를 비교하여 ASCII 값이 작은 쪽이 앞에 위치합니다.
    // 정렬 전: {"119", "97674223", "1195524421"};
    // 정렬 후: {"119", "1195524421", "97674223"};

    sort(phone_book.begin(), phone_book.end());

    for(int i=0; i < phone_book.size()-1; i++){
        string prefix = phone_book[i];
        string check = phone_book[i+1].substr(0, prefix.length());

        if(check == prefix){
            return false;
        }
    }
    return true;
}
```

> 반례를 찾기보다 문자열이 정렬될때의 정확한 기준을 알고 있으면 쉽게 풀 수 있는 문제이다

> 주석에 적어놓았듯이 prefix가 동일한 경우 ascii가 작은 순서대로 정렬된다, 그렇다면 반복문 한번에 확인할 수 있다는 것을 명심할 것!

> substr(0, index)의 사용법도 배울 수 있었다.

의상- 프로그래머스

```
#include <string>
#include <vector>
#include <unordered_map>
#include <iostream>

using namespace std;

int solution(vector<vector<string>> clothes) {
    int num = 0, ans = 1; // 총 가짓 수, 정답
    unordered_map<string, vector<string>> um;

    for (const auto &c: clothes){ // &는 reference(참조자, 복사가 이루어지지 않음)
        string clothing_item = c[0];
        string clothing_type = c[1];
        um[clothing_type].push_back(clothing_item);
        num++;
    }

    int answer = 1;
    for (const auto& value : um) {
```

```
            answer *= (value.second.size() + 1); // +1을 하는 이유는 각 카테고리에서 아
이템을 착용하거나 아무것도 착용하지 않는 선택을 할 수 있기 때문
        }

        return answer-1;
    }
```

> 전형적인 패턴을 찾아서 조합을 구하는 문제, size()+1은 공집합 포함

> n = size() + 1, r = 1이라 할때 각 카테고리 당 nCr로 가능한 조합을 모두 곱한다

> size()+1에 공집합이 포함되어있으므로 -1을 빼서 모두 공집합인 경우의 수를 제외한다.

스택/큐

햄버거 - 프로그래머스

```
def solution(ingredient):
    temp = []
    cnt = 0

    for i in ingredient:
        temp.append(i)

        if temp[-4:] == [1, 2, 3, 1]: # 백준 문자열 폭발 문제와 완전 동일함
            cnt += 1
            # temp = temp[:len(temp)-4]
            del temp[-4:] # slicing은 복사가 이루어지므로 시간초과 발생할 수 있음,
del이 훨씬 빠름

    return cnt
```

> 문제는 백준 문자열 폭발 문제와 완전 동일하다

> slicing하지 말고 del을 자주 쓰자, del이 훨씬 더 빠르다!

카드2 - 백준

누가봐도 덱 써서 푸는 문제

```
import sys
from collections import deque

# 회전하는 큐 문제
n = int(sys.stdin.readline())
cards = deque([i for i in range(1, n+1)])

while len(cards) > 2:
```

```
    cards.popleft()
    cards.append(cards.popleft())

print(cards[-1])
```

> 전형적인 자료구조를 활용하는 문제, 쉽다!

카드 뭉치 - 프로그래머스

```
#include <string>
#include <vector>
#include <queue>

using namespace std;

string solution(vector<string> cards1, vector<string> cards2, vector<string> goal)
{
    string answer = "";

    queue<string> c1;
    queue<string> c2;
    queue<string> g;

    for(int i= 0; i < cards1.size(); i++){
        c1.push(cards1[i]);
    }
    for(int j= 0; j < cards2.size(); j++){
        c2.push(cards2[j]);
    }
    for(int k = 0; k < goal.size(); k++){
        g.push(goal[k]);
    }

    for(int l = 0; l < goal.size(); l++){
        if(g.empty()){
            return "Yes";
        }

        else{
            if(!c1.empty() && g.front() == c1.front()){
                g.pop();
                c1.pop();
            }

            else if(!c2.empty() && g.front() == c2.front()){
                g.pop();
                c2.pop();
            }
        }
    }
```

```
    if(g.empty()){
        return "Yes";
    } else{
        return "No";
    }
}
```

> 맨처음에 return 조건을 `if(c1.empty() && c2.empty()) return "Yes";`로 했는데 어디가 틀린걸까?

> 흠 그냥 안전하게 `g.empty()`로 하자 이게 목적이니까 반례는 생각이 안난다

같은 숫자는 싫어 - 프로그래머스

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

using namespace std;

vector<int> solution(vector<int> arr)
{
    stack<int> s;
    vector<int> ans;

    for(int a: arr){
        if (s.empty()){
            s.push(a);
            ans.push_back(a);
        }

        else if(s.top() != a){
            s.push(a);
            ans.push_back(a);
        }
    }
    return ans;
}
```

> 중복제거 때문에 맨처음에 map을 떠올렸다가 순서를 기억해야 하므로 스택이 더 적합하다는 사실을 깨달음

기능개발 - 프로그래머스

```
#include <string>
#include <vector>
```

```
#include <cmath>
#include <queue>

using namespace std;

vector<int> solution(vector<int> progresses, vector<int> speeds) {
    vector<int> answer;

    queue<int> q;
    for(int i=0; i < speeds.size(); i++){
        double rest = 100.0 - static_cast<double>(progresses[i]);
        int finish = static_cast<int>(ceil(rest / static_cast<double>
(speeds[i])));
        q.push(finish);
    }

    while(!q.empty()){
        int top = q.front(); q.pop();
        int cnt = 1;

        while(!q.empty() && top >= q.front()){
            q.pop();
            cnt += 1;
        }
        answer.push_back(cnt);
    }

    return answer;
}
```

올바른 괄호 - 프로그래머스

```
#include <string>
#include <stack>
#include <iostream>

using namespace std;

bool solution(string s)
{
    if(s[0] == ')') return false; // 바로 리턴

    bool answer = true;
    stack<char> check;

    for(char c: s){
        if(c == '('){
            check.push(c);
        }
```

```
            else{
                if(!check.empty() && c == ')' && check.top() == '('){
                    check.pop();
                }
            }
        }

        if(check.empty()){
            return true;
        } else{
            return false;
        }
    }
```

> top 확인할 때 비어있으면 segmentation falut 발생함, empty()인지 먼저 체크할 것

주식가격 - 프로그래머스

```
#include <string>
#include <vector>
#include <stack>

using namespace std;

vector<int> solution(vector<int> prices) {

    vector<int> answer;

    for(int i=0; i < prices.size(); i++){
        int cnt = 0;
        for(int j=i+1; j < prices.size(); j++){
            cnt++;
            if(prices[i] > prices[j]) break;
        }
        answer.emplace_back(cnt);
    }

    return answer;
}
```

> prices의 최대 길이 100,000 이면 이중 반복문 돌리면 큰일 날법한데 중간에 break 들어가고 j가 N-1만큼 돌아서 시간초과 발생 X

Heap

최소 힙 - 백준

단순하게 힙 써서 푸는 문제

```python
import sys
import heapq

n = int(sys.stdin.readline())
hq = []

for _ in range(n):
    num = int(sys.stdin.readline())
        if len(hq) != 0:
            print(heapq.heappop(hq))
        else:
            print(0)

    else:
        heapq.heappush(hq, num)
```

> heapq.heapify()는 이미 완성된 리스트에만 적용할 수 있다라는 걸 명심하자

> heapq 처음에 선언할 때 리스트로 선언하고 풀면 된다!

> python의 heap의 디폴트 값은 최소 힙, cpp은 최대 힙

더 맵게 - 프로그래머스

```cpp
#include <string>
#include <vector>
#include <queue>

using namespace std;

int solution(vector<int> scoville, int K) {
    priority_queue<int> pq; // 디폴드는 최대 힙!!!

    for(int s: scoville){
        pq.push(-s); // 가장 작은 값이 가장 큰 값이 되도록 변경
    }

    int cnt = 0;

    while (-pq.top() < K){
        if (pq.size() < 2){
            return -1;
        }

        int min1 = -pq.top(); pq.pop();
        int min2 = -pq.top(); pq.pop();
        int n = -(min1 + (min2 * 2));
        pq.push(n);
        cnt++;
    }
```

```
    return cnt;
    }
```

> Heap의 정확한 사용법을 배울 수 있었던 문제, push(), pop()할때 python과 달리 값을 반환하지 않는다!!

정렬

최빈값

```python
def solution(array):
    answer = []
    set_array = set(array)

    for sa in set_array:
        answer.append([array.count(sa), sa])

    answer.sort(key=lambda x: x[0], reverse=True)

    if len(answer) >= 2 and answer[1][0] == answer[0][0]:
        return -1

    else:
        return answer[0][1]
```

```cpp
#include <string>
#include <vector>
#include <unordered_map>
using namespace std;

int solution(vector<int> array) {
    int answer = 0;
    int maxNum = 0;

    // 정수와 해당 정수의 등장 횟수를 저장하기 위한 unordered_map
    unordered_map<int, int> um;

    // 배열을 순회하며 각 숫자의 등장 횟수를 세는 과정
    for (const auto num : array) {
        um[num]++;
    }

    // unordered_map을 순회하며 최빈값을 찾는 과정
    for (const auto& num : um) {
        if (num.second > maxNum) { // 현재 등장 횟수가 최빈값보다 큰 경우
            maxNum = num.second; // 최빈값을 업데이트
            answer = num.first; // 해당 숫자를 정답으로 설정
        } else if (num.second == maxNum) { // 현재 등장 횟수가 최빈값과 같은 경우
```

```
                answer = -1; // 최빈값이 여러 개인 경우 -1을 정답으로 설정
            }
        }
        return answer;
}
```

> cpp하고 python의 풀이방법이 엄청나게 차이난다. python의 count 함수 덕분

문자열 내 마음대로 정리하기 - 프로그래머스

```
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

int N;

// 오름차순은 < , 내림차순은 > 형태이다.
bool comp(string a, string b){
    if (a[N] == b[N]){
        return a < b; // true면 b는 뒤에 정렬, false면 a가 앞에 정렬
    }
    else{
        return a[N] < b[N];
    }
}

vector<string> solution(vector<string> strings, int n) {
    N = n;

    sort(strings.begin(), strings.end(), comp);

    return strings;
}
```

> python sort와 비슷하게 내가 조건을 설정해 줄 수 있다. < 이면 오름차순 > 내림차순 이란걸 쉽게 기억하자!

과일 장수 - 프로그래머스

```
#include <string>
#include <vector>
#include <algorithm>
#include <numeric>

using namespace std;
```

```
// vector를 slice하는 함수
vector<int> slice(vector<int> &score, int a, int b) {
    return vector<int>(score.begin() + a, score.begin() + b); // Corrected slice
bounds
}

int solution(int k, int m, vector<int> score) {
    int price = 0;

    sort(score.begin(), score.end(), greater<>()); // 내림차순 정렬

    int idx = 0;
    while (idx + m - 1 < score.size()) {
        vector<int> segScore = slice(score, idx, idx + m); // Slice m elements
        price += (segScore[m-1] * m); // Multiply smallest value by m
        idx += m; // Move forward by m
    }

    return price;
}
```

> *min_element(segScore.begin(), segScore.end()) 와 같이 사용할 수도 있음

> *이 붙는 이유는 min_element나 max_element 모두 iterator를 반환하기 때문

정수 내림차순으로 배치하기 - 프로그래머스

```
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

long long solution(long long n) {
    string s = to_string(n);
    vector<char> nums(s.length());
    for(char c: s){
        nums.push_back(c);
    }

    sort(nums.begin(), nums.end(), greater<>()); // 내림차순 정렬

    string answer;

    for(char n:nums){
        answer += n;
    }

    return stoull(answer);
}
```

> long long 굉장히 오랜만에 보는 자료형이다

> stoi(): string to int

> stoil(): string to unsigned long

> stof(): string to float

> stod(): string to double

> stold(): string to long double

K번째 수 - 프로그래머스

```cpp
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

vector<int> vecSlice(vector<int> inp, int a, int b) { // vector slicing 함수
    return vector<int>(inp.begin() + a, inp.begin() + b);
}

vector<int> solution(vector<int> array, vector<vector<int>> commands) {
    vector<int> answer;

    for(int i=0; i < commands.size(); i++){
        vector<int> new_arr = vecSlice(array, commands[i][0]-1, commands[i][1]);
// a이상 b미만
        sort(new_arr.begin(), new_arr.end());

        int num = new_arr[commands[i][2]-1];
        answer.emplace_back(num);
    }
    return answer;
}
```

> vector slicing 하는법을 정확히 알아놓자

> index 접근하는 법도 명확하게 알아 놓아야 함, b미만임을 명심

가장 큰 수 - 프로그래머스

```cpp
#include <string>
#include <vector>
#include <algorithm>
```

```cpp
using namespace std;

bool compare(int a, int b) {
    string sa = to_string(a); // 숫자를 문자열로 변환
    string sb = to_string(b); // 숫자를 문자열로 변환

    string n1 = sa + sb; // 문자열 합치기
    string n2 = sb + sa; // 문자열 합치기


    int num1 = stoi(n1);
    int num2 = stoi(n2);

    return num1 > num2;
}

string solution(vector<int> numbers) {
    sort(numbers.begin(), numbers.end(), compare);

    string answer = "";

    for (int i = 0; i < numbers.size(); i++) {
        answer += to_string(numbers[i]);
    }

    if (answer[0] == '0') {
        return "0";
    }

    else{
        return answer;
    }
}
```

> 이미 풀이법을 외워버린 문제, 맨앞에 "0"이 왔다는 것은 입력이 {0, 0, 0, 0} 이라는 의미이므로 바로 "0"을 리턴한다.

H-index - 프로그래머스

```cpp
#include <string>
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

int solution(vector<int> citations) {
    int h_idx = 0;
    int n  = citations.size();
```

```
    while (true) {
        int cnt = 0;
        for(auto c: citations){
            if(c >= h_idx){
                cnt++;
            }
        }
        if(cnt >= h_idx && n-cnt <= h_idx){
            h_idx++;
        }
        else{
            break;
        }
    }
    return h_idx-1;
}
```

> 굳이 정렬을 쓰지 않아도 풀 수 있는문제, 마지막에 h_idx-1을 하는 이유는 조건맞으면 h_idx++해주기 때문에 마지막에는 -1을 빼주는 게 올바른 구현

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


int solution(vector<int> citations) {
    int answer = 0;

    sort(citations.begin(), citations.end(), greater<>()); // 오름차순 정렬

    if (!citations[0]) return 0; // 제일 많은 인용횟수가 0이면 볼 필요도 없음

    for(int i=0; i<citations.size(); i++){ // {6, 5, 3, 1, 0}
        if(citations[i] >= i+1){ // i+1이 h_index
            answer = i+1;
        }
    }
    return answer;
}
```

> 다른 풀이는 i 값을 토대로 순차적으로 풀었다. 이렇게 하면 오름차순 정렬이 필수적이긴 하겠다.

완전탐색

대피소 - 백준

백트래킹과 완전탐색이 섞여 있는 문제

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

using namespace std;

int N, K;
vector<pair<int, int>> homes;
vector<pair<int, int>> shelters; // 쉼터 조합
vector<vector<pair<int, int>>> shelterCombinations; // 모든 쉼터 조합 저장

vector<pair<int, int>> tmp; // 현재 선택한 쉼터 조합

// 백트래킹으로 N개 중에 K개 선택 (nCk)
void makeShelter(int idx) {
    if (tmp.size() == K) {
        shelterCombinations.push_back(tmp); // 쉼터 조합 저장
        return;
    }
    for (int i = idx; i < homes.size(); i++) {
        tmp.push_back(homes[i]);
        makeShelter(i + 1); // 다음 쉼터 선택
        tmp.pop_back();
    }
}

vector<int> solve() {
    int maxDist;
    vector<int> distances;

    // 모든 쉼터 조합에 대해 계산
    for (auto& shelter : shelterCombinations) {
        maxDist = 0;
        // 각 집에 대해
        for (auto& h : homes) {
            int minDist = 10000000000; // 가장 가까운 대피소와 집 사이의 거리 중 가장
큰 값
            for (auto& s : shelter) {
                int dist = abs(h.first - s.first) + abs(h.second - s.second);
                minDist = min(minDist, dist); // 최소 거리 찾기
            }
            maxDist = max(maxDist, minDist); // 집들 중 최대 거리 찾기
        }
        distances.push_back(maxDist); // 각 쉼터 조합에 대해 최대 거리 저장
    }
    return distances;
}

int main(int argc, char** argv) {
    cin >> N >> K;

    // 집 좌표 입력
    for (int i = 0; i < N; i++) {
```

```
        int x, y;
        cin >> x >> y;
        homes.push_back({x, y});
    }

    makeShelter(0); // N개 집 중 K개 쉼터 선택
    vector<int> distances = solve();
    cout << *min_element(distances.begin(), distances.end()) << endl; // 최소 거리
출력
    return 0;
}
```

```
import sys
from itertools import combinations

n, k = map(int, sys.stdin.readline().split())
homes = [tuple(map(int, sys.stdin.readline().split())) for _ in range(n)]
maximum_dist = []

for shelter in combinations(homes, k):
    if n != k:
        n_shelter = list(set(homes) - set(map(tuple, shelter))) # 대피소가 아닌 집
들

    else: # 대피소가 아닌 집들이 없음, 다 대피소임
        n_shelter = shelter

    distances = []

    for i in n_shelter:
        min_dist = 10000000000
        for j in shelter:
            dist = (abs(i[0]-j[0]) + abs(i[1]-j[1]))
            min_dist = min(min_dist, dist)
        distances.append(min_dist)

    maximum_dist.append(max(distances))

print(min(maximum_dist))
```

> 조합을 만드는 방법에 대해서 배웠다, 재귀호출 할 때 idx를 하나씩 밀면 된다

> LIG 넥스원 기출이니까 꼭 풀이법을 알아 놓자

소수 만들기 - 프로그래머스

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

```cpp
#include <algorithm>

using namespace std;

int isPrime(const int &num){
    for(int i=2; i < num; i++){
        if(num % i == 0){
            return 0;
        }
    }
    return 1;
}

int solution(vector<int> nums) {
    sort(nums.begin(), nums.end());

    int answer = 0;

    for(int i = 0; i < nums.size() - 2; i++){ // O(N^3)
        for(int j = i + 1; j < nums.size() - 1; j++){
            for(int k = j + 1; k < nums.size(); k++){
                int number = nums[i] + nums[j] + nums[k];
                answer += isPrime(number);
            }
        }
    }

    return answer;
}
```

> nums의 최대길이가 50이라서 삼중 반복문 안돌아도 시간초과가 나지 않는다!

Greedy

문자열 폭발 - 백준

```python
import sys

stack = []
s = list(sys.stdin.readline().strip())
bomb = sys.stdin.readline().strip()
lb = len(bomb)

for i in range(len(s)):
    stack.append(s[i])

    if (len(stack) >= lb):
        if ''.join(stack[len(stack)-lb:]) == bomb:
            del stack[-lb:]
```

```
if not stack:
    print("FRULA")
else:
    print(''.join(stack))
```

```
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>

using namespace std;

// Function to slice the vector (optional, not strictly needed)
vector<char> slice(const vector<char> &vec, int a) {
    return vector<char>(vec.end() - a, vec.end());
}

int main(int argc, char** argv) {
    string in, bomb;
    cin >> in >> bomb;

    vector<char> stack;

    for(char c: in) {
        stack.push_back(c);
        if (stack.size() >= bomb.length()) {
            bool flag = true;
            for (int i = 0; i < bomb.length(); i++) {
                if (stack[i + stack.size() - bomb.length()] != bomb[i]) {
                    flag = false;
                    break;
                }
            }
            if (flag) {
                for (int i = 0; i < bomb.length(); i++) {
                    stack.pop_back();
                }
            }
        }
    }

    if (stack.empty()) {
        cout << "FRULA\n";
    } else {
        string tmp(stack.begin(), stack.end()); // 복사를 간편하게 할 수 있다, 기억
해 놓을 것
        cout << tmp << "\n";
    }

    return 0;
}
```

> LIG 넥스원 기출이니까 꼭 풀이법을 알아 놓자

수 이어 쓰기 - 백준

```
import sys

series = list(sys.stdin.readline().rstrip())
s_dict = {i: series.count(i) for i in set(series)}
num = 1

while True:
    s_num = str(num)

    for s in s_num:
        if s in s_dict:
            s_dict[s] -= 1

    if sum(s_dict.values()) == 0:
        print(num)
        break

    else:
        num += 1
```

> 시간초과 코드, 나도 안다 num이 최대 3000자리의 숫자이므로 시간초과 뜨는게 당연함

> 무엇보다도 이 코드가 틀린건 s_dict로 바꾸면서 입력받은 수열을 고려하지 않는다는 점

> N=20이고 입력이 234092일때, 12에서 s_dict={'2':0}이 되버림

```
import sys

series = list(sys.stdin.readline().rstrip())

idx = 0
num = 1

while True:
    s_num = str(num)
    for s in s_num:
        if series[idx] == s:
            idx += 1

            if idx == len(series):
                print(num)
                exit()

    num += 1
```

> for 문 돈다음에 조건 맞으면 while 문까지 나가야함, exit()로 코드 종료 시켜버리기

주유소 - 백준

한번에 안풀려서 참고한 문제, 전반적인 생각 방법은 거의 동일했다.

```
import sys

n = int(sys.stdin.readline()) # 도시의 갯수
length = list(map(int, sys.stdin.readline().split())) # 도로의 길이
price = list(map(int, sys.stdin.readline().split())) # 주유소의 가격

minPrice = price[0]
cost = price[0] * length[0] # 처음에 기름 무조건 넣어야 함

# 가격을 최소화 하기 위해, 가장 싼 곳에서 가장 많이 구해야 한다. 따라서 가격의 최솟값을 찾아야 한다.
# 가장 싼곳에서 가야할 길이 만큼 기름을 구매한다.
for i in range(1, n-1):
    minPrice = min(minPrice, price[i])
    cost += minPrice * length[i]

print(cost)
```

> 주석에 써놓은 대로 가격을 최소화 하기 위해 가장 싼 곳에서 가장 많이 구매해야 한다.

> 초반에는 index를 2개를 놓고 while문으로 접근했으나 세상에 어느 문제가 while문을 두개씩이나 쓰면서 접근하겠나

> (다음 주유소와도 비교했을 때) 현재 주유소 가격이 최소라면 거리 값을 그대로 곱해서 비용에 더한다라는게 이 문제를 푸는 방식

보물 - 백준

B가 재배열하면 안된다고 하지만, 재배열해도 뭔상관임? B를 내림차순으로 정렬하자

```
import sys

# S = A[0] × B[0] + ... + A[N-1] × B[N-1] 를 작게하는 최소 값
n = int(sys.stdin.readline())
A = list(map(int, sys.stdin.readline().split()))
B = list(map(int, sys.stdin.readline().split()))

# 전략: B의 max와 A의 min이 곱해지면 된다

A_new = sorted(A) # 오름차순 정렬
B_new = sorted(B, reverse=True) # 내림차순 정렬
total = sum([A_new[i]*B_new[i] for i in range(n)])
```

```
print(total)
```

> A의 최소와 B의 최대가 서로 곱해지면 되는 단순한 문제!

로프 - 백준

```
import sys

# k개의 로프를 사용하여 중량이 w인 물체를 들어올릴 때,
# 각각의 로프에는 모두 고르게 w/k 만큼의 중량이 걸리게 된다.

n = int(sys.stdin.readline()) # 로프의 갯수
weight = [int(sys.stdin.readline()) for _ in range(n)] # 각 로프가 버틸 수 있는 최대 중량

if n == 1:
    print(weight[0])

else:
    weight.sort()
    maxWeight = weight[-1]

    for k in range(n, 0, -1): # 사용할 로프의 갯수
        minWeight = weight[n-k]
        maxWeight = max(maxWeight, minWeight * k)

    print(maxWeight)
```

> 참고한 답, 먼저 최대 무게를 weight의 max값으로 초기화 한다.

> 모든 로프를 사용할 필요가 없으므로, weight를 순회하면서 weight[i]* i를 해준다!

회의실 배정 - 백준

코드트리에서 풀어본 문제, 회의가 빨리 끝나는 순서대로 오름차순 정렬하면 최대한 많은 회의를 진행할 수 있다

```
import sys

n = int(sys.stdin.readline())
meetings = [list(map(int, sys.stdin.readline().split())) for _ in range(n)]

# 회의가 빨리 끝나는 순서대로 정렬, 끝나는 시간이 같으면 시작하는 순서대로 정렬
meetings.sort(key=lambda x: (x[1], x[0]))

cnt = 1 # 최소 1개의 회의는 진행할 수 있으므로 1로 초기화

curr_meeting = meetings[0]
```

```
for i in range(1, len(meetings)):
    next_meeting = meetings[i]

    if curr_meeting[1] <= next_meeting[0]:
        cnt += 1
        curr_meeting = next_meeting

print(cnt)
```

> 조건이 맞을때만 비교해야 할 현재 미팅을 바꿔준다

> 정렬 시 만약 끝나는 시간이 동일하다면, 회의 시작 시간이 빠른 순서대로 정렬하는 것이 맞다!

동전 0 - 프로그래머스

코드트리에서 풀어본 문제, 동전이 서로 배수 관계에 있으므로 가장 높은 금액부터 쓰면 된다.

```
# 동전을 적절히 사용해서 그 가치의 합을 K
# 이때 필요한 동전 개수의 최솟값을 구하는 프로그램

import sys

n, k = map(int, sys.stdin.readline().split())
coins = [int(sys.stdin.readline()) for _ in range(n)] # 동전의 가치 Ai가 오름차순,
배수 관계

cnt = 0

for i in range(len(coins)-1, -1, -1):
    if k == 0:
        break

    cnt += k // coins[i]
    k %= coins[i]

print(cnt)
```

> 동전이 배수관계가 아니라면 적용 불가능한 풀이방법

구명보트 - 프로그래머스

```
def solution(people, limit):
    answer = 0
    people.sort() # 정렬 후 몸무게 큰 사람과 작은 사람을 짝지음

    idx1, idx2 = 0, len(people) - 1

    while(idx1 <= idx2):  # 두 명씩 구출해서 다 구출할 때까지 비교
```

```
        minWeight = people[idx1]
        maxWeight = people[idx2]

        if(minWeight + maxWeight <= limit):  # limit 보다 작거나 같으면 둘다 구출 가
능
            idx1 += 1
            idx2 -= 1

        else: # 안되면 일단 몸무게 큰 사람부터 먼저 구출
            idx2 -= 1

        answer += 1

    return answer
```

```
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

int solution(vector<int> people, int limit) {
    int answer = 0;

    sort(people.begin(), people.end());
    int idx1 = 0, idx2 = people.size() - 1;
    while (idx1 <= idx2){
        int minWeight = people[idx1];
        int maxWeight = people[idx2];

        if(minWeight + maxWeight > limit){
            idx2--;
        }

        else{
            idx1++;
            idx2--;
        }
        answer++;
    }
    return answer;
}
```

> LIG 넥스원 기출이니까 꼭 풀이법을 알아 놓자

> 정렬 후 투포인터로 보트의 갯수를 세는 문제이다. idx1, idx2 증감연산이 조금 헷갈려서 헤맸던 문제

예산 - 프로그래머스

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <numeric>
#include <algorithm>

using namespace std;

int solution(vector<int> d, int budget) {
    if(accumulate(d.begin(), d.end(), 0) == budget){
        return d.size();
    }
    else{
        sort(d.begin(), d.end()); // 오름차순 정렬
        int cnt = 0;

        for(int i=0; i<d.size(); i++){
            budget -= d[i];

            if(budget >= 0){
                cnt++;
            }
            else break;
        }
        return cnt;
    }
}
```

> 자 그럼 왜 정렬을 하느냐? 최대한 많은 부서에게 예산을 분배해야 하므로 가장 작은 부서부터 차례대로 예산을 분배하면 가장 많이 분배할 수 있다

> 그리디 문제는 결국 전략 싸움이다

돌 게임 - 백준

코드 자체가 너무 단순해서 놀랐던 문제, 다이나믹 프로그래밍이 무엇인지 조금 더 알 수 있는 문제 였다고 생각한다. 이러한 유형의 문제를 풀기 위해서는 결국 패턴을 파악해야 한다.

주어진 돌의 갯수가 홀수일 경우 항상 상근이가 이기고 짝수일 경우 항상 창영이가 이긴다라는 패턴을 찾아야 함!

```python
import sys
n = int(sys.stdin.readline())
print("CY" if n % 2 == 0 else "SK")
```

```cpp
#include <iostream>
#include <string>
```

```cpp
using namespace std;

int main(int argc, char** argv){
    int n;
    cin >> n;

    if (n % 2 == 0){
        cout << "CY\n";
    }

    else{
        cout << "SK\n";
    }
    return 0;
}
```

DFS/BFS

무인도 여행 - 프로그래머스

```python
import sys
sys.setrecursionlimit(10000)

n, m = 0, 0
directions = [[-1, 0], [1, 0], [0, -1], [0, 1]]
grid, visited = [] , []

def inRange(x, y):
    return x >= 0 and x < n and y >= 0 and y < m

def isX(x, y):
    return grid[x][y] == "X"

def isVisited(x, y):
    return visited[x][y]

def canGo(x, y):
    return (inRange(x, y)) and (not isX(x, y)) and (not isVisited(x, y))

def dfs(x, y):
    visited[x][y] = True
    area = int(grid[x][y])

    for direction in directions:
        dx, dy = direction[0], direction[1]
        nx, ny = x + dx, y + dy

        if canGo(nx, ny):
            area += dfs(nx, ny)
```

```
    return area

def solution(maps):
    global n, m, grid, visited

    answer = []
    grid = maps[:]
    n = len(maps)
    m = len(maps[0])

    visited = [[False] * m for _ in range(n)]  # visited 리스트 초기화

    for i in range(n):
        for j in range(m):
            if canGo(i, j):
                area = dfs(i, j)
                answer.append(area)

    if not answer:
        return [-1]

    else:
        return sorted(answer)
```

> 백준의 그림 문제와 거의 동일한 문제이다. dfs를 돌면서 무인도의 크기를 계속 누적하면 된다

DFS와 BFS - 백준

복습만이 살길이다. DFS/BFS 복습하자!

```
import sys
from collections import deque
sys.setrecursionlimit(10000)

n, m, v = map(int, sys.stdin.readline().split())
graph = [[] for _ in range(n+1)]

# 양방향 그래프
for _ in range(m):
    start, end = map(int, sys.stdin.readline().split())
    graph[start].append(end)
    graph[end].append(start)

# 정점 번호가 작은 것부터 방문
for i in range(1, n+1):
    graph[i].sort()

def dfs(x):
    visited[x] = True # 현재 기준 정점 방문 처리
    print(x, end=" ")
```

```
    for currV in graph[x]:
        if not visited[currV]: # 방문되지 않았다면
            dfs(currV) # currV와 연결된 다른 정점들 탐색

def bfs(x):
    q = deque([x])
    visited[x] = True
    print(x, end=" ")

    while q:
        currV = q.popleft()
        for i in graph[currV]: # currV와 연결된 다른 정점들 탐색
            if not visited[i]:
                visited[i] = True # 방문 처리
                print(i, end=" ")
                q.append(i)

# DFS
visited = [False for _ in range(n+1)]
dfs(v)
print()

# BFS
visited = [False for _ in range(n+1)]
bfs(v)
```

> 오랜만에 푸니까 기억이 안나네, 양방향 그래프를 만들 때 grid형식보다 연결리스트 방식이 공간복잡도가 훨씬 적다는 것을 명심하자!

그림 - 백준

BFS로 푼 방식

```
#include <iostream>
#include <queue>
#include <vector>
#include <algorithm>

using namespace std;

int n, m;
int grid[500][500] = {0, };
bool visited[500][500] = {false, };
int dirs[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}}; // 상하좌우
queue<pair<int, int>> q;

bool inRange(int x, int y){
    return x >= 0 && x < n && y >= 0 && y < m;
}

bool isVisited(int x, int y){
```

```
    return visited[x][y];
}

bool canGo(int x, int y){
    return inRange(x, y) && !isVisited(x, y) && grid[x][y] == 1;
}

int bfs(pair<int, int> pos){
    int area = 1; // 시작점도 포함하므로 1로 시작

    q.push({pos.first, pos.second}); // 방문 안했으면 큐에 집어 넣는다
    visited[pos.first][pos.second] = true; // 방문처리

    while(!q.empty()){
        int x = q.front().first;
        int y = q.front().second;
        q.pop(); // 방문처리 했으므로 큐에서 제거

        for(int i=0; i < 4; i++){
            int dx = dirs[i][0], dy = dirs[i][1];

            if(canGo(x+dx, y+dy)){
                q.push({x+dx, y+dy});
                visited[x+dx][y+dy] = true;
                area++; // 그림의 넓이증가
            }
        }
    }
    return area;
}

int main(int argc, char** argv){
    int maxArea = 0;

    cin >> n >> m;

    // grid 생성
    for(int i=0; i < n; i++){
        for(int j=0; j < m; j++){
            cin >> grid[i][j];
        }
    }

    int paintingNum = 0;

    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            if(canGo(i, j)){ // 그림인 경우에만 BFS 시작
                int area = bfs({i, j});
                maxArea = max(maxArea, area);
                paintingNum++;
            }
        }
    }
```

```
    cout << paintingNum << "\n" << maxArea;
    return 0;
}
```

DFS로 푼 방식

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

using namespace std;

int n, m;
int grid[500][500] = {0, };
bool visited[500][500] = {false, };
int dirs[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}}; // 상하좌우
int area = 1; // 그림의 넓이

bool inRange(int x, int y){
    return x >= 0 && x < n && y >= 0 && y < m;
}

bool isVisited(int x, int y){
    return visited[x][y];
}

bool canGo(int x, int y){
    return inRange(x, y) && !isVisited(x, y) && grid[x][y] == 1;
}

void dfs(pair<int, int> pos){
    visited[pos.first][pos.second] = true; // 방문처리

    int x = pos.first;
    int y = pos.second;

    for(int i=0; i < 4; i++){
        int dx = dirs[i][0], dy = dirs[i][1];

        if(canGo(x+dx, y+dy)){
            area++; // 그림의 넓이증가
            dfs({x+dx, y+dy});
        }
    }
}

int main(int argc, char** argv){
    int maxArea = 0;
    cin >> n >> m;

    // grid 생성
    for(int i=0; i < n; i++){
```

```
        for(int j=0; j < m; j++){
            cin >> grid[i][j];
        }
    }

    int paintingNum = 0;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            if(canGo(i, j)){ // 그림인 경우에만 BFS 시작
                dfs({i, j});
                maxArea = max(maxArea, area);
                area = 1; // 전역변수 area 초기화
                paintingNum++;
            }
        }
    }
    cout << paintingNum << "\n" << maxArea;
    return 0;
}
```

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

using namespace std;

int n, m;
int grid[500][500] = {0, };
bool visited[500][500] = {false, };
int dirs[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}}; // 상하좌우

bool inRange(int x, int y){
    return x >= 0 && x < n && y >= 0 && y < m;
}

bool isVisited(int x, int y){
    return visited[x][y];
}

bool canGo(int x, int y){
    return inRange(x, y) && !isVisited(x, y) && grid[x][y] == 1;
}

int dfs(pair<int, int> pos){
    visited[pos.first][pos.second] = true; // 방문 처리
    int area = 1; // 자기 자신(현재 위치)도 넓이에 포함

    int x = pos.first;
    int y = pos.second;

    for(int i = 0; i < 4; i++){
```

```
        int dx = dirs[i][0], dy = dirs[i][1];  // dirs로 수정

        if(canGo(x + dx, y + dy)){
            area += dfs({x + dx, y + dy}); // 재귀적으로 연결된 영역의 넓이를 더함
        }
    }

    return area; // 최종적으로 넓이를 반환
}

int main(int argc, char** argv){
    int maxArea = 0;
    cin >> n >> m;

    // grid 생성
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cin >> grid[i][j];
        }
    }

    int paintingNum = 0;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            if(canGo(i, j)){ // 방문하지 않은 그림 부분에 대해 DFS 시작
                int area = dfs({i, j}); // 해당 그림의 넓이를 계산
                maxArea = max(maxArea, area); // 최대 넓이 갱신
                paintingNum++; // 그림의 개수 증가
            }
        }
    }
    cout << paintingNum << "\n" << maxArea;
    return 0;
}
```

> 첫번째 코드는 반환 없이 작동하는 dfs, 두번째 코드는 area를 리턴하는 dfs

> 첫번째 코드에서 BFS와는 다르게 재귀로 작동하므로 dfs호출전에 `area++`를 실행해줘야 한다.

토마토 - 백준

```
#include <iostream>
#include <queue>
#include <vector>
#include <algorithm>

// 만약, 저장될 때부터 모든 토마토가 익어있는 상태이면 0을 출력해야 하고,
// 토마토가 모두 익지는 못하는 상황이면 -1을 출력해야 한다.

using namespace std;
```

```
int n, m;
int grid[1000][1000] = {0, };

int dirs[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}}; // 상하좌우
queue<pair<int, int>> q;

bool inRange(int x, int y) {
    return x >= 0 && x < n && y >= 0 && y < m;
}

bool canGo(int x, int y) {
    return inRange(x, y) && grid[x][y] == 0; // 아직 익지 않은 토마토
}

pair<int, int> bfs() {
    int day = 0;
    int ripeTomato = 0;    // 익은 토마토의 수

    while (!q.empty()) {
        int size = q.size(); // 현재 레벨의 크기

        for (int s = 0; s < size; s++) {
            int x = q.front().first;
            int y = q.front().second;
            q.pop(); // 큐에서 제거
            ripeTomato++; // 익은 토마토 수 증가

            for (int i = 0; i < 4; i++) {
                int dx = dirs[i][0], dy = dirs[i][1];
                if (canGo(x + dx, y + dy)) {
                    grid[x + dx][y + dy] = 1; // 토마토를 익었다고 표시
                    q.push({x + dx, y + dy}); // 익은 토마토를 큐에 추가
                }
            }
        }
        if (!q.empty()) day++; // 큐에 아직 토마토가 남아 있으면 하루가 지나야 함
    }

    return {ripeTomato, day};
}

int main() {
    int totalTomato = 0;
    pair<int, int> ans;
    cin >> m >> n;

    // grid 생성
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> grid[i][j];
            if (grid[i][j] == 0 || grid[i][j] == 1) {
                if(grid[i][j] == 1){
                    q.push({i, j}); // 1인지점 모두 큐에 넣기
                }
```

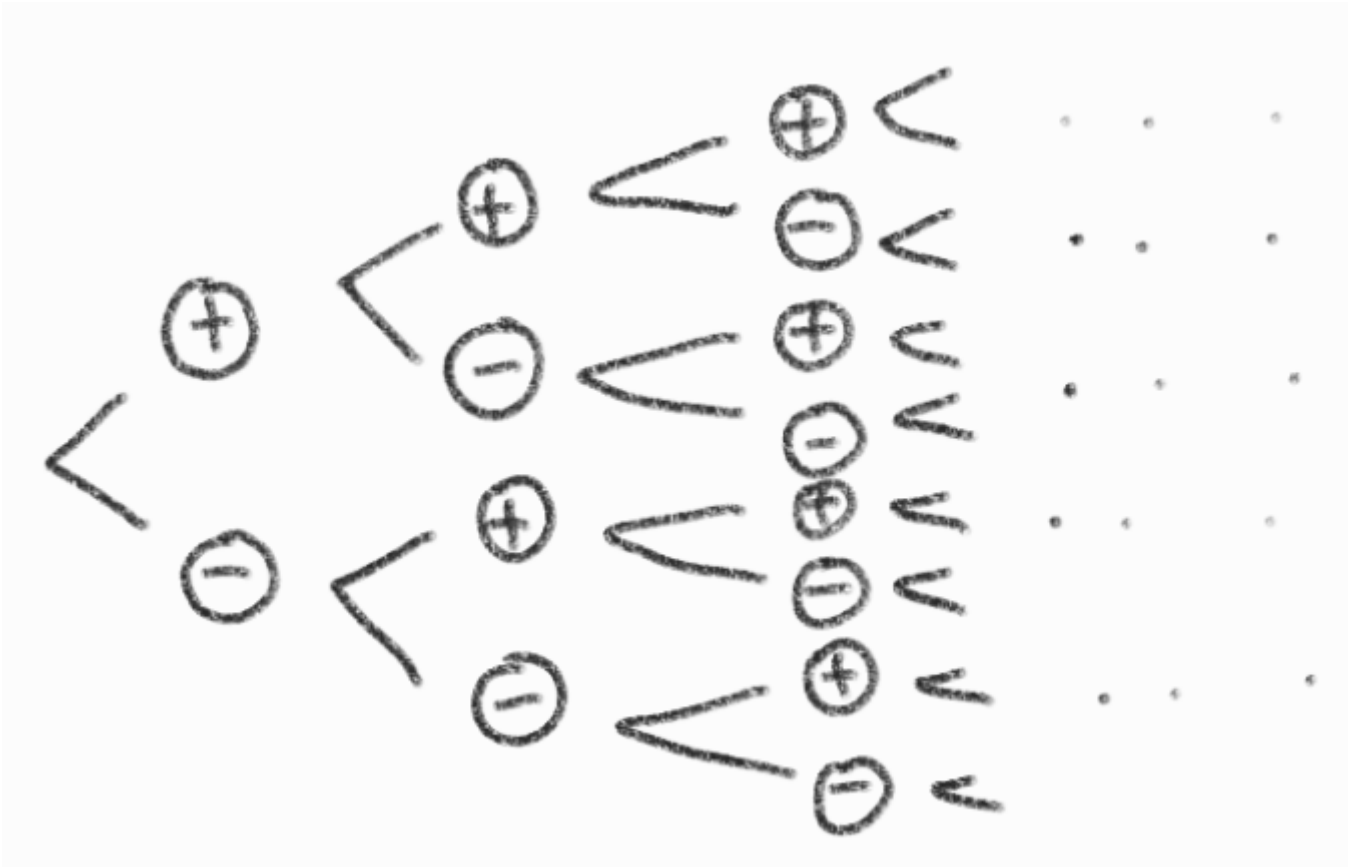
```
                totalTomato++; // 총 토마토의 갯수 세기
            }
        }
    }

    ans = bfs();

    // 모든 토마토가 익었는지 확인
    if (ans.first == totalTomato) {
        cout << ans.second << "\n"; // 모든 토마토가 익었다면 걸린 일수 출력
    } else {
        cout << -1 << "\n"; // 익지 못하는 경우
    }
    return 0;
}
```

> 잘 안풀려서 GPT의 도움을 받은 문제, 익으면서 1로 표시하므로 visited 배열이 필요가 없다!

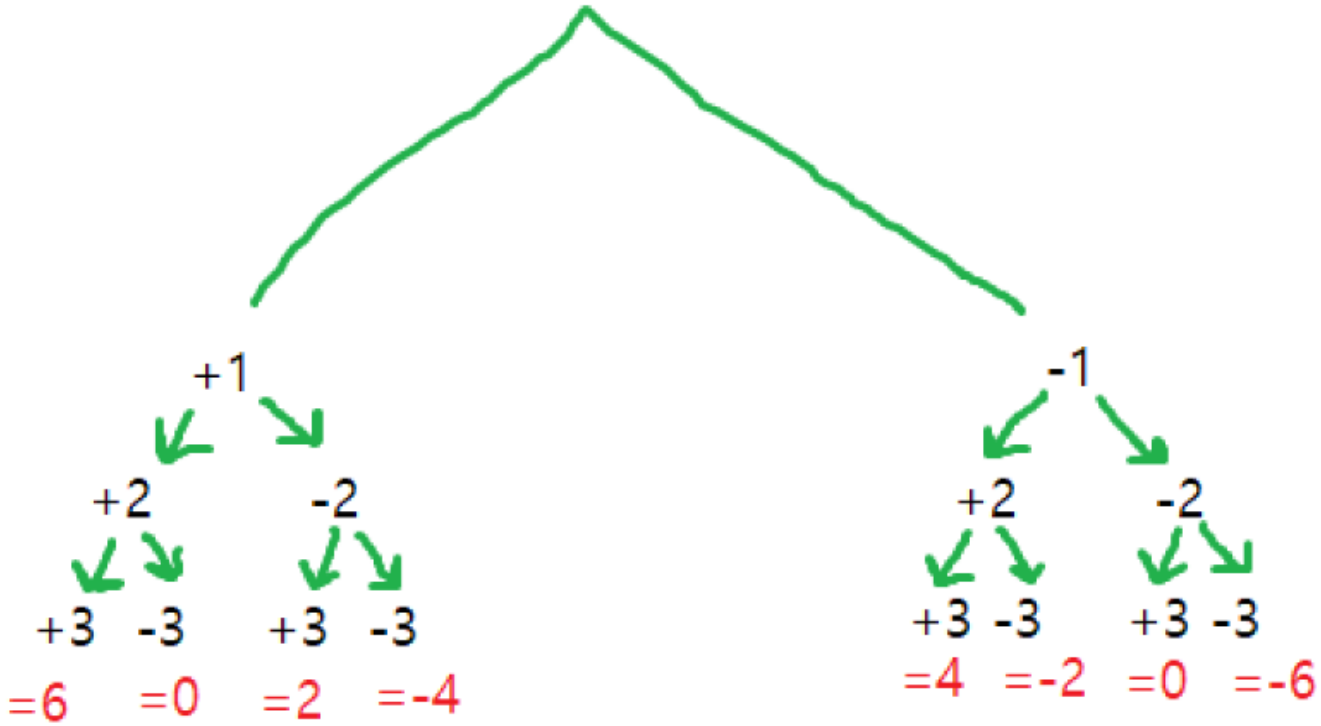
> BFS의 원칙에 따라, 한 번의 반복(day)에서 현재 레벨의 모든 노드를 처리하고 나서야 다음 레벨의 노드로 넘어갈 수 있다

> 이 말은 만약에 grid 탐색하면서 1인지점을 2개 발견했다고 가정하면 이 두 시작지점은 같은날에 익기 시작한다. 따라서 `q.size()`만큼 반복문을 도는 것이다!

타겟 넘버 - 프로그래머스



예로 numbers 에 1, 2, 3 가 들어있고, target으로 4를 들어보겠다.



명백한 트리구조, dfs/bfs를 사용할 수 있다!

```
#include <string>
#include <vector>

using namespace std;

int answer = 0;
```

```
int n;

void dfs(vector<int> &v, int idx, int sum, int &target){
    int tmp1 = sum + v[idx]; // + 조합
    int tmp2 = sum - v[idx]; // - 조합

    // 주어진 numbers에 끝에 도달 했으므로 idx가 끝을 가리킴 (종료조건)
    if((idx == n-1) && (target == tmp1 || target == tmp2)){
        answer++;
    }

    // 재귀조건
    else if(idx < n-1){
        dfs(v, idx+1, tmp1, target);
        dfs(v, idx+1, tmp2, target);
    }
}

int solution(vector<int> numbers, int target) {
    n = numbers.size();
    dfs(numbers, 0, 0, target);

    return answer;
}
```

> dfs이고 재귀 방법으로 푸는 문제, 재귀로 차근차근 접근하는 방식이 옳아보인다.

> 백트래킹을 아직 배우진 않았지만 백트래킹에 가까운 문제이지 않을까 한다.

게임 맵 최단거리 - 프로그래머스

```
#include <vector>
#include <queue>
#include <algorithm>
#include <iostream>

// 최단거리를 구하는 문제니까 BFS로 풀자
using namespace std;

int n, m;
int dirs[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}}; // 인접 4방향
queue<vector<int>> q;

bool isVisited(int x, int y, vector<vector<bool>> &visited){
    return visited[x][y];
}

bool inRange(int x, int y){
    return x >= 0 && x < n && y >= 0 && y < m;
}
```

```
bool canGo(int x, int y, vector<vector<int>> &maps, vector<vector<bool>> &visited)
{
    return inRange(x, y) && !visited[x][y] && maps[x][y] == 1;
}

int bfs(vector<int> root, vector<vector<int>> &maps, vector<vector<bool>>
&visited){
    q.push(root);
    visited[root[0]][root[1]] = true;

    while(!q.empty()){
        int x = q.front()[0];
        int y = q.front()[1];
        int dist = q.front()[2];
        q.pop();

        if(x == n-1 && y == m-1) return dist;

        for(int i=0; i < 4; i++){
            int dx = dirs[i][0], dy = dirs[i][1];
            if(canGo(x+dx, y+dy, maps, visited)){
                q.push({x+dx, y+dy, dist+1});
                visited[x+dx][y+dy] = true;
            }
        }
    }
    return -1;
}

int solution(vector<vector<int>> maps)
{
    n = maps.size();
    m = maps[0].size();
    vector<vector<bool>> visited(n, vector<bool>(m, false)); // vector로 2차원 배열
크기 초기화 하는 방법
    return bfs({0, 0, 1}, maps, visited);
}
```

> 최단 거리를 구현할땐 큐에 거리 정보까지 넣어줘야 한다는 사실을 명심할 것

> 종료조건을 `if(visited[n-1][m-1]) return dist;` 로 맨처음에 작성함

> 아 생각해보니까 n-1하고 m-1이 이때의 dist라는 보장이 없겠구나, 완전히 이해함

구현

숫자 짝꿍 - 프로그래머스

```
def solution(X, Y):
    X_list = list(X)
    Y_list = list(Y)
```

```
    answer = []
    for x in X_list:
        try:
            idx = Y_list.index(x)

            if idx != -1:
                answer.append(Y_list[idx])
                Y_list.pop(idx)

        except ValueError:
            continue

    if len(answer) == 0:
        return "-1"

    else:
        answer.sort(reverse=True)
        if answer[0] == '0':
            return '0'

        else:
            return ''.join(answer)
```

```
def solution(X, Y):
    # 0부터 9까지의 숫자 카운트를 저장할 리스트
    X_count = [0] * 10
    Y_count = [0] * 10

    # X와 Y에서 각 숫자의 등장 횟수를 카운트
    for x in X:
        X_count[int(x)] += 1
    for y in Y:
        Y_count[int(y)] += 1

    # 공통으로 등장하는 숫자들의 최소 횟수를 구해 결과에 추가
    answer = []
    for i in range(10):
        count = min(X_count[i], Y_count[i])
        answer.extend([str(i)] * count)

    # 정답이 없는 경우
    if len(answer) == 0:
        return "-1"

    # 내림차순 정렬
    answer.sort(reverse=True)

    # 정답이 '0'으로만 이루어진 경우
    if answer[0] == '0':
        return '0'
```

```
    return ''.join(answer)
```

> 첫번째 코드는 시간 초과 발생한 코드, 두번째 코드는 GPT가 제안한 코드

> X, Y의 길이가 최대 3,000,000 이므로 시간초과가 발생할 수 밖에 없다

> 1. find, index같은 함수를 사용하지 않고 히스토그램 형식으로 각 숫자가 등장한 횟수를 센다

> 2. X_count와 Y_count를 순회 하면서 둘 중 최솟값을 answer에 extend()함수를 이용한다

> 3. 내림차순 정렬하면 답이다.

둘만의 암호 - 프로그래머스

```python
def setRange(newChar):
    return chr(ord('a') + ((ord(newChar) - ord('a')) % 26))

def solution(s, skip, index):
    answer = ''
    skip = set(skip)  # 효율성을 위해 set으로 변환

    for c in s:
        cnt = 0
        newChar = c

        while cnt < index:
            newChar = setRange(chr(ord(newChar) + 1))  # 다음 문자로 이동

            if newChar not in skip:  # skip에 없으면 count 증가
                cnt += 1

        answer += newChar

    return answer
```

```cpp
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

char setRange(char c){
    return 'a' + ((c - 'a') % 26);
}

string solution(string s, string skip, int index) {
    string answer = "";
```

```
    for(auto c: s){
        int cnt = 0;
        char newC = c;

        while(cnt != index){
            newC = setRange(newC + 1);

            if(skip.find(newC) == string::npos){
                cnt++;
            }
        }
        answer += newC;
    }
    return answer;
}
```

> 알파벳의 범위를 설정하는 방법을 잘 알아 놓자

공원 산책 - 프로그래머스

```python
H = None
W = None
grid = []

def inRange(nx, ny):
    return 0 <= nx < H and 0 <= ny < W

def isnotWall(cx, cy, nx, ny):
    if cx == nx:  # 같은 행에서 이동
        for j in range(min(cy, ny), max(cy, ny) + 1):
            if grid[cx][j] == 'X':
                return False
    elif cy == ny:  # 같은 열에서 이동
        for i in range(min(cx, nx), max(cx, nx) + 1):
            if grid[i][cy] == 'X':
                return False
    return True

def canGo(cx, cy, nx, ny):
    return inRange(nx, ny) and isnotWall(cx, cy, nx, ny)

def solution(park, routes):
    global H, W, grid

    H = len(park)
    W = len(park[0])
    grid = [list(row) for row in park]  # park를 2D 리스트로 변환

    # 시작점 찾기
    for i in range(H):
```

```
            if 'S' in park[i]:
                sx, sy = i, park[i].index('S')
                break

    # 현재 로봇 강아지의 위치 (cx, cy)
    cx, cy = sx, sy

    # 명령에 따라 이동 시작
    for r in routes:
        direction, step = r.split(' ')
        step = int(step)

        if direction == "E" and canGo(cx, cy, cx, cy + step):
            cy += step
        elif direction == "S" and canGo(cx, cy, cx + step, cy):
            cx += step
        elif direction == "W" and canGo(cx, cy, cx, cy - step):
            cy -= step
        elif direction == "N" and canGo(cx, cy, cx - step, cy):
            cx -= step

    return [cx, cy]
```

> 문제 자체는 쉽다, 다만 이동중에 장애물이 있는지 확인해야 하므로 isnotWall 함수를 잘 설정해줘야 한다.

소수 찾기 - 프로그래머스

```
import math

def checkPrimeNumber(x):
    # 2부터 x의 제곱근까지의 숫자
    for i in range(2, int(math.sqrt(x) + 1)):
        if x % i == 0: # 나눠떨어지는 숫자가 있으면 소수가 아님
            return False

    return True

def solution(n):
    answer = 0

    for i in range(2, n+1):
        if checkPrimeNumber(i):
            answer += 1

    return answer
```

```cpp
#include <string>
#include <vector>
#include <cmath>

using namespace std;

bool checkPrimeNumber(int n){
    int root_n = static_cast<int>(sqrt(n));

    for(int i = 2; i <= root_n; i++){
        if(n % i == 0){
            return false;
        }
    }
    return true;
}

int solution(int n) {
    int answer = 0;
    for(int i = 2; i <= n; i++){
        if(checkPrimeNumber(i)) answer++;
    }
    return answer;
}
```

> 약수 찾기와 마찬가지로 소수를 찾을때도 숫자 N이 있을때 sqrt(N)까지만 탐색하면 된다.

> 약수 찾기와 소수 찾기 방법을 숙지해 놓을 것!

이상한 문자열 만들기 - 프로그래머스

```python
def solution(s):
    # 각 단어의 짝수번째 알파벳은 대문자로, 홀수번째 알파벳은 소문자로 바꾼 문자열
    answer = []
    s_list = s.split(" ")

    for i in s_list:
        tmp = ""
        for j in range(len(i)):
            if j == 0:
                tmp += i[j].upper()
            else:
                if j % 2 == 0:
                    tmp += i[j].upper()
                else:
                    tmp += i[j].lower()
        answer.append(tmp)

    answer = ' '.join(answer)
```

```
    return answer
```

```
#include <string>
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

string solution(string s) {
    vector<string> strings;
    int idx = 0;

    // 공백을 기준으로 나누기
    for (int i = 0; i < s.length(); i++) {
        if (s[i] == ' ') {
            strings.push_back(s.substr(idx, i - idx));  // Add word before space
            idx = i + 1;  // Move idx to character after space
        }
        if (i == s.length() - 1) {
            strings.push_back(s.substr(idx, i - idx + 1));  // Add last word
        }
    }

    // 짝수 인덱스는 대문자로, 홀수 인덱스는 소문자로 변환
    vector<string> memory;
    for (auto& word : strings) {
        string tmp = "";
        for (int j = 0; j < word.length(); j++) {  // Fix to word length
            if (j % 2 == 0) {
                tmp += toupper(word[j]);  // 짝수 인덱스는 대문자
            } else {
                tmp += tolower(word[j]);  // 홀수 인덱스는 소문자
            }
        }
        memory.push_back(tmp);
    }
    // Join the words back together with spaces
    string answer = "";
    for (int i = 0; i < memory.size(); i++) {
        answer += memory[i];
        if (i != memory.size() - 1) {
            answer += " ";  // Add space between words
        }
    }

    return answer;
}
```

> python은 string.upper(), string.lower()로 접근

> cpp은 toupper(string), tolower(string) 으로 접근

문자열 다루기 기본 - 프로그래머스

```python
def solution(s):
    if len(s) == 4 or len(s) == 6:
        for c in s:
            if not (c >= '0' and c <= '9'):
                return False

        return True

    else:
        return False
```

```cpp
#include <string>
#include <vector>

using namespace std;

bool solution(string s) {
    if(s.length() == 4 || s.length() == 6){
        for(auto c: s){
            if (!(c >= '0' && c <= '9')){
                return false;
            }
        }
        return true;
    }
    else{
        return false;
    }
}
```

기사단원의 무기 - 프로그래머스

약수를 어떻게 효율적으로 찾을 수 있는지 알 수 있었던 문제

```python
import math

def getDivisor(number):
    # N의 약수를 구할 때는, 1부터 N의 sqrt(N) 까지의 수만 0으로 나누어 떨어지는지 확인
하면 된다!
    if number == 1:
        return 1
```

```
    divisor = []
    for i in range(1, int(math.sqrt(number)) + 1):
        if number % i == 0:
            divisor.append(i)
            if i != number // i: # 똑같은 숫자가 들어가는 것을 방지하기 위한 조건
                divisor.append(number // i)
    return len(divisor)

def solution(number, limit, power):
    ironWeight = 0
    for i in range(1, number + 1):
        divisor_count = getDivisor(i)

        if divisor_count > limit: # 제한수치를 초과한 기사
            ironWeight += power
        else: # 제한 수치 이하인 기사
            ironWeight += divisor_count

    return ironWeight
```

```
#include <string>
#include <vector>
#include <cmath>
#include <iostream>

using namespace std;
int getDivisor(const int &number){
    if(number == 1){
        return 1;
    }

    else{
        vector<int> divisor;
        int rootN = static_cast<int>(sqrt(number));

        for(int i=1; i <= rootN; i++){
            if(number % i == 0){
                divisor.emplace_back(i);
                if(i != number / i){
                    divisor.emplace_back(number / i);
                }
            }
        }
        return divisor.size();
    }
}

int solution(int number, int limit, int power) {
    int answer = 0;
```

```
    for(int i = 1; i <= number; i++){
        int cnt = getDivisor(i);

        if(cnt > limit){
            answer += power;
        }
        else{
            answer += cnt;
        }
    }
    return answer;
}
```

> 숫자 N의 약수를 찾을 때, [1, sqrt(N)] 까지만 탐색하면 된다.

> N % i == 0의 조건을 만족하는 약수를 일단 구한다.

> 조건을 만족하는 i에 대해서 number / i를 수행하면 나머지 약수가 모두 구해진다.

랭킹전 대기열 - 백준

대기자가 있는 방을 어떻게 관리할 것이냐가 이 문제의 핵심이다

```
import sys

p, m = map(int, sys.stdin.readline().split())
rooms = []

for i in range(p):
    level, name = sys.stdin.readline().split()
    level = int(level)

    if i == 0:
        rooms.append([]) # 방 생성
        rooms[-1].append([level, name]) # 유저 추가

    else:
        flag = False

        for j in range(len(rooms)):
            if (level >= (rooms[j][0][0]-10) and level <= (rooms[j][0][0]+10)) and
len(rooms[j]) < m:
                    rooms[j].append([level, name])
                    flag = True
                    break

        if not flag: # 유저가 방에 추가되지 못함
            rooms.append([]) # 방 생성
            rooms[-1].append([level, name]) # 유저 추가
```

```
for r in range(len(rooms)):
    rooms[r].sort(key=lambda x : x[1])
    print("Started!" if len(rooms[r]) == m else "Waiting!")

    for level, name in rooms[r]:
        print(level, name)
```

> 가장 먼저 시도한 방식은 rooms 리스트를 만들때 `[[]] * (p//m)` 으로 진행

> `1<=m<=p` 라는 조건이 없으므로 일단 rooms list를 그냥 `[]`으로 두는게 맞다

> 오랜만에 flag를 사용해서 해결한 문제이다

어두운 굴다리 - 백준

```
import sys

n = int(sys.stdin.readline())
m = int(sys.stdin.readline())
lights = list(map(int, sys.stdin.readline().split()))

# 실패 케이스를 생각해보자
# 1. lights[0] - h > 0 --> 실패
# 2. lights[-1] + h < n --> 실패
# 3. lights[i]+h < light[i+1]-h --> 실패

for h in range(1, n+1):
    flag = True
    # 맨 왼쪽 가로등
    if lights[0] - h > 0:
        flag = False

    elif lights[-1] + h < n:
        flag = False

    else:
        for i in range(m-1):
            if lights[i] + h < lights[i+1] - h:
                flag = False
                break

    if flag:
        print(h)
        break
```

> 케이스 1, 2까지는 생각했다. 케이스 3를 생각못한 문제

> 이분탐색 기법으로 어떻게 푸는지 한번 참고해서 완벽히 이해하자!

크로스 컨트리 - 백준

전형적인 구현인데 dictionary도 sorted() 할 수 있다는 것을 알아야 하는 문제

```
import sys

t = int(sys.stdin.readline())

for _ in range(t):
    n = int(sys.stdin.readline())
    teams = list(map(int, sys.stdin.readline().split())) # 팀 번호를 나타내는 n개의 정수
    notQualified = []

    # 자격없는 팀 걸러내기
    for i in set(teams):
        if teams.count(i) < 6: # 6명 미만이므로 자격 X
            notQualified.append(i)

    # 차집합 이용, 자격있는 팀만 board 생성
    team_list = list(set(teams) - set(notQualified))
    board = {team:[0, 0, 0] for team in team_list} # 자격이 있는 팀 [총점, 카운트 변수, 5번째 선수의 점수]

    # 자격있는 팀만 board 업데이트
    score = 1
    for j in teams:
        if j not in notQualified:
            board[j][1] += 1 # 5번째 선수 카운트를 위한 카운팅 변수

            if board[j][1] <= 4:
                board[j][0] += score

            if board[j][1] == 5: # 5번째 선수라면
                board[j][2] = score

            score += 1

    board = sorted(board, key=lambda x:(board[x][0], board[x][2])) # 반환 값은 key를 기준으로 sorting 된 key값!
    print(board[0])
```

> 구현문제인데도 불구하고 꽤 많은 것을 배울 수 있었던 문제, 특히나 딕셔너리의 sorted() 이용을 알 수 있었다.

> 딕셔너리를 sorted(dict) 할 경우 딕셔너리의 value에 따라 sorting되는 것을 확인 할 수 있음, 반환 값은 key를 기준으로 sorting 된 key값!

> 항상 dictionary는 키 값을 이용해야만 접근 할 수 있다는 사실을 잊지말자

스위치 켜고 끄기 - 백준

```
import sys
import math

def change_state(x):
    return 0 if x == 1 else 1

switch_num = int(sys.stdin.readline())  # 스위치의 갯수 (positive integer <= 100)
switch = list(map(int, sys.stdin.readline().split()))  # 스위치 상태 (1 for on, 0
for off)

student_num = int(sys.stdin.readline())  # 학생 수 (positive integer <= 100)
earned_switch = [list(map(int, sys.stdin.readline().split())) for _ in
range(student_num)]

for s, c in earned_switch:
    if s == 1:  # 남자라면
        switch_index = switch_num // c
        switch_index = [coefficient * c for coefficient in range(1, switch_index +
1)]

        for si in switch_index:
            si -= 1
            switch[si] = change_state(switch[si])

    else:  # 여자라면
        c -= 1
        index = 0

        while (c-index >= 0 and c+index < len(switch) and switch[c+index] ==
switch[c-index]):
            switch[c + index] = change_state(switch[c + index])
            switch[c - index] = change_state(switch[c - index])
            index += 1

        switch[c] = change_state(switch[c]) # 인덱스가 0부터 시작하므로 자기 자신을
한번 더 바꿔야 함

for i in range(switch_num):
    print(switch[i], end=" ")
    if (i + 1) % 20 == 0: # 20개 이상 출력시 줄 바꿈
        print()
```

> 왜 정답률이 20% 언저리인지 알겠다. 출력이 거지 같다, 구현자체는 쉬웠음

등수 구하기 - 백준

```
import sys

# N >= 0, 리스트에 있는 점수의 갯수
# 10 <= P <= 50, 랭킹에 올라갈 수 있는 점수의 갯수
n, taesu, p = map(int, sys.stdin.readline().split())

scores = list(map(int, sys.stdin.readline().split()))
scores = scores[:]

ranking = 1

if n == 0:
    print(ranking)

else:
    if n >= p and min(scores) >= taesu:
        print(-1)

    else:
        for score in scores:
            if score > taesu:
                ranking += 1

        print(ranking)
```

> 입력이 많고, 조건이 까다로운 문제, 문제만 잘읽으면 구현 자체는 너무나 쉽다!

쿠키의 신체 측정 - 백준

```
import sys

n = int(sys.stdin.readline())
cookie = [list(sys.stdin.readline().strip()) for _ in range(n)] # 문자열이라서 통째로 입력받음, split 불가

def find_heart():
    heart_pos = [1, 1]
    for i in range(n):
        for j in range(n):
            if cookie[i][j].count("*") == 1: # cookie[i]가 하나의 문자열이므로 [j]까지 접근해야함
                heart_pos[0] = i + 1 + 1 # 머리는 심장 바로 윗 칸에 1칸 크기로 있다.
                heart_pos[1] = cookie[i].index("*") + 1 # 심장의 열 위치 찾음
                print(*heart_pos)
                heart_pos[0] -= 1
                heart_pos[1] -= 1

                return heart_pos

h_pos = find_heart()
```

```
left_arm = cookie[h_pos[0]][:h_pos[1]].count("*")
right_arm = cookie[h_pos[0]][h_pos[1]+1:].count("*")
waist, left_leg, right_leg = 0, 0, 0

for i in range(h_pos[0]+1, n):
    if cookie[i][h_pos[1]] == "*":
        waist += 1

    else:
        if cookie[i][h_pos[1]-1] == "*":
            left_leg += 1

        if cookie[i][h_pos[1]+1] == "*":
            right_leg += 1

print(left_arm, right_arm, waist, left_leg, right_leg)
```

> 전형적인 구현 문제면서 거지같은 문제였다. 입력이 문자열로 통째로 들어와서 split이 안먹힘, 그래서 cookie[i]가 문자열 하나가 통째로 들어와서 cookie[i][j]까지 해야 각 요소에 접근이 가능하다.

비밀번호 발음하기 - 백준

카운트 변수를 어떻게 적절하게 써야 하는지 알 수 있었던 문제, 특히 지금까지 푼 문제들 경험 상 연속적으로 같은 문자가 나오는지 확인하는 문제가 굉장히 많았다. 이 때 while문 돌릴 필요 없이 cnt 변수 적절히 초기화 +1 해주면 된다!

```
import sys

def condition(string):
    moeum = ['a', 'e', 'i', 'o', 'u']

    # 조건 1: 모음 하나를 반드시 포함하여야 한다.
    moeum_cnt = 0

    for m in moeum:
        if m in string:
            moeum_cnt += 1

    if moeum_cnt == 0:
        return f'<{string}> is not acceptable.'

    # 조건 2: 모음이 3개 혹은 자음이 3개 연속으로 오면 안 된다.
    moeum_cnt = 0
    jaeum_cnt = 0
    for s in string: # 조건 2
        if s in moeum:
            moeum_cnt += 1
            jaeum_cnt = 0

        else:
```

```
            jaeum_cnt += 1
            moeum_cnt = 0

        if moeum_cnt >= 3 or jaeum_cnt >= 3:
            return f'<{string}> is not acceptable.'

    # 조건 3: 같은 글자가 연속적으로 두번 오면 안되나, ee와 oo는 허용한다.
    sequence_cnt = 1
    sequence_init = string[0]

    for i in range(1, len(string)):
        if string[i] != sequence_init:
            sequence_init = string[i]
            sequence_cnt = 1

        else:
            sequence_cnt += 1

        if sequence_cnt == 2:
            if sequence_init == 'e' or sequence_init == 'o':
                sequence_cnt = 1
            else:
                return f'<{string}> is not acceptable.'

    # 조건 1, 2, 3 모두 안겹친 경우 acceptable
    return f'<{string}> is acceptable.'

while True:
    string = sys.stdin.readline().strip()

    if string == "end":
        break

    print(condition(string))
```

덩치 - 백준

N이 최대 50 밖에 안되므로 브루트포스임을 눈치채야 한다.

```
import sys

n = int(sys.stdin.readline())
data = [list(map(int, sys.stdin.readline().split()))for _ in range(n)]

for i in range(len(data)):
    ranking = 1
    for j in range(len(data)):
        if data[i][0] < data[j][0] and data[i][1] < data[j][1]:
            ranking += 1
    print(ranking, end=' ')
```

> ranking을 1로 초기화 하고 자기 자신보다 덩치가 큰 사람이 있을 때만 ranking에 1을 더해주는 방식으로 구현!

올림픽 - 백준

```
import sys

n, k = map(int, sys.stdin.readline().split())
rank = {}

for _ in range(n):
    c_id, gold, silver, bronze = map(int, sys.stdin.readline().split())
    rank[c_id] = [gold, silver, bronze, 0]


for key1, medal1 in rank.items():
    ranking = 1
    for key2, medal2 in rank.items():
        if key1 != key2: # 이중 반복문 돌면서 자기 자신을 검사할 필요 없음

            if medal1[0] < medal2[0]: # 조건 1: 금메달 수가 더 많은 나라
                ranking += 1

            elif medal1[0] == medal2[0]: # 조건 2: 금메달 수가 같으면, 은메달 수가 더 많은 나라
                if medal1[1] < medal2[1]:
                    ranking += 1

            elif (medal1[0] == medal2[0]) and (medal1[1] == medal2[1]): # 조건 3: 금, 은메달 수가 모두 같으면, 동메달 수가 더 많은 나라
                if medal1[2] < medal2[2]:
                    ranking += 1

    rank[key1][-1] = ranking # 랭킹 정보 업데이트

print(rank[k][-1])
```

> 정렬 전혀 안쓰고 이중 반복문 돌리면서 해결한 코드, N값이 작기 때문에 브루트포스 방식으로 해결해도 무방한데 N값이 커지면 시간초과 발생함, 다른 사람이 해결한 방식을 살펴보자

```
import sys

n, k = map(int, sys.stdin.readline().split())
medals = [list(map(int, sys.stdin.readline().split())) for _ in range(n)]
medals.sort(key=lambda x: (x[1], x[2], x[3]), reverse=True) # 내림차순 정렬, 이미 정렬된 상태로 등수 탐색 시작 (동일한 우선순위를 가진다면 입력이 먼저온 순서대로 정렬됨)

for i in range(n):
```

```
    if medals[i][0] == k: # 등수를 알고 싶은 국가 번호 인덱스 찾기
        idx = i #

for i in range(n):
    if medals[idx][1:] == medals[i][1:]: # for문을 돌며 국가 k의 금, 은, 동메달 수와 같은 국가가 나오면 그 국가의 (index +1) 등이므로 출력하고 break
        print(i+1)
        break
```

> python의 sort 함수에서 key까지는 알았는데 reverse에 대한 사용법도 알게 되었다. 특히 정렬 후에 동일한 우선순위를 가진다면 입력이 먼저온 순서대로 정렬된다는 것을 명심하자

> medals[idx][1:] == medals[i][1:] 이 코드가 좀 의심이 가서 GPT에게 물어보았다. 리스트의 주소값을 비교한다고 생각했는데 각 원소가 완전히 동일한지 비교 가능하다는 것이다! 앞으로 자주 써먹자

ZOAC4 - 백준

이 문제에서 얻은 것은 문제를 깊게 고민하고 그다음에 풀이를 시작하자, 시간 급하다고 마구잡이로 하면 시간 초과 뜬다!

```
import sys
import math

H, W, N, M = map(int, sys.stdin.readline().split())

person_in_col = math.ceil(H / (N+1)) # 1열에 앉을 수 있는 사람의 수
person_in_row = math.ceil(W / (M+1)) # 1행에 앉을 수 있는 사람의 수

print(person_in_col * person_in_row)
```

> 1행에 앉을 수 있는 사람수를 구하고, 1열에 앉을 수 있는 사람 수를 구해서 곱하면 전체 앉을 수 있는 사람의 수 이다. 따라서 이중 반복문 돌릴 필요없이 규칙만 찾으면 해결되는 문제

삼각형과 세변 - 백준

총 4개의 조건을 판별해야 한다. 따라서 큰 대분류로 먼저 묶고 난 다음에 세부 분류로 하는 것이 좋은 코드 작성 방법이다!

```
import sys

while True:
    edges = sorted(list(map(int, sys.stdin.readline().split())))

    if edges[0] == 0:
        break

    # 가장 먼저 삼각형을 만들 수 있느냐 없느냐 부터 판단하고 들어가자
```

```
    if edges[0] + edges[1] <= edges[2]: # 삼각형을 만들 수 없는 경우
        print("Invalid")

    else: # 삼각형을 만들 수 있는 경우
        if edges[0] == edges[1] and edges[1] == edges[2]:
            print("Equilateral")

        elif edges[0] != edges[1] and edges[1] != edges[2] and edges[0] !=
edges[2]:
            print("Scalene")

        else:
            print("Isosceles")
```

단어공부 - 백준

```
import sys

string = sys.stdin.readline().strip().upper()
string_set = list(set(string))
char_cnt = [string.count(c) for c in string_set]

maximum = max(char_cnt)

if char_cnt.count(maximum) != 1: # maximum이 여러개 존재
    print("?")

else: # maximum이 1개만 존재
    max_idx = char_cnt.index(maximum)
    print(string_set[max_idx])
```

> for문 안쓸려고 내장함수 index로 접근했는데 실행시간은 똑같다, 어차피 최댓값의 index찾을려면 순회해야하니까 똑같은 듯

디지털 티비 - 백준

```
import sys
n = int(sys.stdin.readline())
channel = [sys.stdin.readline().rstrip() for _ in range(n)]
command = []

idx1 = channel.index("KBS1")
command.append("1" * idx1)
command.append("4" * idx1)

idx2 = channel.index("KBS2")
```

```
if idx1 > idx2:
    idx2 += 1

command.append("1" * idx2)
command.append("4" * (idx2-1))

print(''.join(command))
```

> 입력하나에 여러개의 해가 존재할 수 있는 문제이다. 따라서 1번, 4번 버튼만 이용하도록 제한하고 푸는 문제, 제약 조건에 방법의 길이는 500보다 작아야한다.

> 채널이 최대 100개면, KBS1, KBS2가 리스트에 마지막에 연속적으로 있다고 해도 400언저리이다. 따라서 1번, 4번만 이용

집합 - 백준

내장 함수인 set()을 사용하지 않고 list만을 이용해서 구현하는 문제, list의 clear()함수에 대해 알게 되었다.

```
import sys

m = int(sys.stdin.readline())
s = []
series = [i for i in range(1, 21)]

for _ in range(m):
    command = sys.stdin.readline().split()

    if command[0] == "add":
        if int(command[1]) not in s:
            s.append(int(command[1]))

    elif command[0] == "remove":
        if int(command[1]) in s:
            s.remove(int(command[1]))

    elif command[0] == "check":
        if int(command[1]) in s:
            print(1)
        else:
            print(0)

    elif command[0] == "toggle":
        if int(command[1]) in s:
            s.remove(int(command[1]))
        else:
            s.append(int(command[1]))

    elif command[0] == "all":
        s[:] = series
```

```
    else:
        s.clear()
```

> 맨처음에 toggle 발생할 때 마다 for문 돌리는 방식으로 작성했는데, 매우 비효율적이라고 판단되어 미리 [1, ..., 20] 까지 리스트를 생성하고 이를 복사하는 방식으로 변경했다.

부족한 금액 계산하기 - 프로그래머스

```python
def solution(price, money, count):
    answer = 0
    cnt = 1

    while(cnt <= count):
        answer += (price * cnt)
        cnt += 1

    if answer > money:
        return answer - money

    else:
        return 0
```

```cpp
using namespace std;

long long solution(int price, int money, int count)
{
    long long answer = 0;

    int cnt = 1;

    while(cnt <= count){
        long long newPrice = price * cnt;
        answer += newPrice;
        cnt++;
    }

    if(answer > static_cast<long long>(money)){
        return answer - static_cast<long long>(money);
    }

    else{
        return 0;
    }
}
```

실패율 - 프로그래머스

```python
def solution(N, stages):
    # 스테이지별 [통과 못한 사람, 시도한 사람] 기록
    board = [[0, 0] for _ in range(N+1)]  # 인덱스가 1부터 N까지 매칭되도록 설정

    for s in stages:
        if s <= N:  # 스테이지에 남아있는 사람 카운트
            board[s][0] += 1

        for i in range(1, min(s+1, N+1)):  # 현재 스테이지까지 도달한 사람 카운트
            board[i][1] += 1

    # 실패율을 계산하고 스테이지 번호와 함께 저장
    result = sorted(range(1, N+1), key=lambda x: (board[x][0] / board[x][1] if
board[x][1] != 0 else 0, -x), reverse=True)

    return result
```

> 맨처음에 board를 dictionary 형태로 만들었다. 그렇게 하다보니까 시간초과 발생

> 딕셔너리를 쓰지 않고 list형태로 만든다음에 인덱스가 1부터 N까지 매칭되도록 설정하도록 한다

콜라문제 - 프로그래머스

```python
def solution(a, b, n):
    answer = 0
    while n >= a:
        coke = (n // a) * b  # 얻을 수 있는 콜라의 갯수
        answer += coke
        n = n % a + coke  # 남은 빈 병에 새로 받은 콜라의 빈 병을 더해줌
    return answer
```

```cpp
#include <string>
#include <vector>

using namespace std;

int solution(int a, int b, int n) {
    int answer = 0;

    while(n >= a){
        int coke = (n / a) * b; // 새로 받은 콜라
        answer += coke;
        n = (n % a) + coke;
    }

    return answer;
}
```

> 규칙을 찾으면 된다, 풀이는 따로 필요 없음

비밀지도 - 프로그래머스

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

using namespace std;

pair<string, string> convBin(int num1, int num2, int n) { // 이진법으로 바꾸는 함수
    string s1 = "";
    string s2 = "";

    while (num1 != 0) {
        s1 = to_string(num1 % 2) + s1; // 더하기 순서 중요!!
        num1 /= 2;
    }

    while (num2 != 0) {
        s2 = to_string(num2 % 2) + s2; // 더하기 순서 중요!!
        num2 /= 2;
    }

    while (s1.length() < n) { // zero padding
        s1 = "0" + s1;
    }

    while (s2.length() < n) { // zero padding
        s2 = "0" + s2;
    }
    return {s1, s2};
}

vector<string> solution(int n, vector<int> arr1, vector<int> arr2) {
    vector<string> grid(n); // 지도를 만들 공간 생성

    for(int i = 0; i < n; i++){
        int a = arr1[i];
        int b = arr2[i];
        pair<string, string>res = convBin(a, b, n);

        string row = ""; // grid의 한 행을 나타낼 row
        for(int j = 0; j < n; j++){
            if (res.first[j] == '1' || res.second[j] == '1'){
                row += '#';
            }
            else row += ' ';
        }
        grid[i] = row;
    }
```

```
    return grid;
}
```

```
def convBin(num1, num2, n):
    s1 =""
    s2 = ""

    while(num1 != 0):
        s1 = str(num1 % 2) + s1
        num1 //= 2  # int형으로 나누기 해야한다

    while(num2 != 0):
        s2 = str(num2 % 2) + s2
        num2 //= 2 # int형으로 나누기 해야한다

    # zero padding
    s1 = "0" * (n - len(s1)) + s1
    s2 = "0" * (n - len(s2)) + s2

    return (s1, s2)

def solution(n, arr1, arr2):
    answer = []
    for i in range(n):
        a, b = arr1[i], arr2[i]
        sa, sb = convBin(a, b, n)
        tmp = ""
        for j in range(n):
            if sa[j] == '1' or sb[j] == '1':
                tmp += "#"
            else:
                tmp += " "
        answer.append(tmp)

    return answer
```

> 문제 자체는 쉬운데 구현이 생각보다 오래 걸리는 문제다. 특히나 이진법 변환이나 zero padding이 좀 까다로운듯 하다

제일 작은 수 제거하기 - 프로그래머스

```
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

vector<int> solution(vector<int> arr) {
```

```
    vector<int> answer;
    if(arr.size() == 1){
        return {-1};
    }
    else{
        int minNum = *min_element(arr.begin(), arr.end()); // min_element의 리턴
값은 iterator
        for(const auto &a : arr){
            if(a != minNum){
                answer.emplace_back(a);
            }
        }
    }
    return answer;
}
```

```
def solution(arr):
    if(len(arr) == 1):
        return [-1]
    answer = []
    minNum = min(arr)
    for a in arr:
        if a != minNum:
            answer.append(a)
    return answer
```

> 뭐 너무 쉬워서 풀이가 없다. answer를 쓰지 않고 리턴하는 방법은 없나 찾아보면 좋긴 할 듯

최댓값 만들기 - 프로그래머스

```
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

int solution(vector<int> numbers) {
    sort(numbers.begin(), numbers.end()); // 오름차순 정렬

    if(numbers[0] < 0 && numbers[1] < 0){
        int tmpMax1 = numbers[0] * numbers[1];
        int tmpMax2 = numbers[numbers.size()-2] * numbers[numbers.size()-1];

        if(tmpMax1 > tmpMax2) return tmpMax1;
        else return tmpMax2;
    }
```

```
    return numbers[numbers.size()-2] * numbers[numbers.size()-1];
}
```

```
def solution(numbers):
    answer = 0
    numbers.sort()

    if numbers[0] < 0 and numbers[1] < 0:
        tmpMax1 = numbers[0] * numbers[1]
        tmpMax2 = numbers[-1] * numbers[-2]
        if tmpMax1 > tmpMax2:
            return tmpMax1

    return numbers[-1] * numbers[-2]
```

> 정렬만 잘해놓으면 쉽게 풀 수 있는 문제, 만약 앞에 처음 두 값이 -면 양수가 되므로 뒤에 두값이랑 비교해서 max값을 리턴한다

> 아니라면 맨 뒤에 있는 두 값을 곱해서 리턴하면 된다

숫자 문자열과 영단어 - 프로그래머스

```
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

int solution(string s) {
    string answer = "", tmp = "";
    vector<string> num = {"zero", "one", "two", "three", "four", "five", "six",
"seven", "eight", "nine"};

    for(int i = 0; i < s.length(); i++) {
        if('0' <= s[i] && s[i] <= '9') answer += s[i];
        else tmp += s[i];

        if(find(num.begin(), num.end(), tmp) != num.end()) {
            answer += to_string(find(num.begin(), num.end(), tmp) - num.begin());
// int를 문자열로
            tmp = ""; // tmp를 다시 빈문자열로 바꿈
        }
    }
    return stoi(answer);
}
```

```
def convert(s):
    if s == "zero":
        return '0'
    elif s == "one":
        return '1'
    elif s == "two":
        return '2'
    elif s == "three":
        return '3'
    elif s == "four":
        return '4'
    elif s == "five":
        return '5'
    elif s == "six":
        return '6'
    elif s == "seven":
        return '7'
    elif s == "eight":
        return '8'
    elif s == "nine":
        return '9'

def solution(s):
    answer = ""
    tmp = ""
    numbers = ['zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven',
'eight', 'nine']

    for c in s:
        if c >= '0' and c <= '9':
            answer += c
        else:
            tmp += c
            if tmp in numbers:
                answer += convert(tmp)
                tmp = ""

    return int(answer)
```

> 생각보다 꽤 헤맨 문제이다. tmp 문자열을 활용하면 간단하게 풀 수 있다

크기가 작은 부분문자열 - 프로그래머스

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

using namespace std;

int solution(string t, string p) {
```

```
    int answer = 0;

    if (p.length() == t.length()){
        if(stol(t) <= stol(p)){ // p와 t의 길이가 최대 10000이므로 long long으로 변
경
            return 1;
        }
        return 0;
    }

    for(int i = 0; i <= t.length() - p.length(); i++){
        string seg = t.substr(i, p.length());
        if(stol(seg)<= stol(p)){
            answer++;
        }
    }
    return answer;
}
```

```
def solution(t, p):
    answer = 0
    for i in range(0, len(t)-len(p)+1):
        seg = t[i:i+len(p)]
        if(int(seg) <= int(p)):
            answer += 1
    return answer
```

> c++의 경우 t의 최대 길이가 10000이므로 int 자료형 넘어갈수 도 있다. 따라서 long long으로 변환해야 함

시저 암호 - 프로그래머스

```
#include <string>
#include <vector>


using namespace std;

string solution(string s, int n) {
    string answer = "";

    for(char c: s){
        if (c == ' '){
            answer += ' ';
        }

        else{
            char newC;
```

```
            if(c >= 'a' && c <= 'z'){ // 소문자인 경우
                newC = ((c - 'a' + n) % 26) + 'a';
            }

            else{ // 대문자인 경우
                newC = ((c - 'A' +  n) % 26) + 'A';
            }
            answer += newC;
        }
    }
    return answer;
}
```

> C++에서는 유독 아스키 코드를 다룰일이 많은 거 같다, alphabet의 갯수가 26임을 미리 숙지하고 있자

크레인 인형 뽑기 - 프로그래머스

```
#include <string>
#include <vector>
#include <stack>

using namespace std;

int solution(vector<vector<int>> board, vector<int> moves) {
    int answer = 0;

    stack<int> reward;
    int explode = 0;

    for(int j = 0; j < moves.size(); j++){
        for(int i = 0; i < board.size(); i++){
            if(board[i][moves[j]-1] != 0){
                if(!reward.empty() && reward.top() == board[i][moves[j]-1]){ // 스
택의 탑이랑 같은지 확인
                    reward.pop(); // 같은 모양 폭발
                    explode += 2;
                }
                else{ // 스택이 비어있고 top이랑 같지 않다면 바로 스택에 추가
                    reward.push(board[i][moves[j]-1]);
                }

                board[i][moves[j]-1] = 0; // 비어있다는 표시로 0으로 바꿈
                break; // 인형을 하나 꺼냈으면 그 열에서 더 이상 찾지 않음
            }
        }
    }
    return explode;
}
```

> 구현과 스택이 섞여 있는 문제, 꽤 괜찮은 문제라고 생각한다

정수 제곱근 판별 - 프로그래머스

```
#include <cmath>
#include <iostream>

using namespace std;

long long solution(long long n) {
    long long answer = 0;
    long long a = sqrt(n);

    if (a*a == n) { // 양의 정수x의 제곱인 경우
        answer= (a+1) * (a+1);
    }
    else {  // 아닌 경우
        answer=-1;
    }
    return answer;
}
```

> 풀이 할만한게 딱히 없다. sqrt 했을때 3같은 경우라면 1.xxxx라서 a에 1이 들어갈 것이다.

3진법 뒤집기 - 프로그래머스

```
#include <string>
#include <vector>
#include <cmath>
#include <iostream>

using namespace std;

string convert2Triple(int n){
    string s = "";

    // n이 0보다 클 때까지 3으로 나누면서 나머지를 문자열에 추가
    while(n > 0){
        s += to_string(n % 3); // 나머지를 문자열에 추가
        n /= 3;  // n을 3으로 나눔
    }
    return s;  // 3진법으로 변환된 문자열 (역순으로 저장됨)
}

int solution(int n) {
    string s = convert2Triple(n);  // 3진법으로 변환된 문자열
    int power = s.length()-1;  // 역순으로 처리되기 때문에 power는 처음에 0으로 시작
    int answer = 0;
```

```
    for(char c: s){
        answer += (c - '0') * pow(3, power);  // 3의 제곱을 계산하여 더함
        power--;
    }

    return answer;
}
```

> n진법에 대한 개념을 가지고 있어야 풀 수 있는 문제, 문제 자체는 쉬운것 같다.

공 던지기 - 프로그래머스

```
#include <string>
#include <vector>

using namespace std;

int solution(vector<int> numbers, int k) {
    int answer = 0;

    if(k == 1){
        return numbers[0];
    }

    else{
        int idx = 0, cnt = 0;
        int target;
        while(cnt != k){
            target = numbers[idx % numbers.size()];
            idx += 2;
            cnt++;
        }
        answer = target;
    }
    return answer;
}
```

> idx가 2씩 증가한다, idx 사이즈가 벡터의 크기를 넘어갈 수 있으므로 size()의 나머지 연산자로 segmentation fault가 일어나지 않도록 한다.

로또의 최고순위와 최저순위 - 프로그래머스

```
#include <string>
#include <vector>
#include <iostream>
#include <algorithm>
```

```
using namespace std;

int convert2Rank(const int &cnt){
    switch(cnt){
        case 6:
            return 1;  // 6개 맞춘 경우 1등
        case 5:
            return 2;  // 5개 맞춘 경우 2등
        case 4:
            return 3;  // 4개 맞춘 경우 3등
        case 3:
            return 4;  // 3개 맞춘 경우 4등
        case 2:
            return 5;  // 2개 맞춘 경우 5등
        default:
            return 6;  // 그 외는 6등
    }
}

vector<int> solution(vector<int> lottos, vector<int> win_nums) {
    int matched = 0, zeroCount = 0;

    // 당첨 번호를 쉽게 찾기 위한 정렬
    sort(win_nums.begin(), win_nums.end());

    // 로또 번호와 당첨 번호 비교
    for (int num : lottos) {
        if (num == 0) {
            zeroCount++;  // 0의 개수를 카운트
        } else if (find(win_nums.begin(), win_nums.end(), num) != win_nums.end())
{
            matched++;  // 일치하는 번호가 있으면 카운트
        }
    }

    // 최고 순위: 0의 개수만큼 추가로 맞힐 수 있음
    int maxRank = convert2Rank(matched + zeroCount);
    // 최저 순위: 0을 모두 틀린 것으로 가정
    int minRank = convert2Rank(matched);

    return {maxRank, minRank};
}
```

> find()사용하면 되는구나 간단하네, zeroCount를 안해서 틀린문제 완전히

다트 게임 - 프로그래머스

```
#include <string>
#include <cmath>
#include <iostream>
```

```
using namespace std;

int solution(string dartResult) {
    int answer = 0;
    int prev = 0, score = 0;

    for(int i = 0; i < dartResult.size(); i++)
    {
        if(dartResult[i] >= '0' && dartResult[i] <= '9') // 숫자를 처리하는 부분
        {
            prev = score;

            if(dartResult[i + 1] == '0')
            {
                score = 10;
                i++; // for문 indexing 하나 건너뛰기
            }
            else
                score = dartResult[i] - '0';
        }

        else if(dartResult[i] == 'S' || dartResult[i] == 'D' || dartResult[i] ==
'T') // 점수를 처리하는 부분
        {
            if(dartResult[i] == 'D') {
                score = pow(score, 2);
            }

            else if(dartResult[i] == 'T') {
                score = pow(score, 3);
            }

            if(dartResult[i + 1] == '*') {
                answer -= prev;
                prev *= 2;
                score *= 2;
                i++;
                answer += prev;
            }

            else if(dartResult[i + 1] == '#'){
                score *= -1;
                i++;
            }

            answer += score;
        }
    }

    return answer;
}
```

> 전형적인 개 무식한 구현 문제, for문을 돌면서 i++를 하면 인덱싱 하나를 건너뛸 수 있다!

> 문자열을 하나하나 받아올 때 char 자료형이 들어간다. atoi 이런거 안먹히니까 `char c -'0'` 을 해주자

콜라츠 추측 - 프로그래머스

```
#include <string>
#include <vector>

using namespace std;

int solution(int num) {
    int iteration = 0;

    long long cNum = static_cast<long long>(num); // int 자료형 범위를 넘어갈 수 있으므로 long long으로 변환

    while(iteration < 500){
        if(cNum == 1){
            return iteration;
        }

        else{
            if(cNum % 2 == 0){
                cNum /= 2;
            } else{
                cNum = (cNum * 3) + 1;
            }
            iteration++;
        }
    }
    return -1;
}
```

> int 자료형 범위가 넘어갈 수 있으므로 long long으로 변환해야 함!

합성수 찾기 - 프로그래머스

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

using namespace std;

int solution(int n) {
    int fusionNum = 0;

    for(int i = 1; i <= n; i++){
```

```
            int cnt = 0;
            for(int j = 2; j < i; j++){ // 자기자신과 1은 제외
                if(i % j == 0) cnt++;

                if(cnt >= 1){
                    fusionNum++; // 자기자신과 1은 제외하므로 cnt가 1이상이면 됨
                    break;
                }
            }
        }

        return fusionNum;
    }
```

> 그냥 단순하게 약수 구하는 문제

최대공약수와 최소공배수 - 프로그래머스

```
#include <string>
#include <vector>

using namespace std;

vector<int> solution(int n, int m) {
    vector<int> answer;

    if(m % n == 0){
        answer.push_back(n);
        answer.push_back(m);
    }
    else{
        int maxNum = 1;
        for(int i=1; i < n; i++){
            if(n % i == 0 && m % i == 0){
                maxNum = max(maxNum, i);
            }
        }
        answer.push_back(maxNum);
        int n1 = n / maxNum;
        int n2 = m / maxNum;
        answer.push_back(maxNum * n1 * n2);
    }

    return answer;
}
```

> 테스트 케이스는 `[3, 12]`, `[2, 5]`와 같은 것만 주어졌는데 `[6, 27]` 같은 반례가 있다는 것을 고려해야 함

가장 가까운 글자 - 프로그래머스

```
#include <string>
#include <vector>
#include <algorithm>
#include <unordered_map>

using namespace std;

vector<int> solution(string s) {
    vector<int> answer;

    unordered_map<char, int> um; // 문자와 인덱스

    for(int i=0; i < s.length(); i++){
        auto iter = um.find(s[i]);
        if(iter == um.end()){
            um[s[i]] = i;
            answer.push_back(-1);
        }

        else{
            answer.push_back(i - um[s[i]]);
            um[s[i]] = i;
        }
    }
    return answer;
}
```

> 가장 가까운 글자를 구해야 하므로 um을 순회하면서 해당 글자가 발견되면 index를 업데이트 시켜준다!

팰린드롬 - 프로그래머스

```
#include <string>
#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;

int solution(int n, int m) {
    int answer = 0;

    for(int i=n; i <=m; i++){
        string s = to_string(i);

        if(s.length() == 1){
            answer++;
            continue;
```

```
        }

        int idx1 = 0;
        int idx2 = s.length()-1;

        while(idx1 < idx2){
            if(s[idx1] == s[idx2]){
                idx1++;
                idx2--;
            }
            else break;
        }

        if(idx1 >= idx2){
            answer++;
        }
    }
    return answer;
}
```

> 맨처음에 스택으로 시작할까 생각했음. string에 존재하는 문자를 입력받아서 스택에 집어넣기 위해 string 길이 만큼 반복문을 돌고, 다시 pop하기 위해 string 길이만큼 반복문을 돌고 s와 비교? 무조건 시간초과임

> 투포인터다! 앞 인덱스와 뒤 인덱스를 설정하고 앞 인덱스가 뒤 인덱스보다 커질때까지 while문을 돌면서 비교하면 된다!!

ZOAC 4 - 백준

이 문제에서 얻은 것은 문제를 깊게 고민하고 그다음에 풀이를 시작하자, 시간 급하다고 마구잡이로 하면 시간초과 뜬다!

```
#include <iostream>
#include <cmath>

using namespace std;

int main(int argc, char** argv) {
    double H, W, N, M;

    cin >> H >> W >> N >> M;

    double row = ceil(H / (N + 1.0));
    double col = ceil(W / (M + 1.0));

    int cnt = static_cast<int>(row * col);

    cout << cnt << "\n";

    return 0;
}
```

> 1행에 앉을 수 있는 사람수를 구하고, 1열에 앉을 수 있는 사람 수를 구해서 곱하면 전체 앉을 수 있는 사람의 수 이다. 따라서 이중 반복문 돌릴 필요없이 규칙만 찾으면 해결되는 문제

삼각형과 세변 - 백준

총 4개의 조건을 판별해야 한다. 따라서 큰 대분류로 먼저 묶고 난 다음에 세부 분류로 하는 것이 좋은 코드 작성 방법이다!

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

using namespace std;

int main(int argc, char** argv){
    while (true) {
        int n;
        vector<int> vec;

        for(int i = 0; i < 3; i++){
            cin >> n;
            vec.push_back(n);
        }

        sort(vec.begin(), vec.end());

        if(vec[2] == 0 && vec[1] == 0 && vec[0] == 0){
            break;
        }

        if (vec[2] >= vec[1] + vec[0]) {
            cout << "Invalid\n";
        }
        else {
            if (vec[0] == vec[1] && vec[1] == vec[2]) {
                cout << "Equilateral\n";
            }
            else if (vec[0] == vec[1] || vec[1] == vec[2] || vec[0] == vec[2]) {
                cout << "Isosceles\n";
            }
            else {
                cout << "Scalene\n";
            }
        }
    }
    return 0;
}
```

> sort 알고리즘 사용하려면 `#include <algorithm>` 필수임

단어공부 - 백준

```cpp
#include <iostream>
#include <algorithm>
#include <string>

using namespace std;

int main(int argc, char** argv) {
    string str;
    int arr[26] = {0}; // 알바벳의 갯수

    cin >> str;

    for (auto i = 0; i < str.size(); i++) {
        str[i] = toupper(str[i]); // 대문자로 변환
        arr[str[i] - 'A']++; // A의 아스키 코드 65, a의 아스키 코드 97
    }

    int max = 0; // 가장 많이 나온 문자 횟수
    int index = -1; // 해당 문자의 인덱스
    int max_count = 0; // 최대가 몇번 나왔는지 확인

    for (auto i = 0; i < 26; i++) {
        if (arr[i] > max) {
            max = arr[i];
            index = i;
            max_count = 1;
        } else if (arr[i] == max) {
            max_count++;
        }
    }

    if (max_count > 1) {
        cout << "?\n";
    } else {
        cout << static_cast<char>(index + 'A') << '\n';
    }

    return 0;
}
```

> 감이 안잡혀서 조금 참고한 문제, 알파벳의 갯수는 26개이다.

> `toupper`함수는 대문자로 변환하는 함수이다.

> `arr[str[i] - 'A']++;` 이문제의 핵심

디지털 티비 - 백준

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

// 1번과 4번만 이용해서 풀이

using namespace std;

int main(int argc, char** argv){
    int n;

    vector<string> channels;
    vector<char> commands;

    cin >> n;

    for(int i=0; i < n; i++){
        string channel;
        cin >> channel;
        channels.push_back(channel);
    }

    int idx1 = find(channels.begin(), channels.end(), "KBS1") - channels.begin();
    int idx2 = find(channels.begin(), channels.end(), "KBS2") - channels.begin();

    if (idx1 > idx2){
        idx2++;
    }

    for(int i=0; i < idx1; i++){
        commands.push_back('1');
    }

    for(int j= idx1; j > 0; j--){
        commands.push_back('4');
    }

    for(int i=0; i < idx2; i++){
        commands.push_back('1');
    }

    for(int j=idx2; j > 1; j--){
        commands.push_back('4');
    }

    for (char c : commands) {
        cout << c;
    }
```

```
    return 0;
}
```

> 입력하나에 여러개의 해가 존재할 수 있는 문제이다. 따라서 1번, 4번 버튼만 이용하도록 제한하고 푸는 문제, 제약 조건에 방법의 길이는 500보다 작아야한다.

> 채널이 최대 100개면, KBS1, KBS2가 리스트에 마지막에 연속적으로 있다고 해도 400언저리이다. 따라서 1번, 4번만 이용

집합 - 백준

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

int main(int argc, char** argv) {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);

    int M;
    cin >> M;

    vector<int> set;
    vector<int> all = {1, 2, 3, 4, 5,
                       6, 7, 8, 9, 10,
                       11, 12, 13, 14, 15,
                       16, 17, 18, 19, 20};

    for(int i = 0; i < M; i++) {
        string command;
        cin >> command;

        vector<int>::iterator iter; // 중요!

        if (command == "add" || command == "remove" || command == "check" ||
command == "toggle") {
            int elem;
            cin >> elem;
            iter = find(set.begin(), set.end(), elem);
        }

        if (command == "add") {
            if (iter == set.end()) {
                set.push_back(elem);
            }
        }
```

```
        else if (command == "remove") {
            if (iter != set.end()) {
                set.erase(iter);
            }
        }

        else if (command == "check") {
            if (iter != set.end()) {
                cout << "1\n";
            }
            else {
                cout << "0\n";
            }
        }

        else if (command == "toggle") {
            if (iter == set.end()) {
                set.push_back(elem);
            }
            else {
                set.erase(iter);
            }
        }

        else if (command == "all") {
            set = all;
        }

        else if (command == "empty") {
            set.clear();
        }
    }
    return 0;
}
```

> 맨처음에 toggle 발생할 때 마다 for문 돌리는 방식으로 작성했는데, 매우 비효율적이라고 판단되어 미리 [1, ..., 20] 까지 리스트를 생성하고 이를 복사하는 방식으로 변경했다.

줄세우기 - 백준

정렬 문제를 처음 접하다 보니 어떻게 풀어야할지 감이 안잡혔던 문제, 정렬 방식 중 가장 구현이 간단한 Bubble Sort 방식으로 풀면 되는 문제다.

```
import sys

# 버블 정렬
p = int(sys.stdin.readline())

for _ in range(p):
```

```
    cnt = 0
    students = list(map(int,sys.stdin.readline().split()))

    for i in range(1, len(students)-1):
        for j in range(i + 1, len(students)): # 자신 앞에 키큰 애가 있는 지 확인
            if students[i] > students[j]:  # i가 더 크면
                students[i], students[j] = students[j], students[i]  # 자리바꾸기
                cnt += 1

    print(students[0], cnt)
```

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

using namespace std;

int main(int argc, char** argv){
    int p;
    cin >> p;

    for(int k=0; k<p; k++){
        int c;
        cin >> c;

        int cnt = 0;
        vector<int> students(20);
        for(int i = 0; i < 20; i++){
            cin >> students[i]; // 학생 입력 받기
        }

        for(int i = 0; i < 19; i++){
            for(int j = i + 1; j < 20; j++){
                if(students[i] > students[j] && i != j){ // 앞에 있는 학생이 더 크
면
                    auto tmp = students[i];
                    students[i] = students[j]; // 뒤에 있는 학생을 앞으로 보내고
                    students[j] = tmp; // 앞에 있는 학생을 뒤로 보냄
                    cnt ++;
                }
            }
        }
        cout << c << ' ' << cnt << "\n";
    }
    return 0;
}
```

> 버블정렬이다

Back tracking

평행 - 프로그래머스

```
import math

tmp = []
combs = []
def makeComb(idx, dots):
    if len(tmp) == 2:
        combs.append([tmp[0], tmp[1]])
    else:
        for i in range(idx, len(dots)):
            tmp.append(dots[i])
            makeComb(i+1, dots)
            tmp.pop()

def solution(dots):
    answer = 0
    makeComb(0, dots)

    for c in combs:
        rest = list(set(map(tuple, dots)) - set(map(tuple, c)))

        gradient1 = (rest[1][1] - rest[0][1]) / (rest[1][0] - rest[0][0])
        gradient2 = (c[1][1] - c[0][1]) / (c[1][0] - c[0][0])

        if gradient1 == gradient2:
            return 1

    return 0
```

> `from itertools import combinations`을 사용할 수도 있지만 백트래킹으로 조합을 생성했다.

> gradient1과 gradient2를 비교해서 같으면 1을 리턴한다

구슬을 나누는 경우의 수 - 프로그래머스

```
cnt = 0 # 경우의 수를 카운트할 변수

tmp = []
def makeComb(idx, balls, share):
    global cnt

    if len(tmp) == share:
        cnt += 1

    else:
        for i in range(idx, balls):
            tmp.append(i)
            makeComb(i+1, balls, share)
            tmp.pop()
```

```
def solution(balls, share):
    global cnt
    makeComb(0, balls, share)

    return cnt
```

```
import math

def solution(balls, share):
    if balls == share:
        return 1
    else:
        return math.comb(balls, share)
```

> 재귀로 풀다보니까 시간초과가 발생한다. 단순하게 math의 comb 함수를 이용하자

K중에 1개를 N번 뽑기 - 코드트리

1이상 K이하의 숫자를 하나 고르는 행위를 N번 반복하여 나올 수 있는 모든 서로 다른 순서쌍을 구해주는 프로그램을 작성하시오 예를 들어 K이 3, N이 2인 경우 다음과 같이 9개의 조합이 가능합니다.

[1, 1], [1, 2], [1, 3], [2, 1], [2, 2], [2, 3], [3, 1], [3, 2], [3, 3]

```
'''
       c(1)
       VVV
  1     2     3
  V     V     V
 c(2)  c(2)  c(2)
 VVV   VVV   VVV
 123   123   123
'''
import sys

def choose():
    global answer

    if len(answer) == n:
        print(*answer)
        return

    else:
        for i in range(1, k+1):
            answer.append(i)
            choose()
            answer.pop()

answer = [] # 조합을 담을 리스트
```

```
k, n = map(int, sys.stdin.readline().split())
choose()
```

```cpp
#include <iostream>
#include <vector>
using namespace std;

vector<int> answer;
int K, N;

void choose(){
    if(answer.size() == N){ // answer의 size가 N과 같아졌다면 print 한다
        for(auto a: answer){
            cout << a << " ";
        }
        cout << "\n";
    }

    else{
        for(int i=1; i <= K; i++){
            answer.emplace_back(i); // answer에 i값 추가
            choose(); // 재귀 실행
            answer.pop_back(); // answer에서 i값을 뺀다
        }
    }
}

int main() {
    cin >> K >> N;
    choose();
    return 0;
}
```

> 재귀 조건을 순서대로 생각해보자

> 1. for문에서 1이 가장 먼저 들어간다
> 2. 재귀호출로 1이 또 들어간다
> 3. len(answer == n)에 걸렸으므로 print하고 return 한다, [1, 1]
> 4. pop()이 이루어진다, 1 만 남았다
> 5. 2가 append 된다
> 6. 재귀호출 했는데 if 조건에 걸렸으므로 print하고 return 한다, [1, 2]

아름다운 수 - 코드트리

1이상 4이하의 숫자로만 이루어져 있으면서, 정확히 해당 숫자만큼 연달아 같은 숫자가 나오는 숫자를 아름다운 수 라고 부릅니다.

예를 들어 1333221는 1이 1번, 3이 3번, 2가 2번 그리고 1이 1번 연속하여 나오므로 아름다운 수 입니다.

이때 동일한 숫자에 대해 연달아 같은 숫자의 묶음이 나오는 것 또한 아름다운 수 입니다. 예를 들어 111, 22222222와 같은 수 역시 1이 1번 나온 것이 3번 반복되었고, 2가 2번 나온 것이 4번 반복되었다고 할 수 있기 때문에 아름다운 수라고 할 수 있습니다. 다만, 222의 경우에는 2가 2번 나온 뒤, 다시 2가 1번 나왔으므로 아름다운 수가 아닙니다.

n자리 아름다운 수가 몇 개 있는지를 구하는 프로그램을 작성해보세요.

```python
import sys

answer = 0
n = int(sys.stdin.readline())

# 아름다운 수 인지 확인하는 함수
def check(num_list):
    global answer
    flag = True
    i = 0

    while i < len(num_list):
        standard = num_list[i]
        cnt = 0
        while i < len(num_list) and standard == num_list[i]:
            cnt += 1
            i += 1
        if (cnt != standard) and (cnt % standard != 0): # 반례 추가, 111인경우 1이 1번 나온것이 3번반복되었으므로 아름다운 수
            flag = False
    if flag:
        answer += 1

# back tracking
def makeBeautifulNum():
    global num

    if len(num) == n:
        check(num)
        return

    else:
        for i in range(1, 5):
            num.append(i)
            makeBeautifulNum()
            num.pop()

num = []
makeBeautifulNum()
print(answer)
```

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

```cpp
using namespace std;

int N;
int answer = 0; // 아름다운 수의 갯수
vector<int> vec; // 조합을 담을 벡터 선언

void check(){ // 아름다운 수인지 아닌지 확인하는 함수
    int idx = 0;
    bool flag = true;

    while(idx < vec.size()){
        int standard = vec[idx];
        int cnt = 0;
        while(idx < vec.size() && standard == vec[idx]){
            cnt++;
            idx++;
        }

        if(standard != cnt && cnt % standard != 0){ // 반례조건 명심할 것
            flag = false;
        }
    }
    if(flag) answer++;
}

void makeNum(){ // 경우의 수를 생성하는 함수
    if(vec.size() == N){
        check();
        return;
    }
    else{
        for(int i = 1; i <= 4; i++){
            vec.emplace_back(i);
            makeNum();
            vec.pop_back();
        }
    }
}

int main() {
    cin >> N;
    makeNum();
    cout << answer <<"\n";
    return 0;
}
```

> 경우의 수 만드는 방법을 잘 알아놓자, 가장 먼저 트리 구조를 생각한 후 반복문 만큼 재귀를 호출하면 된다

> 재귀가 끝나면 벡터에 존재하는 함수를 출력하거나 확인한 후 맨 뒤에 있는 값을 빼서 다음 경우의 수를 검사한다

N과 M (1) - 백준

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

- 1부터 N까지 자연수 중에서 중복 없이 M개를 고른 수열

```python
candidate = []
def makeComb():
    if len(candidate) == m:
        print(*candidate)

    else:
        for s in series:
            if (len(candidate) == 0) or (s not in candidate):
                candidate.append(s)
                makeComb()
                candidate.pop()

n, m = map(int, input().split())
series = [i for i in range(1, n+1)]
makeComb()
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int n, m;
vector<int> series; // 수열 미리 만들어 놓기
vector<int> combs; // 경우의 수를 담아놓을 벡터

void makeComb(){
    if(combs.size() == m){
        for(auto c: combs){
            cout << c << " "; // 경우의 수 출력하기
        }
        cout << "\n";
    }
    else{
        for(auto s: series){
            if(combs.size() == 0 || find(combs.begin(), combs.end(), s) ==
combs.end()){
                combs.push_back(s);
                makeComb();
                combs.pop_back();
            }
        }
    }
```

```
}

int main(int argc, char** argv){
    cin >> n >> m;

    for(int i = 1; i <= n; i++){
        series.push_back(i);
    }
    makeComb();

    return 0;
}
```

> 경우의 수를 만드는 방법에 대해서 드디어 배우게 되었다!!!

N과 M (2) - 백준

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.

```
candidate = []
def makeComb():
    if len(candidate) == m:
        flag = True
        for i in range(len(candidate)-1):
            if candidate[i] > candidate[i+1]:
                flag = False
                break

        if flag:
            print(*candidate)

    else:
        for s in series:
            if (len(candidate) == 0) or (s not in candidate):
                candidate.append(s)
                makeComb()
                candidate.pop()

n, m = map(int, input().split())
series = [i for i in range(1, n+1)]
makeComb()
```

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

```
using namespace std;

int n, m;
vector<int> series; // 수열 미리 만들어 놓기
vector<int> combs; // 경우의 수를 담아놓을 벡터

void makeComb(){
    if(combs.size() == m){
        bool flag = true;
        int idx = 0;
        while(idx < combs.size()-1){
            if(combs[idx] > combs[idx + 1]){ // 오름차순이 아님
                flag = false;
                break;
            }
            idx++;
        }

        if(flag){
            for(auto c: combs){
            cout << c << " "; // 경우의 수 출력하기
            }
            cout << "\n";
        }
    }

    else{
        for(auto s: series){
            if(combs.size() == 0 || find(combs.begin(), combs.end(), s) ==
combs.end()){
                combs.push_back(s);
                makeComb();
                combs.pop_back();
            }
        }
    }
}

int main(int argc, char** argv){
    cin >> n >> m;

    for(int i = 1; i <= n; i++){
        series.push_back(i);
    }
    makeComb();

    return 0;
}
```

> 조건이 하나가 추가된 문제, 출력할 때 오름차순인지 검사가 필요하다.