목 차

문법 및 함수 정리

for문 매크로

```
#define FOR(x,n,m)   for(int x=n;x<(m);x++)
```

lower_bound, upper_bound

lower_bound(first, last, val);
-> first부터 last까지 중 val 보다 크거나 같은 값의 첫 위치
upper_bound(first, last, val);
-> first부터 last까지 중 val 보다 큰 값의 첫 위치

C++ 구조체 생성자 오버로딩

```
typedef struct S{
        int x,y,z;
        S(){z=1;}
        S(int X, int Y, int Z) : x(X),y(Y),z(Z){}
};
```

위처럼 구조체도 생성자 오버로딩이 가능함.

map 순회

```
for(auto it=m.begin(); it!=m.end(); it++)
```

해당 위치의 데이터 값 접근

```
it->first, it->second
```

2차원 배열 90도 회전

```
int temp_arr[100][100];
FOR(i,0,n) FOR(j,0,n) temp_arr[i][j] = arr[n-j-1][i];
memmove(arr,temp_arr,sizeof(arr));
```

2차원 행,열 바꾸기

```
int temp_arr[100][100];
FOR(i,0,n) FOR(j,0,n) temp_arr[i][j] = arr[j][i];
memmove(arr,temp_arr,sizeof(arr));
```

배열 내 최소, 최대값 구하기

*min_element(zSum, zSum+5);

*max_element(zSum, zSum+5);

배열 내 최소,최대를 구하는 함수로써 비교 함수도 인자로 넣어줄 수 있다.

주의할 점은 배열의 return 값이 배열의 주소 값이기 때문에 *을 붙여줘야한다.

배열 복사

memmove(temp, arr, sizeof(arr));

이런 경우 다차원 배열도 복사가 가능함.

2차원 배열의 일부분도 복사가 가능함.

memmove(temp, arr[x], sizeof(temp));

temp는 1차원 arr는 2차원인 경우에도 가능함.

2차원 vector 크기 및 값 초기화

vector<vector<int> > arr(6, vector<int>(5, 0));

이런 경우 6x5의 vector를 0으로 초기화

C++ split 함수 구현

```
vector<string> split(string s, string d){
    vector<string> result;

    auto start = 0U;
    auto end = s.find(d);
    while (end != string::npos){
        result.push_back(s.substr(start, end - start));
        start = end + d.length();
        end = s.find(d, start);
    }
    result.push_back(s.substr(start, end));
```

```
    return result;
}
```

C++ split 함수 구현 2

split delimeter를 여러개 지정하고 싶을 때 사용.

단 strtok의 인자로 string이 아닌 char배열이 들어가야함.

```
char* tok3 = strtok(str3,"-+*");
while(tok3!=NULL){
    cout<<tok3<<endl;
    tok3 = strtok(NULL,",-; ");
}
```

C++ replace all 구현

x를 y로 replace

```
replace( s.begin(), s.end(), 'x', 'y');
```

cmp 함수 선언

```
bool cmp(string s, string s2){
    if(s.size() < s2.size()) return true;
    return false;
}
```

이 경우 string 길이에 따른 cmp 함수 선언.

Union Find

여러 서로소 집합의 정보를 저장하고 있는 자료구조를 의미함.

트리 구조로써 집합을 표현하고 경로 압축과 레벨 최적화 도입해서 유용함.

```
int find(int n) {
    if (room.find(n) == empty_room) return n; // 비어있으면 바로 배정

    room[n] = find(room[n]); // 비어있지 않으면 해당 번호가 가르키는 곳 다시 찾기
    return room[n];
}
```

틀린 문제

백준 16235 - 나무 재테크

-> 단순 simulation 문제
-> 각각의 나무를 vector의 2차원 배열로 표현했음.
-> 봄 계절에 양분을 얻지 못하는 나무는 죽은 나무로 표현하고 삭제하지 않음.
-> 이유는 vector의 erase를 쓰고 싶지 않았기 때문에.
-> 하지만 데이터를 남기고 있던 것이 오히려 시간을 더 오래 걸리게 했음.
-> vector의 pop_back() 함수를 이용하면 해결 가능함.
-> 원래는 pop_back을 안써서 vector의 뒤에서부터 탐색하며 erase를 했는데 대안이 생김.

백준 17143 - 낚시왕

-> simulation 문제
-> 상어 잡기, 상어 움직이기 모두 구현했는데, 상어가 겹칠 때 효율적으로 삭제시키는 방법을 알아야함.
-> 구현은 할 수 있겠지만 너무 비효율적인 방법인 것 같음.
-> 삭제할 때 해당 좌표에 상어들을 다 넣어놓고 나중에 비교하고 삭제하려고 했음.
-> 이렇게 하면 가장 큰 상어를 구할지라도 삭제하는 과정에서 vector의 erase를 사용하고 코드가 길어짐.
-> 구조체 S 타입으로 2차원 배열 선언하고 상어의 이동이 끝날 때마다 해당 위치 상어와 비교
-> 크기가 크다면 덮어 씌우고 크기가 작다면 다음으로 넘어감.

SWEA 2382 - 미생물 격리

-> simulation 문제
-> 이전에 삼성 기출 낚시왕 문제를 풀면서 동일한 자리에 왔을 때 삭제하는 방법 적용.
-> 하지만 i 번째 미생물과 해당 자리에 있는 미생물을 비교하는 즉시 값을 더해버림.
-> 이러면 같은 자리에 계속 값이 누적되기 때문에 뒤에 미생물들을 다 삼켜버림.
-> 답은 겹치는 미생물 중에서 가장 큰 미생물의 방향을 따라야함.
-> 그래서 미생물을 삼킬 때마다 값을 바로 더하지말고 sum[nx][ny]에 누적시키고
-> 최종적으로 남은 미생물 크기에 더해줌.

백준 17136 - 색종이 붙이기

-> dfs 문제
-> (0,0)에서 시작해 한 칸씩 움직이면서 5개 색종이를 붙이고 dfs 재귀 사용.
-> 색종이를 붙일 수 있는지에 대한 조건식에서 실수해버림.
-> visit 변수에 대해서는 체크하지 않아서 시간 초과 및 오답이 나와버려서 해맸음.
-> 조건 잘 보고 따지고, 굳이 visit 변수를 만들지 않고 배열 값을 0으로 만들면 간단해짐.

SWEA 2112 - 보호 필름

-> dfs 문제
-> 각 줄마다 0 또는 1로 바꾸기 또는 바꾸지 않고 다음 depth로 넘어가기
-> 문제의 구현자체는 크게 어렵지 않았지만 코드가 조금 더러운 느낌이 있음.
-> 숏코딩 참고해보자
-> 모든 행에 dfs를 진행하고 끝 행에 왔을 때 return 시키는 방법을 사용해서 시간초과 뜸.
-> 문제의 답은 무조건 k보다 클 수 없다는 점을 이용해서 n이 k보다 크면 return 시켜야함.
-> dfs는 깊이 끝까지 간다는 점을 잘 생각하고 문제를 풀어야할 듯.

카카오 겨울 인턴 2번 - 튜플

-> string 처리
-> split, replace 숙지할 것!
-> 입력 string을 vector에 저장해 갯수만큼 정렬시켰지만 그다음을 몰라 못품.
-> string 처리 후 string vector에 저장하고 정렬시켰는데 int vector로 저장시키고 합을 구함.
-> 그러면 sum[i] - sum[i-1]의 값이 새로운 원소가 됨.

카카오 겨울 인턴 3번 - 불량 사용자

-> string 처리
-> ban id와 user id를 비교하는 작업까지는 어렵지 않았음.
-> 하지만 여러 ban id에 적용되는 user id 처리를 하는데 어려웠음.
-> n의 크기가 8 이하라서 next_permutation 이용하면 됨.
-> bit mask 사용법 알아보자.

카카오 겨울 인턴 4번 - 호텔 방 배정

-> 효율성 문제
-> 문제 자체는 간단한 문제였지만 입력 값의 범위를 보면 분명 시간초과가 날 듯함.
-> 역시나 단순 구현을 했을 때는 시간초과가 났음.
-> 방의 갯수가 10의 12승이기 때문에 배열로써 만들 수 없음.
-> Union find 사용해야함 - 노드들이 해당되는 부모 노드를 가르킴.
-> 현재 위치의 방은 차 있을 경우 다음 방의 위치를 가리킴.
-> 이렇게 하면 배열을 만들지 않아도 됨.

SWEA 1949 - 등산로 조성

-> dfs 문제
-> 문제 자체는 간단한 문제였지만 문제를 제대로 읽지 않음.
-> 문제에서는 한 칸을 최대 k만큼 깎을 수 있다고 했기때문에 꼭 k만큼 깎지 않아도 됨.
-> 그리고 마지막으로 48개에서 통과 못하고 못 풀었던 부분은
-> 봉우리를 깎을 때 현재 위치의 값보다 -1만큼만 깎으면 됨.
-> 오히려 더 깎으면 다음 진행에 있어서 악영향이 있기 때문.

맞은 문제

백준 14890 - 경사로
-> 단순 simulation 문제
-> 경사로를 넣는 조건이 다양해서 조금 까다로웠음.
-> 행, 열에 대해 답을 구해야 하는데 이런 경우 2차원 배열을 90도 돌리면 쉬움.

백준 15684 - 사다리 조작
-> 단순 simulation 문제 + dfs
-> 문제 자체는 어려워 보이지 않는데 구현이 귀찮았음.
-> 히든케이스(사다리를 놓지 않아도 통과되는 경우를 생각 못함.
-> 코드 너무 더럽게 짰음. 숏코딩 참고하자.
-> 꼭 다시 풀어보기.

백준 14891 - 톱니바퀴
-> 단순 simulation 문제
-> 돌리는 바퀴 기준으로 방향 변수에 대해 실수함.

백준 15686 - 드래곤커브
-> 단순 simulation 문제
-> 지금까지 만들어진 방향을 vector에 저장
-> 1세대를 그릴 때마다 vector의 역순으로 탐색.
-> 탐색을 하면서 각 방향에 +1을 해주고 그려줌.
-> Stack을 사용할까 생각했지만 vector로도 충분히 가능할 것 같아서 vector로 구현함.
-> 생각해보니 Stack은 탐색이 힘들어서 적합하지 않을 듯.

백준 16236 - 아기 상어
-> bfs의 반복 문제
-> 먹이를 찾았다고 bfs를 끝내면 안되고 해당 깊이까지는 다 돌아야함.
-> 해당 깊이에서 후보군에 넣을 때 조건이 필요함(0이 아니고 자신보다 작은 크기일 때)
-> bfs 순회 중 현재 깊이가 먹이를 먹은 깊이보다 많을 때 업데이트하고 return 시킴.
-> 이러면 먹이를 먹고 다음 깊이가 없다면 조건에 걸리지 않아 업데이트가 안됨.
-> 업데이트를 while문 밖에 두면 됨.
-> 그러면 조건에 걸려서 break되거나 더이상 순회할 좌표가 없을 때도 업데이트가 됨.
-> 단, 먹이를 먹을 때 값이 변하는 dd 값이 -1이라면 먹이를 못 먹은 경우이므로 return false.

백준 17142 - 연구소 3
-> 조합 찾기 후 bfs 문제
-> 처음에 단순히 bfs가 끝나면 답을 업데이트 시켰음.
-> 이런 경우 빈칸이 다 없어져도 활성화되지 않은 바이러스도 활성화 시키느라 오답이 됨.
-> bfs while에 들어가기 전에 빈칸의 개수를 세고 0개가 되면 while 탈출하도록 함.
-> 조합 찾을 때 dfs로 했는데 return 조건에서 실수함.

백준 17140 - 이차원 배열과 연산
-> simulation 문제
-> 배열을 검사해서 새로운 배열을 만드는 것은 구현했음.
-> 하지만 시험장에서 풀었다면 시간이 부족했을 것 같음.
-> 행마다 검사, 열마다 검사를 각각 따로 만들려고 했기때문.
-> 배열을 90도 돌리는 방법도 생각했지만 배열 모양이 정사각형이 아니기 때문에 불가능.
-> 생각해보니 회전이 아니라 행,열을 바꾸면 되는 문제였음.
-> 계산하는 함수는 1개만 만들고 행,열을 바꿔주며 계속 진행함.

백준 17779 - 게리맨더링2
-> simulation 문제
-> 처음에 구역을 어떻게 나눌지에 대해 고민함.
-> 구역을 모두 5로 채운 다음 각 지역에 맞게 채우면 됨.
-> 1,3 구역은 열 0부터 시작해서 경계선에 만나면 다음 행으로 넘어감.
-> 2,4 구역은 열 n-1부터 시작해서 경계선에 만나면 다음 행으로 넘어감.

백준 17837 - 새로운 게임2
-> simulation 문제
-> 기본적인 구현은 다 했지만 부분부분 실수로 인해 시간이 지체됨.
-> 배열 값 업데이트 할 때 index 값도 업데이트 해야한다면 순서 주의하기.
-> 말이 이동할 때 if, else if로 파랑 or 빨강으로 했는데 이 부분에서 틀렸음.
-> 말이 다음 칸이 파랑색이고 반대 방향도 빨강색일 때 if,else if에 의해 걸리지 않음.
-> 파랑 빨강일 때를 각각 if문으로 구현한다.

백준 17822 - 원판 돌리기
-> simulation, bfs 문제
-> 기본적인 구현은 다 했지만 역시나 실수로 해맸음.
-> Queue 조건에서 값이 같을 때만 추가하는 것을 빼먹음.
-> 삭제할 index 추가할 때 Queue 시작하는 지점을 넣지 않음.
-> 역시나 코드 한줄 복붙하는 과정에서 값을 고치지 않아 계속 답이 안나옴.
-> 절대로 코드 복붙하지말고 실수 줄이려면 직접 쓰자!!

백준 16637 - 괄호 추가하기
-> dfs 문제
-> 문제에서 모든 연산자 우선순위는 같다고 했기 때문에 앞에서부터 차례대로 연산함.
-> 핵심은 괄호를 어디에 넣을지인데 괄호안에는 연산자가 한개만 존재하기 때문에 쉬움.
-> dfs를 이용하여 현재 지점에서 다음 숫자를 바로 연산할지 아니면
-> 다음, 다다음 숫자를 괄호로 묶어서 연산할지 재귀로 들어가면 됨.

백준 17281 - ⚾
-> 시뮬레이션
-> next_permutation을 쓰면 쉽게 풀리는 문제.
-> 매 선수마다 각각의 경우를 구현해준다.
-> 루 배열을 만들어놓고 안타의 경우 루 배열에 따라 점수가 나도록 함.

백준 17406 - 배열 돌리기4

-> 시뮬레이션
-> next_permutation을 쓰면 쉽게 풀리는 문제.
-> 회전할 때 4방향을 모두 for문으로 구현한다.
-> 배열 index 값 실수함.
-> 회전시킬 때 4방향의 코드가 모두 비슷하므로 배열 index 값 실수 주의.

백준 17471 - 게리맨더링

-> bfs 탐색, 조합
-> next_permutation을 쓰면 쉽게 풀리는 문제.
-> 고정 갯수의 조합이 아니기 때문에 for문 안에 do while(next_permutation) 사용.
-> 서로 다 연결되어있는지 확인하기 위해서 bfs를 사용했음.

카카오 겨울 인턴 1번 - 크레인 인형뽑기 게임

-> 명령에 맞게 인형을 옮기는 시뮬레이션
-> vector의 back(), pop_back()을 사용하면 쉬움.
-> 처음에 입력을 제대로 확인하지 않아서 틀림.
-> 입력은 위에부터 받기 때문에 아래부터 저장하는 새로운 2차원 vector를 만들어야함.

SWEA 2105 - 디저트 카페

-> 시뮬레이션
-> 백준 게리맨더링2와 같이 4중 for문을 이용.
-> 쉬운 문제였음.

SWEA 1953 - 탈주범 검거

-> bfs 문제
-> 주의해야할 점은 터널에 따른 탐색 조건을 잘 구현해야함.