**<시간 복잡도 팁>**

[**https://school.programmers.co.kr/questions/43408**](https://school.programmers.co.kr/questions/43408)

문제를 보면, 무언가 특별한 알고리즘이 존재할 것 같지만,

실제로는 그냥 무지성으로 모든 경우를 때려박아도 상관없습니다.

이건 약간의 꿀팁입니다..

답변을 먼저 제출하여, 효율성 테스트를 채점하는지 우선 확인합니다.

효율성을 채점하지 않는다면, 대부분 알고리즘이 10초이내에 돌아가게 만들면 됩니다.

그 다음, 문제의 제한 사항을 확인합니다.

여기서에서 배열을 총 길이들을 확인하여, 최대 가능한 연산횟수를 대충 유추할 수 있습니다.

users가 최대 100, emoticons가 최대 7의 길이를 갖습니다.

또한, 문제에서 할인율은 [10, 20, 30, 40] 4종류의 할인율만 가진다고 하였습니다.

대충 가질 수 있는 할인율을 계산해보자면, 16000(4^7)가지가 될겁니다.

그럼, 100 \* 7 \* 16000 = 1100만번의 연산을 수행합니다.

여기서 보조 연산이 들어가거나, 언어의 속도에 따라 추가적인 시간 소요가 일어납니다.

많은 연산이기는 하지만, 무지성으로 탐색해도, 1억번을 넘지 않으므로, 충분히 시간안에 통과가 가능합니다.

연산횟수가 1억 단위가 넘어가면, 시간 초과가 뜰가능성이 존재한다는 사실을 늘 염두하셔야 합니다.

이런 팁을 알면, 실제 코딩테스트를 풀 때, 사용할 알고리즘을 빠르게 유추해볼 수 있겠습니다.

이미 다들 아실거라 생각하는 내용이지만,

저와 같은 초보자들에게는 매우 중요한 팁이 되지 않나 싶습니다.

<https://school.programmers.co.kr/questions/30876>

문제를 다 읽고 드는 생각은 아마 전부 계산하는 방법일겁니다. 그리고 제한 사항을 확인해보겠죠. N \* M = 1,000,000이 나옵니다. skill 길이가 25만이므로 최악의 경우 25만 \* 100만까지 계산하게 될겁니다. 시간복잡도는 O(NMlen(skill))이 됩니다. 해당 방법으로는 정확도는 어떻게든 맞추더라도 효율성에서 점수를 많이 깎아 먹을겁니다. 또한 문제가 요구하는 풀이도 아니겠죠. 자 그렇다면 계산 횟수를 어떻게 줄일까요?

바꿀 수 있고 바꿀 수 없는 것을 나눠봅시다. skill 길이는 최대 25만입니다. 바꿀 수 없죠. NM은 최대 100만이지만 모든 구간을 다 계산하지 않으면 이를 줄일 수도 있을겁니다. 따라서 NM은 우리가 바꿀 수 있는 요소입니다. 자 그럼 이를 어느 정도 까지 줄여야할까요? 파이썬 1초 == 2천만번 연산이라고 합시다. skill 길이는 최대 25만이므로 한 skill 연산당 최대 2천만/25만=8000번 계산할 수 있을겁니다. 쉽지 않군요. (NM)log2(NM)보다 낮아야합니다. log(N\*M)=20 혹은 상수 시간만에 skill 한개에 대한 작업을 완료해야합니다. 어떻게 하면 skill 한개의 연산을 줄일 수 있을지 하나 하나 뜯어봅시다.

**<DFS와 BFS>**

**DFS**는 **완전 탐색**시 사용하며, **BFS**는 **최단경로 및 임의의 경로**를 찾을 때 사용합니다.

int check[101][101];

check[0][0] = true;

int dx[] = {0, 0, 1, -1};

int dy[] = {1, -1, 0, 0};

**<무한 표시>**

#include <climits>

= INT\_MAX;

**<구조체 활용>**

struct **load**

{

int a, b, c;

};

queue<**load**> que;

que.push(load{0,0,0});

**문자간의 거리(간격)** = 경우의 수 총합 / 문자열 길이의 경우의 수(1=5, 2= 25, 3= 125, 4=625, 5=3125)

**<문자열의 공백 무시>**

**#include <sstream>**

string strA, strB, strC;

string str = "ab cd ef";

stringstream ss(str);

ss>>strA;//ab 저장됨

ss>>strB;//cd

ss>>strC;//ef

**<대문자 소문자 변환>**

c=toupper(c); //**대문자 소문자**

else c=tolower(c); //char 1개만

transform(str.begin(), str.end(), str.begin(), ::tolower); //str**문자열 전체**를 **lower**

string charStr(1, 'c'); //**char**` **string**으로 **변환**

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z //알파벳 순서

(char)(‘a’+1) = ‘b’; //**정수형**에서 **Char**으로 변환

if(a>=**97**) answer[a-'a'+26]**++**; //소문자일 경우(97~122) //26개

if(a<**97**) answer[a-'A']++; //대문자일 경우(대문자는 **65** ~ 90) //26개

if (c < 'a') answer += **(char)(((c - 'A') + n) % 26 + 'A');** //’A’~’Z’와리가리(c는 char)

else answer += **(char)(((c - 'a') + n) % 26 + 'a');**  //’a’~’z’와리가리

**#include <vector>**

배열,문자열,벡터 정렬: **sort(**벡터명**.begin(),** 벡터명**.end());**  **(문자열도 가능함, 처음 위치 필요시 따로 저장)**

**또는**

sort(strings.begin(), strings.end(), **compare**);

bool **compare** (string a, string b)

{

**return a[i] == b[i]** ? **a < b** : **a[i] < b[i]**;

} //b가 큰게 기본 값인가 봄(우상향)

bool customCompare(int a, int b) {return to\_string(a) + to\_string(b) > to\_string(b) + to\_string(a);} // 숫자인 a와 b를 문자열로 변환하여 비교

**vector<int> dp(n, 1);** //벡터의 크기를 n으로 설정 후, 모든 값을 1로

억 단위 길이로 만들면 **타임 오버** 발생

맨 뒤 요소 삭제: vec.pop\_back();

배열,문자열,벡터 역정렬: **sort(**벡터명**.rbegin(),** 벡터명**.rend());**  **(문자열도 가능함)**

배열 뒤집기**: reverse(**벡터명**.begin(),** 벡터명**.end());** //문자열도 가능함

**N의 갯수 새기: 변수 = count(**벡터명**.begin(),** 벡터명**.end(),n);**

**배열 안의 요소 삭제: 벡터명.erase(**벡터명**.begin()+i); //한칸씩 옮기는 처리(O(n))**

**배열 안의 중복 제거: 벡터명.erase(unique(v.begin(), v.end()), v.end());**

벡터의 크기: 벡터명**.size();**

**벡터의 맽 끝 요소: m.back();**

벡터에 요소 삽입: **벡터.emplace\_back()**;

벡터 안에서 **최대 값** 출력: \*max\_element(vec.begin(), vec.end())

**벡터에서 특정 요소 찾기**: ( **str.find('z');** )

벡터 초기화:vec.clear();

int target = 30;

auto it = find(vec.begin(), vec.end(), target);

if (it != vec.end())

{

int pos = distance(vec.begin(), it);

cout << "요소 " << target << "의 인덱스: " << pos << endl;

}

else cout << "요소를 찾지 못했습니다." << endl;

**벡터의 가능한 모든 종류 출력**:

**vector<int> vec = {1, 2, 3, 4};**

do

{

for (int i = 0; i < vec.size(); i++) //for문 제외하면 같은 길이로 출력

{

cout<<vec[i]<<' '; //같은 자리 크기로 출력

}

cout<<endl;

} while (**next\_permutation**(vec.begin(), vec.end())); //number는 문자열(1234->1243….)(넘어가는 동안 true)

**벡터 중간에 삽입**:

//1. 벡터에 하나의 요소 삽입

vector<int> vec = {1, 2, 3, 4, 5};

vec.insert(vec.begin() + 2, 100);//1 2 100 3 4 5

//2. 벡터에 다수의 요소 삽입

vec.insert(vec.begin() + 4, 3, 200);//1 2 100 3 200 200 200 4 5

//3. 다른 벡터의 요소를 맨 앞에 삽입

vector<int> anotherVec = {300, 400};

vec.insert(vec.begin(), anotherVec.begin(), anotherVec.end());//300 400 1 2 100 3 200 200 200 4 5

//4. 다른 벡터의 요소를 맨 뒤에 삽입

vec.insert(vec.end(), {500, 600, 700});//300 400 1 2 100 3 200 200 200 4 5 500 600 700

vector<vector<int>> vec;

for(**vector<int> tmpVec: vec**) //에서 tmpVec을 수정해도 vec에는 **영향 없음**

초기화는 무조건 하자(이전에 사용한 값 딸려옴)

vector<vector<int>> vec(rows, vector<int>**(cols, 1)**); //2차원 배열 초기화

vec.push\_back(**{1, 2}**);

cout<<**vec[0][1]**;

**특수 구조**

**배열의 초기화**: int dp[10]; //**배열 구조**

fill(dp, dp+10, 10); //전부 초기화

fill(dp+1,dp+3, 10); //1번째부터 2번째까지 초기화

**#include <stack>** //**스택 구조**

stack<char> sStack; //정의

sStack.push(chr); //푸쉬

sStack.pop(); //팝

sStack.top()==’s’ //맨 위 출력

sStack.empty

**#include <queue>** //**큐 구조**

queue<int> q;;

q.push(10);

q.pop();

cout<<q.front();

cout<<q.back();

queue<pair<int, int>> q; //**큐에 2가지 정보 저장**

q.push({1,1});

cout<<q.front().first;<< q.front().second;

deque<int> DQ; //**이중 큐 구조**

push\_back();

push\_front();

pop\_back();

pop\_front();

**priority\_queue**<int,vector<int>,**greater**<int>> que (vec.begin(), vec.end()); //우선순위 **큐 구조, 작은 것부터** 출력함, 반대는: less(자동 정렬)

아니면 priority\_queue<int> pq;

cout<<que.top();//우선순위 큐 이미지의 맨 위 상상, 벡터의 0번이 큐의 탑(사용하는 함수가 다름)

값을 갱신하려면 pop하고 나서, push

**pair< >우선순위 큐**

struct ComparePair {

bool operator()( pair<int, int> a, pair<int, int> b) {

return a.first > b.first; // 오름차순으로 정렬(이것만 순서가 반대로임)

}

};

priority\_queue<pair<int, int>, vector<pair<int, int>>, **ComparePair**> pq; //구조체 안쓰고는 이렇게 못함

**#include <map>**  //**map 구조 O(log n)**

map<string, int> m;

m[s] = true;

cout << **m["a"]** << endl;

**for(auto at: m)cout<<at.second<<' ';**

**map<int, vector<int>>m2; //맵에 벡터를 저장**

for (auto entry : m2) {

int key = entry.first;

vector<int> values = entry.second;

cout << "Key: " << key << ", Values: ";

for (int value : values)

{

cout << value << " ";

}

}

**#include <unordered\_map> // unordered\_map 구조 O(1), 중복 없다고 했나**

unordered\_map<string,int> um;

**pair<int, double> myPair(10, 3.14);**  //**pair 구조**

cout << myPair.first << myPair.second;

**set<int>S;** //**set 구조(벡터와 유사하지만 중복 불가)(**foreach 반복문**으로만 출력 가능), 정렬 후 벡터를 넣어 중복 방지 가능**

S.insert(1);

st.count("abc2") //**해당 요소가 있는지 확인**하고 있으면 1, 없으면 0;push\_bac

set<int> bro\_1(topping.begin(), topping.end()) //벡터 안 중복 제거

string str = **s.substr**(index, pos - index); // string first = s.substr(1, 2) = s.substr(1); //**시간 복잡도** = O**(생성 할 문자열의 길이**)

//-------------------------------------------------------------------------

for문에서 범위를 for문 내부에서 조절하는 경우 오류 발생

while문에서 조건부 완전 초기화는 효율 쓰레기

char -> int형: **c - 48;**

int-> char형: **str[0]**

**<간단한 참조 호출>**

void add(int &b)

b+= 1;

int a = 5;

add(a);

cout<<a<<endl;

void add(int ar[][2]); //**자동 참조됨**

void add(int ar[][2]) {

ar[0][0] = 10000;

}

for(**string &str**: vec) //foreach문이여도 내부 변경 가능

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

벡터 참조 호출: void select(**vector<vector<int>>& arr2** , **vector<string> arr3**);

큰 숫자를 매 번 나누는 이유는 숫자가 크면 오버플로우가 발생해서

**<코어 덤프가 발생하는 경우>**

배열의 크기 문제

타입 문제(stoi말고 stol, stoll) //타입 안맞으면 실패 나옴

**<타입에 따른 시간>**

long 안쓰고 그냥 비교하면 45초

int로 하면 52초

long long은 72초

//--------------------------------------------------------------------------------

**10진수->2진수 변환하는 함수**

int decimalToBinary(int decimal) {

int binary = 0;

int power = 0;

while (decimal > 0) {

int remainder = decimal % 2;

binary += remainder \* **pow(10, power++)**;

decimal /= 2;

}

return binary;

}

**<소수인지 확인하는 함수>**

bool IsPrime(int number)

{

if (number < 2)

return false;

for (int i = 2; i <= **sqrt**(number); i++) //제곱근(자동으로 floor)

{

if (number % i == 0)

return false;

}

return true;

}

**<부동소수점이 존재하는지 확인하는 함수>**(2.0과 2.1 구분)

bool hasFloatingPoint(double num)

return num != static\_cast<long long>(num);

**<시간 차이 계산 함수>**

**#include <sstream>**

int calculateTimeDifference(string time1 = "12:00", string time2 = "12:14") {

// 시간 문자열을 시간과 분으로 분할

istringstream stream1(time1);

istringstream stream2(time2);

int hour1, minute1, hour2, minute2;

char colon;

stream1 >> hour1 >> colon >> minute1;

stream2 >> hour2 >> colon >> minute2;

// 시간 차이 계산

int time1\_minutes = hour1 \* 60 + minute1;

int time2\_minutes = hour2 \* 60 + minute2;

return time2\_minutes - time1\_minutes;

}