백준 문제 풀이 1/4

1. 프로그래머스 역순 정렬하기, 최댓값 구하기

SQL 기본을 익히자

**문제 설명**

ANIMAL\_INS 테이블은 동물 보호소에 들어온 동물의 정보를 담은 테이블입니다. ANIMAL\_INS 테이블 구조는 다음과 같으며, ANIMAL\_ID, ANIMAL\_TYPE, DATETIME, INTAKE\_CONDITION, NAME, SEX\_UPON\_INTAKE는 각각 동물의 아이디, 생물 종, 보호 시작일, 보호 시작 시 상태, 이름, 성별 및 중성화 여부를 나타냅니다.

| **NAME** | **TYPE** | **NULLABLE** |
| --- | --- | --- |
| ANIMAL\_ID | VARCHAR(N) | FALSE |
| ANIMAL\_TYPE | VARCHAR(N) | FALSE |
| DATETIME | DATETIME | FALSE |
| INTAKE\_CONDITION | VARCHAR(N) | FALSE |
| NAME | VARCHAR(N) | TRUE |
| SEX\_UPON\_INTAKE | VARCHAR(N) | FALSE |

동물 보호소에 들어온 모든 동물의 이름과 보호 시작일을 조회하는 SQL문을 작성해주세요. 이때 결과는 ANIMAL\_ID 역순으로 보여주세요. SQL을 실행하면 다음과 같이 출력되어야 합니다.

| **NAME** | **DATETIME** |
| --- | --- |
| Rocky | 2016-06-07 09:17:00 |
| Shelly | 2015-01-29 15:01:00 |
| Benji | 2016-04-19 13:28:00 |
| Jackie | 2016-01-03 16:25:00 |
| \*Sam | 2016-03-13 11:17:00 |

답

SELECT NAME, DATETIME FROM ANIMAL\_INS ORDER BY ANIMAL\_ID DESC

관련 정보

데이터조회하기 (SELECT)

**SELECT** 열명1, 열명2 **FROM** 테이블명; // 테이블의 열명1 열명2에 대한 행을 조회

* 예약어의 데이터베이스 객체명은 대소문자를 구별하지 않는다.

## 테이블 구조 참조하기 (DESC)

DESC 테이블명;

## 검색 조건 지정하기 (WHERE)

**SELECT** 열1, 열2 **FROM** 테이블명 **WHERE** 조건식;

* WHERE절 예제

**SELECT** \* **FROM** 테이블명 **WHERE** **no** = 2;

//no열의 값이 2인 경우만 조회

**SELECT** \* **FROM** 테이블명 **WHERE** **no** <> 2;

//no열의 값이 2가 아닌 경우만 조회

**SELECT** \* **FROM** 테이블명 **WHERE** **name**='홍길동';

//name열이 홍길동인 경우만 조회

//숫자가 아닌 문자열이나 날짜에 경우 '' 싱글 쿼트롤 둘러싼다.

**SELECT** \* **FROM** 테이블명 **WHERE** **name** **IS** NULL;

//name 열이 NULL인 경우만 조회

## WHERE절 조건 조합하기

**SELECT** \* **FROM** 테이블명 **WHERE** 조건1 **AND** 조건2;

**SELECT** \* **FROM** 테이블명 **WHERE** 조건1 **OR** 조건2;

**SELECT** \* **FROM** 테이블명 **WHERE** **NOT** 조건;

* AND는 OR에 비해 우선순위가 높다. 그러므로 괄호를 통해서 우선수위를 바꿀 수 있다.

**SELECT** \* **FROM** 테이블명 **WHERE** (a=1 **OR** a=2) **AND** (b=1 **OR** b=2);

## 패턴매칭에 의한 검색

**SELECT** \* **FROM** 테이블명 **WHERE** text **LIKE** 'SQL%';

//text라는 열에서 SQL로 시작하는 내용이 있다면 검색한다. (전방매치)

**SELECT** \* **FROM** 테이블명 **WHERE** text **LIKE** '%SQL%';

//text라는 열에서 SQL을 포함하는 내용이 있다면 검색한다. (중간매치)

//예를들어 'SQL은 RDBMS를 조작하는 언어이다'

//'LIKE는 SQL에서 사용할 수있는 술어중 하나이다'

**SELECT** \* **FROM** 테이블명 **WHERE** text **LIKE** '%\%%';

//이스케이프를 통해서 % 검색하기

// \_를 검색할떄도 이스케이프 (\\_) 시켜야한다.

참고도서  
SQL 첫걸음 - 아사이 아츠시

※예제

**1. SELECT empno FROM emp ORDER BY empno DESC**

**2-1. SELECT empno FROM emp ORDER BY empno ASC**

최대,최소

MAX와 MIN는 테이블에 조회되는 데이터의 최대값 및 최소값을 구한다

## MAX(최대값), MIN(최소값) 문법

최대값을 구하는 함수는 아래와 같다.

**SELECT** **MAX**("필드명")

**FROM** "테이블명";

최소값을 구하는 함수는 아래와 같다.

**SELECT** **MIN**("필드명")

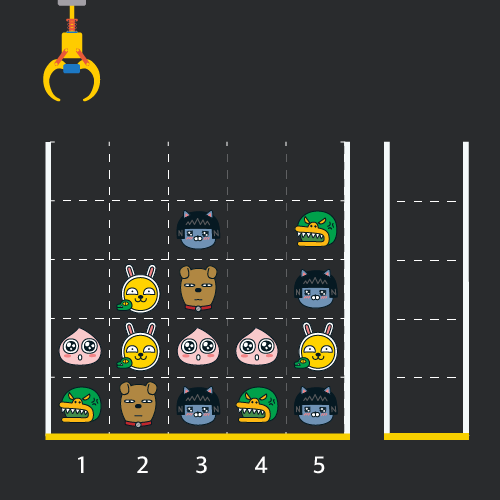
**FROM** "테이블명";

출처: <https://araikuma.tistory.com/519> [프로그램 개발 지식 공유]

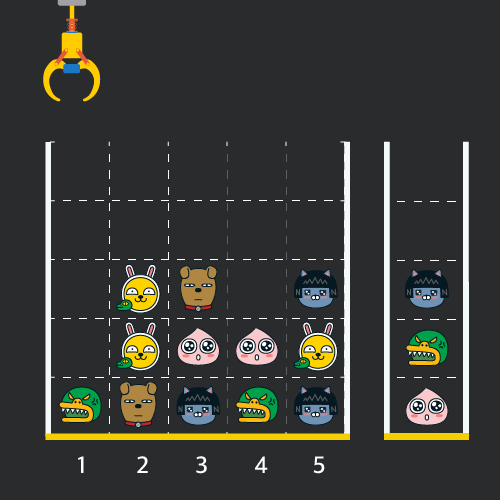
1. 크레인 인형뽑기 게임

###### 문제 설명

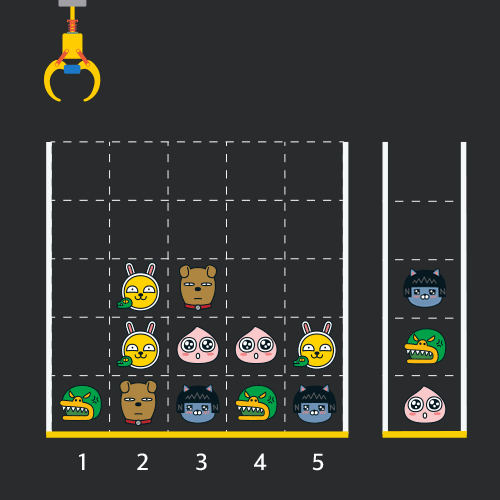
게임개발자인 죠르디는 크레인 인형뽑기 기계를 모바일 게임으로 만들려고 합니다.  
죠르디는 게임의 재미를 높이기 위해 화면 구성과 규칙을 다음과 같이 게임 로직에 반영하려고 합니다.



게임 화면은 **1 x 1** 크기의 칸들로 이루어진 **N x N** 크기의 정사각 격자이며 위쪽에는 크레인이 있고 오른쪽에는 바구니가 있습니다. (위 그림은 5 x 5 크기의 예시입니다). 각 격자 칸에는 다양한 인형이 들어 있으며 인형이 없는 칸은 빈칸입니다. 모든 인형은 1 x 1 크기의 격자 한 칸을 차지하며 **격자의 가장 아래 칸부터 차곡차곡 쌓여 있습니다.** 게임 사용자는 크레인을 좌우로 움직여서 멈춘 위치에서 가장 위에 있는 인형을 집어 올릴 수 있습니다. 집어 올린 인형은 바구니에 쌓이게 되는 데, 이때 바구니의 가장 아래 칸부터 인형이 순서대로 쌓이게 됩니다. 다음 그림은 [1번, 5번, 3번] 위치에서 순서대로 인형을 집어 올려 바구니에 담은 모습입니다.



만약 같은 모양의 인형 두 개가 바구니에 연속해서 쌓이게 되면 두 인형은 터뜨려지면서 바구니에서 사라지게 됩니다. 위 상태에서 이어서 [5번] 위치에서 인형을 집어 바구니에 쌓으면 같은 모양 인형 **두 개**가 없어집니다.



크레인 작동 시 인형이 집어지지 않는 경우는 없으나 만약 인형이 없는 곳에서 크레인을 작동시키는 경우에는 아무런 일도 일어나지 않습니다. 또한 바구니는 모든 인형이 들어갈 수 있을 만큼 충분히 크다고 가정합니다. (그림에서는 화면표시 제약으로 5칸만으로 표현하였음)

게임 화면의 격자의 상태가 담긴 2차원 배열 board와 인형을 집기 위해 크레인을 작동시킨 위치가 담긴 배열 moves가 매개변수로 주어질 때, 크레인을 모두 작동시킨 후 터트려져 사라진 인형의 개수를 return 하도록 solution 함수를 완성해주세요.

##### **[제한사항]**

* board 배열은 2차원 배열로 크기는 5 x 5 이상 30 x 30 이하입니다.
* board의 각 칸에는 0 이상 100 이하인 정수가 담겨있습니다.
  + 0은 빈 칸을 나타냅니다.
  + 1 ~ 100의 각 숫자는 각기 다른 인형의 모양을 의미하며 같은 숫자는 같은 모양의 인형을 나타냅니다.
* moves 배열의 크기는 1 이상 1,000 이하입니다.
* moves 배열 각 원소들의 값은 1 이상이며 board 배열의 가로 크기 이하인 자연수입니다.

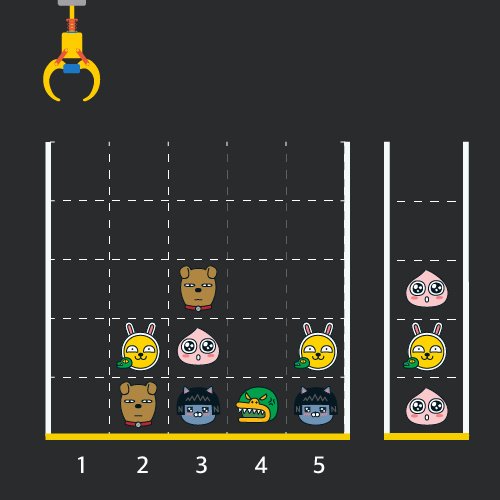
##### **입출력 예**

| **board** | **moves** | **result** |
| --- | --- | --- |
| [[0,0,0,0,0],[0,0,1,0,3],[0,2,5,0,1],[4,2,4,4,2],[3,5,1,3,1]] | [1,5,3,5,1,2,1,4] | 4 |

##### **입출력 예에 대한 설명**

**입출력 예 #1**

인형의 처음 상태는 문제에 주어진 예시와 같습니다. 크레인이 [1, 5, 3, 5, 1, 2, 1, 4] 번 위치에서 차례대로 인형을 집어서 바구니에 옮겨 담은 후, 상태는 아래 그림과 같으며 바구니에 담는 과정에서 터트려져 사라진 인형은 4개 입니다.



* 내 답안

#include <string>

#include <vector>

#include <stack>

#include <iostream>

using namespace std;

int solution(vector<vector<int>> board, vector<int> moves) {

int answer = 0;

stack<int> r;

int n = board[0].size();

vector<stack<int>> s(n);

for(int i=0;i<n;i++)

for(int j=0;j<n;j++) {

if(board[n-1-j][i])

s[i].push(board[n-1-j][i]);

}

int r\_top=0;

for(auto i: moves)

if(!s[i-1].empty()) {

if(!r.empty())

r\_top = r.top();

r.push(s[i-1].top());

s[i-1].pop();

if(r\_top == r.top()) {

r.pop();

r.pop();

answer++;

}

}

return answer\*2;

}

1. 여기서 첫번째로 배열을 transpose 하는데 O(n^2)이외의 시간이 들고 싶었으나 그런 방법은 찾지 못하였다.
2. 스택 배열을 만들기 위해 vector<stack<int>> s(n) 와 같이 선언하였다.

#이차원 벡터를 만들기 위해서는 다음과 같이 선언한다

vector<vector<int>> board\_t(10,vector<int>(10,0));

다른 풀이

#include <string>

#include <vector>

#include <stack>

using namespace std;

int solution(vector<vector<int>> board, vector<int> moves) {

int answer = 0;

stack<int> s;

for(int i=0;i<moves.size();i++)

{

int check = moves[i] - 1;

for(int j=0;j<board.size();j++)

{

if(board[j][check] != 0)

{

if(!s.empty() && s.top() == board[j][check])

{

s.pop();

answer += 2;

}

else

s.push(board[j][check]);

board[j][check] = 0;

break;

}

}

}

return answer;

}

moves 크기가 m 이라고 한다면 시간복잡도는 O(mn)이 된다. m이 n보다 작은경우에 더 효율적일 것 같다.

1. 두 개 뽑아서 더하기

* 두 개 뽑아서 더하기

darklight

sublimevimemacs

C++

###### 문제 설명

정수 배열 numbers가 주어집니다. numbers에서 서로 다른 인덱스에 있는 두 개의 수를 뽑아 더해서 만들 수 있는 모든 수를 배열에 오름차순으로 담아 return 하도록 solution 함수를 완성해주세요.

* 내 답

#include <string>

#include <vector>

#include <set>

using namespace std;

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

vector<int> answer;

set<int> s;

for(int i=0;i<numbers.size();i++)

for(int j=i+1;j<numbers.size();j++)

s.insert(numbers[i]+numbers[j]);

set<int>::iterator iter;

for(iter=s.begin();iter!=s.end();iter++)

answer.push\_back(\*iter);

return answer;

}

iter 쓰는 방법 알아두기!

저렇게 말고 그냥

answer.assign(s.begin(), s.end());

다음과 같이 해줄 수 있음.

1. 주식 가격문제 설명

초 단위로 기록된 주식가격이 담긴 배열 prices가 매개변수로 주어질 때, 가격이 떨어지지 않은 기간은 몇 초인지를 return 하도록 solution 함수를 완성하세요.

##### **제한사항**

* prices의 각 가격은 1 이상 10,000 이하인 자연수입니다.
* prices의 길이는 2 이상 100,000 이하입니다.

##### **입출력 예**

| **prices** | **return** |
| --- | --- |
| [1, 2, 3, 2, 3] | [4, 3, 1, 1, 0] |

##### **입출력 예 설명**

* 1초 시점의 ₩1은 끝까지 가격이 떨어지지 않았습니다.
* 2초 시점의 ₩2은 끝까지 가격이 떨어지지 않았습니다.
* 3초 시점의 ₩3은 1초뒤에 가격이 떨어집니다. 따라서 1초간 가격이 떨어지지 않은 것으로 봅니다.
* 4초 시점의 ₩2은 1초간 가격이 떨어지지 않았습니다.
* 5초 시점의 ₩3은 0초간 가격이 떨어지지 않았습니다.
* 내 답

#include <string>

#include <vector>

#include <stack>

using namespace std;

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

int n = prices.size();

vector<int> answer;

vector<stack<int>> sa(n);

int num=0;

for(int i=0;i<n;i++) {

int j=i+1;

for(;j<n;j++) {

if(prices[i]>prices[j])

break;

sa[i].push(prices[j]);

}

num = sa[i].size()+1;

if(j==n)

num-=1;

answer.push\_back(num);

}

return answer;

}

* Best 답 (1)

#include <string>

#include <vector>

#include <stack>

using namespace std;

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

vector<int> answer(prices.size());

stack<int> s;

int size = prices.size();

for(int i=0;i<size;i++){

while(!s.empty()&&prices[s.top()]>prices[i]){

answer[s.top()] = i-s.top();

s.pop();

}

s.push(i);

}

while(!s.empty()){

answer[s.top()] = size-s.top()-1;

s.pop();

}

return answer;

}

스택에 인덱스를 저장하고, 그 인덱스를 이용하여 prices 값에 접근하며

스택을 관리한다. 상당히 좋은 접근 법이다.

* 다른 답 (2)

#include <string>

#include <vector>

using namespace std;

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

vector<int> answer;

int size=prices.size();

for(int i=0; i<size; i++){

int time=0;

for(int j=i+1; j<size; j++){

time++;

if(prices[j]<prices[i]||j==size-1){ //마지막에서 두번째 값은 만약 마지막 값이 더 크다면 push할 수가 없기때문에 조건에 추가

answer.push\_back(time);

break;

}

}

}

answer.push\_back(0); //마지막은 시간이 없음

return answer;

}

굳이 스택을 안쓰고, 이런식으로 풀어줄 수 도 있다.

1. 기능개발

###### 문제 설명

프로그래머스 팀에서는 기능 개선 작업을 수행 중입니다. 각 기능은 진도가 100%일 때 서비스에 반영할 수 있습니다.

또, 각 기능의 개발속도는 모두 다르기 때문에 뒤에 있는 기능이 앞에 있는 기능보다 먼저 개발될 수 있고, 이때 뒤에 있는 기능은 앞에 있는 기능이 배포될 때 함께 배포됩니다.

먼저 배포되어야 하는 순서대로 작업의 진도가 적힌 정수 배열 progresses와 각 작업의 개발 속도가 적힌 정수 배열 speeds가 주어질 때 각 배포마다 몇 개의 기능이 배포되는지를 return 하도록 solution 함수를 완성하세요.

##### **제한 사항**

* 작업의 개수(progresses, speeds배열의 길이)는 100개 이하입니다.
* 작업 진도는 100 미만의 자연수입니다.
* 작업 속도는 100 이하의 자연수입니다.
* 배포는 하루에 한 번만 할 수 있으며, 하루의 끝에 이루어진다고 가정합니다. 예를 들어 진도율이 95%인 작업의 개발 속도가 하루에 4%라면 배포는 2일 뒤에 이루어집니다.

##### **입출력 예**

| **progresses** | **speeds** | **return** |
| --- | --- | --- |
| [93, 30, 55] | [1, 30, 5] | [2, 1] |
| [95, 90, 99, 99, 80, 99] | [1, 1, 1, 1, 1, 1] | [1, 3, 2] |

##### **입출력 예 설명**

입출력 예 #1  
첫 번째 기능은 93% 완료되어 있고 하루에 1%씩 작업이 가능하므로 7일간 작업 후 배포가 가능합니다.  
두 번째 기능은 30%가 완료되어 있고 하루에 30%씩 작업이 가능하므로 3일간 작업 후 배포가 가능합니다. 하지만 이전 첫 번째 기능이 아직 완성된 상태가 아니기 때문에 첫 번째 기능이 배포되는 7일째 배포됩니다.  
세 번째 기능은 55%가 완료되어 있고 하루에 5%씩 작업이 가능하므로 9일간 작업 후 배포가 가능합니다.

따라서 7일째에 2개의 기능, 9일째에 1개의 기능이 배포됩니다.

입출력 예 #2  
모든 기능이 하루에 1%씩 작업이 가능하므로, 작업이 끝나기까지 남은 일수는 각각 5일, 10일, 1일, 1일, 20일, 1일입니다. 어떤 기능이 먼저 완성되었더라도 앞에 있는 모든 기능이 완성되지 않으면 배포가 불가능합니다.

따라서 5일째에 1개의 기능, 10일째에 3개의 기능, 20일째에 2개의 기능이 배포됩니다.

* 내 답

#include <string>

#include <vector>

#include <queue>

#include <iostream>

using namespace std;

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

vector<int> answer;

int size = progresses.size();

queue<int> price\_q;

queue<int> speeds\_q;

int i=0;

price\_q.push(progresses[i]);

speeds\_q.push(speeds[i]);

while(!price\_q.empty()) {

int count = 0;

int days = ((100-price\_q.front()) / speeds\_q.front()) + bool((100-price\_q.front()) % speeds\_q.front());

while(price\_q.front()+days\*speeds\_q.front()>=100) {

count++;

i++;

price\_q.pop();

speeds\_q.pop();

if(i==size)

break;

price\_q.push(progresses[i]);

speeds\_q.push(speeds[i]);

}

answer.push\_back(count);

}

return answer;

}

어차피 전의 것이 완료가 되어야 뒤의 것이 배포가 될 수 있으므로 전체를 보지 않고 바로 전에것만 봐주면서 코드를 진행하였다. 따라서 O(n) 시간에 작동하는 알고리즘을 작성하였다.

* 남의 답(Best)

#include <string>

#include <vector>

#include <iostream>

using namespace std;

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

vector<int> answer;

int day;

int max\_day = 0;

for (int i = 0; i < progresses.size(); ++i)

{

day = (99 - progresses[i]) / speeds[i] + 1;

if (answer.empty() || max\_day < day)

answer.push\_back(1);

else

++answer.back();

if (max\_day < day)

max\_day = day;

}

return answer;

}

감탄이 나오는 알고리즘이다. vector.back() 은 마지막 원소를 참조하는 함수인데, 완료 시 까지 필요한 day 를 기준으로 max\_day 보다 작으면 추가 될 수 있으므로 answer에 push해준 마지막 원소를 더해주고, max\_day 보다 더 필요한 경우에는 새로 1을 추가하면서 갱신해준다.

O(n)에 작동하고 훨씬 깔끔한 알고리즘이다. 그리고 day를 구하는 경우에도 아예 99를 기준으로 나눈 다음에 1을 더해주는게 더 깔끔했다.