



ACM Craft - 1005번

ACM Craft 성공



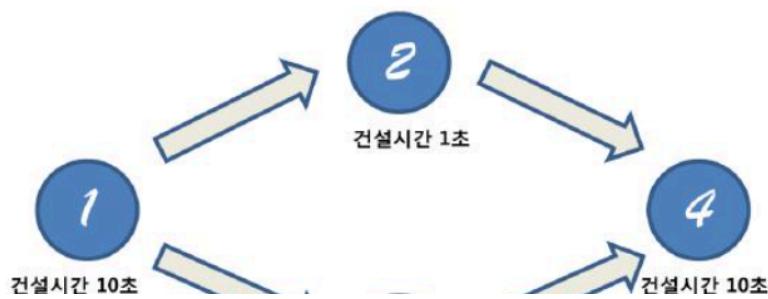
3 골드 III

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	512 MB	87914	25933	18119	30.441%

문제

서기 2012년! 드디어 2년간 수많은 국민들을 기다리게 한 게임 ACM Craft (Association of Construction Manager Craft)가 발매되었다.

이 게임은 지금까지 나온 게임들과는 다르게 ACM크래프트는 다이나믹한 게임 진행을 위해 건물을 짓는 순서가 정해져 있지 않다. 즉, 첫 번째 게임과 두 번째 게임이 건물을 짓는 순서가 다를 수도 있다. 매 게임시작 시 건물을 짓는 순서가 주어진다. 또한 모든 건물은 각각 건설을 시작하여 완성이 될 때까지 Delay가 존재한다.



트리 하나를 예시로 보자

```
graph TD
    a((1))
    b((2))
    c((3))
    a --> b
    a --> c
```

```
style a fill:#6afDCF,stroke:#333,stroke-width:2px
```

방향 비순환 그래프에서 모든 노드의 부모 노드는 최대 1개이다. 그러면 자식 노드로부터 위로 올라가면 될 것 같다.

DFS를 통해서 자식 노드에서부터 올라가서 값을 계산하자.

이때, 값은 DP를 이용해서 가장 위 부모 노드부터 값을 누적하면서 구해보자.

```
import io, os, sys
input = io.BytesIO(os.read(0, os.fstat(0).st_size)).readline
sys.setrecursionlimit(100000)
def sol():
    n,k = map(int,input().split())
    t = [0]+list(map(int,input().split()))
    g = [[] for _ in range(n+1)]
    for _ in range(k):
        a,b = map(int,input().split())
        g[b].append(a)
    dp = [-1]*(n+1)
    def search(now):
        if dp[now] != -1:
            return dp[now]
        dp[now] = t[now]
        if g[now]:dp[now] += max(search(i) for i in g[now])
        return dp[now]
    print(search(int(input())))
for _ in range(int(input())): sol()
```

<내 코드가 아닌 다른 사람 코드다>

또 다른 방법으로는 bfs로 하는 것인데 이건 위상정렬을 알아야 한다. 사실 DFS도 위상 정렬의 한 종류이지만, 발상으로도 가능하다면, 이건 진짜 위상 정렬을 알아야 한다.

자세한 것은 링크로 확인해보고,

♡ 위상정렬

```
from collections import deque
import sys
input = sys.stdin.readline

T = int(input())

for _ in range(T):
    N,K = map(int,input().split())
    D = list(map(int,input().split()))
    graph = [[] for _ in range(N+1)]
    indegree = [0] * (N+1)

    for i in range(K):
        X,Y = map(int,input().split())
        graph[X].append(Y)
        indegree[Y] += 1

    start = int(input())

    def topological_sort():
        dp = [0] * (N+5)
        q = deque()

        for i in range(1, N+1):
```

```
if indegree[i] == 0:  
    q.append(i)  
    dp[i] = D[i-1]  
  
while q:  
    now = q.popleft()  
    for i in graph[now]:  
        indegree[i] -= 1  
        dp[i] = max(dp[i],dp[now]+D[i-1])  
  
    if indegree[i] == 0:  
        q.append(i)  
  
return dp[start]  
  
print(topological_sort())  
  
#10 20 15 8 7 1 43
```



스터디 1주차 보고서

(3/15 ~ 3/22)

작성자 : 이강민

작성일 : 2024.03.22

문제 풀이 (BOJ 10562 나이트)

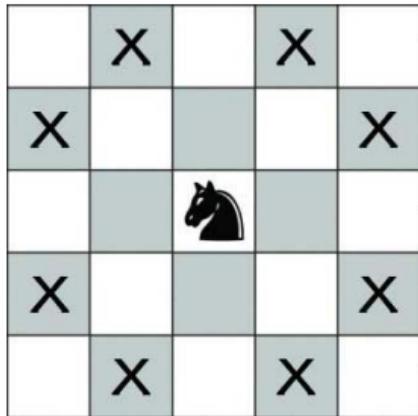
(문제 분석 및 프로그램 설계)

문제설명

<https://www.acmicpc.net/problem/10562>

M행 N열 크기의 체스판 위에 나이트를 놓으려고 한다. 각각의 칸에는 최대 1개의 나이트가 놓여질 수 있다.

이때, 체스판 위에 있는 나이트가 서로 공격을 할 수 있으면 안 된다. 나이트가 놓여져 있을 때, 공격할 수 있는 칸의 위치는 아래 그림에 X로 표시되어 있다.



체스판의 크기가 주어졌을 때, 나이트를 놓을 수 있는 방법의 수를 구하는 프로그램을 작성하시오.

문제 분석

읽어보면 N-Queen의 나이트버전과 유사하다. 특징으로는 M이 4이하이고 N은 1e9 이하라는 것이다. 즉 어떻게 이전 상황을 잘 활용해 dp를 사용할 수 있어보인다.

이 문제는 M이 4 이하 이기 때문에, 이전 두칸의 상황에 대해서 모두 전파가 가능하고, $O(N \times 8^M)$ 인 비트마스크로 방문정보를 처리하면서 풀 수 있다. 또한 이러한 방식에서 행렬곱으로 시간을 줄이면 충분히 **60초**라는 시간제한 안에 들어오게 된다.

(문제를 해결하는데 필요한 개념)

Berlekamp-Massey 알고리즘, 비스마스킹DP

<https://koosaga.com/231>

(소스 코드)

CPP

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

#define ll long long
#define X first
#define Y second
#define pii pair<int,int>
#define pll pair<ll,ll>

const int mod = 1000000009;

// 구사과의 벌캠 템플릿 : https://gist.github.com/koosaga/d4afc4434dbaa348d5bef0d60ac36aa4

ll ipow(ll x, ll p){
    ll ret = 1, piv = x;
    while(p){
        if(p & 1) ret = ret * piv % mod;
        piv = piv * piv % mod;
        p >>= 1;
    }
    return ret;
}

vector<int> berlekamp_massey(vector<int> x){
    vector<int> ls, cur;
    int lf, ld;
    for(int i = 0; i < x.size(); i++){
        ll t = 0;
        for(int j = 0; j < cur.size(); j++){
            t = (t + 1ll * x[i - j - 1] * cur[j]) % mod;
        }
        if((t - x[i]) % mod == 0) continue;
        if(cur.empty()){
            cur.resize(i + 1);
            lf = i;
            ld = (t - x[i]) % mod;
            continue;
        }
        cur[lf] = t;
        lf++;
        ld = (t - x[i]) % mod;
    }
}
```

```

    }

    ll k = -(x[i] - t) * ipow(lf, mod - 2) % mod;
    vector<int> c(i - lf - 1);
    c.push_back(k);
    for(auto &j : ls) c.push_back(-j * k % mod);
    if(c.size() < cur.size()) c.resize(cur.size());
    for(int j = 0; j < cur.size(); j++){
        c[j] = (c[j] + cur[j]) % mod;
    }
    if(i - lf + (int)ls.size() >= (int)cur.size()){
        tie(ls, lf, lf) = make_tuple(cur, i, (t - x[i]) % mod);
    }
    cur = c;
}

for(auto &i : cur) i = (i % mod + mod) % mod;
return cur;
}

int get_nth(vector<int> rec, vector<int> dp, ll n){
    int m = rec.size();
    vector<int> s(m), t(m);
    s[0] = 1;
    if(m != 1) t[1] = 1;
    else t[0] = rec[0];
    auto mul = [&rec](vector<int> v, vector<int> w){
        int m = v.size();
        vector<int> t(2 * m);
        for(int j = 0; j < m; j++){
            for(int k = 0; k < m; k++){
                t[j + k] += 1ll * v[j] * w[k] % mod;
                if(t[j + k] >= mod) t[j + k] -= mod;
            }
        }
        for(int j = 2 * m - 1; j >= m; j--){
            for(int k = 1; k <= m; k++){
                t[j - k] += 1ll * t[j] * rec[k - 1] % mod;
                if(t[j - k] >= mod) t[j - k] -= mod;
            }
        }
        t.resize(m);
        return t;
    };
    while(n){
        if(n & 1) s = mul(s, t);
        t = mul(t, t);
        n >>= 1;
    }
}

```

```

ll ret = 0;
for(int i = 0; i < m; i++) ret += 1ll * s[i] * dp[i] % mod;
return ret % mod;
}

int guess_nth_term(vector<int> x, ll n){
    if(n < x.size()) return x[n];
    vector<int> v = berlekamp_massey(x);
    if(v.empty()) return 0;
    return get_nth(v, x, n);
}

struct elem { int x, y, v; }; // A_(x, y) <- v, 0-based. no duplicate please..

vector<int> get_min_poly(int n, vector<elem> M){
    // smallest poly P such that A^i = sum_{j < i} {A^j * P_j}
    vector<int> rnd1, rnd2;
    mt19937 rng(0x14004);
    auto randint = [&rng](int lb, int ub){
        return uniform_int_distribution<int>(lb, ub)(rng);
    };
    for(int i = 0; i < n; i++){
        rnd1.push_back(randint(1, mod - 1));
        rnd2.push_back(randint(1, mod - 1));
    }
    vector<int> gobs;
    for(int i = 0; i < 2 * n + 2; i++){
        int tmp = 0;
        for(int j = 0; j < n; j++){
            tmp += 1ll * rnd2[j] * rnd1[j] % mod;
            if(tmp >= mod) tmp -= mod;
        }
        gobs.push_back(tmp);
        vector<int> nxt(n);
        for(auto &i : M){
            nxt[i.x] += 1ll * i.v * rnd1[i.y] % mod;
            if(nxt[i.x] >= mod) nxt[i.x] -= mod;
        }
        rnd1 = nxt;
    }
    auto sol = berlekamp_massey(gobs);
    reverse(sol.begin(), sol.end());
    return sol;
}

ll det(int n, vector<elem> M){
    vector<int> rnd;

```

```

mt19937 rng(0x14004);
auto randint = [&rng](int lb, int ub){
    return uniform_int_distribution<int>(lb, ub)(rng);
};
for(int i = 0; i < n; i++) rnd.push_back(randint(1, mod - 1));
for(auto &i : M){
    i.v = 1ll * i.v * rnd[i.y] % mod;
}
auto sol = get_min_poly(n, M)[0];
if(n % 2 == 0) sol = mod - sol;
for(auto &i : rnd) sol = 1ll * sol * ipow(i, mod - 2) % mod;
return sol;
}

// 여기까지가 템플릿임

void solve() {
    int M;
    ll N;
    cin >> M >> N;

    if(N == 0){
        cout << 1 << '\n';
        return;
    }
    if(N == 1){
        // 어떤 칸이든 선택 가능
        int ans = (1 << M) % mod;
        cout << ans << '\n';
        return;
    }

    // 모든 한 열의 배치
    vector<int> masks;
    for(int mask = 0; mask < (1 << M); mask++){
        masks.push_back(mask);
    }

    auto validAdj = [=](int a, int b) -> bool {
        int lim = (1 << M) - 1;
        // a에서 좌우로 2칸 이동한 자리와 b가 겹치면 안됨
        if( ((a << 2) & lim) & b ) return false;
        if( (a >> 2) & b ) return false;
        return true;
    };

    // 두 열 차이가 2인 경우
}

```

```
auto validTwogap = [=](int a, int b) -> bool {
    int lim = (1 << M) - 1;
    if( ((a << 1) & lim) & b ) return false;
    if( (a >> 1) & b ) return false;
    return true;
};

struct State { int a, b; };
vector<State> states;
for(auto a : masks){
    for(auto b : masks){
        if(validAdj(a, b)){
            states.push_back({a, b});
        }
    }
}
int S = states.size();

vector<vector<int>> trans(S);
for (int i = 0; i < S; i++){
    int a = states[i].a, b = states[i].b;
    for (int j = 0; j < S; j++){
        // 상태 j 는 (b, c)여야 함
        if(states[j].a != b) continue;
        int c = states[j].b;
        if(!validAdj(b, c)) continue;
        if(!validTwogap(a, c)) continue;
        trans[i].push_back(j);
    }
}

// f(n):= M 행 n 열 체스판에 대해 나이트를 놓는 경우의 수
vector<int> seq;
// seq[0] = f(1), seq[1] = f(2)
int f1 = (1 << M) % mod;
seq.push_back(f1);
seq.push_back(S % mod);

// dp:= 체스판 열이 2 개일 때, 각 상태의 경우의 수
vector<int> dp(S, 1);
// 충분한 초기값의 개수
int L = max(2 * S, 50);
for (int n = 3; n <= L; n++){
    vector<int> ndp(S, 0);
    for (int i = 0; i < S; i++){
        if(dp[i] == 0) continue;
        for (int j : trans[i]){
            ndp[j] += dp[i];
        }
    }
    dp = ndp;
}
```

```

        ndp[j] = (ndp[j] + dp[i]) % mod;
    }
}
dp = ndp;
ll tot = 0;
for (int x : dp){
    tot = (tot + x) % mod;
}
seq.push_back((int) tot);
}

// seq[k] = f(k+1)
// N >= 2 면 f(N) = seq[N-1]
int ans = 0;
if(N <= (int)seq.size()){
    ans = seq[N - 1];
} else {
    // 벌캡 돌리기
    ans = guess_nth_term(seq, N - 1) % mod;
}
cout << ans % mod << '\n';

}

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

    int Tc = 1;
    cin >> Tc;
    for(int tt = 1; tt <= Tc; tt++) {
        solve();
    }
    return 0;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

일단 이번에 이 문제는 Berlekamp-Massey 알고리즘을 공부하기 위해 잡은 문제이다.

Berlekamp-Massey 알고리즘은 특정한 DP의 점화식을 찾아주는 알고리즘이다.

이걸 사용하면 초기 몇 개의 항만으로 이후의 값이 바로 계산된다.

증명은 매우 복잡하기 때문에 생략했다.

만약 증명이 궁금하다면 아래 키워드들을 모두 공부한 뒤에 시도해보는 것이 좋다...

(생성 함수, 행렬 거듭제곱NlogN, 케일리-해밀턴 정리, 키타마사법)

사용하는것만으로도 큰 의미가 있으니 그냥 쓰면된다.

아래 koosaga님의 템플릿을 사용하였다.

<https://gist.github.com/koosaga/d4afc4434dbaa348d5bef0d60ac36aa4>

우리는 문제상황에서 MOD값이 $1e9 + 7$ 이므로 이것을 변경해주자

우리는 이제 이 dp에서 전파되는 21가지의 경우의 수를 모두 처리해주면 된다. 실제로 나이트가 있을 수 있는 경우는

1. 같은 열에 있을 때
2. 차이가 2인 열에 있을 때

다음 두가지를 통해서 case_work를 해주면 된다.

(실제로 그려보면 대칭꼴이 많이 나오고 되는경우가 많이 없어서 금방구해진다.)

이렇게 초기에 대략 50개를 구해주고, guess_nth_term를 돌려주면 벌래캠프-메시가 알아서 n번째 항을 구해줄것이다.

만약 벌래캠프 메시를 쓰지 않는다면, 이전 상황에서 전파되는 것을 행렬곱으로 변환하여 NlogN으로 행렬곱을 하며 전파하는 방법도 있다고 하나, 정확한 풀이는 찾지 못했다.

이러한 어렵고 복잡한 풀이에 비해서 벌래캠프-메시 알고리즘이 참 유용한듯하다.

물론 해당 DP 점화식이 선형점화식인 것을 판별한 뒤에 적용해야한다는 것이 문제다.

그래서 어떻게 점화식이 선형인지를 빠르게 판별하는지 그 테크닉이 궁금해져 좀 더 찾아봐야할듯하다.



스터디 1주차 보고서

(3/15~3/22)

작성자:이유나

작성일:2025.03.21

문제 풀이 (2744번 대소문자 바꾸기)

(문제 분석 및 프로그램 설계)

영어 소문자와 대문자로 이루어진 단어를 입력 받은 뒤, 대문자는 소문자로, 소문자는 대문자로 바꾸어 출력한다. 단, 단어의 길이는 최대 100이다.

(문제를 해결하는데 필요한 개념)

c언어와 다르게 c++은 string이 있기 때문에 영어 단어를 입력 받을 때 char배열이 아닌 string변수를 사용한다.

(소스코드)

cpp

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

int main() {
    string sWord;
    cin >> sWord;
    if (sWord.length() <= 100) {
        for (int i = 0; i < sWord.length(); i++) {
            if (isupper(sWord[i])) {
                sWord[i] = tolower(sWord[i]);
            }
            else if (islower(sWord[i])) {
                sWord[i] = toupper(sWord[i]);
            }
            else if (!isalpha(sWord[i])) {
                cout << "잘못된 단어입니다. 다시 입력하세요.";
                break;
            }
        }
        cout << sWord;
    }
}
```

```
else {
    cout << "잘못된 단어입니다. 다시 입력하세요.";
}
return 0;
}
```

(코드 설명)

영어 단어를 입력 받은 후 ①단어의 길이가 100 이하인지. ②영어 단어가 대문자 인지 소문자인지. ③영어 단어가 아닌 다른 단어가 들어가 있는지. 이 3가지를 판단한 후 모두 참이면 대소문자를 바꾼 sWord를 다시 출력한다.

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

처음에는 sWord[i] == NULL일때까지 반복하면 된다고 생각했는데, 컴파일은 되지만 틀렸다고 떠서 왜 그런가 고민해보았더니 만약 C의 char배열이었다면, 문자열을 입력할 때 마지막에 \0이 자동으로 들어가기 때문에 “\0까지 반복한다”라는 논리가 맞았겠지만, 여기서는 C++ string변수를 사용했기 때문에 “string의 길이만큼 반복한다”라는 논리를 사용해야 한다는 사실을 깨달았다.

아직 C언어와 C++이 많이 헷갈리고, 방학 동안 공부를 하지 않아 기억이 안 나는 부분이 많았다. 빨리 백준 문제를 많이 풀면서 기억을 되살리고, 실력을 향상시키고 싶다.



스터디 1주차 보고서

(3/16 ~ 3/22)

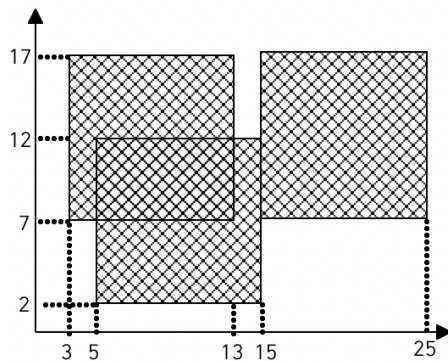
작성자 : 김영민

작성일 : 2025.03.21

문제 풀이 (BOJ 2563 색종이)

(문제 분석 및 프로그램 설계)

가로, 세로의 크기가 각각 100인 정사각형 모양의 흰색 도화지가 있다. 이 도화지 위에 가로, 세로의 크기가 각각 10인 정사각형 모양의 검은색 색종이를 색종이의 변과 도화지의 변이 평행하도록 붙인다. 이러한 방식으로 색종이를 한 장 또는 여러 장 붙인 후 색종이가 붙은 영역의 넓이를 구하자



(문제를 해결하는데 필요한 개념)

2차원 배열을 초기화 하고 배열 값을 수정하는 방법

(소스 코드)

```
C
#pragma warning(disable:4996)

#include <stdio.h>

int main(void)
{
    int paper[100][100];

    for (int i = 0; i < 100; i++)
    {
        for (int j = 0; j < 100; j++)
        {
            paper[i][j] = 0; // [100][100]행렬의 값을 전부 0으로 초기화
        }
    }
}
```

```
}

int num; //색종이의 수

int X; //색종이의 좌하단 꼭짓점의 x좌표

int Y; //색종이의 좌하단 꼭짓점의 y좌표

int count = 0; //색종이가 차지하고 있는 넓이에 대한 변수 초기값은 0

scanf("%d", &num); //색종이의 수를 결정

for (int z = 0; z < num; z++) //색종이의 수에 따라 아래의 for문을 반복할 횟수를 결정

{

    scanf("%d", &X); //색종이의 좌하단 꼭짓점의 x좌표를 결정

    scanf("%d", &Y); //색종이의 좌하단 꼭짓점의 y좌표를 결정

    for (int a = X; a < (X + 10); a++)

    {

        for (int b = Y; b < (Y + 10); b++)

        {

            paper[a][b] = 1; //색종이의 좌하단 꼭짓점 (x,y)좌표에서 우상단 꼭짓점

(x+10,y+10)를 지정하고 내부를 1로 변경

        }

    }

}

for (int c = 0; c < 100; c++)

{

    for (int d = 0; d < 100; d++)

    {

        if (paper[c][d] != 0)
```

```
{  
    count++; //배열의 값을 검사하면서 0이 아니면 넓이를 차지하고 있  
    는것으로 간주하고 1씩 더해나감  
}  
}  
  
printf("%d", count); //넓이 출력  
  
return 0;  
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

문제에서 주어진 넓이라는 단위를 어떻게 표현할지 생각해 보았다. 넓이라는 단위는 1×1 의 단위 넓이의 총 합으로 변환 할 수 있으므로 배열의 메모리 한 칸을 단위 넓이로 치환하여 내부를 채우는 방식으로 문제를 전환할 수 있었다. 따라서 배열을 좌표로 생각하고 좌하단과 우상단을 지정하고 그 사이를 전부 1로 채움으로서 배열의 값을 넓이로 표현할 수 있었다.

처음 문제를 봤을때는 좌표라는 개념을 구현하는 방법이 무엇이 있을지 고민하다. 2차원 배열의 개념을 떠올리고 2차원 배열과 좌표의 구조의 유사성을 떠올렸다. 이처럼 연산만이 아닌 기하적인 문제를 수학적 개념을 통해 문제를 보는 관점을 바꾸는 것의 중요함을 느꼈다.

또한 자바에 대한 기초적인 문법공부가 완료되면 같은 문제를 자바로 풀어보면서 추가적인 언어 공부를 이어나갈 것이다.



스터디 5주차 보고서

(5/10~ 5/17)

작성자 : 박가령

작성일 : 2025.05.17

#2908 상수

문제

상근이의 동생 상수는 수학을 정말 못한다. 상수는 숫자를 읽는데 문제가 있다. 이렇게 수학을 못하는 상수를 위해서 상근이는 수의 크기를 비교하는 문제를 내주었다. 상근이는 세 자리 수 두 개를 칠판에 써주었다. 그 다음에 크기가 큰 수를 말해보라고 했다.

상수는 수를 다른 사람과 다르게 거꾸로 읽는다. 예를 들어, 734와 893을 칠판에 적었다면, 상수는 이 수를 437과 398로 읽는다. 따라서, 상수는 두 수중 큰 수인 437을 큰 수라고 말할 것이다.

두 수가 주어졌을 때, 상수의 대답을 출력하는 프로그램을 작성하시오.

풀이과정

1. 변수 - 숫자 2개를 저장할 배열
2. 입력 - 숫자 2개를 입력
3. 처리 - 숫자를 뒤집기
4. 출력 - strcmp 함수를 이용해서 큰 수를 출력

코드

```
#include <stdio.h>
#include <string.h>

int main()
{
    char str[4];
    char str1[4];

    scanf("%s %s",str, str1);

    char s;
    s=str[0];
    str[0]=str[2];
    str[2]=s;

    s=str1[0];
    str1[0]=str1[2];
    str1[2]=s;
```

```
if(strcmp(str,str1)>0) printf("%s",str);
else printf("%s",str1);

return 0;

}
```

알게된 점

하나씩 비교하면서 크면 출력하고 반복을 중단 크지도 작지도 않다면 다음 반복을 continue 하는 방법이 있

```
#include <stdio.h>
```

```
void swap(char * a, char * b)
{
    char v=*a;
    *a=*b;
    *b=v;
}
```

```
void print(char a[],int size)
```

```
{
    for(int i=0;i<size;i++)
        printf("%c",a[i]);
}
```

```
int main()
```

```
{
    char str1[4];
    char str2[4];
```

```
scanf("%s %s",str1, str2);
```

```
swap(&str1[0],&str1[2]);
```

```
swap(&str2[0],&str2[2]);\n\nfor(int i=0;i<3;i+ +)\n{\n    if(str1[i]>str2[i]){\n        print(str1,3);\n        return 0;\n    }\n    else if(str1[i]<str2[i])\n    { printf(str2,3);\n        return 0;}\n\n    else\n        continue;\n}\n\nreturn 0;\n}
```

문자열 배열을 만들어서 저장할 수도 있다.

문자열은 널문자 저장을 위해 하나 더 배열의 크기를 설정해야한다.

```
#include <stdio.h>\n#include <string.h>\n\nint main()\n{\n    char str[2][3];\n    scanf("%s %s",str[0],str[1]);\n\n    for(int i=0;i<2;i+ +)\n    {\n        char v=str[i][0];\n
```

```
str[i][0]=str[i][2];
str[i][2]=v;
}

printf("%s",(strcmp(str[0],str[1])>0)?str[0]:str[1]);
return 0;
}
```

#5622 다이얼

문제

상근이의 할머니는 아래 그림과 같이 오래된 다이얼 전화기를 사용한다.



전화를 걸고 싶은 번호가 있다면, 숫자를 하나를 누른 다음에 금속 펀이 있는 곳 까지 시계방향으로 돌려야 한다. 숫자를 하나 누르면 다이얼이 처음 위치로 돌아가고, 다음 숫자를 누르려면 다이얼을 처음 위치에서 다시 돌려야 한다.

숫자 1을 걸려면 총 2초가 필요하다. 1보다 큰 수를 거는데 걸리는 시간은 이보다 더 걸리며, 한 칸 옆에 있는 숫자를 걸기 위해선 1초씩 더 걸린다.

상근이의 할머니는 전화 번호를 각 숫자에 해당하는 문자로 외운다. 즉, 어떤 단어를 걸 때, 각 알파벳에 해당하는 숫자를 걸면 된다. 예를 들어, UNUCIC는 868242와 같다.

할머니가 외운 단어가 주어졌을 때, 이 전화를 걸기 위해서 필요한 최소 시간을 구하는 프로그램을 작성하시오.

풀이과정

1. 변수 - 문자열을 저장할 배열 / 걸린 시간을 저장할 정수형 변수

2. 입력 - 문자열을 입력
3. 처리 - 문자열을 0부터 시작하는 숫자로 변경 / 숫자의 범위에 따라서 전체 시간에 더함
4. 출력 - 전체시간을 출력

코드

```
#include <stdio.h>
```

```
int main()
{
    char str[16];
    int count=0;

    scanf("%s",str);
    int i=0;
    while(str[i]!='\0')

    {
        int a=str[i]-'A';

        if(a>=0&&a<=2) count+=3;

        if(a>=3&&a<=5) count+=4;

        if(a>=6&&a<=8) count+=5;

        if(a>=9&&a<=11) count+=6;

        if(a>=12&&a<=14) count+=7;

        if(a>=15&&a<=18) count+=8;

        if(a>=19&&a<=21) count+=9;

        if(a>=22&&a<=25) count+=10;

        i++;
    }
}
```

```
    printf("%d",count);

    return 0;

}
```

알게된 점

while 문 대신 배열의 크기만큼 반복할 수도 있고 / if 문 대신에 switch 문으로 케이스를 구분해서 풀 수도 있다.

```
#include <stdio.h>

#include <string.h>

int main()

{

    char str[16];

    scanf("%s",str);

    int count=0;

    for(int i=0;i<strlen(str);i+ +)

    {

        switch(str[i])

        {

            case 'A': case 'B' : case 'C':

                {count+=3;

                break;}

            case 'D': case 'E' : case 'F':

                {count+=4;

                break;}

            case 'G': case 'H' : case 'I':

                {count+=5;

                break;}

            case 'J': case 'K' : case 'L':

                {count+=6;

                break;}
```

```
case 'M': case 'N' : case 'O':  
{count+=7;  
break;}  
  
case 'P': case 'Q' : case 'R': case 'S':  
{count+=8;  
break;}  
  
case 'T': case 'U' : case 'V':  
{count+=9;  
break;}  
  
case 'W': case 'X' : case 'Y':case 'Z':  
{count+=10;  
break;}  
  
default:  
break;  
}  
  
}  
  
printf("%d",count);  
return 0;  
}
```

#11718 그대로 출력하기

문제

입력 받은 대로 출력하는 프로그램을 작성하시오.

풀이과정

1. 변수 - 행이 최대 100인 배열
2. 입력 - 배열에다가 문자열을 입력

코드

```
#include <stdio.h>
```

```
int main()
{
    char str[101];

    while(fgets(str, sizeof(str),stdin))
    {
        printf("%s",str);
    }
    return 0;
}
```

알게된 점

 설명

- fgets()는 줄 단위로 입력을 받고, Wn 포함해서 저장해 줌
- while (fgets(...))는 입력이 끝날 때까지(EOL, Ctrl+ D 또는 파일 끝) 반복
- printf("%s", line)으로 그대로 출력



스터디 1주차 보고서

(03/22)

작성자 : 우지민

작성일 : 2025.03.22

문제 풀이 (백준 1463번: 1로 만들기)

(문제 분석 및 프로그램 설계)

정수 X에 사용할 수 있는 연산은 다음과 같이 세 가지 이다.

1. X가 3으로 나누어 떨어지면, 3으로 나눈다.
2. X가 2로 나누어 떨어지면, 2로 나눈다.
3. 1을 뺀다.

정수 N이 주어졌을 때, 위와 같은 연산 세 개를 적절히 사용해서 1을 만들려고 한다. 연산을 사용하는 횟수의 최솟값을 출력하시오.

입력으로는 첫째 줄에 1보다 크거나 같고, 106보다 작거나 같은 정수 N이 주어진다.

(문제를 해결하는데 필요한 개념)

동적 계획법 설명에 사용하는 동전의 최소 개수로 목표 금액을 만드는 문제를 살짝 변형한 문제.. 동적 계획법을 알면 간단하게 해결 가능함

(소스 코드)

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

// 1주차 알고리즘 풀기
// 백준 1463, 1로 만들기
void baek_1463() {
    int N = 0;
    std::cin >> N;
    std::vector<int> calnumDP(N + 1, 0);

    for (int i = 2; i < N + 1; i++) { // 0, 1은 빼주기~

        calnumDP[i] = calnumDP[i - 1] + 1; // 모든 경우에 비교대상이니까..

        if (i % 2 == 0) {
            calnumDP[i] = std::min(calnumDP[i], calnumDP[i / 2] + 1);
        }

        if (i % 3 == 0) {
            calnumDP[i] = std::min(calnumDP[i], calnumDP[i / 3] + 1);
        }
    }
    std::cout << calnumDP[N];
}

int main()
```

```
{  
    baek_1463();  
    return 0;  
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

N+1 크기의 연산 수행 개수 기록 DP 테이블을 만들어서 0에서 N까지 만들어 가는 방향으로, DP 테이블을 채우면 된다.

즉 $dp[i] = \min\{ dp[i-1]+1, dp[i/2]+1, dp[i/3]+1 \}$. (단, $i/2$ $i/3$ 의 경우는 2랑 3으로 나누어 떨어지는 경우에만 해당되므로 if 조건을 걸어주면 좋을 듯)

실3문제 중에서 가장 많은 사람들이 푼 문제라 골랐음. 딱 보자마자 이건 DP로 풀어야 한다는 걸 알아서 문제 자체는 레벨이 더 낮은 문제들보다 쉬웠다. 그러나 그래서 어떻게 프로그래밍을 해야 하는 지는 생각나지가 않았고, 결국 DP 와 C++ 기본 문법부터 다시 공부해야 했었다.... 앞으로도 꾸준히 알고리즘 문제를 풀어야겠다.

캡스 스터디

1주차 보고서

작성자: 전정환
작성일: 2025.03.23

2557번

[제출](#) [맞힌 사람](#) [숏코딩](#) [채점 결과](#) [채점 현황](#) [내 제출](#) [난이도 기여](#) [질문 게시판](#)


Hello World 성공

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	128 MB	1320988	535015	367622	39.447%

문제

Hello World!를 출력하시오.

입력

없음

출력

Hello World!를 출력하시오.

예제 입력 1 복사

예제 출력 1 복사

Hello World!

알고리즘 분류

- 구현

메모

2557번

[제출](#)[맞힌 사람](#)[숏코딩](#)[채체점 결과](#)[채점 현황](#)[내 제출](#)[난이도 기여](#)[질문 게시판](#)

Hello World

언어

C++17

언어 설정

소스 코드 공개

 공개 비공개 맞았을 때만 공개

소스 코드

```

1 #include<stdio.h>
2 int main()
3 {
4     printf("Hello World!");
5 }
```

[제출](#)

문제 풀이 후 느낀점: 백준에 가입하고 처음 풀게 된 문제입니다. 고등학교 때 C언어를 잠깐 배운 기억이 있어, 기억을 더듬으며 풀이해보았습니다. 단순히 출력하는 문제이므로 `printf` 함수를 통해 출력하고자 하는 결과값을 얻었습니다.

1000번

제출

맞힌 사람

숏코딩

재채점 결과

채점 현황

내 제출

질문 게시판

A+B

다국어



한국어 ▾

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	128 MB	1252098	489122	332636	38.553%

문제

두 정수 A와 B를 입력받은 다음, A+B를 출력하는 프로그램을 작성하시오.

입력

첫째 줄에 A와 B가 주어진다. ($0 < A, B < 10$)

출력

첫째 줄에 A+B를 출력한다.

예제 입력 1 복사

1 2

예제 출력 1 복사

3

힌트

1000번

제출

맞힌 사람

숏코딩

재채점 결과

채점 현황

내 제출

난이도 기여

질문 게시판

A+B

언어

C++17

언어 설정

소스 코드 공개 공개
 비공개
 맞았을 때만 공개

소스 코드

```
1 #include<stdio.h>
2 int main()
3 {
4     int a, b;
5     scanf("%d %d", &a, &b);
6     printf("%d", a+b);
7 }
```

제출

문제 풀이 후 느낀점: 다음 문제는 정수의 간단한 연산들이었습니다. 따라서 이 부분도 어렵지 않게 풀어낼 수 있었습니다.

float과 double의 차이가 무엇이길래 이러는 걸까요?

1008번 - A/B

khg3031 9달 전

좋아요

float형으로 출력할 경우 틀리다고 뜨고, double을 쓰면 정답이라고 출력하는데 자료형의 어떤점으로 인해 발생하는 오류일까요?

두개의 소스코드는 아래와 같습니다.

```
#include <stdio.h>

int main(){
    int a, b;
    float div;

    scanf("%d %d", &a,&b);
    div = (float)a/(float)b;
    printf("%.12lf",div);
}

1 #include <stdio.h>
2
3 int main(){
4     int a, b;
5     double div;
6
7     scanf("%d %d", &a,&b);
8     div = (double)a/(double)b;
9     printf("%.12lf",div);
10 }
```

wizardrabbit 9달 전 10 4

좋아요

float는 매우 정확도가 떨어지는 실수 자료형입니다. 문제에서 요구하는 오차는 10^{-9} 입니다. 반면 double은 float보다는 훨씬 정밀하게 수를 저장할 수 있는 자료형입니다.

특별한 상황이 아니라면 PS에서는 앞으로도 실수를 사용해야 할 경우 double형을 사용할 것을 권해드립니다.

문제 풀이 후 느낀점: 이후 문제를 푸는 과정에서 변수 정의를 할 때, float과 double이 뭐가 다른지, 또 이거 때문에 오류까지 나는지 궁금해져서 질의응답이나 구글을 뒤져보았습니다. 근데 마침 저랑 똑같은 생각을 한 사람이 질문을 했었던 걸 볼 수 있었습니다. 답변에 따르면, float과 double 둘 다 별 차이는 없지만, double이 float보다 좀 더 정밀하고 정확하게 값을 저장할 수 있다고 합니다. 앞으로 실수형을 쓰는 일이 생긴다면 float보다 double을 쓰는 것이 더 효율적일 수 있다는 것을 알 수 있었습니다.

이렇게 오늘은 간단한 출력 문제와 정수의 연산 문제들이었습니다. 개인적으로 좀 문제를 평소에 풀어서 다음 차 시때는 좀 더 수준 있는 문제로 도전해보겠습니다.

27896번 – 특별한 순회

김세진

문제

???: 가지라니, 비슷하지도 않잖아요...

NLCS Jeju에서는 파묻튀(파마산을 묻혀 튀긴 소고기)를 서빙하는 것을 좋아한다.

그러나, 학생들은 파묻튀보다는 신선한 가지를 먹고 싶어한다!

급식실에 $N \leq N \leq 200000$ 명의 학생들이 차례로 서 있다. 줄의 앞에서부터 $i \leq i \leq N$ 번째 학생이 가지 대신 파묻튀를 받았을 경우 $x_i \leq x_i \leq 10^9$ 만큼 불만도가 늘어나고, 가지를 받았을 경우에는 $x_i \leq x_i \leq 10^9$ 만큼 불만도가 내려간다. 단, 불만도의 초기값은 0이다.

음식을 앞에 서있는 학생부터 순서대로 서빙할 때, 어떤 한 순간이라도 불만도가 $M \leq M \leq 10^9$ 이상이 되면 학생들은 '가지 운동'을 일으키게 된다.

가지 운동을 일으키지 않게 하기 위한 가지의 최소 개수를 구하는 프로그램을 작성하시오.

입력

첫 번째 줄에 $N \leq N \leq 200000$ 과 $M \leq M \leq 10^9$ 이 공백으로 구분되어 주어진다.

두 번째 줄에 $x_i \leq x_i \leq 10^9$ 를 나타내는 $N \leq N \leq 200000$ 개의 정수가 공백으로 구분되어 주어진다.

출력

첫 번째 줄에 학생들이 가지 운동을 일으키지 않게 하기 위한 가지의 최소 개수를 출력한다.

제한

- $1 \leq N \leq 200000$ $1 \leq N \leq 200,000$
- $1 \leq M \leq 10^9$ $1 \leq M \leq 10^9$
- $0 \leq x_i \leq 10^9$ $0 \leq x_i \leq 10^9$

예제 입력 1 복사

5 3

0 0 2 0 2

예제 출력 1 복사

1

예제 입력 2 복사

10 90

14 6 12 16 14 6 20 19 16 12

예제 출력 2 복사

2

● 해결과정

N개의 불만도가 순서대로 주어진다. 이때 입력된 불만도 M의 값을 한 순간도 넘기지 않게 하기 위한 최소 가지의 개수를 구하는 문제이다. 먼저 N번 반복하면서 불만도를 입력받는다. 이때 누적 불만도를 계산하여 최대힙에 입력한다. 만약 누적 불만도가 M보다 작으면 입력을 계속 받는다. 어느 순간 누적 불만도가 M 이상이 되면 M보다 낮아질 때 까지 최대힙에서 수 x를 꺼내 누적 불만도에서 빼준다. X는 누적 불만도에 이미 한 번 더해져 있으므로 2를 곱해 빼야 한다. 최대힙에서 수를 뺄 때마다 gaji변수에 1을 더해준다. 이렇게 반복문이 N까지 순회하면 가지의 최소 개수를 구할 수 있다.

● 느낀점

우선순위 큐의 이해도를 높이고자 이 문제를 풀어보았다. 이 문제에서 우선순위 큐의 사용은 간단했지만 우선순위 큐를 복습하는 것에 있어서는 좋은 공부가 되었다.

● 소스코드

```
#include <stdio.h>
#define SIZE 200000

int N, M, sum=0, gaji=0;
int heap_size=0, heap[SIZE+1];

void swap(int *a, int *b){
    int t = *a;
    *a = *b;
    *b = t;
}

void heap_push(int x){
    heap_size++;
    heap[heap_size] = x;

    int cur = heap_size;
    while(cur>1&&heap[cur]>heap[cur/2]){
        swap(&heap[cur], &heap[cur/2]);
        cur /= 2;
    }
}

int heap_pop(){
    if(heap_size==0) return 0;
```

```

int ret = heap[1];
heap[1] = heap[heap_size--];
int cur=1;
while(1) {
    int left = 2*cur;
    int right = left + 1;
    int next = cur;

    if(left <= heap_size && heap[left] > heap[next])
        next = left;
    if(right <= heap_size && heap[right] > heap[next])
        next = right;
    if(next == cur)
        break;
    swap(&heap[cur], &heap[next]);
    cur = next;
}
return ret;
}

int main(void){
    scanf("%d %d", &N, &M);
    int arr[SIZE] = {0};
    for(int i=0; i<N; i++){
        scanf("%d", &arr[i]);
        heap_push(arr[i]);
        sum+=arr[i];
        while(sum>=M){
            int x=heap_pop();
            sum -= 2*x;
            gaji++;
        }
    }
    printf("%d\n", gaji);

    return 0;
}

```

세 용액

성공 스페셜 저지



3 골드 III

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	43044	12843	9177	27.258%

문제

KOI 부설 과학연구소에서는 많은 종류의 산성 용액과 알칼리성 용액을 보유하고 있다. 각 용액에는 그 용액의 특성을 나타내는 하나의 정수가 주어져있다. 산성 용액의 특성값은 1부터 1,000,000,000까지의 양의 정수로 나타내고, 알칼리성 용액의 특성값은 -1부터 -1,000,000,000까지의 음의 정수로 나타낸다.

같은 양의 세 가지 용액을 혼합한 용액의 특성값은 혼합에 사용된 각 용액의 특성값의 합으로 정의한다. 이 연구소에서는 같은 양의 세 가지 용액을 혼합하여 특성값이 0에 가장 가까운 용액을 만들려고 한다.

예를 들어, 주어진 용액들의 특성값이 [-2, 6, -97, -6, 98]인 경우에는 특성값이 -97와 -2인 용액과 특성값이 98인 용액을 혼합하면 특성값이 -1인 용액을 만들 수 있고, 이 용액이 특성값이 0에 가장 가까운 용액이다. 참고로, 세 종류의 알칼리성 용액만으로나 혹은 세 종류의 산성 용액만으로 특성값이 0에 가장 가까운 혼합 용액을 만드는 경우도 존재 할 수 있다.

산성 용액과 알칼리성 용액이 주어졌을 때, 이 중 같은 양의 세 개의 서로 다른 용액을 혼합하여 특성값이 0에 가장 가까운 용액을 만들어내는 세 용액을 찾는 프로그램을 작성해보시오.

이 문제의 입력은 3개 이상 5000개 이하의 용액을 입력받는다. 용액의 특성값은 $-1e9$ 부터 $1e9$ 까지이다. 시간제한은 1초니까 러프하게 잡았을 때 $O(N^2)$ 으로 끝내야한다. 그리고 최소를 찾는 것이니 초기 최댓값을 안전하게 float("INF")로 잡는다.

용액의 개수가 3개이니 하나를 고정해두고 투포인터로 문제를 풀어야겠다는 생각을 했다. 특성값이 지금까지 나온 용액 중에 최소값이라면 최소를 갱신하고 용액의 특성값이 음수이면 포인터1을 +1 하고 양수이면 포인터2를 -1을 반복하여 루프 2개로 $O(N^2)$ 으로 코드를 완성했다.

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

N = int(input())
arr = sorted(list(map(int, input().split())))
ans = []
max_acid = float("INF")
for i in range(N-2):
    p1 = i+1
    p2 = N-1
    while p1 < p2:
        acid = arr[i] + arr[p1] + arr[p2]
        if abs(acid) < max_acid:
            max_acid = abs(acid)
            ans = [arr[i], arr[p1], arr[p2]]
        if acid > 0:
            p2 -= 1
        elif acid < 0:
            p1 += 1
```

```
p1 += 1
else:
    break

print(" ".join(map(str, sorted(ans))))
```

이 문제는 단순한 투포인터 보다는 하나를 고정 시키고 투포인터를 적용할 수 있나를 묻는 문제였다. 요즘 투포인터랑 이분탐색 문제를 많이 풀어보고 있는데 도움이 많이 되는 문제인 것 같다. 다른 투포인터랑 이분탐색 문제를 풀면서 더욱 실력을 늘려야겠다고 생각했다.



스터디 1주차 보고서

(3/17 ~ 3/22)

작성자 : 이서연

작성일 : 2025.03.22

문제 풀이(백준 10989 수 정렬하기 3)

이 문제는 시간 제한과 메모리 제한이 모두 까다로운 문제다. 그리고, 결론부터 말하자면 나는 기능 구현에는 성공했지만 이 문제를 완전히 잘못 이해하고 풀었기에 메모리 제한에 실패했다. 잘못된 문제 분석을 적는 것보다는 내가 어째서 잘못 풀게 되었는지 되돌아보는 게 도움이 될 것 같아 소스 코드부터 첨부한 다음 문제를 푼 과정 및 느낀점을 소개하겠다.

(소스 코드)

CPP

```
#include <stdio.h>
#include <stdlib.h>

int compare(const void* a, const void* b) {
    //void: 자료형의 크기가 정해지지 않은 포인터(연산 안 됨)

    if (*(short*)a < *(short*)b) // 오름차순으로 하고 싶을 경우 첫번째 변수가 더 클 때 1 반환,
        작을 때 -1 반환
        return -1;

    if (*(short*)a > *(short*)b)
        return 1;

    return 0;
}

int main() {
    int n;
    scanf("%d", &n);
    short* num = (short*)malloc(sizeof(short) * n);

    for (int i = 0; i < n; i++) {
        scanf("%d", num+i);
    }

    qsort(num, n, sizeof(short), compare);
    for (int i = 0; i < n; i++) {
        printf("%d\n", *(num + i));
    }
    return 0;

    free(num);
}
```

(문제를 푼 과정 및 느낀점)

나는 백준 문제를 푸는 것도 처음이고 메모리를 신경 써야 하는 프로그래밍을 하는 것도 처음이라 8MB라는 메모리 제한이 얼마나 작은지 실감하지 못했기에 이런 잘못된 코드가 나온 것 같다. int로 적었던 자료형을 short로 바꾸는 등의 노력을 해보며 3번 정도 채점을 받아봤지만 계속 메모리 제한에 실패해 다른

사람들의 풀이를 참고하려 살펴보니 나와는 전혀 다른 코드였다. 주로 계수 정렬을 사용하는 코드가 많았다. 다른 사람의 코드를 확인했을 시점이 발표날이 얼마 남지 않았을 시점이라 이 코드를 참고해서 프로그래밍을 해도 베낀 것밖에 되지 않을 거라는 생각이 들어 이번에는 실패한 코드를 되돌아보는 기회를 가지고 메모리 관리에 대해 공부한 후 이 스터디가 끝나기 전에 다시 문제를 풀어 발표해보려 한다.

계수 정렬: 주어진 배열의 값의 범위가 작은 경우 빠른 속도를 갖는 정렬 알고리즘이다. 원소 값 개수의 누적값을 저장하는 배열을 따로 만들어 정렬을 수행한다.



스터디 3주차 보고서

(4/28~ 5/4)

작성자 : 이서연

작성일 : 2025.05.05

문제 풀이(백준 10989 수 정렬하기 3)

지난 시간에 메모리 관리에 실패했던 문제를 다른 사람들의 풀이를 참고해 다시 풀어본 것이다. 이 문제는 천만 개의 수가 주어지더라도 8MB 안으로 메모리 제한을 해야 하기에 메모리 제한이 매우 까다로운 문제고, 대부분의 사람이 계수정렬을 사용해 이 문제를 해결했다. 계수정렬을 위한 배열을 만들고 몇 개의 수를 입력 받을 것인지 scanf 함수를 통해 입력받는다. 이렇게 정한 개수만큼 숫자를 카운트하는 코드를 반복하고, 1부터 10000까지 반복하면서 해당 수 개수만큼 출력하는 코드를 반복하면 된다.

(소스 코드-틀림)

C

```
#include<stdio.h>

int main() {
    int N;
    int i, j;
    int num;

    int count[10000] = { 0, };
    scanf("%d", &N);

    for (i = 0; i < N; i++) {
        scanf("%d", &num);
        count[num-1] = count[num-1] + 1; //숫자 카운트
    }

    for (i = 0; i < 10000; i++) {
        for (j = 1; j <= count[i]; j++) { //숫자만큼 반복 출력
            printf("%d\n", i+1);
        }
    }
    return 0;
}
```

(문제를 푼 과정 및 느낀점)

이 문제는 천만 개의 수가 주어지더라도 8MB 안으로 메모리 제한을 해야 하기에 메모리 제한이 매우 까다로운 문제고, 나는 지난번에 8MB라는 메모리 제한이 얼마나 작은지 이해하지 못하고 쿼드렬을 이용하여 문제를 풀었기에 기능구현은 했지만 메모리 제한에 실패하여 해당 문제를 해결하지 못했다. 다른 사람의 코드를 참고하여 코드를 적었지만 계수를 잘못 적어 10000을 넣었을 때 오류가 발생했고, 이를 수정하여 지금과 같은 코드가 되었다. 지금껏 메모리를 신경쓰며 코딩해본 적이 없는데 이번 기회를 통해 메모리를 줄이는 여러 방법을 알아보았고, 계수정렬이라는 효율적인 알고리즘을 알게 되었다. 사실 브론즈 문제는 쉬울 거라 예상하고 이번에만 브론즈 문제를 풀고 다음부터는 더 높은 티어 문제에 도전해보려 했으나 아직 부족함을 느꼈고, 다음에도 브론즈 문제 중 너무 쉽지 않은 문제에 도전해볼 생각이다.

계수 정렬: 주어진 배열의 값의 범위가 작은 경우 빠른 속도를 갖는 정렬 알고리즘이다. 원소 값 개수의 누적값을 저장하는 배열을 따로 만들어 정렬을 수행한다.

미로 탐색 - 2178

1 2178번 제출 맞힌 사람 솔코딩 재채점 결과 채점 현황 내 제출 난이도 기여 ⌂
강의 ▾ 질문 게시판

미로 탐색 성공



1 실버 I

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	192 MB	244624	115648	72273	45.571%

문제

N×M크기의 배열로 표현되는 미로가 있다.

그냥 간단한 bfs 구현 문제 경우 나누는 과정이 귀찮았다.

```
from collections import deque
import sys
input = sys.stdin.readline

N,M = map(int,input().split())

maze = []
for _ in range(N):
    maze.append(input().strip())

def bfs():
    visited = [[False]*(M) for _ in range(N)]
    queue = deque([(0,0,1)])
    visited[0][0] = True
```

```
while queue:  
    x,y,dist = queue.popleft()  
  
    if x == M-1 and y == N-1:  
        print(dist)  
        break  
  
    if maze[y][x] == '1':  
        if x > 0:  
            if not visited[y][x-1]:  
                visited[y][x-1] = True  
                queue.append((x-1, y, dist+1))  
  
        if y<N-1:  
            if not visited[y+1][x]:  
                visited[y+1][x] = True  
                queue.append((x,y+1,dist+1))  
  
        if y>0:  
            if not visited[y-1][x]:  
                visited[y-1][x] = True  
                queue.append((x,y-1,dist+1))  
  
    if x<M-1:  
        if not visited[y][x+1]:  
            visited[y][x+1] = True  
            queue.append((x+1,y,dist+1))  
  
bfs()
```



트리의 부모 찾기 - 11725

트리 문제인데, 트리 문제는 가장 중요한 것이 시작 탐방 노드이다. 문제에서는 루트가 1이라고 했으니 1 노드부터 탐방해보자.

풀이는 크게 두 가지가 존재한다

1. bfs

2. dfs

나는 dfs를 짜본 적이 없어 bfs로 짜보기로 하였다. bfs로 탐방을 할 때, 현재 노드와 queue에 노드 값을 추가 해 줄 때, 추가되는 노드들은 전부 부모 노드가 현재 노드이다.

```
graph TD
    a((1))
    b((6))
    c((4))
    d((2))
    a --> b
    a --> c
    c --> d

    style b fill:#ff9999,stroke:#333,stroke-width:2px
    style c fill:#ff9999,stroke:#333,stroke-width:2px
    style a fill:#6afdcf,stroke:#333,stroke-width:2px
```

그림에서 볼 수 있 듯이 현재노드 1이 다음 탐방하는 빨간색 노드의 부모이다.



Python

```
from collections import deque
import sys
input = sys.stdin.readline

def bfs(graph,start,N):
    queue = deque([start])
    ans = [0]*(N+1)

    while queue:
        now_node = queue.popleft()
        for i in graph[now_node]:
            if (ans[i] == 0):
                ans[i] = now_node
                queue.append(i)

    return map(str,ans[2:])

N = int(input())
graph = [[] for _ in range(N+1)]
for i in range(N-1):
    x,y = map(int,input().split())
    graph[x].append(y)
    graph[y].append(x)

bfs(graph,1,N)
print('\n'.join(bfs(graph,1,N)))
```

dfs코드도 같은 원리이다.

Python

```
from collections import deque
import sys
input = sys.stdin.readline

def dfs(start, graph, N):
    ans = [0]*(N+1)

    stack = deque([start])

    while stack:
        now_node = stack.pop()
        for i in reversed(graph[now_node]):
            if not ans[i]:
                stack.append(i)
                ans[i] = now_node

    return map(str,ans[2:])

N = int(input())
graph = [[] for _ in range(N+1)]

for _ in range(N-1):
    x,y = map(int,input().split())
    graph[x].append(y)
    graph[y].append(x)

print('\n'.join(dfs(1,graph,N)))
```

계단 오르기 - 2579

1. 최대 값을 구하며,
2. 이전 값을 통해 최종 결과 값이 구해진다.

즉 이 문제는 dp 문제로 의심해 볼 수 있다.

예를 들어 입력이

```
2  
2  
2
```

라고 해보자.

계단이 3개가 연속되지 않으려면, 현재 계단에서 바로 이전 계단을 밟았을 때, 바로 다음 계단을 밟아서는 안된다.

	시작점	계단 1	계단 2
이전 칸을 안밟음	0	2	2
이전 칸을 밟음	0	0	4

즉 이전 칸을 밟았을 때는 상태 전이가 칸을 안밟은 칸으로만 옮겨지며, 2칸 올라간다.

이전 칸을 밟지 않았을 때는 한 칸 올라갔을 때와, 두 칸 올라갔을 때 모두 상태 전이가 발생 한다.

```
import sys  
input = sys.stdin.readline  
  
n = int(input())  
dp = [[0]*(n+5),[0]*(n+5)]  
point = [0]
```

```
for _ in range(n):
    point.append(int(input()))

point.append(0)

dp[0][1] = point[1]
dp[0][2] = point[2]

for i in range(1,n):
    dp[1][i+1] = max(dp[1][i+1],dp[0][i]+point[i+1])
    dp[0][i+2] = max(dp[0][i+2],dp[0][i]+point[i+2])
    dp[0][i+2] = max(dp[0][i+2],dp[1][i]+point[i+2])

print(max(dp[0][n],dp[1][n]))
print(dp)
```



집합 - 11723

5 11723번 제출 맞힌 사람 속코딩 재채점 결과 채점 현황 내 제출 난이도 기여 ↗
강의▼ 질문 게시판

집합 성공 언어 제한



5 실버 V

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1.5 초	4 MB (하단 참고)	146535	45506	33650	29.776%

문제

비어있는 공집합 S가 주어졌을 때, 아래 연산을 수행하는 프로그램을 작성하시오.

- add x : S에 x를 추가한다. ($1 \leq x \leq 20$) S에 x가 이미 있는 경우에는 연산을 무시한다.
- remove x : S에서 x를 제거한다. ($1 \leq x \leq 20$) S에 x가 없는 경우에는 연산을 무시한다.
- check x : S에 x가 있으면 1을, 없으면 0을 출력한다. ($1 \leq x \leq 20$)
- toggle x : S에 x가 있으면 x를 제거하고, 없으면 x를 추가한다. ($1 \leq x \leq 20$)
- all : S를 $\{1, 2, \dots, 20\}$ 으로 바꾼다.
- empty : S를 공집합으로 바꾼다.

일단 문제 그대로 풀었다.

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

M = int(input())
arr = [0]*(21)

for i in range(M):
    temp = input().strip().split()
```

```
if temp[0] == 'add':
    arr[int(temp[1])] = 1

elif temp[0] == 'check':
    print(arr[int(temp[1])])

elif temp[0] == 'remove':
    arr[int(temp[1])] = 0
elif temp[0] == 'toggle':
    if arr[int(temp[1])]:
        arr[int(temp[1])] = 0

else:
    arr[int(temp[1])] = 1

elif temp[0] == 'all':
    for j in range(21):
        arr[j] = 1

else:
    for j in range(21):
        arr[j] = 0
```

pypy로 제출했는데 메모리 초과가 떴다. pypy는 python에 비해 빠른 대신 메모리를 많이 차지하는데, python으로 바꾸자마자 해결되었다.

DFS와 BFS - 1260

The screenshot shows a web-based programming competition interface. At the top, there's a navigation bar with links like '제출', '맞힌 사람', '숏코딩', '제체점 결과', '체점 현황', '내 제출', '난이도 기여', '강의', and '질문 게시판'. Below the navigation bar, the title 'DFS와 BFS' is displayed with a status '성공' (Success). A progress bar indicates '2 실버 II'. The main area shows a table with performance metrics:

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	128 MB	339128	136884	80790	38.890%

Below the table, there are two sections: '문제' (Problem) and '입력' (Input). The '문제' section contains the problem statement: "그래프를 DFS로 탐색한 결과와 BFS로 탐색한 결과를 출력하는 프로그램을 작성하시오. 단, 방문할 수 있는 정점이 여러 개인 경우에는 정점 번호가 작은 것을 먼저 방문하고, 더 이상 방문할 수 있는 점이 없는 경우 종료한다. 정점 번호는 1번부터 N번까지이다." The '입력' section contains the input specification: "첫째 줄에 정점의 개수 N(1 ≤ N ≤ 1,000), 간선의 개수 M(1 ≤ M ≤ 10,000), 탐색을 시작할 정점의 번호 V가 주어진다. 다음 M개의 줄에는 간선이 연결하는 두 정점의 번호가 주어진다. 어떤 두 정점 사이에 여러 개의 간선이 있을 수 있다. 입력으로 주어지는 간선은 양방향이다."

그냥 DFS와 BFS를 구현하면 된다.

```
visited = [False]*(N+1)
def DFS(now_node):
    visited[now_node] = True
    print(now_node, end = ' ')
    for i in graph[now_node]:
        if not visited[i]:
            dfs(i)
```

```
dfs(V)
```

```
visited = [False]*(N+1)
```

```
def BFS():
```

```
queue = deque([V])
visited = [False] * (N+1)
visited[V] = True

while queue:
    now_node = queue.popleft()
    for i in graph[now_node]:
        if not visited[i]:
            visited[i] = True
            queue.append(i)

print(now_node, end = ' ')
```

```
BFS()
```

두 코드를 합치면 된다.

Python

```
from collections import deque

N,M,V = map(int,input().split())

graph = [[] for _ in range(N+1)]

for _ in range(M):
    x,y = map(int,input().split())
    graph[x].append(y)
    graph[y].append(x)

for i in range(1,N+1):
    graph[i].sort()

visited = [False] * (N+1)
```

```

def dfs(now_node):
    visited[now_node] = True
    print(now_node, end=' ')
    for i in graph[now_node]:
        if not visited[i]:
            dfs(i)

def bfs():
    queue = deque([V])
    visited = [False]*(N+1)
    visited[V] = True

    while queue:
        now_node = queue.popleft()
        for i in graph[now_node]:
            if not visited[i]:
                visited[i] = True
                queue.append(i)

        print(now_node, end = ' ')

    dfs(V)
    print("")
    bfs()

```

C++

```

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

vector<vector<int>> graph;
vector<bool> visited;
int N, M, V;

```

```

// DFS 코드
void dfs(int now_node)
{
    visited[now_node] = true;
    cout << now_node << ' ';
    for (int x : graph[now_node])
    {
        if (!visited[x])
        {
            dfs(x);
        }
    }
}

// BFS 코드
void bfs(int start)
{
    queue<int> que;
    vector<bool> vis(N + 1, false);
    que.push(start);
    vis[start] = true;

    while (!que.empty())
    {
        int now_node = que.front();
        que.pop();
        cout << now_node << ' ';

        for (int x : graph[now_node])
        {
            if (!vis[x])
            {
                vis[x] = true;
                que.push(x);
            }
        }
    }
}

```

```
}

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

    cin >> N >> M >> V;

    graph.resize(N + 1);
    visited.assign(N + 1, false);

    for (int i = 0; i < M; ++i)
    {
        int x, y;
        cin >> x >> y;
        graph[x].push_back(y);
        graph[y].push_back(x);
    }

    for (int i = 1; i <= N; ++i)
    {
        sort(graph[i].begin(), graph[i].end());
    }

    dfs(V);
    cout << '\n';
    bfs(V);

    return 0;
}
```

도미노 게임 - 34053

도미노 게임 성공 다국어

☆ 한국어 ▾

3 실버 III

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초 (추가 시간 없음)	1024 MB (추가 메모리 없음)	241	124	104	59.091%

문제

$N \times M$ 크기의 격자가 주어진다. i 행 j 열의 칸에는 음이 아닌 정수 a_{ij} 가 적혀 있다. ($1 \leq i \leq N; 1 \leq j \leq M$)

민지는 각 차례마다 아래와 같은 행동을 한다.

1. 격자에서 상하좌우로 인접한 두 칸을 선택한다. 이때 두 칸에 적힌 수 중 적어도 하나는 양의 정수이어야 한다.

2. 선택한 칸에 적힌 수에서 1씩 뺀다. 만약 어떤 칸에 적힌 수가 0이라면 그대로 둔다.

모든 칸에 적힌 수가 0이 될 때 게임이 종료된다. 민지는 이 게임을 최대한 오래 하려고 한다. 민지가 최선을 다해 게임을 오래 진행했을 때, 진행할 수 있는 최대 차례의 수를 구해보자.

입력

첫째 줄에 양의 정수 N, M 이 공백으로 구분되어 주어진다. ($2 \leq N, M \leq 1000$)

이후 N 개의 줄에 걸쳐 격자에 적힌 수가 주어진다. 그 중 i 번째 줄에는 $a_{i1}, a_{i2}, \dots, a_{iM}$ 이 공백으로 구분되어 주어진다. ($0 \leq a_{ij} \leq 10^9$)

출력

진행할 수 있는 최대 차례의 수를 출력한다.

ucpc 문제인데, 간단한 게임 이론이다. 먼저 0이 하나가 존재한다고 해보자. 그렇다면, 판에 있는 모든 숫자의 횟수만큼이 최대 차례가 된다. 그런데, 0이 없다면, 가장 작은 수를 제외한 횟수가 최대 횟수가 된다.

Python

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

N,M = map(int,input().split())
graph = []
ans = 0
mn = float('inf')
for i in range(N):
    k = list(map(int,input().split()))
    graph.append(k)
```

```
ans += sum(k)
mn = min(mn,min(k))

ans -= mn
print(ans)
```

나이트의 이동 - 7562

검색▼ 실준 게시판

나이트의 이동 성공 다른어

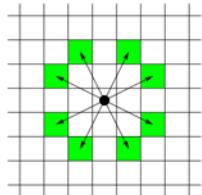
☆ 한국어 ▾

1 실버 I

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	73970	40380	29984	53.392%

문제

체스판 위에 한 나이트가 놓여져 있다. 나이트가 한 번에 이동할 수 있는 칸은 아래 그림에 나와있다. 나이트가 이동하려고 하는 칸이 주어진다. 나이트는 몇 번 움직이면 이 칸으로 이동할 수 있을까?



유명한 문제를 이제 풀어본다. bfs를 좀 많이 풀었더니 이제 눈에 보이기 시작한다.

```
from collections import deque
import sys
input = sys.stdin.readline

testcase = int(input())

for _ in range(testcase):
    l = int(input())
    start_x,start_y = map(int,input().split())
    end_x,end_y = map(int,input().split())

#####BFS 코드
visited = [[False]*l for i in range(l)]
queue = deque([(start_x, start_y, 0)])
```

```
visited[start_y][start_x] = True

while queue:
    x,y,dist = queue.popleft()

    if (x == end_x) and (y == end_y):
        print(dist)
        break

    if (x+1<l) and (y+2<l):
        if not visited[y+2][x+1]:
            visited[y+2][x+1] = True
            queue.append((x+1,y+2,dist+1))

    if (x+2<l) and (y+1<l):
        if not visited[y+1][x+2]:
            visited[y+1][x+2] = True
            queue.append((x+2,y+1,dist+1))

    if (x+2<l) and (y-1>=0):
        if not visited[y-1][x+2]:
            visited[y-1][x+2] = True
            queue.append((x+2,y-1,dist+1))

    if (x+1<l) and (y-2>=0):
        if not visited[y-2][x+1]:
            visited[y-2][x+1] = True
            queue.append((x+1,y-2,dist+1))

    if (x-1>=0) and (y-2>=0):
        if not visited[y-2][x-1]:
            visited[y-2][x-1] = True
            queue.append((x-1,y-2,dist+1))

    if (x-2>=0) and (y-1>=0):
        if not visited[y-1][x-2]:
            visited[y-1][x-2] = True
```

```

queue.append((x-2,y-1,dist+1))

if (x-2>=0) and (y+1<l):
    if not visited[y+1][x-2]:
        visited[y+1][x-2] = True
        queue.append((x-2,y+1,dist+1))

if (x-1>=0) and (y+2<l):
    if not visited[y+2][x-1]:
        visited[y+2][x-1] = True
        queue.append((x-1,y+2,dist+1))

```

- 다른 사람 코드는 좀 더 깔끔하게 짰는데 좋은 테크닉이 쓰였다

```

from collections import deque
import sys
input = sys.stdin.readline

testcase = int(input())

# 나이트 이동 방향
dx = [1, 2, 2, 1, -1, -2, -2, -1]
dy = [2, 1, -1, -2, -2, -1, 1, 2]

for _ in range(testcase):
    l = int(input())
    start_x, start_y = map(int, input().split())
    end_x, end_y = map(int, input().split())

    visited = [[False] * l for _ in range(l)]
    queue = deque([(start_x, start_y, 0)])

```

```
visited[start_y][start_x] = True

while queue:
    x, y, dist = queue.popleft()

    if x == end_x and y == end_y:
        print(dist)
        break

    for i in range(8):
        nx = x + dx[i]
        ny = y + dy[i]

        if 0 <= nx < l and 0 <= ny < l and not visited[ny][nx]:
            visited[ny][nx] = True
            queue.append((nx, ny, dist + 1))
```

- 다른 좌표 문제에서 유용할 듯하다.,

스터디 1주차 보고서

(7/7 ~ 7/13)

작성자 : 모수진

작성일 : 2025.07.13

1. BOJ 5397 : 키로거

(문제 분석)

키로거는 사용자가 키보드를 누른 명령을 모두 기록한다. 따라서, 사용자가 비밀번호를 입력할 때, 화살표나 백스페이스를 입력해도 정확한 비밀번호를 알아낼 수 있다.

비밀번호 창에서 입력한 키가 주어졌을 때, 비밀번호를 알아내는 프로그램을 작성하시오. 키보드로 입력한 키는 알파벳 대문자, 소문자, 숫자, 백스페이스, 화살표이다.

(소스 코드)

Python3

```
import sys

tests=int(sys.stdin.readline())
for _ in range(tests):
    inputL=sys.stdin.readline().rstrip()
    pwd=[]
    cursor=0
    for i in inputL:
        if i=='<':
            cursor=max(cursor-1,0)
        elif i==">>":
            cursor=min(cursor+1,len(pwd))
        elif i=='-' :
            if cursor!=0:
                pwd.pop(cursor-1) #특정 인덱스의 원소를 pop 하는 것은 시간이 오래 걸림
                cursor-=1
            else:
                pwd.insert(cursor,i) #특정 인덱스의 위치에 원소를 삽입하는 것은 시간이 오래 걸림
                cursor+=1
    print(''.join(pwd))
```

- pop 과 insert 함수로 특정 인덱스의 원소를 제거하거나 삽입하는 것은 시간복잡도가 크기 때문에, 입력 문자열의 크기가 클수록 시간이 오래 걸린다.
- 해당 코드는 실제로 백준 사이트에 정답으로 입력 시 시간초과 에러로 실패한다.
- 따라서, 이를 해결하기 위해서는 원소 삽입으로 insert 대신 append를, pop은 마지막 원소를 제거하는 용도로 사용해야 할 것이다.

Python3

```

import sys

#시간 단축을 위해 input() 대신 sys.stdin.readline() 사용
tests=int(sys.stdin.readline())
for _ in range(tests):
    inputL=sys.stdin.readline().rstrip() #마지막 개행문자 제거
    cursor_left=[] #커서 기준 왼쪽 배열
    cursor_right=[] #커서 기준 오른쪽 배열(역순)

    for i in inputL:
        if i=='<': #커서 왼쪽으로 한 칸 이동
            if cursor_left:
                cursor_right.append(cursor_left.pop())
        elif i==">>": #커서 오른쪽으로 한 칸 이동
            if cursor_right:
                cursor_left.append(cursor_right.pop())
        elif i=='-': #커서 왼쪽에서 한 글자 삭제
            if cursor_left:
                cursor_left.pop()
        else: #커서 왼쪽에 한 글자 삽입
            cursor_left.append(i)

    cursor_left.extend(reversed(cursor_right)) #역순인 오른쪽배열을 다시 뒤집어
    왼쪽 뒤에 더하기
    print(''.join(cursor_left)) #출력

```

- 커서를 기준으로, 커서 왼쪽의 문자열은 cursor_left 배열에, 커서 오른쪽의 문자열은 cursor_right 배열에 저장하도록 설계한다.
- '<'입력 시 커서가 왼쪽으로 한 칸 이동한 것이므로, cursor_left의 마지막 원소가 커서 오른쪽 문자열의 첫번째가 될 것이다. 따라서 cursor_left의 마지막 원소를 pop한 후 해당 문자를 cursor_right에 추가한다.
- '>'입력 시 커서가 오른쪽으로 한 칸 이동한 것이므로, cursor_right의 마지막 원소가 (=커서 오른쪽 문자열의 첫 번째 문자) 커서 왼쪽 문자열의 마지막이 될 것이다. 따라서 cursor_right의 마지막 원소를 pop한 후 해당 문자를 cursor_left에 추가한다.
- ‘-’는 커서 왼쪽의 문자열 마지막 글자를 삭제하는 것이므로 cursor_left의 마지막 원소를 pop한다.
- 그 외의 문자 입력은 모두 커서 왼쪽의 문자열 마지막에 추가하는 것이므로 cursor_left에 추가한다.
- 이때, pop함수는 빈 배열에서 실행할 수 없으므로 조건문을 사용하여 빈 배열이 아닐 때만 실행하도록 한다.

2. BOJ 2108 : 통계학

(문제 분석)

통계학에서 N개의 수를 대표하는 기본 통계값에는 다음과 같은 것들이 있다. 단, N은 홀수라고 가정하자.

1. 산술평균 : N개의 수들의 합을 N으로 나눈 값
2. 중앙값 : N개의 수들을 증가하는 순서로 나열했을 경우 그 중앙에 위치하는 값
3. 최빈값 : N개의 수들 중 가장 많이 나타나는 값
4. 범위 : N개의 수들 중 최댓값과 최솟값의 차이

N개의 수가 주어졌을 때, 네 가지 기본 통계값을 구하는 프로그램을 작성하시오.
로 나타낼 수 있다.

(소스 코드)

Python3

```
import sys

n=int(sys.stdin.readline())
arr=[]
for _ in range(n):
    arr.append(int(sys.stdin.readline()))
arr.sort()
sum=arr[0]
cnt=0
maxcnt=0
maxcnt_n=arr[0]
flag=0
for i in range(1,n):
    sum+=arr[i]
    cnt+=1
    if arr[i]!=arr[i-1]:
        if maxcnt==cnt:
            if flag==0:
                maxcnt_n=arr[i-1]
                flag=1
        elif maxcnt<cnt:
            maxcnt=cnt
            maxcnt_n=arr[i-1]
            flag=0
    cnt=0
```

```

cnt+=1
if ((flag==0) & (maxcnt==cnt))|(maxcnt<cnt):
    maxcnt_n=arr[n-1]

print(int(round(sum/n,0))) #산술평균
print(arr[n//2]) #중앙값
print(maxcnt_n) #최빈값
print(arr[n-1]-arr[0]) #범위

```

- 산술평균은 각 입력값을 sum에 모두 더하고 이를 n으로 나누어 구한다. 이때 소수 첫째 자리에서 반올림하여 평균값을 구하므로 round 함수를 사용한다.
- 중앙값은 인덱스가 $(n-1)/2$ 인 원소를 출력한다.
- 최빈값을 구하기 위하여 입력 받는 각 원소의 개수를 센다. 오름차순으로 정렬한 후 연속된 값을 세는데, 이때 이전 원소와 다른 값이 되는 경계일 때 해당 숫자의 빈도수를 체크한다. 이전 최대 빈도수와 같다면 flag값을 주어 두 번째로 작은 값을 확인한다.
- 범위는 마지막 원소 값에서 첫 번째 원소 값을 빼 출력한다.

스터디 3주차 보고서

(7/21 ~ 7/27)

작성자 : 모수진

작성일 : 2025.07.27

3. BOJ 1003 : 피보나치 함수

(문제 분석)

`fbonacci(3)` 을 호출하면 다음과 같은 일이 일어난다.

- `fbonacci(3)` 은 `fbonacci(2)` 와 `fbonacci(1)` (첫 번째 호출)을 호출한다.
- `fbonacci(2)` 는 `fbonacci(1)` (두 번째 호출)과 `fbonacci(0)` 을 호출한다.
- 두 번째 호출한 `fbonacci(1)` 은 1을 출력하고 1을 리턴한다.
- `fbonacci(0)` 은 0을 출력하고, 0을 리턴한다.
- `fbonacci(2)` 는 `fbonacci(1)` 과 `fbonacci(0)` 의 결과를 얻고, 1을 리턴한다.
- 첫 번째 호출한 `fbonacci(1)` 은 1을 출력하고, 1을 리턴한다.
- `fbonacci(3)` 은 `fbonacci(2)` 와 `fbonacci(1)` 의 결과를 얻고, 2를 리턴한다.

1은 2번 출력되고, 0은 1번 출력된다. N이 주어졌을 때, `fbonacci(N)` 을 호출했을 때, 0과 1이 각각 몇 번 출력되는지 구하는 프로그램을 작성하시오.

(소스 코드)

```
Python3
```

```

import sys

n=int(sys.stdin.readline())
for _ in range(n): #각 케이스별 입력 -> 출력 반복
    cnt0=[1,0] #f(0) 호출 횟수
    cnt1=[0,1] #f(1) 호출 횟수
    i=int(sys.stdin.readline()) #케이스 입력
    if i>1:
        for k in range(2,i+1):
            cnt0.append(cnt0[-2]+cnt0[-1])
            cnt1.append(cnt1[-2]+cnt1[-1])

    print(cnt0[i],cnt1[i])

```

- cnt0의 초기값:f(0)은 f(0)을 1번, f(1)은 f(0)을 0번 호출한다.
- cnt1의 초기값:f(0)은 f(1)을 0번, f(1)은 f(1)을 1번 호출한다.
- $F(n)=F(n-1)+F(n-2)$
- f(n)의 f(0) 호출 횟수는 f(n-1)과 f(n-2)의 f(0) 호출 횟수를 더한 것과 같다.
- 위의 알고리즘으로 cnt0과 cnt1을 구하여 출력한다.

4. BOJ 2108 : 통계학

(문제 분석)

N장의 카드가 있다. 각각의 카드는 차례로 1부터 N까지의 번호가 붙어 있으며, 1번 카드가 제일 위에, N번 카드가 제일 아래인 상태로 순서대로 카드가 놓여 있다.

이제 다음과 같은 동작을 카드가 한 장 남을 때까지 반복하게 된다. 우선, 제일 위에 있는 카드를 바닥에 버린다. 그 다음, 제일 위에 있는 카드를 제일 아래에 있는 카드 밑으로 옮긴다.

예를 들어 N=4인 경우를 생각해 보자. 카드는 제일 위에서부터 1234의 순서로 놓여 있다. 1을 버리면 234가 남는다. 여기서 2를 제일 아래로 옮기면 342가 된다. 3을 버리면 42가 되고, 4를 밑으로 옮기면 24가 된다. 마지막으로 2를 버리고 나면, 남는 카드는 4가 된다.

N이 주어졌을 때, 제일 마지막에 남게 되는 카드를 구하는 프로그램을 작성하시오

(소스 코드)

```
Python3
import collections

n=int(input())
deq=collections.deque(list(range(1,n+1)))

while len(deq)>1:
    deq.popleft()
    deq.append(deq.popleft())

print(deq.pop())
```

- 해당 문제를 해결하기 위하여 python의 모듈 중 리스트의 왼쪽과 오른쪽 모두에서 삽입과 삭제를 할 수 있는 deque를 사용한다.
- 1~n까지의 리스트를 deque의 원소로 담고, deque의 길이가 1이 될 때까지 문제의 조건을 수행한다.
- Popleft()로 첫 번째 원소를 제거하고, 그 다음 원소는 pop한 후 다시 마지막에 append한다.
- while문이 종료되면 deque에 마지막으로 남은 원소를 pop하여 출력한다.



스터디 2주차 보고서

(7/14 ~ 7/20)

작성자 : 성준영

작성일 : 2025.07.20

문제 풀이 (BOJ 1867 돌멩이 제거)

(문제 분석 및 프로그램 설계)

문제

n 행 n 열의 격자로 나뉜 운동장이 있다. 이 위에 k 개의 돌멩이가 있는데, 하나의 돌멩이는 격자 한 칸에 정확히 들어가 있으며, 두 개 이상의 돌멩이가 한 칸에 들어가 있는 경우는 없다.

사고의 위험을 없애기 위해 이 돌멩이를 모두 제거하고 깨끗한 운동장을 만들려고 한다. 돌멩이를 제거할 때에는, 한 행이나 한 열을 따라 직선으로 달려가면서 그 행이나 열에 놓인 돌멩이를 모두 줍는 방식을 쓴다.

여러분이 할 일은 운동장의 상태가 주어졌을 때 최소 몇 번이나 달려가야 돌멩이 줍기를 끝낼 수 있는지 계산하는 것이다.

입력

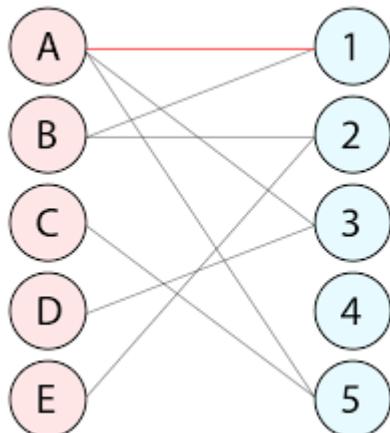
첫째 줄에 n 과 k 가 주어진다. ($1 \leq n \leq 500, 1 \leq k \leq 10,000$) 이후 k 개의 줄에는 돌멩이의 위치가 한 줄에 하나씩 주어진다. 줄마다 첫 번째 숫자는 행 번호, 두 번째 숫자는 열 번호를 나타낸다. 입력으로 주어지는 돌멩이의 위치는 중복되지 않는다.

출력

첫 줄에 몇 번의 달리기를 통해 돌멩이를 주울 수 있는지 출력한다.

(문제를 해결하는데 필요한 개념)

이분매칭 알고리즘



이분 매칭은 레이어가 2인 **fow**의 일종이라 보면 나름 편하겠다.

1. 원쪽 알파벳을 순회하면서

- 숫자칸으로 갈 수 있는 데를 간다
- 숫자칸이 이미 가 있다?
 - backtracking으로 숫자칸으로 가는 알파벳으로 간다
 - 그 놈이 다른 숫자칸으로 갈 수 있으면 간다.

(소스 코드)

CPP

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

```

using namespace std;
vector<int> adj_list[501];
int P[501];
int visited[501];
int BI_DFS(int x){
    for(auto a:adj_list[x]){
        if(visited[a]) continue;
        visited[a]=1;
        if(P[a]==0 or BI_DFS(P[a])){
            P[a]=x;
            return 1;
        }
    }
    return 0;
}
int main(void){
    ios::sync_with_stdio(0);
    cin.tie(0);
    int N,Q;
    cin>>N>>Q;
    while(Q--){
        int a,b;cin>>a>>b;
        adj_list[a].push_back(b);
    }
    int answer=0;
    for(int i=1;i<=N;i++){
        if(adj_list[i].empty()) continue;
        fill(visited,visited+N+1,0);
        answer+=BI_DFS(i);
    }
    cout<<answer;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

Flow 가 좀 어렵다가 이분매칭을 풀었다. 이분 매칭은 backtracking 의 느낌이 물씬 나는 문제 풀이었다. 이분 매칭이 손에 잘 읽게 평소에도 잘 풀어야겠다. 특히 구현상에서 실수가 잦다.

문제 풀이 (BOJ 2414 게시판 구멍 막기)

(문제 분석 및 프로그램 설계)

문제

$N \times M$ 모양의 게시판에 구멍이 뚫려 있다. 이를 폭이 1인 테이프를 이용하여 막으려 한다. 테이프의 길이는 무한하다고 생각해도 좋지만, 테이프를 끊어내는 횟수를 최소로 하려 한다. 테이프를 붙일 때에는 구멍이 뚫려 있지 않은 부분을 막아서는 안 된다. 하지만 테이프가 한 번 붙은 곳에 테이프를 또 붙여도 된다. 또한, 테이프를 붙일 때에는 가로나 세로로 붙이는 경우만 허용한다.

입력

첫째 줄에 N, M ($1 \leq N, M \leq 50$)이 주어진다. 다음 N 개의 줄에는 M 개의 문자로 게시판의 모양이 주어진다. 각각의 문자는 붙어 있으며, 구멍이 없는 부분은 '.', 구멍이 있는 부분은 '*'으로 주어진다.

출력

첫째 줄에 테이프를 끊어 내는 횟수의 최솟값을 출력한다.

(문제를 해결하는데 필요한 개념)

이번 맹꽁 문제이다. Rook 문제와 비슷할 수 있지만 최소 베타스 커버라는 게 있다. 그걸 이해를 더 잘 하면 쉽게 더 잘 풀 수 있을 거다.

(소스 코드)

CPP

```
#include<iostream>
#include<vector>
using namespace std;
vector<int> adj_list[501];
int P[501];
int visited[501];
char _map[51][51];
int row_map[51][51];
int col_map[51][51];
int BI_DFS(int x){
    for(auto a:adj_list[x]){
        if(visited[a]) continue;
        visited[a]=1;
        if(P[a]==0 or BI_DFS(P[a])){
            P[a]=x;
            return 1;
        }
    }
    return 0;
}
int main(void){
    ios::sync_with_stdio(0);
    cin.tie(0);
    int N,M;
    cin>>N>>M;
    for(int i=1;i<=N;i++) for(int j=1;j<=M;j++) cin>>_map[i][j];
    int count=0;
    for(int i=1;i<=N;i++){
        for(int j=1;j<=M;j++){
            if(_map[i][j]=='*'){
                if(_map[i][j-1]=='.') or _map[i][j-1]==0) count++;
                row_map[i][j]=count;
            }
        }
    }
    // for(int i=1;i<=N;i++){
    //     for(int j=1;j<=M;j++) cout<<row_map[i][j];
    //     cout<<"\n";
    // }
    count=0;
    for(int i=1;i<=M;i++){
        for(int j=1;j<=N;j++) {
```

```

        if(_map[j][i]=='*'){
            if(_map[j-1][i]=='.'
               or _map[j-1][i]==0)count++;
            col_map[j][i]=count;
        }
    }
// for(int i=1;i<=N;i++){
//     for(int j=1;j<=M;j++)cout<<col_map[i][j];
//     cout<<"\n";
// }
for(int i=1;i<=N;i++)
    for(int j=1;j<=M;j++)
        if(row_map[i][j] and col_map[i][j])
            adj_list[row_map[i][j]].push_back(col_map[i][j]);
int answer=0;
for(int i=1;i<=500;i++){
    if(adj_list[i].empty())continue;
    // for(auto a:adj_list[i])cout<<a<<" ";
    // cout<<"\n";
    fill(visited,visited+501,0);
    answer+=BI_DFS(i);
}
cout<<answer;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

1. 최소 버텍스 커버와 이분 매칭 간의 연결점이 아직 몸에 와닿지 않는다.
2. 격자점에서의 최소 버텍스 커버와 이분 매칭의 감을 익혀야 한다.
3. 최소 버텍스 커버에 집중된 문제를 제대로 풀어봐야겠다.



스터디 3주차 보고서

(7/20 ~ 7/27)

작성자 : 성준영

작성일 : 2025.07.27

문제 풀이 (BOJ 11280 2-SAT - 3)

(문제 분석 및 프로그램 설계)

2-SAT은 N개의 불리언 변수 x_1, x_2, \dots, x_n 있을 때, 2-CNF 식을 true로 만들기 위해 각

x_i 를 어떤 값으로 정해야 하는지를 구하는 문제다. 2-CNF 식은 $(x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$ 와 같은 형태이며, 여기서 괄호로 묶인 각각의 식을 절(clause)이라고 부른다. 절은 2개의 변수를 OR(\vee) 연산한 형태로 이루어지며, \vee 는 논리합(OR), \wedge 는 논리곱(AND), \neg 는 부정(NOT)을 의미한다. 변수의 개수 N , 절의 개수 M , 그리고 식 f 가 주어졌을 때, 식 f 를 true로 만들 수 있는지를 판단하는 프로그램을 작성해야 한다.

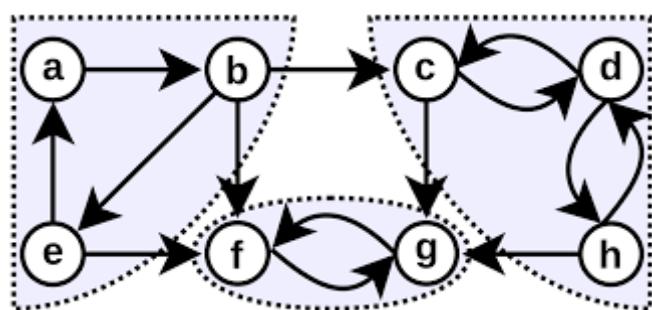
예를 들어, $N = 3, M = 4$ 이고 $f = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (x_1 \vee x_3) \wedge (x_3 \vee x_2)$ 인 경우, x_1 을 false, x_2 를 false, x_3 를 true로 정하면 식 f 를 true로 만들 수 있다. 하지만 $N = 1, M = 2$ 이고 $f = (x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_1)$ 인 경우에는 x_1 에 어떤 값을 넣어도 식 f 를 true로 만들 수 없다.

입력은 다음과 같은 형식으로 주어진다. 첫째 줄에 변수의 개수 N ($1 \leq N \leq 10,000$)과 절의 개수 M ($1 \leq M \leq 100,000$)이 주어지며, 둘째 줄부터 M 개의 줄에는 절이 주어진다. 각 절은 두 정수 i 와 j ($1 \leq |i|, |j| \leq N$)로 이루어지며, i 와 j 가 양수인 경우에는 각각 x_i, x_j 를 의미하고, 음수인 경우에는 각각 $\neg x_i, \neg x_j$ 를 의미한다.

출력은 식 f 를 true로 만들 수 있으면 1을, 만들 수 없으면 0을 출력하면 된다.

(문제를 해결하는데 필요한 개념)

강한 연결 요소가 어떤 알고리즘인지 알고 있어야 한다.



(소스 코드)

```
CPP
#include<iostream>
#include<vector>
#include<utility>
#include<algorithm>
using namespace std;
vector<int> s;
vector<int> adj[20001];
int p[20001];
bool fix[20001];
int ind=1;
int DFS(int x){
    p[x]=ind;
    int pnum=ind;
    s.push_back(x);
    ind++;
    for(int i=0;i<adj[x].size();i++){
        if(!p[adj[x][i]])pnum=min(pnum,DFS(adj[x][i]));
        else if(!fix[adj[x][i]])pnum=min(pnum,p[adj[x][i]]);
    }
    if(p[x]==pnum){
        while(s.back()!=x){
```

```

        fix[s.back()]=1;
        p[s.back()]=x;
        s.pop_back();
    }
    fix[s.back()]=1;
    p[s.back()]=x;
    s.pop_back();
}
return pnum;
}
int main(void){
ios::sync_with_stdio(0);
cin.tie(0);
int N,M;
cin>>N>>M;
for(int i=0;i<M;i++){
    int x,y;
    cin>>x>>y;
    //false true
    if(x>0 and y>0){
        adj[abs(x)+N].push_back(abs(y)); //++
        adj[abs(y)+N].push_back(abs(x)); //++
    }
    else if(x<0 and y<0){
        adj[abs(x)].push_back(abs(y)+N); //--
        adj[abs(y)].push_back(abs(x)+N); //--
    }
    else if(x<0){
        adj[abs(x)].push_back(abs(y)); //-
        adj[abs(y)+N].push_back(abs(x)+N); //-
    }
    else if(y<0){
        adj[abs(x)+N].push_back(abs(y)+N); //+-
        adj[abs(y)].push_back(abs(x)); //-
    }
}
// cout<<adj[1][0];
for(int i=1;i<=N*2;i++)if(!p[i])DFS(i);
// for(int i=1;i<=N*2;i++)cout<<p[i]<<" ";
for(int i=1;i<=N;i++)if(p[i]==p[i+N]){\cout<<0;return 0;}
cout<<1;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

1. 2-sat 문제에서는 강한연결요소라는 알고리즘이 사용되는데 간선을 어떻게 설정할지가 중요하다. 이때 비슷한 간선 설정이 이분매칭에서도 비슷하면서 다르다. 이분매칭에서는 모순되는 애들끼리 맺었는데, 2-sat에서는 모순이 발생할 가능성이 있는 놈들끼리 엮는다. 간선을 만들고, x 와 !x 가 같은 사이클 안에 속해있다면 모순이다! 라는 논리로 2-sat가 진행된다.

문제 풀이 (BOJ 13505 두 수 XOR)

(문제 분석 및 프로그램 설계)

문제

N개의 수가 주어졌을 때, XOR한 값이 가장 큰 두 수를 찾는 프로그램을 작성하시오.

즉, A1, A2, ..., AN 중에서 $i \neq j$ 이면서 $A_i \oplus A_j$ 가 가장 큰 것을 찾아야 한다.

입력

첫째 줄에 N ($2 \leq N \leq 100,000$)이 주어진다.

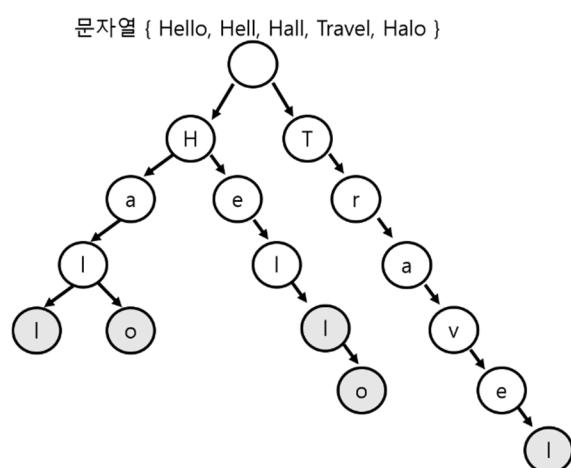
둘째 줄에는 N개의 수가 주어진다. 입력으로 주어지는 수는 1,000,000,000보다 작거나 같은 음이 아닌 정수이다.

출력

첫째 줄에 XOR한 값이 가장 큰 두 수의 XOR한 결과를 출력한다.

(문제를 해결하는데 필요한 개념)

트라이이라는 문자열 알고리즘이 필요하다.



(소스 코드)

CPP

```
#include<iostream>
#include<vector>
#include<cmath>
using namespace std;
int ind=2;
int trie[2][7'000'000];
string int_to_str(int x){
    string tmp;
    for(int i=30;i>=0;i--)tmp.push_back(((x&(1<<i))>0)?'0':1);
    return tmp;
}
void update_dfs(string str, int e, int i){
    if(e<0)return;
    int op=str[30-e]-'0'; //element
    if(!trie[op][i]){

    }
```

```

        trie[op][i]=ind;
        ind++;
        update_dfs(str,e-1,trie[op][i]);
    }
    else{
        update_dfs(str,e-1,trie[op][i]);
    }
}
int xor_find(string str, int e, int i){
    if(e<0) return 0;
    int op=str[30-e]-'0'; //element
    if(trie[op^1][i])return xor_find(str,e-1,trie[op^1][i])+(1<<e);
    else return xor_find(str,e-1,trie[op][i]);
}
int main(void){
    ios::sync_with_stdio(0);
    cin.tie(0);
    int N;
    cin>>N;
    vector<int> v(N);
    int answer=0;
    for(int i=0;i<N;i++)cin>>v[i];
    for(int i=0;i<N;i++)update_dfs(int_to_str(v[i]),30,1);
    for(int i=0;i<N;i++)answer=max(xor_find(int_to_str(v[i]),30,1),answer);
    cout<<answer ;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

1. 수를 이진수로 바꾼 string 으로 trie 를 구성했다.
2. 고정 어레이로 다룬다면 크리는 늘 (MX)라는 걸 유의해야 한다.
 - A. 그럼에도 **outofbound** 를 조심
3. dfs 느낌이라 생각하면 구현이 쉬워진다.
4. 차수가 높은 이진수부터 다른 수로 가는 아이디어가 중요했다.



스터디 4주차 보고서

(7/27 ~ 8/03)

작성자 : 성준영

작성일 : 2025.08.03

문제 풀이 (BOJ 1126 같은 탐)

(문제 분석 및 프로그램 설계)

문제

홍준이는 N개의 직사각형 블록을 가지고 있다. 홍준이는 블록 위에 또 다른 블록을 올려놓는 방식으로 탑을 만들 수 있다. 이때, 두 개의 탑을 만드는데, 이 두 탑의 높이가 같게 만들려고 한다. 각 탑은 적어도 한 개의 블록을 포함해야 한다. 홍준이는 되도록이면 탑의 높이를 최대로 하려고 한다. 그리고 모든 블록을 사용할 필요는 없다.

각 블록의 높이가 주어질 때, 홍준이가 만들 수 있는 탑의 높이의 최댓값을 출력하는 프로그램을 작성하시오.

입력

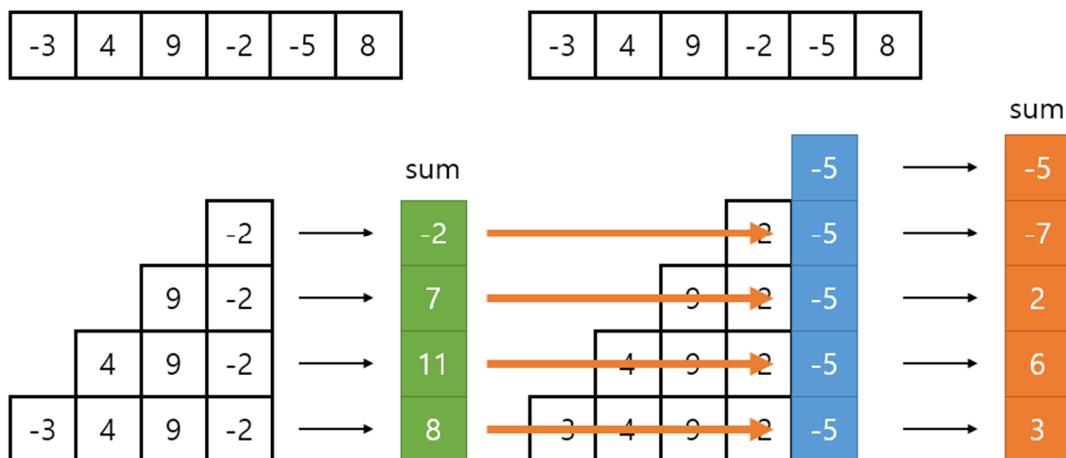
첫째 줄에 조각의 개수 N이 주어진다. N은 50보다 작거나 같은 자연수이다. 둘째 줄에 각 조각의 높이가 주어진다. 높이는 500,000보다 작거나 같은 자연수이고, 모든 조각의 높이의 합은 500,000을 넘지 않는다.

출력

첫째 줄에 문제의 정답을 출력한다. 불가능할 때는 -1을 출력한다.

(문제를 해결하는데 필요한 개념)

DP



(소스 코드)

CPP

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int dp[51][500'001];
vector<int> v;
int main(void) {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int N; cin>>N;
    v.resize(N);
    dp[0][0]=1;
    for(int i=0;i<N;i++) {
        int x; cin>>x;
        for(int j=0;j<=500'000;j++) {
            if(!dp[i][j]) continue;
            if(j+x<=500'000) dp[i+1][j+x] = 1;
        }
    }
}
```

```

if(j+x<=500'000) dp[i+1][j+x]=max(dp[i+1][j+x],dp[i][j]+x);
if(j<x) dp[i+1][x-j]=max(dp[i+1][x-j],dp[i][j]+(x-j));
if(j>=x) dp[i+1][j-x]=max(dp[i+1][j-x],dp[i][j]);
dp[i+1][j]=max(dp[i+1][j],dp[i][j]);
}
}
if(dp[N][0]==1) cout<<"-1";
else cout<<dp[N][0]-1;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

1. DP 에서는 특히 변수를 어떻게 압축하고 변형해서 매력적이게 쓸 수 있느냐다.
2. 1차원 DP 에서는 index 와 value 총 두 부분을 매력적이게 쓸 수 있어야 한다.
그럼 변수를 두 개로 압축하고 보기 좋게 변형해야 한다는 거다.
3. 2차원 DP 에서는 행과 열, value 총 세 부분을 매력적이게 쓸 수 있어야 한다.
그럼 변수를 세 개로 압축하고 보기 좋게 변형해야 한다.
 1. 압축에는 늘 최댓값 최솟값이라는 개념이 많은 걸 보장해준다.
4. 해당 문제에서 생각할 수 있는 변수는 총 세 개이다
 1. N: 현재 들고 있는 조각
 2. L: 왼쪽 탑
 3. R: 오른쪽 탑
5. 그럼 이 상황에서는 어떤 걸 이끌어 낼 수 있을까?
 1. dp[L][R] 을 새워보자.
 1. L 과 R 이 둘 다 최대 500,000이 될 수 있다.
 2. 이때 만들어 질 수 있는 경우의 수는 3^{50} 꼴일 거다. 이 모든 경우의 수를 저 matrix 안에 넣는다고 하면 시간 부터가 비효율일 거다.
 2. dp[R][L]
 1. wolog
 3. dp[N][L]
 1. N 이 최대 50이니 될 만 하다고 느낄 수 있다.
 2. 그럼 자연스레 value 에는 R 과 관련된 무언가가 들어갈 텐데 여전히 까다롭다.
6. 압축과 변형
 1. 압축을 하는 데에는 특히 최댓값 최솟값이 아주 좋다. dp 에서는 최대 최솟값을 value 에 연관되어 푸는 경우가 아주 많다. 그럼 value 에는 탑의 최댓값을 넣는 게 좋지 않을까?

2. 탑은 두 개고 symmetric 관계에 있는 경우를 생각하자. L 과 R 을 스왑하면서 풀어야 할까?

7. 변형

1. $\max(x,y)$
 2. $\text{abs}(x-y)$
 3. N
8. 이렇게 세 변수로 변형하면 모든 경우의 수를 나타낼 수 있다. 그리고 특히 symmetric 성질을 생각하지 않아도 max 에 따라 고민하면 된다.
9. $\text{dp}[N][\text{abs}(x-y)] = \max(x,y)$
1. max 는 항상 value 에 가면 좋다. value 값에 max 를 염두에 지수의 시간복잡도를 다행의 시간복잡도로 끌어내리는 경우가 아주 많다.
 2. N 이 최대 50이고 abs(x-y)는 최대 500,000이기에 매력적이다.

문제 풀이 (BOJ 13263 나무 자르기)

(문제 분석 및 프로그램 설계)

문제

높이가 a_1, a_2, \dots, a_n 인 나무 n개를 전기톱을 이용해서 자르려고 한다.

i 번 나무에 전기톱을 사용할 때마다 그 나무의 높이는 1만큼 감소한다. 전기톱은 사용할 때마다 충전해야 한다. 전기톱을 충전하는 비용은 완전히 자른 나무의 번호에 영향을 받는다. 즉, 높이가 0이 되어버린 나무의 번호에 영향을 받는다. 완전히 잘려진 나무의 번호 중 최댓값이 i 이면, 전기톱을 충전하는 비용은 b_i 이다. 완전히 잘려진 나무가 없다면 전기톱은 충전할 수가 없다. 가장 처음에 전기톱은 충전되어져 있다.

나무의 높이 a_i 와 각각의 나무에 대한 충전 비용 b_i 가 주어졌을 때, 모든 나무를 완전히 자르는데 (높이를 0으로 만드는데) 필요한 충전 비용의 최솟값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 $n(1 \leq n \leq 100,000)$ 이 주어진다. 둘째 줄에는 a_1, a_2, \dots, a_n , 셋째 줄에는 b_1, b_2, \dots, b_n 주어진다. ($1 \leq a_i \leq 10^9, 0 \leq b_i \leq 10^9$)

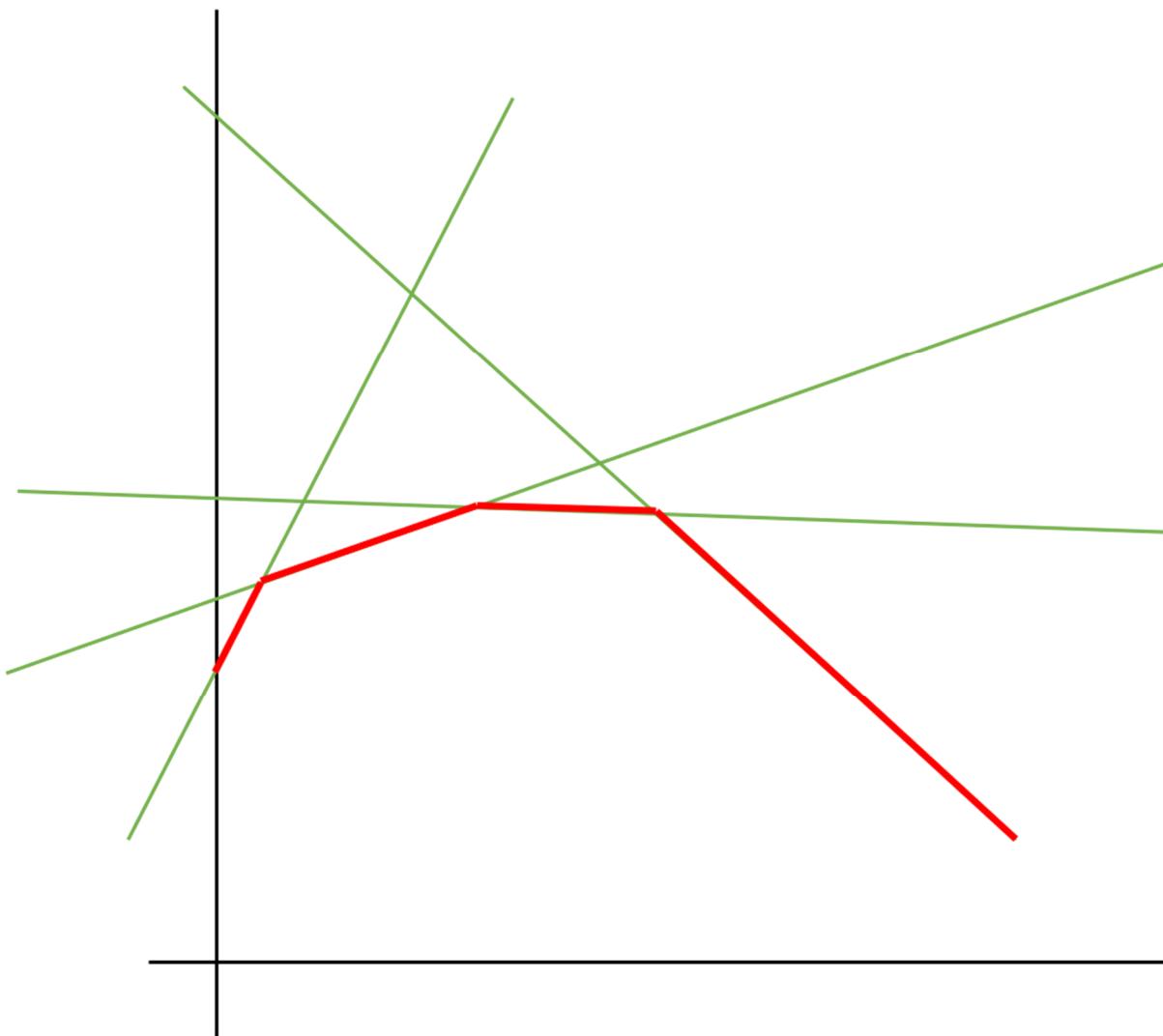
$a_1 = 1$ 이고, $b_n = 0$ 이며, $a_1 < a_2 < \dots < a_n, b_1 > b_2 > \dots > b_n$ 을 만족한다.

출력

나무를 완전히 자르는 충전 비용의 최솟값을 출력한다.

(문제를 해결하는데 필요한 개념)

컨벡스헬dp 알고리즘을 알고 있어야 한다.



(소스 코드)

CPP

```
#include<iostream>
#include<vector>
#include<utility>
#include<algorithm>
#define ll long long
using namespace std;
ll a[100'001];
ll b[100'001];
ll dp[100'001];
vector<pair<double, int>> s;
int main(void) {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int N;
    cin>N;
    for(int i=0;i<N;i++) cin>>a[i];
    for(int i=0;i<N;i++) cin>>b[i];
    s.push_back({0,0});
    //dp[0]=b[0];
    // b[j]*a[i]+dp[j]
    for(int i=1;i<N;i++) {
```

```

int curx=a[i];
int point=upper_bound(s.begin(),s.end(),make_pair((double)curx,0))-s.begin()-1;

dp[i]=dp[s[point].second]+a[i]*b[s[point].second];

double x=-((double)dp[s.back().second]-dp[i])/(b[s.back().second]-b[i]);
while(x<s.back().first){
    s.pop_back();
    x=-((double)dp[s.back().second]-dp[i])/(b[s.back().second]-b[i]);
}

s.push_back({x,i});
}
// for(int i=0;i<N;i++)cout<< dp[i]<<" ";
cout<<dp[N-1];
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

컨버스헐 dp 를 배워봤다. dp 식이 $ax+b$ 로 일반화 할 수 있을 때, 그에 기하적인 방법으로 그러한 일차 방정식들이 있는 그래프들을 상상할 수 있다. 이때 모든 x 에 대하여 최댓값이나 최솟값인 부분은 컨버스 헐처럼 ‘보인다’. 이때 $ax+b$ 를 어떻게 일반화 할 수 있는지가 중요한 거 같다. 그리고 이때 x 를 찾는 방법또한 이분탐색, 그리고 최댓값을 보장하는 일차 방정식들을 stack에 쌓아가면서 체크하는 게 중요하다.



스터디 5주차 보고서

(8/03 ~ 8/10)

작성자 : 성준영

작성일 : 2025.08.10

문제 풀이 (BOJ 16404 주식회사 승범이네)

(문제 분석 및 프로그램 설계)

문제

승범이는 평소 래퍼 도끼를 흔모해왔지만, 도끼만큼 랩을 잘할 수 없다는 것을 깨닫고 도끼만큼 돈이라도 벌자는 결심을 한다. 그래서 휴학 후 (주)승범이네를 창업했다.

(주)승범이네는 판매원들로만 이루어진 다단계 회사이다.

승범이를 제외한 모든 판매원은 사수가 배정되는데, 사수는 한 회원당 단 한 명씩만 배정된다. 만약 판매원 A가 B의 사수라면, B를 A의 부사수라고 부른다.

(주)승범이네의 수익구조는 기형적이다.

판매원들은 제품을 자비로 사서 판매한다. 이때 제품을 구매가격보다 저렴하게 판매하게 되면 손해를 보게 되는데, 어떤 회원 A가 손해를 보면 그 회원의 모든 부사수도 같은 만큼의 손해를 보게 된다. 그러면 부사수들의 부사수들도 손해를 보게 되고, 그들의 부사수들도 손해를 보게 되고, ..., 결국 A와 A 밑의 모든 판매원이 A가 잃은 만큼의 손해를 보게 된다.

반대로 판매원 A가 제품을 비싸게 팔아 이익이 생길 경우, A와 A 밑의 모든 판매원이 같은 이익을 얻을 수 있다.

승범이는 직원들이 현재 얼마만큼 돈을 벌었는지 감시하기 위해 다음 두 종류의 명령을 처리하려고 한다.

1 $i w$: 직원 i 가 w 만큼 이익/손해를 본다. (이익은 양수로, 손해는 음수로 주어진다)

2 i : 직원 i 의 현재 통장 잔액을 출력한다.

직원들은 빈 통장을 갖고 일을 시작하며, 이익과 손해가 실시간으로 통장에 기록된다. 물론 통장 잔액은 음수일 수도 있다.

일을 시작하기 직전에 플레이님 승급전을 남겨두고 온 것을 떠올린 승범이는 우리에게 일을 맡기고 집으로 달려가버렸다.

만년 골드 승범이를 위해 문제를 대신 해결해주자.

입력

첫 번째 줄에 승범이를 포함한 판매원들의 수 $N(1 \leq N \leq 100,000)$, 명령의 수 $M(1 \leq M \leq 100,000)$ 이 주어진다. 판매원들은 1번부터 N 번까지 번호가 매겨지며, 승범이는 항상 1번이다.

두 번째 줄에 판매원 1번부터 N 번까지의 사수가 순서대로 공백으로 구분되어 주어진다. 승범이는 사수가 없으므로 -1이 주어진다.

세 번째 줄부터 M 개의 줄에 걸쳐 위에서 설명한 명령(i, w 는 정수, $1 \leq i \leq N$, $-10,000 \leq w \leq 10,000$)이 주어진다.

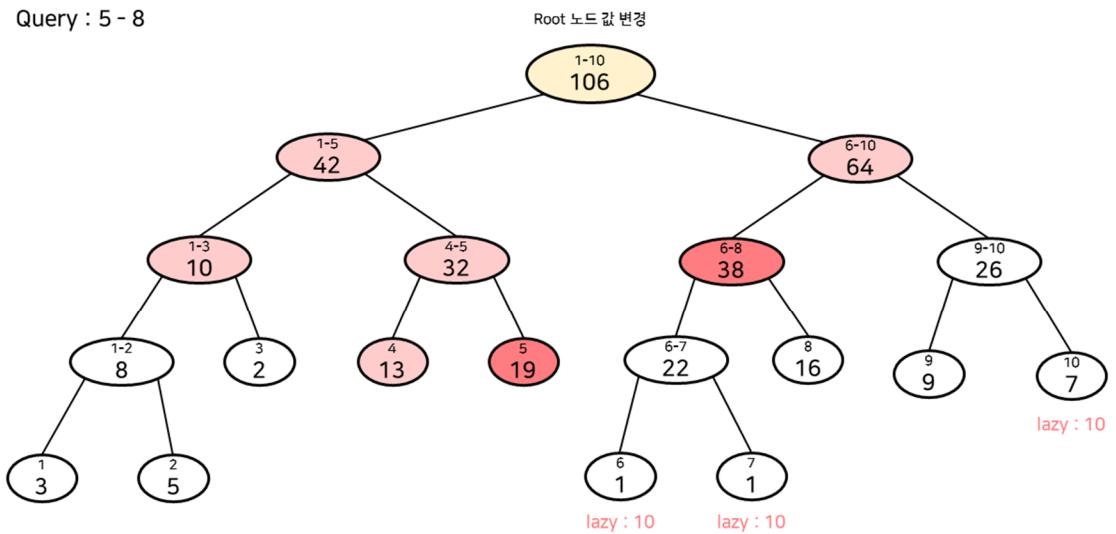
출력

2번 명령이 주어질 때마다 한 줄에 하나씩 해당하는 직원 i 의 잔고 상황을 출력한다.

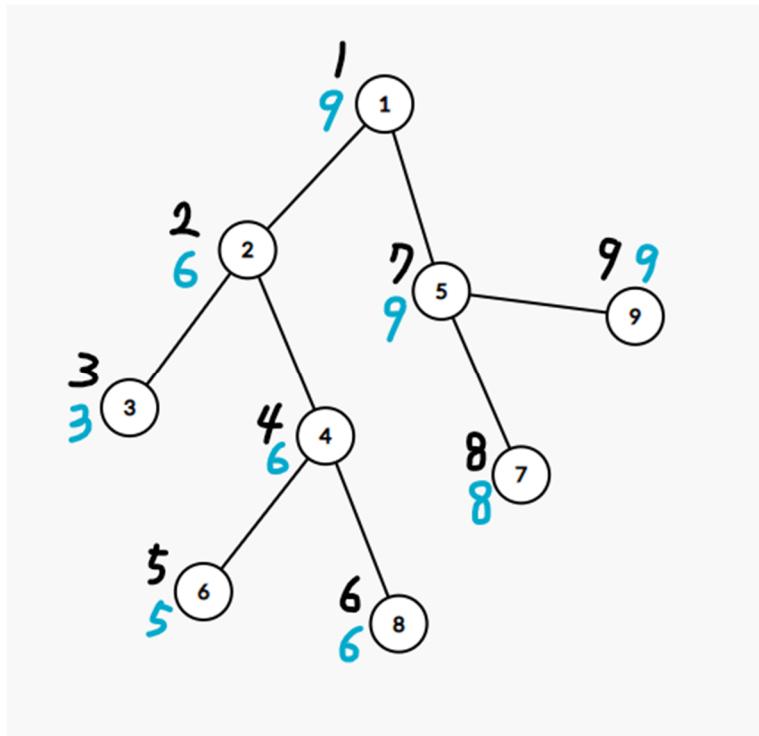
(문제를 해결하는데 필요한 개념)

세그먼트트리

Query : 5 - 8



오일러 경로 테크닉



(소스 코드)

CPP

```
#include<iostream>
#include<vector>
#include<cmath>
#define ll long long
using namespace std;
ll seg_tree[300'000];
ll lazy[300'000];
vector<int> adj[100'001];
```

```

int ind_arr[100'001]; // 사람->오일러
int range[100'001]; // 사람->오일러
int N,evenN;
int ind=-1;
int init_dfs(int x);
void lazy_func(int x,int h){
    int many=min((1<<evenN)+N, ((x+1)<<(evenN-h))-1)-((x)<<(evenN-h))+1;
    seg_tree[x]+=lazy[x]*many;
    if(x*2<=(1<<evenN)+N) lazy[x*2]+=lazy[x];
    if(x*2+1<=(1<<evenN)+N) lazy[x*2+1]+=lazy[x];
    lazy[x]=0;
}

void setting_dfs(int x,int h,int s,int e,int item){
    if(lazy[x]) lazy_func(x,h);

    int ind1=((x)<<(evenN-h));
    int ind2=((x+1)<<(evenN-h))-1;

    if(s<=ind1 and ind2<=e) lazy[x]+=item;
    else if(ind2<s or e<ind1) return;
    else {
        if(x*2<=(1<<evenN)+N) setting_dfs(x*2,h+1,s,e,item);
        if(x*2+1<=(1<<evenN)+N) setting_dfs(x*2+1,h+1,s,e,item);
    }
}

void find_dfs(int x,int h,int s,int e){
    if(lazy[x]) lazy_func(x,h);

    int ind1=((x)<<(evenN-h));
    int ind2=((x+1)<<(evenN-h))-1;

    if(s<=ind1 and ind2<=e) {
        cout<<seg_tree[x]<<"\n";
        return;
    }
    else if(ind2<s or e<ind1) return;
    else {
        if(x*2<=(1<<evenN)+N) find_dfs(x*2,h+1,s,e);
        if(x*2+1<=(1<<evenN)+N) find_dfs(x*2+1,h+1,s,e);
    }
}

int main(void){
    ios::sync_with_stdio(0);
    cin.tie(0);
    int Q;
    cin>>N>>Q;
    evenN=(N==1)?0:(log2(N-1)+1);
    for(int i=1;i<=N;i++) {
        int in;cin>>in;
        if(in==-1) continue;
        adj[in].push_back(i);
    }
    init_dfs(1);
    while(Q--){
        int op;cin>>op;
        if(op==1){
            int x,item;
            cin>>x>>item;
            int s=(1<<evenN)+ind_arr[x];
            int e=(1<<evenN)+range[x];
            // cout<<range[x]<<" ";
            setting_dfs(1,0,s,e,item);
        }
        else{
            int x;
        }
    }
}

```

```

        cin>>x;
        x=(1<<evenN)+ind arr[x];
        find_dfs(1,0,x,x);
    }
}
int init_dfs(int x) {
    ind++;
    ind_arr[x]=ind;
    int maxnum=ind;
    for(int i=0;i<adj[x].size();i++) {
        int nx=adj[x][i];
        maxnum=max(maxnum,init_dfs(nx));
    }
    range[x]=maxnum;
    return maxnum;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

1. 오일러 경로는 트리 상에서 부노의 자식간의 관계를 구간으로 만들어낼 수 있게끔 할 수 있다. 평상시에는 아무 생각없이 구현해 풀어왔는데 이름이 있는지 몰랐다
2. 돈을 벌면 대충 체크해두고, 쿼리를 요구할 때 특정 노드의 자식들을 한번에 구간으로 체크. 이때 세그먼트가 쓰인다

문제 풀이 (BOJ 4002 특공대)

(문제 분석 및 프로그램 설계)

문제

1부터 n 까지 번호가 붙여진 n 명의 병사들로 이루어진 군대의 지휘관이 있다. 이 지휘관은 앞으로의 전투를 위하여 n 명의 병사들을 여러 개의 특공대로 나누고자 한다. 결속력과 사기를 높이기 위하여 각 특공대는 $\{i, i+1, \dots, i+k\}$ 형태의 번호가 연속하는 병사들로 구성된다.

각 병사 i 의 전투력은 x_i 이다. 병사들 $\{i, i+1, \dots, i+k\}$ 로 구성된 특공대의 전투력 x 는 원래는 각 병사의 전투력의 합으로 계산되었다. 달리 말하면 $x = x_i + x_{i+1} + \dots + x_k$ 이었다.

그러나 여러 해의 영광스러운 승리를 통하여 특공대의 전투력을 다음과 같이 조정해야 하는 것으로 결론을 내렸다: 특공대의 조정된 전투력 x' 는 등식 $x' = ax^2 + bx + c$ 로 계산한다. 여기서 a, b, c 는 알려져 있는 계수들로서 $a < 0$ 이고, x 는 특공대의 원래 정의된 전투력이다.

여러분이 할 일은 모든 특공대의 조정된 전투력의 합을 최대화하도록 병사들을 특공대로 나누는 것이다.

예를 들어, 4명의 병사들이 있고, 각 병사의 전투력 $x_1 = 2, x_2 = 2, x_3 = 3,$

$x_4 = 4$ 라 하자. 특공대의 조정된 전투력 등식에 있는 계수가 $a=-1, b=10, c=-20$ 이라 하자. 이러한 경우, 최적인 해는 병사들을 다음과 같이 세 개의 특공대로 나누는 것이다: 첫 번째 특공대는 병사 1과 2로 구성하고, 두 번째 특공대는 병사 3으로 구성하고, 세 번째 특공대는 병사 4로 구성한다. 이들 세 특공대의 원래의 전투력은 각각 4, 3, 4이고 조정된 전투력은 각각 4, 1, 4이다. 이렇게 나눌 때 조정된 전체 전투력은 각 특공대의 조정된 전투력의 합인 9이며, 이보다 더 좋은 해가 없음을 알 수 있다.

입력

입력은 세 줄로 구성된다. 첫 번째 줄에 전체 병사들 수인 양의 정수 n 이 주어진다. 두 번째 줄에 특공대의 조정된 전투력 계산 등식의 계수인 세 정수 a, b, c 가 주어진다. 마지막 줄에 병사들 $1, 2, \dots, n$ 의 전투력을 나타내는 n 개의 정수 x_1, x_2, \dots, x_n 이 공백을 사이에 두고 주어진다.

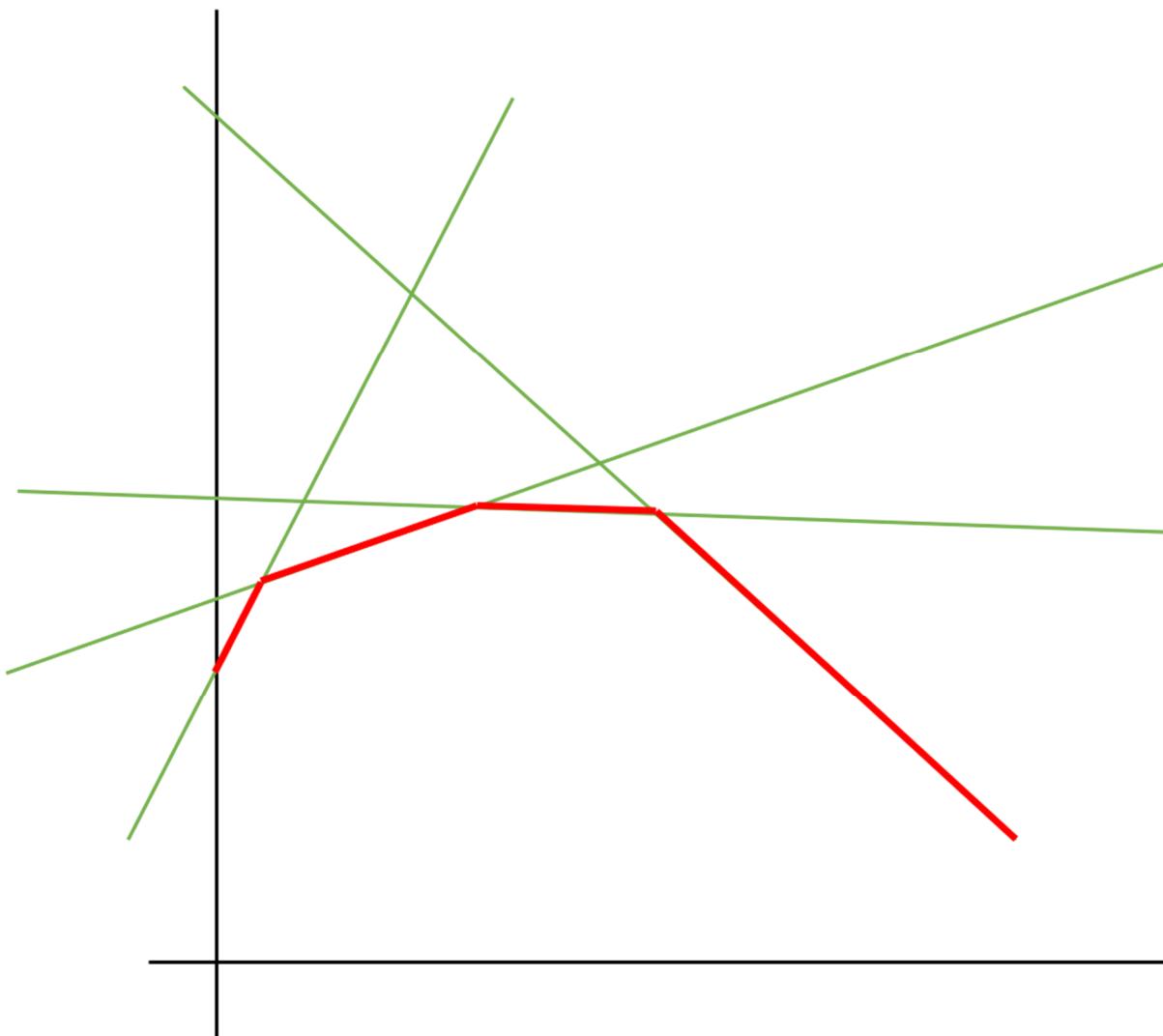
$$n \leq 1,000,000, -5 \leq a \leq -1, |b| \leq 10,000,000, |c| \leq 30,000,000, 1 \leq x_i \leq 100$$

출력

얻을 수 있는 최대의 조정된 전체 전투력을 나타내는 하나의 정수를 한 줄에 출력한다.

(문제를 해결하는데 필요한 개념)

컨벡스헬dp 알고리즘을 알고 있어야 한다.



(소스 코드)

CPP

```
#include<iostream>
#include<vector>
#include<algorithm>
#include<utility>
#define ll long long
using namespace std;
ll A,B,C;
ll sum[1'000'001];
ll dp[1'000'001];
vector<pair<double,int>> v;
ll a(int i){
    ll tmp=-2LL*A*sum[i];
    return tmp;
}
ll b(int i){
    ll tmp=A*sum[i]*sum[i]-B*sum[i]+dp[i];
    return tmp;
}
bool bigger(int i){
    double l=v.back().first;
    double r=-( (double)b(i)-b(v.back().second) ) / (a(i)-a(v.back().second) );
    if (l < r)
        return true;
    else
        return false;
}
```

```

        return r<1;
    }
int main(void){
    ios::sync_with_stdio(0);
    cin.tie(0);
    int N;
    cin>>N>>A>>B>>C;
    for(int i=1;i<=N;i++) {
        cin>>sum[i];
        sum[i]+=sum[i-1];
    }
    v.push_back({0,0});
    for(int i=1;i<=N;i++) {
        int y=upper_bound(v.begin(),v.end(),make_pair((double)sum[i],0))-v.begin()-1;
        dp[i]=a(v[y].second)*sum[i]+b(v[y].second);
        dp[i]+=A*sum[i]*sum[i]+B*sum[i]+C;
        while(bigger(i)){
            v.pop_back();
        }
        double x=-((double)b(i)-b(v.back().second))/(a(i)-a(v.back().second));
        v.push_back({x,i});
    }
    cout<<dp[N];
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

컨버스헬 dp 를 배워봤다. dp 식이 $ax+b$ 로 일반화 할 수 있을 때, 그에 기하적인 방법으로 그러한 일차 방정식들이 있는 그래프들을 상상할 수 있다. 이때 모든 x 에 대하여 최댓값이나 최솟값인 부분은 컨버스 헬처럼 ‘보인다’. 이때 $ax+b$ 를 어떻게 일반화 할 수 있는지가 중요한 거 같다. 그리고 이때 x 를 찾는 방법 또한 이분탐색, 그리고 최댓값을 보장하는 일차 방정식들을 stack에 쌓아가면서 체크하는 게 중요하다.

이때 이 문제에서 주는 수식을 일차 방정식으로 바꾸는 게 중요한데, x^2 는 계산에 영향이 안 가므로 \min 함수의 외부로 빼도된다. 이때 x 는 y 와 곱해져 있기에 함부로 빼면안 된다.



스터디 1주차 보고서

(7/06 ~ 7/13)

작성자 : 성준영

작성일 : 2025.07.13

문제 풀이 (BOJ 1214 쿨한 물건 구매)

(문제 분석 및 프로그램 설계)

문제

구사과는 지폐를 오직 두 종류만 가지고 있다. 바로 P원 지폐와 Q원 지폐이다. 이 두 종류의 지폐를 구사과는 무한대만큼 가지고 있다.

오늘 구사과가 구매하려고 하는 물건의 가격은 D원이다. 구사과가 이 물건을 구매하기 위해서 지불해야 하는 금액의 최솟값은 얼마일까?

물건을 구매하기 위해서는 물건의 가격보다 크거나 같은 금액을 지불해야 한다.

입력

첫째 줄에 D, P, Q가 주어진다. 모두 10^9 보다 작거나 같은 자연수이다.

출력

첫째 줄에 물건을 구매하기 위해 구사과가 지불해야 하는 금액의 최솟값을 출력한다.

(문제를 해결하는데 필요한 개념)

브루트포스 알고리즘은 가능한 모든 경우의 수를 대입하는 기법이다.

이때 푸는 방식이 크게 두 개인데,

1. 최대 공약수를 빠르게 구할 수 있는 알고리즘을 구현해서 브루트포스를 해야하는 range를 줄이거나
2. 수학적 성질을 통해서 브루트포스를 해야하는 range를 줄일 수 있다. 이때 b c 중 작은 수를 b라고 할 때 c에 곱해지는 수가 b에 분배 법칙을 적용할 수 있는지 고민해 보는 게 좋다.

(소스 코드)

CPP

```
#include<iostream>
#include<vector>
#define ll long long
using namespace std;
ll gcd(ll x,ll y){
    if(y==0) return x;
    else return gcd(y,x%y);
}
int main(void){
    ios::sync_with_stdio(0);
    cin.tie(0);
    ll a,b,c;
    cin>>a>>b>>c;
    if(b<c) swap(b,c);
    ll g=gcd(b,c);
    ll cur=b/g*c/g*g;
    // cout<<cur<<"\n";
    if(a>cur){
        cur=cur*(a/cur-1);
        a-=cur;
    }
    else{
        cur=0;
    }
    // cout<<g<<"\n";
    ll answer=2'000'000'000;
    for(ll i=a/b+1;i>=0;i--){
        ll tmpa=a-b*i;
```

```

    ll j=0;
    if(tmpa>0){
        j=tmpa/c;
        if(tmpa-j*c>0)j++;
    }
    answer=min(answer,b*i+c*j);
    if(answer==a)break;

}
cout<<answer+cur;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

최소공배수를 구하고 a/cur 이 아닌 $a/cur-1$ 을 했어야 했다.

문제 풀이 (BOJ 16975 수열과 쿼리 21)

(문제 분석 및 프로그램 설계)

길이가 N 인 수열 A_1, A_2, \dots, A_N 이 주어진다. 이때, 다음 쿼리를 수행하는 프로그램을 작성하시오.

- 1 i j k: A_i, A_{i+1}, \dots, A_j 에 k 를 더한다.
- 2 x: A_x 를 출력한다.

입력

첫째 줄에 수열의 크기 N ($1 \leq N \leq 100,000$)이 주어진다.

둘째 줄에는 A_1, A_2, \dots, A_N 이 주어진다. ($1 \leq A_i \leq 1,000,000$)

셋째 줄에는 쿼리의 개수 M ($1 \leq M \leq 100,000$)이 주어진다.

넷째 줄부터 M 개의 줄에는 쿼리가 한 줄에 하나씩 주어진다. 1번 쿼리의 경우 $1 \leq i \leq j \leq N$, $-1,000,000 \leq k \leq 1,000,000$ 이고, 2번 쿼리의 경우 $1 \leq x \leq N$ 이다. 2번 쿼리는 하나 이상 주어진다.

출력

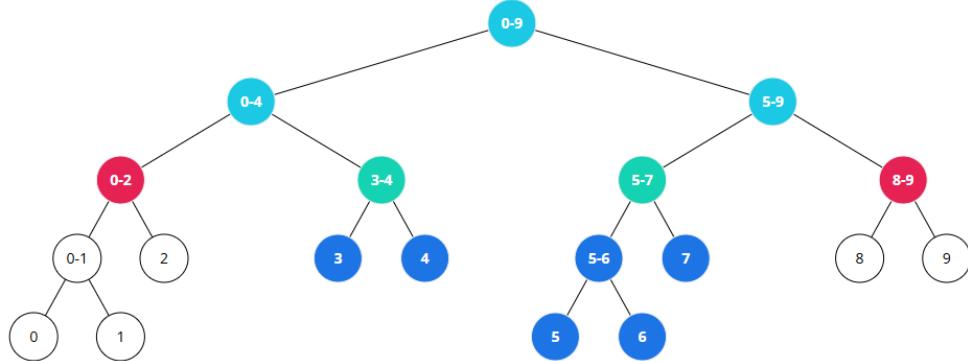
2번 쿼리가 주어질 때마다 출력한다.

(문제를 해결하는데 필요한 개념)

느리게 갱신되는 세그먼트 트리

변경해야 하는 노드

다음은 $N = 10$, $\text{left} = 3$, $\text{right} = 7$ 인 경우 변경해야 하는 노드를 표시했습니다.



은 $[\text{left}, \text{right}]$ 를 변경하기 위해서 합을 변경해야 하는 노드입니다.
여기서 을 루트로하는 서브트리는 모든 노드에 들어있는 합을 변경해야 합니다. 이런 경우 은 합을 변경하지 않고 나중에 다시 변경을 수행하려 그 노드에 방문했을 때 변경을 진행해도 됩니다.

갱신을 미루어도 되는 데이터는 갱신을 미루고, 당장 써야 하는 데이터는 갱신을 하면서 데이터를 변경 및 탐색을 하는 알고리즘이다.

(소스 코드)

CPP
<pre>#include<iostream> #include<cmath> using namespace std; long long arr[300'000]; long long lazy[300'000]; int N, evenN, degree; int many(int x, int h){ return min(degree+N, (x+1)<<(evenN-h)-(x<<(evenN-h))); } void lazy_update(int x, int h){ if(lazy[x]){ arr[x] += many(x, h)*lazy[x]; if(x*2<(degree)*2) lazy[x*2] += lazy[x]; if(x*2+1<(degree)*2) lazy[x*2+1] += lazy[x]; lazy[x] = 0; } } void update_DFS(int x, int h, int l, int r, int k){ lazy_update(x, h); int ind1 = (x<<(evenN-h)); int ind2 = ((x+1)<<(evenN-h))-1; if(ind2 < l r < ind1) return; if(l <= ind1 & ind2 <= r) lazy[x] += k; else{ arr[x] += (min(r, ind2)-max(ind1, l)+1)*k;//여집합 if(x*2<degree+N) update_DFS(x*2, h+1, l, r, k); if(x*2+1<degree+N) update_DFS(x*2+1, h+1, l, r, k); } }</pre>

```

void update_func(void){
    int i,j,k;
    cin>>i>>j>>k;
    i--;j--;
    update_DFS(1,0,degree+i,degree+j,k);
}
void print_DFS(int x,int h,int l,int r){//cover range, target range
    lazy_update(x,h);
    int ind1=(x<<(evenN-h)); // cover range
    int ind2=((x+1)<<(evenN-h))-1;
    if(ind2<l or r<ind1) return;
    if(l<=ind1 and ind2<=r){
        cout<<arr[x]<<"\n";
        return;
    }
    else{
        if(x*2<degree+N)print_DFS(x*2,h+1,l,r);
        if(x*2+1<degree+N)print_DFS(x*2+1,h+1,l,r);
    }
}
void print_func(void){
    int x;cin>>x;x--;
    print_DFS(1,0,degree+x,degree+x);
}
int main(void){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin>>N;
    evenN=(N==1)?0:(log2(N-1)+1);
    degree=(1<<evenN);
    for(int i=0;i<N;i++)cin>>arr[degree+i];
    for(int h=degree/2;h>0;h/=2)for(int i=h;i<h*2;i++)arr[i]=arr[i*2]+arr[i*2+1];
    int Q;cin>>Q;
    while(Q--){
        int op;cin>>op;
        if(op==1)update_func();
        else print_func();
    }
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

1. Int N 을 전역변수에 선언을 하고 실수로 main 문 안에 int N 을 다시 선언을 했다.
그래서 외부 함수들을 쓸 때 N 이 계속 0 인 상태로 구현이 되어서 많이 혼났다.
2. Degree 를 이번에 처음 써 봤는데, 이제부터 유용하게 쓸 거 같다.
3. if(x*2<degree+N)update_DFS(x*2,h+1,l,r,k);
이렇게 재귀 함수 함수의 조건을 달았는데, 습관이 되어야 할 거 같다.
4. arr[x]+=(min(r,ind2)-max(ind1,l)+1)*k;//여집합
여집합을 체크하는 테크닉이 기본으로 쓸 수 있어야 한다. 이걸 놓쳐서 개신이 이상하게 됐다.

```
5. int many(int x,int h){  
    return min(degree+N,(x+1)<<(evenN-h))-(x<<(evenN-h));  
}
```

이렇게 many 문을 써 봤는데, 괜찮은 거 같다. 딱 알맞은 노드에 대해서 밑의 애들의 수를 찾아준다. 이때 min 의 테크닉을 집중해야 할 거 같다.

6. 래이지를 업데이트 하는 조건문을 어디에 둘까 고민했다.

7. 래이지를 구현하는 속도가 매우 느리다. 흄흣..



스터디 1주차 보고서

(7/7 ~ 7/13)

작성자 : 김승우

작성일 : 2025.07.20

문제 풀이 (BOJ 2439 별 찍기 - 2)

(문제 분석 및 프로그램 설계)

문제

전형적인 별 찍기 문제이다.

원하는 줄의 수를 먼저 입력 받으면 첫째 줄부터 마지막 줄까지 별의 개수가 한 개씩 늘어나면서 피라미드의 절반 형태가 되도록 출력해야 한다.

입력

첫째 줄에 $N(1 \leq N \leq 100)$ 이 주어진다.

출력

첫째 줄부터 N 번째 줄까지 차례대로 별을 출력한다.

(문제를 해결하며 고안한 전략)

첫 줄에는 1개부터이다. 즉, n 번째 줄에는 별이 n 개 있어야 한다.

이 문제의 경우, 공백이 먼저 출력되고 별이 후에 출력되도록 설계되어야 하기 때문에 입력 받은 숫자 N 에 주목해야 한다.

공백의 수와 별의 수의 합이 N 이 나와야 하므로 이중으로 반복문을 사용하였다.

(소스 코드)

```
C
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void) {
    int a;

    scanf("%d", &a);

    for (int i = 0; i < a; i++) {
        for (int j = i; j < a - 1; j++) {
            printf(" ");
        }
        for (int k = 0; k <= i; k++) {
            printf("*");
        }
    }
}
```

```

        printf("\n");
    }
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

아직 프로그래밍 왕초보라 그런지 반복문을 이중으로 생각하자니 머리가 깨지는 줄 알았다. 그래도 직접 만든 코드가 원하는 대로 작동하니 기분이 좋다.

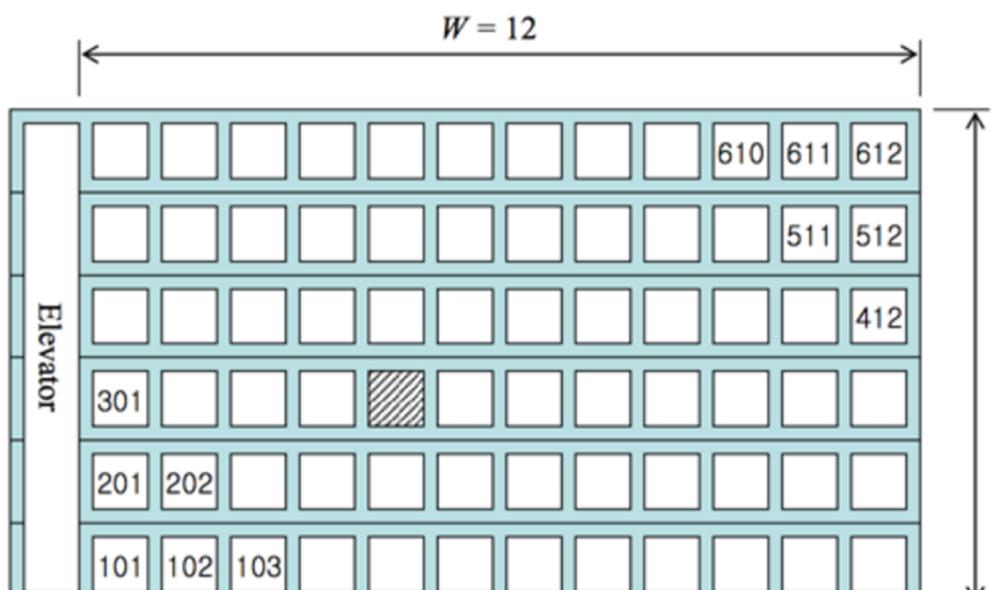
문제 풀이 (BOJ 10250 ACM 호텔)

(문제 분석 및 프로그램 설계)

문제

ACM 호텔 매니저 지우는 손님이 도착하는 대로 빈 방을 배정하고 있다. 고객 설문조사에 따르면 손님들은 호텔 정문으로부터 걸어서 가장 짧은 거리에 있는 방을 선호한다고 한다. 여러분은 지우를 도와 줄 프로그램을 작성하고자 한다. 즉 설문조사 결과 대로 호텔 정문으로부터 걷는 거리가 가장 짧도록 방을 배정하는 프로그램을 작성하고자 한다.

문제를 단순화하기 위해서 호텔은 직사각형 모양이라고 가정하자. 각 층에 W 개의 방이 있는 H 층 건물이라고 가정하자 ($1 \leq H, W \leq 99$). 그리고 엘리베이터는 가장 왼쪽에 있다고 가정하자(그림 1 참고). 이런 형태의 호텔을 $H \times W$ 형태 호텔이라고 부른다. 호텔 정문은 일층 엘리베이터 바로 앞에 있는데, 정문에서 엘리베이터까지의 거리는 무시한다. 또 모든 인접한 두 방 사이의 거리는 같은 거리(거리 1)라고 가정하고 호텔의 정면 쪽에만 방이 있다고 가정한다.



방 번호는 YXX 나 YYXX 형태인데 여기서 Y나 YY는 층 수를 나타내고 XX는 엘리베이터에서부터 세었을 때의 번호를 나타낸다. 즉, 그림 1에서 빛금으로 표시한 방은 305 호가 된다.

손님은 엘리베이터를 타고 이동하는 거리는 신경 쓰지 않는다. 다만 걷는 거리가 같을 때에는 아래층의 방을 더 선호한다. 예를 들면 102 호 방보다는 301 호 방을 더 선

호하는데, 102 호는 거리 2 만큼 걸어야 하지만 301 호는 거리 1 만큼만 걸으면 되기 때문이다. 같은 이유로 102 호보다 2101 호를 더 선호한다.

여러분이 작성할 프로그램은 초기에 모든 방이 비어있다고 가정하여 이 정책에 따라 N 번째로 도착한 손님에게 배정될 방 번호를 계산하는 프로그램이다. 첫 번째 손님은 101 호, 두 번째 손님은 201 호 등과 같이 배정한다. 그림 1의 경우를 예로 들면, H = 6이므로 10 번째 손님은 402 호에 배정해야 한다.

입력

프로그램은 표준 입력에서 입력 데이터를 받는다. 프로그램의 입력은 T 개의 테스트 데이터로 이루어져 있는데 T 는 입력의 맨 첫 줄에 주어진다. 각 테스트 데이터는 한 행으로서 H, W, N, 세 정수를 포함하고 있으며 각각 호텔의 층 수, 각 층의 방 수, 몇 번째 손님인지를 나타낸다($1 \leq H, W \leq 99, 1 \leq N \leq H \times W$).

출력

프로그램은 표준 출력에 출력한다. 각 테스트 데이터마다 정확히 한 행을 출력하는데, 내용은 N 번째 손님에게 배정되어야 하는 방 번호를 출력한다.

(문제를 해결하며 고안한 전략)

어려워 보이지만 생각보다 단순한 문제이다.

사람들은 가까운 방을 항상 선호하기 때문에 모든 방이 비어 있는 이 시점에서는 쉽게 문제가 풀린다. 그냥 101, 201, 301, 401, ~~ 이 순서로 사람들이 선호하며 이는 앞쪽 세로 줄부터, 아래에서 위 순서로 배정된다는 규칙이 있다.

(소스 코드)

```
C
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main() {
    int T, H[100], W[100], N[100], Y[100], X[100];

    scanf("%d", &T);

    for (int i = 0; i < T; i++) {
        scanf("%d %d %d", &H[i], &W[i], &N[i]);
    }
    for (int j = 0; j < T; j++) {
        if (N[j] % H[j] != 0) {
            Y[j] = N[j] % H[j];
            X[j] = N[j] / H[j] + 1;
        } else {
            Y[j] = H[j];
            X[j] = N[j] / H[j];
        }
        printf("%d\n", Y[j] * 100 + X[j]);
    }
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

처음에 제출했을 때 틀렸다고 나와 당황했다. 전혀 문제될 것이 없는 완벽한 답안이라고 생각했는데 알고 보니 배열은 0에서 시작한다는 사실을 까먹고 1부터 생각한 것이었다. 정말 바보였다.

또한, 마지막에 방의 호수를 출력할 때 아예 숫자로 만들어 출력할 수도 있지만 그냥 %d%d 로 해서 출력하는 방법도 있다는 것을 뒤늦게 알았다. 그렇게 하면 조금이라도 길이가 줄었을 것 같은데.



스터디 2주차 보고서

(7/14 ~ 7/20)

작성자 : 김승우

작성일 : 2025.07.20

문제 풀이 (BOJ 3052 나머지)

(문제 분석 및 프로그램 설계)

문제

두 자연수 A와 B가 있을 때, A%B는 A를 B로 나눈 나머지이다. 예를 들어, 7, 14, 27, 38을 3으로 나눈 나머지는 1, 2, 0, 2이다.

수 10개를 입력 받은 뒤, 이를 42로 나눈 나머지를 구한다. 그 다음 서로 다른 값이 몇 개 있는지 출력하는 프로그램을 작성하시오.

입력

첫째 줄부터 열 번째 줄 까지 숫자가 한 줄에 하나씩 주어진다. 이 숫자는 1,000보다 작거나 같고, 음이 아닌 정수이다.

출력

첫째 줄에, 42로 나누었을 때, 서로 다른 나머지가 몇 개 있는지 출력한다.

(문제를 해결하며 고안한 전략)

어떻게 하면 개수를 세도록 할 수 있을지가 관건이었다. 그리고 어떻게 하면 간편하고 직관적으로 할 수 있을까가 고민되었다.

생각해낸 방법은 어차피 42로 나눈 나머지는 0~41까지로 한정되니, 42개의 배열 ($b[0] \sim b[41]$)을 만들어서 0으로 초기값을 넣어주었다. 각 나머지의 값(n)이 나올 때 $b[n]$ 에 해당하는 값을 1로 정의했고, 이후 마지막에 $b[0] \sim b[41]$ 에 들어있는 값을 모두 더해주기만 하면 나머지가 몇 종류 나왔는지 알 수 있게 된다.

(소스 코드)

```
C
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main() {
    int a[10];
    int b[42];
    int n;
    int count = 0;

    for (int j = 0; j < 42; j++) {
        b[j] = 0;
```

```

}
for (int i = 0; i < 10; i++) {
    scanf("%d", &a[i]);
    n = a[i] % 42;
    b[n] = 1;
}
for (int k = 0; k < 42; k++) {
    count += b[k];
}
printf("%d", count);
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

직접적으로 개수를 세는 것이 아니라 약간 간접적으로 표현할 방법을 찾는 과정이 재미있었고 완성하니 굉장히 뿌듯했다.

문제 풀이 (BOJ 10951 A+B - 4)

(문제 분석 및 프로그램 설계)

문제

두 정수 A와 B를 입력 받은 다음, A+B를 출력하는 프로그램을 작성하시오.

입력

입력은 여러 개의 테스트 케이스로 이루어져 있다.

각 테스트 케이스는 한 줄로 이루어져 있으며, 각 줄에 A와 B가 주어진다. ($0 < A, B < 10$)

예제 입력 1 복사

```

1 1
2 3
3 4
9 8
5 2

```

예제 출력 1 복사

```

2
5
7
17
7

```

출력

각 테스트 케이스마다 A+B를 출력한다.

(문제를 해결하며 고안한 전략)

EOF의 개념을 배울 수 있는 기회였다.

(소스 코드)

```
C
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main() {
    int a, b;

    for (int i = 0; scanf("%d %d", &a, &b) != EOF; i++) {
        printf("%d\n", a + b);
    }
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

솔직히 굉장히 쉬운 문제이고 보고서에 올릴 수 있는 다른 문제들도 많았지만 이 문제를 선정한 이유는 다음과 같다.,

EOF라는 처음 듣는 개념을 배울 수 있었다.

지금까지는 무언가 입력을 받으면 입력 받는 값의 개수가 정해져 있거나 '나 여기까지만 입력할 거예요'라는 표시가 되어주는 값을 입력하는 식으로 입력 받는 양을 한정 지어줄 수 있었는데 이 문제는 얼마나 입력 받을 것인지 정해져 있지 않아서 난관이었다.

열심히 찾아본 결과 방법을 알아낼 수 있었고 쉬운 문제였지만 뜻밖의 큰 성과가 있어서 보고서에 실었다.



스터디 3주차 보고서

(7/21 ~ 7/27)

작성자 : 김승우

작성일 : 2025.07.29

문제 풀이 (BOJ 1676 팩토리얼 0의 개수)

(문제 분석 및 프로그램 설계)

문제

N!에서 뒤에서부터 처음 0이 아닌 숫자가 나올 때까지 0의 개수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N이 주어진다. ($0 \leq N \leq 500$)

출력

첫째 줄에 구한 0의 개수를 출력한다.

(문제를 해결하며 고안한 전략)

뒤에 0이 온다는 건 곧 10의 배수라는 것!

10의 배수라는 것은 곧 2와 5의 배수라는 것. 그런데 팩토리얼을 계산할 때 2번에 1번 꼴로 곱하는 것이 짹수이기에 2는 충분히 많을 것이다. 그렇다면 핵심은 5의 개수! 즉, 뒷자리 0의 개수는 곱해진 5의 개수와 같다.

5의 개수를 구하기 위해 1부터 입력 받은 n까지 모든 숫자에서 각 숫자가 5로 몇 번 나누어 떨어지는지 확인하고자 함.

(소스 코드)

```
C
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void) {
    int n;
    int count = 0;
    int a;

    scanf("%d", &n);

    for (int i = 1; i <= n; i++) {
        a = i;
        if (a % 5 == 0) {
            count += 1;
            for (; (a / 5) % 5 == 0;) {
                a = a / 5;
                count += 1;
                if (a == 0) {
                    break;
                }
            }
        }
        else {
            continue;
        }
    }
    printf("%d", count);

    return 0;
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

처음으로 해결한 실버 티어의 문제여서 굉장히 뿌듯했고 무작정 10으로 나누어서 세는 것은 바이트 수의 한계가 있을 것이라 생각하고 5의 개수를 구해야 한다는 접근을 이끌어낸 자신이 대견했다.

문제 풀이 (BOJ 2609 최대공약수와 최소공배수)

(문제 분석 및 프로그램 설계)

문제

두 개의 자연수를 입력 받아 최대 공약수와 최소 공배수를 출력하는 프로그램을 작성 하시오.

입력

첫째 줄에는 두 개의 자연수가 주어진다. 이 둘은 10,000이하의 자연수이며 사이에 한 칸의 공백이 주어진다.

출력

첫째 줄에는 입력으로 주어진 두 수의 최대공약수를, 둘째 줄에는 입력으로 주어진 두 수의 최소 공배수를 출력한다.

(문제를 해결하며 고안한 전략)

조건문을 이용해 나머지가 0이냐 아니냐로 나누어 구성하였고 조건에 적합한 값이 나올 때까지 구하기 위해 반복문을 이용했다.

(소스 코드)

```
C
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void) {
    int a, b, n;
    int max, min;

    scanf("%d %d", &a, &b);

    if (a - b >= 0) {
        n = b;
    }
    else {
        n = a;
    }
    for (int i = 1; i <= n; i++) {
        if ((a % i == 0) && (b % i == 0)) {
            max = i;
        }
        else {
            continue;
        }
    }
}
```

```
for (int j = n; ;j++) {  
    if ((j % a == 0) && (j % b == 0)) {  
        min = j;  
        break;  
    }  
    else {  
        continue;  
    }  
}  
  
printf("%d\n", max);  
printf("%d", min);  
  
return 0;  
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

인간은 직접 생각하고 판단하는 능력 덕분에 별다른 과정 없이 직접적으로 최대공약수, 최소공배수 등의 값을 구할 수 있지만, 컴퓨터는 그런 걸 내가 직접 하나하나 다 구축 시켜줘야 해서 컴퓨터라는 것은 정말 똑똑한 것이 맞는 건지 의문이 드는 시간이었다.



스터디 4주차 보고서

(7/28 ~ 8/3)

작성자 : 김승우

작성일 : 2025.08.03

문제 풀이 (BOJ 14626 ISBN)

(문제 분석 및 프로그램 설계)

문제

ISBN(International Standard Book Number)은 전 세계 모든 도서에 부여된 고유번호로, 국제 표준 도서번호이다. ISBN에는 국가명, 발행자 등의 정보가 담겨 있으며 13자리의 숫자로 표시된다. 그중 마지막 숫자는 체크기호로 ISBN의 정확성 여부를 점검할 수 있는 숫자이다. 이 체크기호는 일련번호의 앞에서부터 각 자리마다 가중치 1, 3, 1, 3....를 곱한 것을 모두 더하고, 그 값을 10으로 나눈 나머지가 0이 되도록 만드는 숫자 m을 사용한다. 수학적으로는 다음과 같다.

ISBN이 abcdefghijklm 일 때, $a+3b+c+3d+e+3f+g+3h+i+3j+k+3l+m \equiv 0 \pmod{10}$

즉, 체크기호 $m = 10 - (a+3b+c+3d+e+3f+g+3h+i+3j+k+3l) \bmod 10$ 이다.

단, 10으로 나눈 나머지 값이 0일 경우 체크기호는 0이다.

전북대학교 중앙도서관에서 사서로 일하고 있는 영훈이는 책 정리를 하다가 개구쟁이 광현이에 의해서 ISBN이 훼손된 도서들을 발견했다. 광현이 때문에 야근해야 하는 불쌍한 영훈이를 위해서 손상된 자리의 숫자를 찾아내는 프로그램을 작성해주자.

입력

ISBN 13자리 숫자가 입력된다. 훼손된 숫자는 *로 표시한다. (훼손된 일련번호는 체크기호를 제외한 무작위 한 자리이다.)

출력

훼손된 숫자 *에 알맞은 숫자를 출력한다.

(문제를 해결하며 고안한 전략)

*의 위치가 짝수 자리일 때와 홀수 자리일 때를 나누어 생각한다.

*을 제외한 모든 값의 합을 구한 뒤 미지수를 이용해 방정식을 세운다.

*은 어차피 0~9까지의 자연수 중 하나이므로 반복문을 이용해 대입해보고 알맞은 값을 찾는다.

(소스 코드)

```
C
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    char n[13];
    int x;
    int sum = 0;
    int N[13];

    for (int i = 0; i < 13; i++)
    {
        n[i] = 0;
        N[i] = 0;
    }

    scanf("%s", &n);

    for (int j = 0; j < 13; j++)
    {
        if (n[j] == '*')
        {
            x = j;
            continue;
        }
        N[j] = n[j] - '0';
        if (j % 2 == 0 && n[j] != '*')
        {
            sum += N[j];
        }
        else if (j % 2 != 0 && n[j] != '*')
        {
            sum += N[j] * 3;
        }
    }

    if (x % 2 == 0)
    {
        N[x] = 10 - sum % 10;
    }
    else
    {
        for (int k = 1; k <= 9; k++)
        {
            if (3 * k % 10 == 10 - sum % 10)

```

```

    {
        N[x] = k;
        break;
    }
    else
    {
        continue;
    }

}

if (sum % 10 == 0)
{
    N[x] = 0;
}

printf("%d\n", N[x]);

return 0;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

처음에는 0부터 9까지 대입해 찾는 방법이 마음에 들지 않았다.

풀이가 안 예뻤기 때문에 마음에 들지 않았기 때문이다.

하지만 다른 풀이를 떠올리기 쉽지 않은 문제였다. 그래서 그냥 저 방법을 사용했다.
하지만 여전히 마음에 들지 않는다.

더 깔끔하고 엘레강스하게 풀고 싶다.

문제 풀이 (BOJ 1259 팰린드롬수)

(문제 분석 및 프로그램 설계)

문제

어떤 단어를 뒤에서부터 읽어도 똑같다면 그 단어를 팰린드롬이라고 한다. 'radar', 'sees'는 팰린드롬이다.

수도 팰린드롬으로 취급할 수 있다. 수의 숫자들을 뒤에서부터 읽어도 같다면 그 수는 팰린드롬수다. 121, 12421 등은 팰린드롬수다. 123, 1231은 뒤에서부터 읽으면 다르므로 팰린드롬수가 아니다. 또한 10도 팰린드롬수가 아닌데, 앞에 무의미한 0이 올 수 있다면 010이 되어 팰린드롬수로 취급할 수도 있지만, 특별히 이번 문제에서는 무의미한 0이 앞에 올 수 없다고 하자.

입력

입력은 여러 개의 테스트 케이스로 이루어져 있으며, 각 줄마다 1 이상 99999 이하의 정수가 주어진다. 입력의 마지막 줄에는 0이 주어지며, 이 줄은 문제에 포함되지 않는다.

출력

각 줄마다 주어진 수가 팰린드롬수면 'yes', 아니면 'no'를 출력한다.

(문제를 해결하며 고안한 전략)

입력 받는 수가 몇 자리 수인지 정해져 있지 않아 범위 내의 경우를 일일이 나누어 푼다.

그런 후 10으로 계속 나누어 각 자리 숫자를 얻어내고 역순으로 10의 거듭제곱 수를 곱해 푼다.

→ 지금 생각해보면 너무 원초적이었음. 그냥 문자열로 받았으면 될 것 같은데. 바보.

(소스 코드)

```
C
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void) {
    int x[1000];
    int a[1000], b[1000], c[1000], d[1000], e[1000], n;
    int p[1000];
    int result[1000];

    for (int i = 0; i < 1000; i++) {
        x[i] = 0;
    }

    for (int j = 0; ; j++) {
        scanf("%d", &x[j]);
        if (x[j] == 0) {
            break;
        }
    }
    for (int k = 0; x[k]!=0; k++) {
        n = x[k];
        a[k] = n / 10000;
        n = n % 10000;
        b[k] = n / 1000;
        n = n % 1000;
        c[k] = n / 100;
        n = n % 100;
        d[k] = n / 10;
        e[k] = n % 10;

        if (a[k] == 0 && b[k] == 0 && c[k] == 0 && d[k] == 0) {
            n = e[k];
        }
        else if (a[k] == 0 && b[k] == 0 && c[k] == 0) {
            n = e[k] * 10 + d[k];
        }
        else if (a[k] == 0 && b[k] == 0) {
            n = e[k] * 100 + d[k] * 10 + c[k];
        }
        else if (a[k] == 0) {
            n = e[k] * 1000 + d[k] * 100 + c[k] * 10 + b[k];
        }
    }
}
```

```

else {
    n = e[k] * 10000 + d[k] * 1000 + c[k] * 100 + b[k] * 10 + a[k];
}

if (n == x[k]) {
    result[k] = 1;
}
else {
    result[k] = 0;
}

for (int p = 0;x[p] != 0;p++) {
    if (result[p] == 1) {
        printf("yes\n");
    }
    else {
        printf("no\n");
    }
}

return 0;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

이 문제를 문자열로 받아 풀지 않은 나는 바보 멍청이 똥개 해삼 말미잘 멍계다.



스터디 5주차 보고서

(8/4 ~ 8/10)

작성자 : 김승우

작성일 : 2025.08.10

문제 풀이 (BOJ 2675 문자열 반복)

(문제 분석 및 프로그램 설계)

문제

문자열 S 를 입력 받은 후에, 각 문자를 R 번 반복해 새 문자열 P 를 만든 후 출력하는 프로그램을 작성하시오. 즉, 첫 번째 문자를 R 번 반복하고, 두 번째 문자를 R 번 반복하는 식으로 P 를 만들면 된다. S 에는 QR Code "alphanumeric" 문자만 들어있다.

QR Code "alphanumeric" 문자는

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ\%\$*+-.:/

이다.

입력

첫째 줄에 테스트 케이스의 개수 $T(1 \leq T \leq 1,000)$ 가 주어진다. 각 테스트 케이스는 반복 횟수 $R(1 \leq R \leq 8)$, 문자열 S 가 공백으로 구분되어 주어진다. S 의 길이는 적어도 1이며, 20글자를 넘지 않는다.

출력

각 테스트 케이스에 대해 P 를 출력한다.

(문제를 해결하며 고안한 전략)

2차원 배열을 이용하여 각 케이스에 대한 문자열을 받고 문자가 들어간 각 배열을 반

복문으로 반복해준다.

(소스 코드)

```
C
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int n = 0;
    int m[1000];
    char a[1000][21];

    scanf("%d", &n);

    for (int j = 0; j < n; j++)
    {
        scanf("%d", &m[j]);
        scanf("%s", &a[j]);
    }

    for (int j = 0; j < n; j++)
    {
        for (int k = 0; a[j][k] != '\0'; k++)
        {
            for (int x = 0; x < m[j]; x++)
            {
                printf("%c", a[j][k]);
            }
        }
        printf("\n");
    }

    return 0;
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

처음에는 배열의 초기화를 위해서 모든 배열에 'a'를 넣었었는데, 그런 과정에서
널문자가 무수히 많이 배열에 저장되며 문제가 생겼다.

이번 기회에 문자열에서 마지막에 널문자를 저장한다는 것을 알게 되었다.

문제 풀이 (BOJ 8958 OX 퀴즈)

(문제 분석 및 프로그램 설계)

문제

"00XX0XX000"와 같은 OX퀴즈의 결과가 있다. 0은 문제를 맞은 것이고, X는 문제를 틀

린 것이다. 문제를 맞은 경우 그 문제의 점수는 그 문제까지 연속된 0의 개수가 된다. 예를 들어, 10번 문제의 점수는 3이 된다.

"00XX0XX000"의 점수는 $1+2+0+0+1+0+0+1+2+3 = 10$ 점이다.

OX퀴즈의 결과가 주어졌을 때, 점수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 테스트 케이스의 개수가 주어진다. 각 테스트 케이스는 한 줄로 이루어져 있고, 길이가 0보다 크고 80보다 작은 문자열이 주어진다. 문자열은 0과 X만으로 이루어져 있다.

출력

각 테스트 케이스마다 점수를 출력한다.

(문제를 해결하며 고안한 전략)

먼저 2차원 배열로 문자열을 받아주고 조건문과 반복문을 이용해서 0일 때는 1점을 부여하며

만약, 그 전이 0였다면 이번 문제의 점수를 그 전 점수 + 1로 매긴다.

(소스 코드)

```
C
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int n;
    char a[100][80];
    int score[80];
    int sum = 0;
    for (int i = 0; i < 80; i++)
    {
        score[i] = 0;
    }

    scanf("%d", &n);

    for (int i = 0; i < n; i++)
    {
        scanf("%s", &a[i]);
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; a[i][j] != '\0'; j++)
        {
            if (a[i][j] == '0')
            {
                score[j] = 1;
            }
        }
    }
}
```

```
        if (a[i][j - 1] == '0' && j != 0)
        {
            score[j] = score[j - 1] + 1;
        }
    else
    {
        score[j] = 0;
    }
    sum += score[j];
}
printf("%d\n", sum);
sum = 0;
}

return 0;
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

난 개 똑똑했다.



스터디 6주차 보고서

(8/11 ~ 8/17)

작성자 : 김승우

작성일 : 2025.08.17

문제 풀이 (BOJ 10809 알파벳 찾기)

(문제 분석 및 프로그램 설계)

문제

알파벳 소문자로만 이루어진 단어 S가 주어진다. 각각의 알파벳에 대해서, 단어에 포함되어 있는 경우에는 처음 등장하는 위치를, 포함되어 있지 않은 경우에는 -1을 출력하는 프로그램을 작성하시오.

입력

첫째 줄에 단어 S가 주어진다. 단어의 길이는 100을 넘지 않으며, 알파벳 소문자로만 이루어져 있다.

출력

각각의 알파벳에 대해서, a가 처음 등장하는 위치, b가 처음 등장하는 위치, ... z가 처음 등장하는 위치를 공백으로 구분해서 출력한다.

만약, 어떤 알파벳이 단어에 포함되어 있지 않다면 -1을 출력한다. 단어의 첫 번째 글자는 0번째 위치이고, 두 번째 글자는 1번째 위치이다.

(문제를 해결하며 고안한 전략)

아스키 코드를 이용하여 문자열로 받는데 받을 때 각 알파벳에 해당하는 아스키 코드 숫자와 같은 숫자의 배열에 저장하여 해결하였다. 이때 저장하는 값은 알파벳이 위치했던 자리이다.

(소스 코드)

```
C  
#define _CRT_SECURE_NO_WARNINGS  
#include <stdio.h>
```

```

int main(void)
{
    char a[100];
    char b[26];
    int n[26];

    scanf("%s", &a);

    for (int i = 97; i <= 122; i++)
    {
        for (int j = 0; a[j] != '\0'; j++)
        {
            if (a[j] == i)
            {
                n[i - 97] = j;
                break;
            }
            else
            {
                n[i - 97] = -1;
            }
        }
    }
    for (int i = 0; i < 26; i++)
    {
        printf("%d ", n[i]);
    }

    return 0;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

나는 이번 문제에서 굉장히 똑똑했다.

문제 풀이 (BOJ 2798 블랙잭)

(문제 분석 및 프로그램 설계)

문제

카지노에서 제일 인기 있는 게임 블랙잭의 규칙은 상당히 쉽다. 카드의 합이 21을 넘지 않는 한도 내에서, 카드의 합을 최대한 크게 만드는 게임이다. 블랙잭은 카지노마다 다양한 규정이 있다.

한국 최고의 블랙잭 고수 김정인은 새로운 블랙잭 규칙을 만들어 상근, 창영이와 게임하려고 한다.

김정인 버전의 블랙잭에서 각 카드에는 양의 정수가 쓰여 있다. 그 다음, 딜러는 N장의 카드를 모두 숫자가 보이도록 바닥에 놓는다. 그런 후에 딜러는 숫자 M을 크게 외친다.

이제 플레이어는 제한된 시간 안에 N장의 카드 중에서 3장의 카드를 골라야 한다. 블랙잭 변형 게임이기 때문에, 플레이어가 고른 카드의 합은 M을 넘지 않으면서 M과 최대한 가깝게 만들어야 한다.

N장의 카드에 써져 있는 숫자가 주어졌을 때, M을 넘지 않으면서 M에 최대한 가까운 카드 3장의 합을 구해 출력하시오.

입력

첫째 줄에 카드의 개수 N($3 \leq N \leq 100$)과 M($10 \leq M \leq 300,000$)이 주어진다. 둘째 줄에는 카드에 쓰여 있는 수가 주어지며, 이 값은 100,000을 넘지 않는 양의 정수이다.

합이 M을 넘지 않는 카드 3장을 찾을 수 있는 경우만 입력으로 주어진다.

출력

첫째 줄에 M을 넘지 않으면서 M에 최대한 가까운 카드 3장의 합을 출력한다.

(문제를 해결하며 고안한 전략)

반복문을 디파 많이 쓰면서 모든 경우를 훑으면 나올 것이다.

(소스 코드)

```
C
#include <stdio.h>

int main() {
    int N, M;
    scanf("%d %d", &N, &M);

    int cards[100];

    for (int i = 0; i < N; i++) {
        scanf("%d", &cards[i]);
    }

    int max = 0;

    for (int i = 0; i < N - 2; i++) {
        for (int j = i + 1; j < N - 1; j++) {
            for (int k = j + 1; k < N; k++) {
                int sum = cards[i] + cards[j] + cards[k];
                if (sum <= M && sum > max) {
                    max = sum;
                }
            }
        }
    }

    printf("%d\n", max);
    return 0;
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

반복문 안에 반복문 안에 반복문 안에 조건문을 하니 머리가 부서지는 줄 알았다.

잠시 저세상 갔다 올 뻔 했다.



스터디 1주차 보고서

(7/6 ~ 8/13)

작성자 : 박재관

작성일 : 2025.07.13

문제 풀이 (BOJ 1874 스택 수열)

(문제 분석 및 프로그램 설계)

문제

스택 (stack)은 기본적인 자료구조 중 하나로, 컴퓨터 프로그램을 작성할 때 자주 이용되는 개념이다. 스택은 자료를 넣는 (push) 입구와 자료를 뽑는 (pop) 입구가 같아 제일 나중에 들어간 자료가 제일 먼저 나오는 (LIFO, Last in First out) 특성을 가지고 있다.

1부터 n까지의 수를 스택에 넣었다가 뽑아 늘어놓음으로써, 하나의 수열을 만들 수 있다. 이때, 스택에 push하는 반드시 오름차순을 지키도록 한다고 하자. 임의의 수열이 주어졌을 때 스택을 이용해 그 수열을 만들 수 있는지 없는지, 있다면 어떤 순서로 push와 pop 연산을 수행해야 하는지를 알아낼 수 있다. 이를 계산하는 프로그램을 작성하라.

입력

첫 줄에 n ($1 \leq n \leq 100,000$)이 주어진다. 둘째 줄부터 n개의 줄에는 수열을 이루는 1이상 n이하의 정수가 하나씩 순서대로 주어진다. 물론 같은 정수가 두 번 나오는 일은 없다.

출력

입력된 수열을 만들기 위해 필요한 연산을 한 줄에 한 개씩 출력한다. push연산은 +로, pop 연산은 -로 표현하도록 한다. 불가능한 경우 NO를 출력한다.

(문제를 해결하는데 필요한 개념)

stack은 LIFO(Last In First Out) 자료형이다.

(소스 코드)

```
CPP
#include<iostream>
#include<stack>
#include<vector>
using namespace std;

int main(void){
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n,N;
    stack<int> stack;
    vector<char> ans;
```

```

cin >> n;
N = n;
int i = 1;
while(n--){
    int num;
    cin >> num;
    if(stack.size() == 0){
        for(;i<=num;i++){
            stack.push(i);
            ans.push_back('+');
            //cout << i << "\n";
        }
    }

    if(stack.top() == num){
        stack.pop();
        ans.push_back('-');
        //cout << "x\n";
    }
    else if(num < i){
        cout << "NO";
        return 0;
    }
    else if(num >= i){
        for(;i<=num;i++){
            stack.push(i);
            ans.push_back('+');
            //cout << "c" << num << "\n";
        }
        stack.pop();
        ans.push_back('-');
    }
}
for(int j = 0;j<2*N;j++){
    cout << ans[j] << "\n";
}
return 0;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

- 먼저 n 을 입력 받고, n 개 만큼의 수를 또 입력 받는다.

2. 스택에 들어가는 수는 1부터 오름차순으로 계속 증가하기 때문에 초기값 int i = 1 선언.
3. stack.size()가 0 일 때 i 부터 입력된 수까지 stack에 push.
4. 초기에는 입력된 수까지 stack에 push 했기 때문에 stack.top() == num 임을 확인하고 pop.
5. 다음에 입력되는 수들에 대하여 조건문 반복.
6. 입력되는 수보다 i 가 크다면(예를 들어, [1,2]가 스택에 있고 [3,4]는 pop 된 상태라서 현재 i 가 5이고, 입력된 수가 1이라면, 입력된 수가 top(=2)가 아니라는 가정하에 문제 조건을 맞출 수 없으므로) NO 를 출력하고 종료.
7. 입력되는 수가 i 보다 크거나 같다면(예를 들어, [1,2]가 스택에 있고 [3,4]는 pop 된 상태라서 현재 i 가 5이고, 입력된 수가 7이라면, [1,2,5,6,7]까지 push 한 뒤, 7을 pop) 입력된 수까지 push 하고 맨 위 값을 pop.
8. 위 전체 과정에서 선언해둔 vector<char> ans 에 stack에 push 할 때마다 '+'를, pop 할 때마다 '-'를 순서대로 저장.
9. ans 를 출력.

처음에 문제를 봤을 때에는 오랫동안 문제 자체가 이해되질 않았는데, 입력과 출력을 보면서 직접 차례대로 써가면서 해보니, 스택의 수를 pop 하는 순서대로 입력에 있는 수열을 만들 수 있는지 없는지를 판단하는 문제라는 걸 알게 되었다.

문제 풀이 (BOJ 11047 동전0)

(문제 분석 및 프로그램 설계)

문제

준규가 가지고 있는 동전은 총 N 종류이고, 각각의 동전을 매우 많이 가지고 있다.

동전을 적절히 사용해서 그 가치의 합을 K로 만들려고 한다. 이때 필요한 동전 개수의 최솟값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N 과 K 가 주어진다. ($1 \leq N \leq 10, 1 \leq K \leq 100,000,000$)

둘째 줄부터 N 개의 줄에 동전의 가치 A_i 가 오름차순으로 주어진다. ($1 \leq A_i \leq 1,000,000, A_1 = 1, i \geq 2$ 인 경우에 A_i 는 A_{i-1} 의 배수)

출력

첫째 줄에 K 원을 만드는데 필요한 동전 개수의 최솟값을 출력한다.

먼저 문제를 처음 봤을 때는 꽤나 복잡한 문제라는 생각이 들었다. 만약 각 동전의 가치가 예제에 있는 것처럼(1, 5, 10, 50, 100, 500, 1000 ...) 배수의 형태가 아닌, 이상한(?)

수가 포함(ex- 2237)됬을 때를 대비해서 처음 합이 만들어지기 전까지 모든 경우의 수를 돌아야하기 때문이다.

근데 문제의 입력 조건에 둘째 줄부터 N 개의 줄에 동전의 가치 A_i 가 오름차순으로 주어진다. ($1 \leq A_i \leq 1,000,000$, $A_1 = 1$, $i \geq 2$ 인 경우에 A_i 는 A_{i-1} 의 배수) 를 보면 A_i 가 A_{i-1} 의 배수라는 조건이 있어서 위와 같은 복잡한 생각은 하지 않아도 되는 문제였다.

(소스 코드)

```
CPP
#include<iostream>
#include<vector>
using namespace std;
int main(void){
    ios::sync_with_stdio(0);
    cin.tie(0);
    int N,K,ans=0;
    vector<int> value;
    cin >> N >> K;
    for(int i = 0;i<N;i++){
        int temp;
        cin >> temp;
        if(temp > K){}
        else{
            value.push_back(temp);
        }
    }
    for(int i = value.size()-1;i>=0;i--){
        int temp;
        temp = K/value[i];
        ans += temp;
        K -= temp*value[i];
        if(K == 0){
            break;
        }
    }
    cout << ans;
    return 0;
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

1. N 과 K 를 입력받고, N 개 만큼의 동전 가치를 입력 받으면서, K 보다 큰 값들은 버리고, 작거나 같은 값들을 `vector<int> value` 에 push 함.
2. 반복문으로 `value` 의 마지막(제일 큰)원소부터 첫번째 원소까지 돌면서 그 동전을 최대로 사용(큰 동전을 최대로 사용한다는 것 = 동전의 개수를 최소로 사용하는 것)하고 그 만큼을 K에서 빼줌. 사용한만큼 ans에 추가.
3. $K = 0$ 이 되었을 때 반복문을 종료하고 ans를 출력.



스터디 2주차 보고서

(7/14 ~ 7/20)

작성자 : 박재관

작성일 : 2025.07.20

문제 풀이 (BOJ 11399 ATM)

(문제 분석 및 프로그램 설계)

문제

인하은행에는 ATM이 1대밖에 없다. 지금 이 ATM앞에 N명의 사람들이 줄을 서 있다. 사람은 1번부터 N번까지 번호가 매겨져 있으며, i번 사람이 돈을 인출하는데 걸리는 시간은 P_i 분이다.

사람들이 줄을 서는 순서에 따라서, 돈을 인출하는데 필요한 시간의 합이 달라지게 된다. 예를 들어, 총 5명이 있고, $P_1 = 3, P_2 = 1, P_3 = 4, P_4 = 3, P_5 = 2$ 인 경우를 생각해보자. [1, 2, 3, 4, 5] 순서로 줄을 선다면, 1번 사람은 3분만에 돈을 뽑을 수 있다. 2번 사람은 1번 사람이 돈을 뽑을 때 까지 기다려야 하기 때문에, $3+1 = 4$ 분이 걸리게 된다. 3번 사람은 1번, 2번 사람이 돈을 뽑을 때까지 기다려야 하기 때문에, 총 $3+1+4 = 8$ 분이 필요하게 된다. 4번 사람은 $3+1+4+3 = 11$ 분, 5번 사람은 $3+1+4+3+2 = 13$ 분이 걸리게 된다. 이 경우에 각 사람이 돈을 인출하는데 필요한 시간의 합은 $3+4+8+11+13 = 39$ 분이 된다.

줄을 [2, 5, 1, 4, 3] 순서로 줄을 서면, 2번 사람은 1분만에, 5번 사람은 $1+2 = 3$ 분, 1번 사람은 $1+2+3 = 6$ 분, 4번 사람은 $1+2+3+3 = 9$ 분, 3번 사람은 $1+2+3+3+4 = 13$ 분이 걸리게 된다. 각 사람이 돈을 인출하는데 필요한 시간의 합은 $1+3+6+9+13 = 32$ 분이다. 이 방법보다 더 필요한 시간의 합을 최소로 만들 수는 없다.

줄을 서 있는 사람의 수 N과 각 사람이 돈을 인출하는데 걸리는 시간 P_i 가 주어졌을 때, 각 사람이 돈을 인출하는데 필요한 시간의 합의 최솟값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 사람의 수 N($1 \leq N \leq 1,000$)이 주어진다. 둘째 줄에는 각 사람이 돈을 인출하는데 걸리는 시간 P_i 가 주어진다. ($1 \leq P_i \leq 1,000$)

출력

첫째 줄에 각 사람이 돈을 인출하는데 필요한 시간의 합의 최솟값을 출력한다.

(문제를 해결하는데 필요한 개념)

(소스 코드)

```
CPP
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main(void){
    int N;
    cin >> N;
    vector<int> P;
```

```

for(int i = 0;i<N;i++){
    int temp;
    cin >> temp;
    P.push_back(temp);
}
sort(P.begin(),P.end());
int s = 0;
int sum = 0;
for(int i = 0;i<N;i++){
    s += P[i];
    sum += s;
}
cout << sum;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

출력해야 하는 정답 값이 ‘시간의 합의 최솟값’이므로 입력 받은 수(시간)들을 sort로 오름차순으로 정렬해주고, s라는 변수를 선언해서 sum에 계속 누적해서 더해주면 최솟값이 나오게 된다.

문제 풀이 (BOJ 2630 색종이 만들기)

(문제 분석 및 프로그램 설계)

문제

아래 <그림 1>과 같이 여러개의 정사각형칸들로 이루어진 정사각형 모양의 종이가 주어져 있고, 각 정사각형들은 하얀색으로 칠해져 있거나 파란색으로 칠해져 있다. 주어진 종이를 일정한 규칙에 따라 잘라서 다양한 크기를 가진 정사각형 모양의 하얀색 또는 파란색 색종이를 만들려고 한다.

1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
0	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0
1	0	0	0	1	1	1	1
0	1	0	0	1	1	1	1
0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1

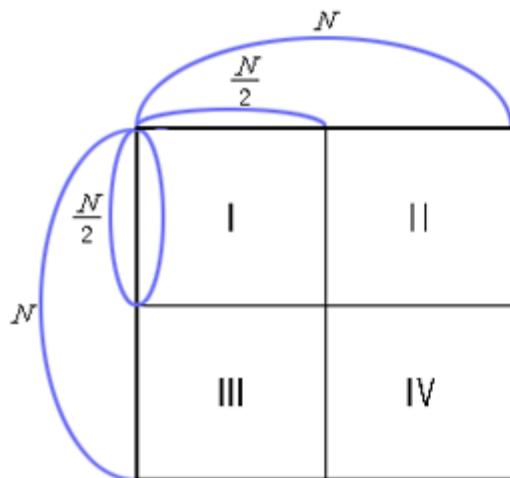
<그림 1> 8×8 종이

전체 종이의 크기가 $N \times N$ ($N=2^k$, k 는 1 이상 7 이하의 자연수) 이라면 종이를 자르는 규칙은 다음과 같다.

전체 종이가 모두 같은 색으로 칠해져 있지 않으면 가로와 세로로 중간 부분을 잘라서 <그림 2>의 I, II, III, IV 와 같이 똑같은 크기의 네 개의 $N/2 \times N/2$ 색종이로

나눈다. 나누어진 종이 I, II, III, IV 각각에 대해서도 앞에서와 마찬가지로 모두 같은 색으로 칠해져 있지 않으면 같은 방법으로 똑같은 크기의 네 개의 색종이로 나눈다. 이와 같은 과정을 잘라진 종이가 모두 하얀색 또는 모두 파란색으로 칠해져 있거나, 하나의 정사각형 칸이 되어 더 이상 자를 수 없을 때까지 반복한다.

위와 같은 규칙에 따라 잘랐을 때 <그림 3>은 <그림 1>의 종이를 처음 나눈 후의 상태를, <그림 4>는 두 번째 나눈 후의 상태를, <그림 5>는 최종적으로 만들어진 다양한 크기의 9장의 하얀색 색종이와 7장의 파란색 색종이를 보여주고 있다.



<그림 2>

1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
0	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0
1	0	0	0	1	1	1	1
0	1	0	0	1	1	1	1
0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1

<그림 3> 처음 나눈 후의 상태

1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
0	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0
1	0	0	0	1	1	1	1
0	1	0	0	1	1	1	1
0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1

<그림 4> 두 번째 나눈 후의 상태

1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
0	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0
1	0	0	0	1	1	1	1
0	1	0	0	1	1	1	1
0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1

<그림 5> 최종적으로 나누어진 색종이들

입력으로 주어진 종이의 한 변의 길이 N 과 각 정사각형칸의 색(하얀색 또는 파란색)이 주어질 때 잘라진 하얀색 색종이와 파란색 색종이의 개수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에는 전체 종이의 한 변의 길이 N 이 주어져 있다. N 은 $2, 4, 8, 16, 32, 64, 128$ 중 하나이다. 색종이의 각 가로줄의 정사각형칸들의 색이 윗줄부터 차례로 둘째 줄부터 마지막 줄까지 주어진다. 하얀색으로 칠해진 칸은 0, 파란색으로 칠해진 칸은 1로 주어지며, 각 숫자 사이에는 빈칸이 하나씩 있다.

출력

첫째 줄에는 잘라진 하얀색 색종이의 개수를 출력하고, 둘째 줄에는 파란색 색종이의 개수를 출력한다.

(소스 코드)

```
CPP
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int N;
int arr[129][129];
int white,blue;
void sol(int a, int b, int k){
    int first = arr[a][b];
    int cut = 0;
    for(int i = a; i<a+k; i++){
        for(int j = b;j<b+k;j++){
            if(arr[i][j] != first){
                cut = 1;
                break;
            }
        }
    }
    if(cut == 1){
        sol(a,b,k/2);
        sol(a,b+k/2,k/2);
        sol(a+k/2,b,k/2);
        sol(a+k/2,b+k/2,k/2);
    }
    else{
        if(first == 0){
            white++;
        }
        else{
            blue++;
        }
    }
}
int main(void){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> N;
    for(int i = 0;i<N;i++){
        for(int j = 0;j<N;j++){
            int temp;
            cin >> temp;
            arr[i][j] = temp;
        }
    }
    sol(0,0,N);
    cout << white << "\n";
    cout << blue;
    return 0;
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

문제를 봤을 때 색종이를 네 장으로 나눈 뒤, 또 나눈 각각의 색종이들에서 판단을 통해 네 장으로 나누는 작업을 계속해서 진행해야 하기 때문에 단순 반복문 보다는 재귀로 푸는게 좋을 것 같다고 생각했다.

재귀함수의 로직은 이렇다:

1. 시작점의 위치(a, b), 색종이 크기(k)를 매개변수로 받는다.
2. 잘라야되는지 말아야되는지를 판단하기 위해 $cut = 0$ 이라는 변수를 선언.
3. a 행부터 $a+k$ 행까지, b 열부터 $b+k$ 열까지 시작점의 색과 다른게 있다면, $cut = 1$ 로 바꾼 뒤 for 문을 종료.
4. 그 뒤 조건문을 통해 $cut == 1$ 이라면 함수의 시작점을 색종이를 네 장으로 나눴을 때 기준으로 재설정하고, 색종이 크기 단위를 2로 나눠서 다시 함수를 돌린다.
5. Cut == 0이라면 $frst == 0$ 일 때 white++, $frst == 1$ 일 때 blue++
6. main 함수에서 `sol(0,0,색종이 처음 크기);`로 돌린 뒤, white 와 blue 를 순서대로 출력.



스터디 3주차 보고서

(7/21 ~ 7/27)

작성자 : 박재관

작성일 : 2025.07.29

문제 풀이 (BOJ 1145 적어도 대부분의 배수)

(문제 분석 및 프로그램 설계)

문제

다섯 개의 자연수가 있다. 이 수의 적어도 대부분의 배수는 위의 수 중 적어도 세 개로 나누어 지는 가장 작은 자연수이다.

서로 다른 다섯 개의 자연수가 주어질 때, 적어도 대부분의 배수를 출력하는 프로그램을 작성하시오.

입력

첫째 줄에 다섯 개의 자연수가 주어진다. 100보다 작거나 같은 자연수이고, 서로 다른 수이다.

출력

첫째 줄에 적어도 대부분의 배수를 출력한다.

(문제를 해결하는데 필요한 개념)

(소스 코드)

CPP

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

```

int main(void){
    ios::sync_with_stdio(0);
    cin.tie(0);
    int arr[5];
    for(int i = 0; i<5; i++){
        cin >> arr[i];
    }
    int k = 1;
    while(1){
        int cnt = 0;
        for(int i = 0; i<5; i++){
            if(k%arr[i] == 0){
                cnt++;
            }
        }
        if(cnt >= 3){
            cout << k;
            return 0;
        }
        k++;
    }
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

먼저 arr[5]에 자연수 5개를 입력받는다.

그 후, k=1 부터 다섯 개의 자연수 중 적어도 3개의 수로 나누어떨어지는 k를 찾는다.

K를 출력.

문제 풀이 (BOJ 1268 임시 반장 정하기)

(문제 분석 및 프로그램 설계)

문제

오민식 선생님은 올해 협택초등학교 6학년 1반 담임을 맡게 되었다. 오민식 선생님은 우선 임시로 반장을 정하고 학생들이 서로 친숙해진 후에 정식으로 선거를 통해 반장을 선출하려고 한다. 그는 자기반 학생 중에서 1학년부터 5학년까지 지내오면서 한번이라도 같은 반이었던 사람이 가장 많은 학생을 임시 반장으로 정하려 한다.

그래서 오민식 선생님은 각 학생들이 1학년부터 5학년까지 몇 반에 속했었는지를 나타내는 표를 만들었다. 예를 들어 학생 수가 5명일 때의 표를 살펴보자.

	1학년	2학년	3학년	4학년	5학년
1번 학생	2	3	1	7	3
2번 학생	4	1	9	6	8
3번 학생	5	5	2	4	4
4번 학생	6	5	2	6	7
5번 학생	8	4	2	2	2

위 경우에 4번 학생을 보면 3번 학생과 2학년 때 같은 반이었고, 3번 학생 및 5번 학생과 3학년 때 같은 반이었으며, 2번 학생과는 4학년 때 같은 반이었음을 알 수 있다. 그러므로 이 학급에서 4번 학생과 한번이라도 같은 반이었던 사람은 2번 학생, 3번 학생과 5번 학생으로 모두 3명이다. 이 예에서 4번 학생이 전체 학생 중에서 같은 반이었던 학생 수가 제일 많으므로 임시 반장이 된다.

각 학생들이 1학년부터 5학년까지 속했던 반이 주어질 때, 임시 반장을 정하는 프로그램을 작성하시오.

입력

첫째 줄에는 반의 학생 수를 나타내는 정수가 주어진다. 학생 수는 3 이상 1000 이하이다. 둘째 줄부터는 1번 학생부터 차례대로 각 줄마다 1학년부터 5학년까지 몇 반에 속했었는지를 나타내는 5개의 정수가 빈칸 하나를 사이에 두고 주어진다. 주어지는 정수는 모두 1 이상 9 이하의 정수이다.

출력

첫 줄에 임시 반장으로 정해진 학생의 번호를 출력한다. 단, 임시 반장이 될 수 있는 학생이 여러 명인 경우에는 그 중 가장 작은 번호만 출력한다.

(소스 코드)

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

int ans[100001];

int main(void){
    ios::sync_with_stdio(0);
    cin.tie();
    int v[10000][5];
    int N;
    cin >> N;
    for(int i = 0;i<N;i++){
        int arr[5];
        for(int j = 0;j<5;j++) {
            cin >> arr[j];
        }
        sort(arr, arr+5);
        if(v[i][0] < arr[4]) {
            v[i][0] = arr[4];
        }
    }
    cout << v[0][0];
}
```

```

        cin >> v[i][j];
    }
}
for(int i = 0;i<N;i++){
    for(int j = 0;j<N;j++){
        for(int k = 0;k<5;k++){
            if(v[i][k] == v[j][k]){
                ans[i]++;
                break;
            }
        }
    }
}
int maxIdx;
int max = 0;
for(int i = N-1;i>=0;i--){
    if(max <= ans[i]){
        maxIdx = i+1;
        max = ans[i];
    }
}
cout << maxIdx;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

먼저 1번 학생부터 N 번 학생까지 같은 반을 했었던 친구의 수를 구한다.

그 후 (~~~친구의 수가 같을 경우 번호가 낮은 학생을 출력해야 하므로) N 번 학생부터 1번 학생까지 순회하면서 최대 수를 가진 학생의 번호를 저장하고 번호를 출력한다.



스터디 1주차 보고서

(7/7 ~ 7/13)

작성자 : 심현서

작성일 : 2025.07.13

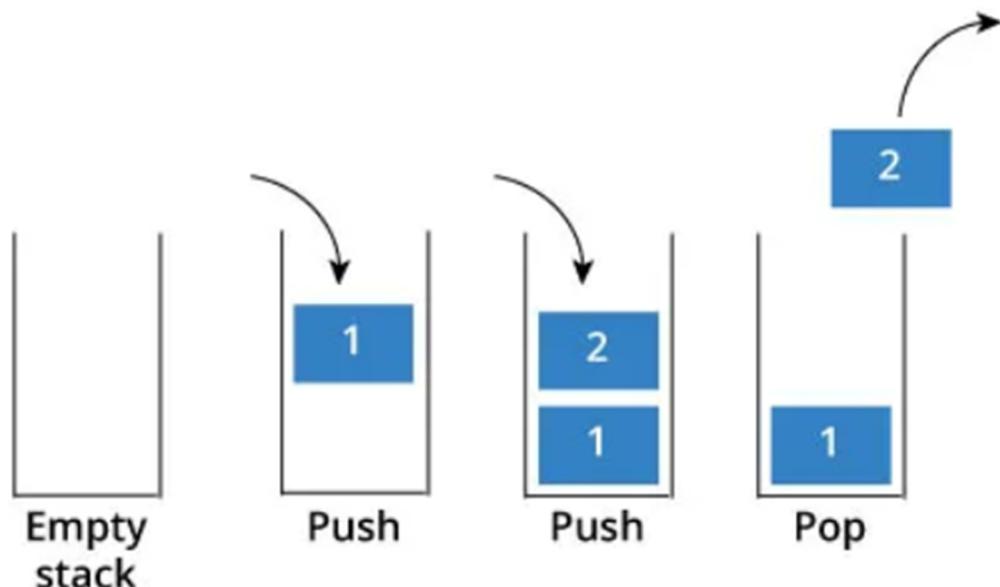
문제 풀이 (BOJ 1874 스택 수열)

(문제 분석 및 프로그램 설계)

첫째 줄에 스택에 넣는 마지막 수(1부터 N까지) $N(1 \leq N \leq 100,000)$ 이 주어진다. 다음 N 개의 줄에는 수열을 이루는 1이상 N이하의 정수가 하나씩 순서대로 주어진다. 1부터 N까지의 수를 자료구조인 스택에 넣었다가 뽑아 늘어놓음으로써 이 수열을 만들 수 있음을 판단하고, push와 pop 연산을 어떤 순서로 수행해야 하는지 '+'와 '-'를 통해 출력한다.

(문제를 해결하는데 필요한 개념)

stack은 last in first out 자료형이다.



(소스 코드)

CPP

```
#include<iostream>
#include<stack>
#include<vector>
using namespace std;
stack<int> st; //수열을 넣어볼 스택
vector<char> p; // '+'와 '-'를 넣을 vector 선언
void ppush(int x); //push 와 '+' 동시에
void ppop(); //pop 과 '-' 동시에

int main() {
```

```

ios::sync_with_stdio(0);
cin.tie(0);
int N, max, tmp;
cin >> N;
cin >> max; // 첫번째 수를 기준점으로 잡기

for (int i = 1; i <= max; i++) {
    ppush(i); // 1부터 첫번째 수까지 수를 push하고 '+'를 추가
}
ppop(); // 첫번째 수를 pop하고 '-'추가
N--;

while (N--) {
    cin >> tmp; // 나머지 수열은 tmp로 받고 그때그때 처리
    if (tmp > max) { // 만약 입력됐던 제일 큰 수보다 크다면
        for (int i = 1 + max; i <= tmp; i++) {
            ppush(i); // 차이만큼 넣기
        }
        ppop(); // 스택에 입력된 마지막 수는 빼기
        max = tmp; // max 갱신 후 반복문
    }
    else {
        if (!st.empty() && st.top() == tmp) {
//비어 있지 않고, 제일 위에 있는 수와 빼야 하는 수가 같다면
            ppop(); // 그 수를 빼고 다시 반복문
        }
    }
} // 만약 이 규칙에 맞는다면 스택이 최종적으로는 비어 있을 것이다

if (st.empty()) { // '!st.empty()'는 비어 있지 않으면 실행하겠다는 뜻
    for (int i = 0; i < p.size(); i++) {
        cout << p[i] << "\n"; // 부호 출력
    }
}
else {
    cout << "NO" << "\n";
}
return 0;
}

```

```

void ppush(int x) {
    st.push(x);
    p.push_back('+');
}

void ppop() {
    st.pop();
    p.push_back('-');
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

스택에 대한 감이 좀 잡히기 시작했다.

처음 문제를 봤을 땐 도대체 어떻게 접근해야 하는지 몰랐다. 스택을 어떻게 이용해서 수열의 적합성을 판단할까 오래 고민했었다. 심지어 push pop 순서까지 출력해야 한다니! 머릿속에 스택에 1부터 N 까지 오름차순으로 넣은 상태에서 수열을 만드는 상황을 떠올려봤다.

일단, 첫번째로 입력된 수(수열의 첫 번째 수)까지 1부터 차례대로 넣은 후 그 수를 pop 하면, 그 수가 수열의 첫 번째 수가 된다는 아이디어를 이용하기로 했다. 그리고 그 다음으로 입력되는 수들이 pop 할 때까지 스택에 수를 push 한 후 pop 하거나 그냥 pop 하는 과정을 통해서 입력된 수열을 다 도출할 수 있다면, 그 스택이 결국엔 비어 있을 거라는 점을 이용했다.

vector 를 사용해본 적이 많지 않았는데 그 유용성을 느끼게 됐다.

문제 풀이 (BOJ 11866 요세푸스 문제 0)

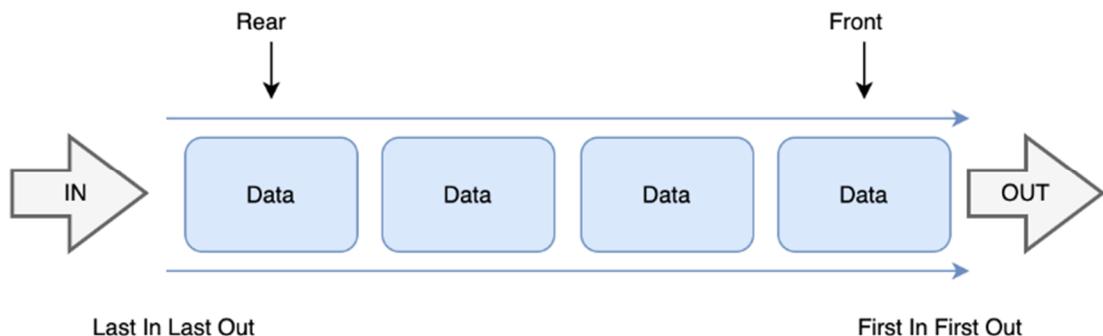
(문제 분석 및 프로그램 설계)

1번부터 N번까지 N명의 사람이 원을 이루며 앉아있고, 그 원을 따라서 N명의 사람이 모두 제거될 때까지 순서대로 K번째 사람을 제거한다. 원에서 사람들이 제거되는 순서를 (N,K)-요세푸스 순열이라고 한다.

N과 K가 주어지면 이에 맞는 요세푸스 순열을 출력해야 한다.

(문제를 해결하는데 필요한 개념)

Queue는 first in first out 자료형이다.



(소스 코드)

CPP

```
#include<iostream>
#include<queue>
#include<vector>
using namespace std;
int main(void) {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int N, K, tmp;
    vector<int> v; // 빠지는 수를 순서대로 넣을 vector
    queue<int> q; // 요세푸스 순열을 실행할 queue
    cin >> N >> K;
    for (int i = 1; i <= N; i++) {
        q.push(i); // 끝까지 넣기
    }

    while (!q.empty()) {
        for (int i = 1; i <= K - 1; i++) {
            tmp = q.front();
```

```

        q.pop();
        q.push(tmp); // K - 1 번 만큼 뺐다가 다시 넣기.
    }

    v.push_back(q.front()); // 제거할 K번째 변수를 vector에 넣기
    q.pop(); // 그 후에 제거
}

cout << "<";
for (int i = 0; i < v.size(); i++) {
    if (i == v.size() - 1) {
        cout << v[i] << ">" << "\n";
    }
    else {
        cout << v[i] << ", ";
    }
}

return 0;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

요세푸스 순열은 결국 queue의 자료구조를 이용하여 쉽게 풀 수 있었다. K번째 수가 아니면 queue의 First in First out 성질을 이용해서 pop 해주고, 다시 queue에 넣어주고 K번째 수이면 pop 해서 아예 제거해버리는 걸 반복해서 queue가 비어 있게 만들어주면 됐다. 이걸 이용해서 구현해봤다.

하지만 더 간단한 코드가 있는 것 같다. 너무 무식하고 naïve하게 코드를 짜는 경향이 있으며 다른 사람들의 코드를 참고해보지 않는 경향이 있는데 발전에 전혀 도움이 되지 않을 것이다. 코드를 좀더 명료하게 짜는 연습을 하며 내실을 다져야겠다.



스터디 2주차 보고서

(7/14~ 7/20)

작성자 : 심현서

작성일 : 2025.07.20

문제 풀이 (BOJ 11660 구간 합 구하기 5)

(문제 분석 및 프로그램 설계)

첫째 줄에 $N \times N$ 크기 표의 크기를 지정해줄 $N(1 \leq N \leq 100,000)$ 과 합을 구해야 하는 횟수 $M(1 \leq M \leq 100,000)$ 이 주어진다. 둘째 줄부터 표에 채워져 있는 수가 1행부터 차례대로 주어지고 그 다음 M 개의 줄에는 구할 범위를 지정해주는 두 좌표 (x_1, y_1) 과 (x_2, y_2) 가 주어진다.

그리고 각각의 경우에서 두 좌표 (x_1, y_1) 과 (x_2, y_2) 가 만드는 사각형에서의 숫자의 합을 구해야 한다. 이 경우 구간합을 이용해서 시간 복잡도가 n^3 이 되도록 프로그램을 작성해야 한다.

(문제를 해결하는데 필요한 개념)

● DP(Dynamic Programming)

하나의 복잡한 큰 문제를 여러 개의 작은 문제로 나누어서 그 결과를 저장한 후 다시 큰 문제를 해결할 때 사용하는 것을 기본적인 아이디어로 한다. 큰 문제를 작은 문제로 쪼개서 그 답을 저장해두고 재활용한다고 할 수 있는데, 이전의 결과들을 바탕으로 다음 결과를 도출한다고 생각하면 된다.(예를 들어 피보나치 수열이 있다.)

DP 가 적용되기 위해서는 2가지 조건을 만족해야 한다.

1) Overlapping Subproblems

동일한 작은 문제들이 반복하여 나타나는 경우에 사용이 가능하다.

2) Optimal Substructure

부분 문제의 최적 결과 값을 사용해 전체 문제의 최적 결과를 낼 수 있는 경우에 사용한다. 부분 문제에서 구한 최적 결과가 전체 문제에서도 동일하게 적용되어 결과가 변하지 않을 때 DP를 사용할 수 있게 된다. 피보나치 수열도 동일하게 이전의 계산 값을 그대로 사용하여 전체 답을 구할 수 있어 최적 부분 구조를 갖고 있다.

● 구간 합

주어진 배열에서 특정 구간의 연속된 요소들의 합을 의미한다. 예를 들어, 배열에서 2번째 요소부터 4번째 요소까지의 구간 합은 $2 + 3 + 4 = 9$ 가 되는 식이다. 이러한 구간 합은 시간 복잡도를 줄이는 데 유용하게 활용될 수 있다.

array	1	8	7	4	3	5	6
prefix sum	1	9	16	20	23	28	31



(소스 코드)

CPP

```

#include<iostream>
using namespace std;
int arr[1025][1025];
int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int N, M;
    cin >> N >> M;

    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            cin >> arr[i][j];
            if (j == 1) arr[i][j] += arr[i - 1][N]; // 1 열인 경우 따로 처리
            else arr[i][j] += arr[i][j - 1];
        }
    }

    // 한 행씩 따로 처리하기. 두 행 이상 차이 나는 경우 주의
    while (M--) {
        int x1, y1, x2, y2;
        cin >> x1 >> y1 >> x2 >> y2;
        int sum = 0;
        for (int i = x1; i <= x2; i++) { // 행별로 계산
            if (y1 == 1) sum += (arr[i][y2] - arr[i - 1][N]);
            else sum += (arr[i][y2] - arr[i][y1 - 1]);
        }
        cout << sum << '\n';
    }
    return 0;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

구간 합을 이용하여 풀어야 한다는 생각은 했으나 1차원 배열일 때와 달리 사각형의 범위 내의 수를 어떻게 구해야 할지가 많이 고민됐던 부분이다. 도대체 어떻게 접근해야 하는지 몰랐으나 누적 합을 통해서 구간 합을 구하는 방법의 원리에 대해서 고민해보니까 한 행씩 처리하면 되겠다는 생각에 이르게 되었다.

일단 이차원 배열에 수를 받을 때 (1,1)의 첫 원소부터 (i,j)원소까지의 누적합을 각 배열에 저장했다. 그리고 나서 주어진 (x1,y1)과 (x2,y2)로 구성된 사각형을 행 단위로

쪼개서 각각의 행에서 구간합을 구한 다음 구간합들을 더하는 방식으로 문제를 해결했다.

너무 복잡하게 생각할 필요 없이 작은 부분으로 쪼개는 것만으로 문제가 풀린다는 점을 느꼈다. 앞으로는 같은 태그의 문제를 풀 때, 전에 풀었던 문제를 적극적으로 참고하며 이걸 어떻게 적용할지 고민하면서 많은 문제를 풀어야겠다.

문제 풀이 (BOJ 1002 터렛)

(문제 분석 및 프로그램 설계)

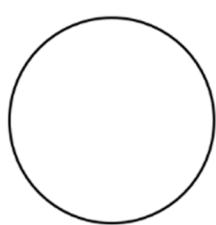
첫째 줄에 총 테스트 케이스의 개수 T 가 주어진다. 그리고 각 테스트 케이스에는 터렛에 위치한 ‘조규현’의 좌표 (x_1, y_1) 과 ‘조규현’의 ‘류재명’과의 거리 r_1 , ‘백승환’의 좌표 (x_2, y_2) 와 ‘류재명’과의 거리 r_2 가 주어진다. 두 터렛의 좌표와 각각에서의 류재명과의 떨어진 거리를 고려하여 류재명이 있을 수 있는 위치의 수를 출력해야 한다.

터렛의 좌표를 원의 중심으로 하고 타겟과의 떨어진 거리를 원의 반지름으로 하는 두 원의 위치 관계를 바탕으로 위치의 경우의 수를 구하는 프로그램을 만들 수 있다.

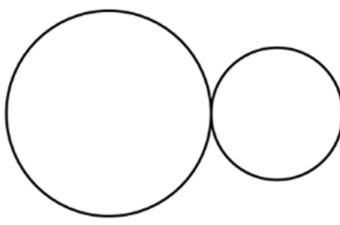
(문제를 해결하는데 필요한 개념)

두 원의 위치 관계

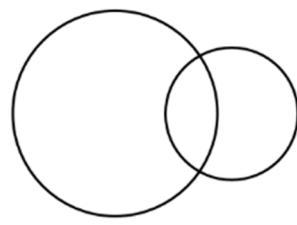
평면 위에서 두 개의 원의 위치 관계는 다음 6가지 중 하나이다.



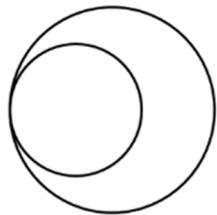
다른 원의 외부
(만나지 않는다)



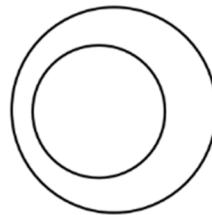
두 원이 외접



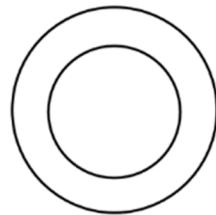
두 점에서 만난다



두 원이 내접



다른 원의 내부



동심원

(소스 코드)

CPP

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

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int x1, y1, r1, x2, y2, r2, T;
    double dis;
    cin >> T;
    while (T--) {
        cin >> x1 >> y1 >> r1 >> x2 >> y2 >> r2;
        dis = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2)); // 두 중심의 거리

        if (dis == 0) { // 두 점의 좌표가 같은 경우
            if (r1 == r2) { // 두 점이 같고 거리가 같으면 무한대
                cout << -1 << "\n";
            }
            else { // 같은데 거리가 다르면 안 돼요
                cout << 0 << "\n";
            }
        }
        else if (dis == r1 + r2 || dis == abs(r1 - r2)) { // 한 점에서 만남(접점)
            cout << 1 << "\n";
        }
        else if (dis > r1 + r2 || dis < abs(r1 - r2)) { // 만나는 점이 없음
            cout << 0 << "\n";
        }
        else { // 그 외 두 점에서 만나는 경우
            cout << 2 << "\n";
        }
    }

    return 0;
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

두 원의 위치 관계를 모두 생각하면 풀린다는 걸 깨달은 후 빨리 풀린 문제이다.
하지만 그 경우의 수를 꼼꼼히 고려하지 못한 점과 원이 뜻하는 의미를 정확히
파악하지 못한 점이 이 문제를 여러 번 틀리게 만든 원인이었다.

원의 자취 그 자체가 류재명이 있을 수 있는 위치이므로 두 개의 원이 만나는 점의
개수만 세면 됐는데 그 원의 범위 내에서는 류재명이 모두 있을 수 있다는
황당무계한 착각을 해버려서 두 터렛의 좌표만 같다면 가능한 위치의 수가
무한대이고 한 원이 다른 원을 감싸기만 하면 가능한 위치의 수가 무한대라는 판단을
했었다.

결국 뒤늦게 깨닫긴 했지만, 앞으로는 문제를 풀기 위한 코드를 설계하는 과정에서
그 과정을 정확히 이해하고 모든 경우를 미리 생각해보는 훈련이 필요하겠다.



스터디 3주차 보고서

(7/21 ~ 7/27)

작성자 : 심현서

작성일 : 2025.07.28..

문제 풀이 (BOJ 1806 부분합)

(문제 분석 및 프로그램 설계)

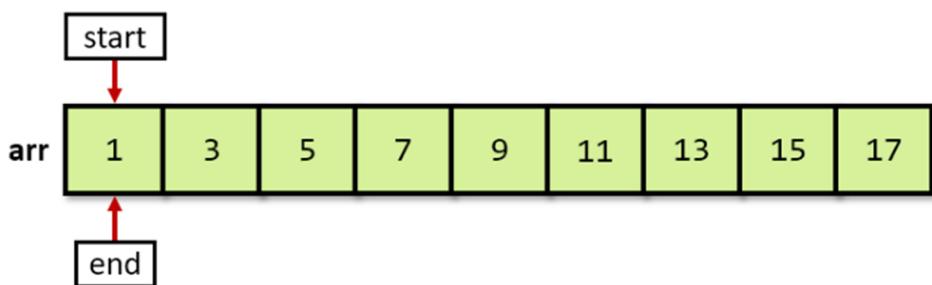
첫째 줄에 10000 이하의 자연수로 이루어진 수열의 길이 N 과 부분합의 기준이 될 S 가 주어진다. 그러면 이 수열에서 연속된 수들의 부분합 중에 그 합이 S 이상이 되는 것 중 가장 짧은 것의 길이를 구해야 한다. 만일 그러한 합을 만드는 것이 불가능하다면 0 을 출력한다.

이 경우 투 포인터와 구간합을 이용해서 시간 복잡도가 너무 커지지 않도록 프로그램을 작성해야 한다.

(문제를 해결하는데 필요한 개념)

● 투 포인터 알고리즘

1차원 배열에서 활용할 수 있는 알고리즘이다. 배열에서 특정 원소를 가리키는 2개의 포인터를 사용하여 목표하는 값을 찾을 수 있다. 리스트에 순차적으로 접근해야 할 때 두 개의 점의 위치를 기록하면서 처리하는 알고리즘이라고 할 수 있다.



● 구간 합

주어진 배열에서 특정 구간의 연속된 요소들의 합을 의미한다. 예를 들어, 배열에서 2번째 요소부터 4번째 요소까지의 구간 합은 $2 + 3 + 4 = 9$ 가 되는 식이다. 이러한 구간 합은 시간 복잡도를 줄이는 데 유용하게 활용될 수 있다.

array	1	8	7	4	3	5	6
prefix sum	1	9	16	20	23	28	31



(소스 코드)

CPP

```
#include<iostream>
#define ll long long
using namespace std;

ll arr[100001];
int main(void) {
    ios::sync_with_stdio(0);
    cin.tie(0);
    ll N, S;
    cin >> N >> S;
    for (int i = 1; i <= N; i++) {
        cin >> arr[i];
        arr[i] += arr[i - 1]; //누적합을 저장하기
    }

    int start = 1; //첫 번째 포인터. 합이 목표보다 크거나 같으면 오른쪽으로
    int end = 1; // 두 번째 포인터. 합이 목표보다 작으면 오른쪽으로

    int ans = 0; //합을 만드는 게 가능한지 체크
    int minL = 100000; //최소가 되는 길이

    while (start <= end) {
        ll sum = arr[end] - arr[start - 1];
        int L = end - start + 1;

        if (sum >= S) { //문제 잘 읽자 합이 S 이상이 되는 것을 보는 것이다
            if (L < minL) minL = L; //최소 길이 업데이트
        }
    }
}
```

```

        start++;
        ans = 1;
    }

    else {
        if (end == N) break; //end 가 더 커지지 않도록, N 이면 반복 멈추기
        end++;
    }

}

if (ans) cout << minL << '\n';
else cout << 0 << '\n';
return 0;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

투 포인터 알고리즘을 모른 채로 다이나믹 프로그래밍과 구간 합 문제들만 푼 상태에서 문제를 풀려고 했을 때 도저히 답이 나오지 않았던 문제이다. 애초에 배열을 가리키는 두 개의 포인터를 써야 한다는 생각을 못 했다. 그래서 인터넷에 투 포인터 알고리즘을 찾아본 후, 백준의 ‘투 포인터’ 태그에 들어가 가장 쉬운 문제부터 풀어보며 알고리즘을 익혀봤다. 그제서야 어떤 방향으로 나아갈지에 대한 가닥이 잡혀나갔다.

일단 1차원 배열을 선언한 후에, N 개의 수를 입력 받을 때 누적합을 저장하도록 코드를 짰다. 그리고 투 포인터 알고리즘을 사용하기 위해 배열을 가리키는 `start` 와 `end` 변수를 선언해주고, 두 포인터가 첫 번째 인덱스를 가리키도록 설정했다. `start` 와 `end` 사이의 합을 구간합을 구하도록 해야 하기 때문에 `start > end` 가 되면 반복문을 종료하도록 설정하고, 각각의 시행에서 누적합을 이용하여 구간 합 `sum` 과 최소 길이를 구하도록 했다. 그리고 이 `sum` 이 기준 값 `S` 보다 크거나 같으면 그 길이를 기반으로 최소 길이를 업데이트 하고, `start` 포인터가 가리키는 인덱스의 위치를 오른쪽으로 한 칸 옮겨주어 구간합의 크기를 줄이고, 기준 값 `S` 보다 작으면 구간 합을 키우기 위해 `end` 포인터를 오른쪽으로 옮기는 방식으로 투 포인터 알고리즘을 이용했다. 그리고 `end` 가 가리키는 인덱스가 N 보다 크면 안 되므로 더 커지면 반복을 멈추는 방식으로 코드를 짰다.

직접 알고리즘을 찾아서 공부해본 후 많은 시간을 들여 풀어본 문제라 의미가 깊고, 첫 골드 문제라 뜻깊은 것 같다. 아직 모르는 알고리즘이 많은데, 더 많이 공부해서 더 많은 문제를 풀 수 있으면 좋겠다.

문제 풀이 (BOJ 1004 어린왕자)

(문제 분석 및 프로그램 설계)

첫째 줄에 총 테스트 케이스의 개수 T 가 주어진다. 그리고 각 테스트 케이스에 대한 첫째 줄에는 출발점의 좌표 (x_1, y_1) 과 도착점의 좌표 (x_2, y_2) 가 주어진다. 그리고 둘째 줄에는 행성계의 개수 n 이 주어지며, 이후 n 줄에 걸쳐 n 개의 행성계의 각각의 중점과 반지름이 주어진다.

각 테스트 케이스에 대해 어떤 왕자가 출발점에서 도착점으로 가기 위해 거쳐야 할 최소의 행성계 진입/ 이탈 횟수를 출력한다.

두 좌표가 같은 원에 위치하는지를 판단하여 한 원에 하나만 포함돼 있는 경우에 횟수를 증가시키는 프로그램을 만들 수 있다.

(문제를 해결하는데 필요한 개념)

- 한 점과 원 사이의 위치 관계

점의 위치에 따라 크게 세 가지로 나눌 수 있다.

- 1) 원 밖에 있는 점

원의 중심과 점 사이의 거리가 원의 반지름보다 길다.

- 2) 원 위에 있는 점

원의 중심과 점 사이의 거리가 원의 반지름과 같다.

- 3) 원 내부에 있는 점

원의 중심과 점 사이의 거리가 원의 반지름보다 짧다.

(소스 코드)

CPP

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

typedef struct psys { //planet system
    int cx;
    int cy;
    int r;
} PLANET; //깔끔한 대입을 위한 구조체 설정
PLANET sys[51];

int InCheck(int x, int y, PLANET s); //Include Check 함수
```

```
int solve(void) {
    int x1, y1, x2, y2, cnt = 0;
    cin >> x1 >> y1 >> x2 >> y2;
    int n;
    cin >> n;
```

```

        for (int i = 0; i < n; i++) {
            cin >> sys[i].cx >> sys[i].cy >> sys[i].r;
            int DepCheck = InCheck(x1, y1, sys[i]); //Departures-Including check
            int ArrCheck = InCheck(x2, y2, sys[i]); //Arrivals-Including check
            if (DepCheck == 1 && ArrCheck == 1) continue;
            //둘 다 포함하면 경계 지날 필요 없음
            else if (DepCheck == 1) cnt++; //하나만 포함하면 경계를 지나야 하므로 count
            else if (ArrCheck == 1) cnt++;
        }
        cout << cnt << '\n';
        return 0;
    }

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int T;
    cin >> T;
    while (T--) {
        solve();
    }
    return 0;
}

int InCheck(int x, int y, PLANET s) {
    double CenDis = sqrt((x - s.cx) * (x - s.cx) + (y - s.cy) * (y - s.cy));
    if (CenDis < s.r) return 1; //단순히 두 중심을 이은 거리가 반지름보다 작으면 포함!
    else return 0;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

어떤 행성계에 두 좌표가 모두 속하거나 하나만 속하거나 둘 다 안 속하는 경우를 생각하면 풀린다는 걸 깨달은 후 빨리 푼 문제다. 함수에 간단하게 대입하기 위해 구조체를 선언해서 풀었는데, 외려 더 복잡하게 푼 것이 아닌가 하는 생각도 듈다.

그리고 if-else if 문에서 조건을 짜는 실력이 너무 부족함을 느낀다. 그냥 마음 가는 대로 설정하다가 반례를 고려 못하는 경우가 많아서 정확한 조건을 설정할 필요성을 느꼈다.

얼핏 보기엔 복잡해 보이는 문제도 어떤 지식을 알아야 하는지 알면 쉽게 풀린다는 느낌을 받았다. 현재 잘 알지 못해서 못 푸는 문제들이 수두룩한데, 알기만 하면 잘 풀 수 있겠다는 생각이 들고, 알고리즘 공부를 본격적으로 열심히 해봐야겠다.



스터디 5주차 보고서

(8/03 ~ 8/10)

작성자 : 심현서

작성일 : 2025.08.10

문제 풀이 (BOJ 11758 CCW)

(문제 분석 및 프로그램 설계)

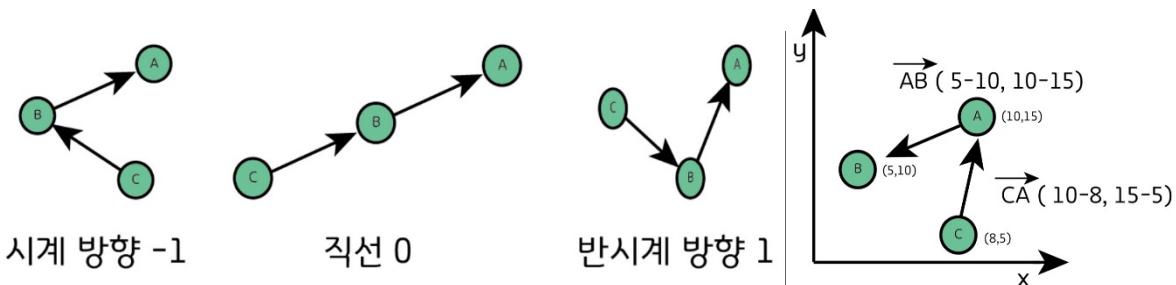
2차원 좌표 평면 위에 있는 서로 다른 점 3개 P1, P2, P3의 좌표가 주어진다. P1, P2, P3를 순서대로 이은 선분이 어떤 방향을 이루고 있는지 구하고 선분이 반시계 방향을 나타내면 1, 시계 방향이면 -1, 일직선이면 0을 출력한다.

(문제를 해결하는데 필요한 개념)

● CCW 알고리즘

CCW(Counter Clock Wise) 알고리즘은 3개의 점 A, B, C가 있을 때 이 점 3개를 이은 직선의 방향을 알고자 할 때 유용한 기하 알고리즘이다.

경우의 수는 총 3가지가 있으며 시계, 반시계, 직선이 있으며 이는 외적을 통하여 구할 수 있으며 외적의 결과가 음수이면 시계 방향, 직선은 0, 반시계 방향일 경우 양수가 나오게 된다.



● 외적

외적: 두 개의 3차원 공간상의 벡터에 대한 연산. 외적은 다음과 같은 특징이 있다.

-기호로는 \times 를 사용한다

-두 벡터를 외적 하면 두 벡터의 수직 벡터 즉 벡터가 나온다 내적은 스칼라 값이 나온다.

-외적은 교환 법칙이 성립하지 않는다. $a \times b$ 와 $b \times a$ 는 다르다.

-두 벡터의 외적은 두 벡터가 만들어내는 평행사변형의 넓이이다.

Matrix notation [edit]

The cross product can also be expressed as the formal determinant:^[note 1]

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

This determinant can be computed using Sarrus's rule or cofactor expansion. Using Sarrus's rule, it expands to

$$\begin{aligned} \mathbf{a} \times \mathbf{b} &= (a_2 b_3 \mathbf{i} + a_3 b_1 \mathbf{j} + a_1 b_2 \mathbf{k}) - (a_3 b_2 \mathbf{i} + a_1 b_3 \mathbf{j} + a_2 b_1 \mathbf{k}) \\ &= (a_2 b_3 - a_3 b_2) \mathbf{i} + (a_3 b_1 - a_1 b_3) \mathbf{j} + (a_1 b_2 - a_2 b_1) \mathbf{k}. \end{aligned}$$

Using cofactor expansion along the first row instead, it expands to^[6]

$$\begin{aligned} \mathbf{a} \times \mathbf{b} &= \begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix} \mathbf{i} - \begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} \mathbf{j} + \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} \mathbf{k} \\ &= (a_2 b_3 - a_3 b_2) \mathbf{i} - (a_1 b_3 - a_3 b_1) \mathbf{j} + (a_1 b_2 - a_2 b_1) \mathbf{k}, \end{aligned}$$

which gives the components of the resulting vector directly.

(소스 코드)

CPP

```
1 #include<iostream>
2 #include<utility>
3 #define ll long long
4 using namespace std;
5 pair<int, int> P[3];
6 //외적을 알면 쉽다는 것에 대한 무력감.
7 ll CCW(pair<int, int> p1, pair<int, int> p2, pair<int, int> p3);
8 int solve(void) {
9     for (int i = 0; i < 3; i++) cin >> P[i].first >> P[i].second;
10
11    ll check = CCW(P[0], P[1], P[2]);
12    if (check > 0) cout << 1;
13    else if (check < 0) cout << -1;
14    else cout << 0;
15
16    return 0;
17 }
18
19 ll CCW(pair<int, int> p1, pair<int, int> p2, pair<int, int> p3) {
20     ll ux = p2.first - p1.first; //1LL 곱하는 처리도 가능하대
21     ll uy = p2.second - p1.second;
22     ll vx = p3.first - p1.first;
23     ll vy = p3.second - p1.second;
24     return 1LL * ux * vy - uy * vx;
25 }
26
27
28 int main() {
29     ios::sync_with_stdio(0);
30     cin.tie(0);
31     solve();
32     return 0;
33 }
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

3. 처음에는 아무것도 모르고 P1P2의 직선을 설정한 후, 그 직선을 기준으로 P3가 어디에 있는지 판단하는 방식을 생각했었다. 그래서 매우 복잡한 방식으로 코드가 작성되었고, 너무 많은 경우를 잘 처리하지 못해서 어디가 잘못됐는지 모르는 채로 틀렸던 것 같다. 그래서 CCW 와 외적을 찾아보게 되었다.
4. 외적을 알면 매우 쉬운 방법으로 결과가 나온다는 걸 알았다. naive 하게 생각하는 것도 좋지만 필요하면 관련된 새로운 알고리즘을 찾아보는 게 최선일 수 있겠다.

문제 풀이 (BOJ 1850 최대공약수)

(문제 분석 및 프로그램 설계)

모든 자리가 1로만 이루어져 있는 두 자연수 A와 B가 주어진다. 이 때, A와 B의 최대 공약수를 구하는 프로그램을 작성해야 한다. 예를 들어, A가 111이고, B가 1111인 경우에 A와 B의 최대공약수는 1이고, A가 111이고, B가 111111인 경우에는 최대공약수가 111이다. A와 B를 이루는 1의 개수가 주어진다.

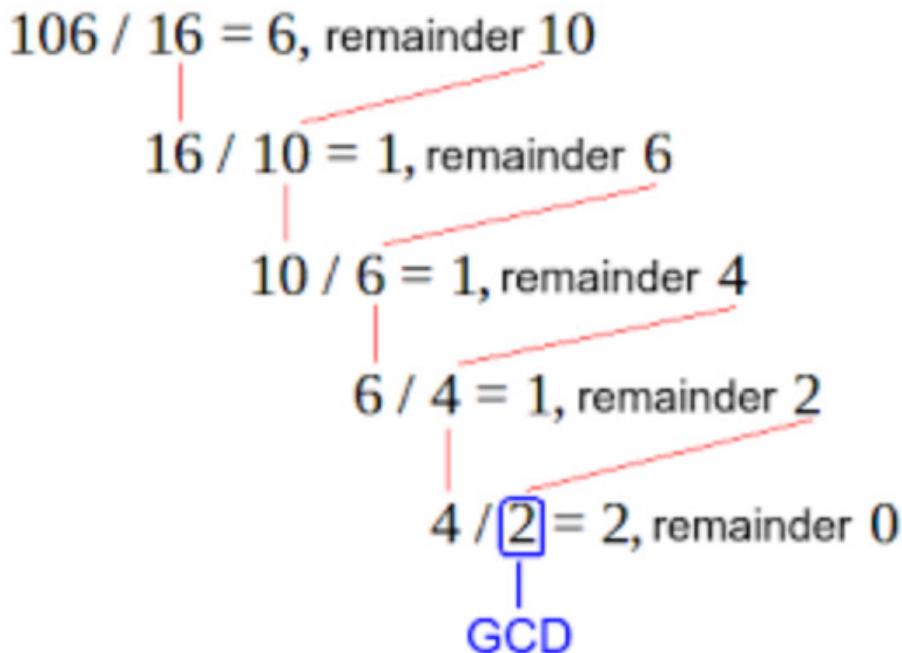
유클리드 호제법을 사용하면, 1의 개수를 그대로 이용해서 간단하게 최대공약수를 구할 수 있다.

(문제를 해결하는데 필요한 개념)

● 유클리드 호제법

유클리드 호제법(-互除法, Euclidean Algorithm)은 2개의 자연수 또는 정식(整式)의 최대 공약수(Greatest Common Divisor)를 구하는 알고리즘의 하나이다. 호제법이란 말은 두 수가 서로(互) 상대방 수를 나누어(除)서 결국 원하는 수를 얻는 알고리즘을 나타낸다.

2개의 자연수(또는 정식) a, b 에 대해서 a 를 b 로 나눈 나머지를 r 이라 하면(단, $a > b$), a 와 b 의 최대공약수는 b 와 r 의 최대공약수와 같다. 이 성질에 따라, b 를 r 로 나눈 나머지 r' 를 구하고, 다시 r 를 r' 로 나눈 나머지를 구하는 과정을 반복하여 나머지가 0이 되었을 때 나누는 수가 a 와 b 의 최대공약수이다.



(소스 코드)

CPP

```
1 #include<iostream>
2 #include<algorithm>
3 #define ll long long
4 using namespace std;
5 int gcd(ll A, ll B) {
6     if (B == 0) return A;
7     else return gcd(B, A % B); // 유클리드 호제법
8 }
9 //1이 몇 개씩인지를 A, B로 주어져도 그냥 유클리드 호제법으로 끝남!
10 //11111과 111 -> 111이 1번 들어감, 111111과 111 -> 111이 2번 들어감
11 //5와 3 -> 3이 1번 들어감, 6과 3 -> 3이 2번 들어감
12
13 int solve(void){
14     ll A, B;
15     cin >> A >> B;
16     if (A < B) swap(A, B); // A > B로 세팅!
17     int ones = gcd(A, B);
18     for (int i = 0; i < ones; i++) cout << 1;
19     cout << '\n';
20
21     return 0;
22 }
23
24 int main() {
25     ios::sync_with_stdio(0);
26     cin.tie(0);
27     solve();
28     return 0;
29 }
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

알고리즘 스터디에서 배운 최대공약수를 구하는 효율적인 방법인 유클리드 호제법을 이용했는데 간단해서 빠르게 풀 수 있었다. DFS에 대한 이해를 높일 수 있는 문제였던 거 같다. 그런데 유클리드 호제법에 대한 이해는 좀 부족해서 좀더 공부를 해봐야겠다.



스터디 6주차 보고서

(8/11 ~ 8/17)

작성자 : 심현서

작성일 : 2025.08.19..

문제 풀이 (BOJ 1620 나는야 포켓몬 마스터 이다솜)

(문제 분석 및 프로그램 설계)

첫째 줄에는 도감에 수록되어 있는 포켓몬의 개수 N 이랑 맞춰야 하는 문제의 개수 M 이 주어진다. 둘째 줄부터 N 개의 줄에 포켓몬의 번호가 1번인 포켓몬부터 N 번에 해당하는 포켓몬까지 한 줄에 하나씩 입력으로 들어온다. 그 다음 줄부터 총 M 개의 줄에 맞춰야 하는 문제가 입력으로 들어온다. 이때 문제가 알파벳으로만 들어오면 포켓몬 번호를 말해야 하고, 숫자로만 들어오면, 포켓몬 번호에 해당하는 문자를 출력해야 한다. 해시를 이용하여 시간 초과가 나지 않게 프로그램을 짜야한다.

(문제를 해결하는데 필요한 개념)

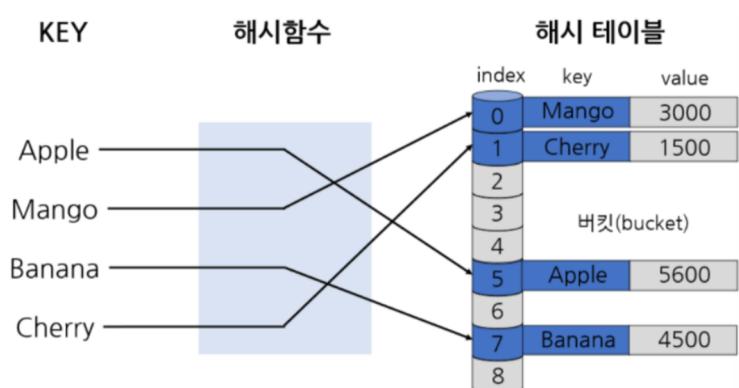
● 해시

컴퓨팅에서 키를 값에 매핑할 수 있는 구조인, 연관 배열 추가에 사용되는 자료 구조이다. 저장 또는 검색 등에서 자주 활용된다. 해시 테이블은 해시 함수를 사용하여 색인(index)을 버킷(bucket)이나 슬롯(slot)의 배열로 계산한다. 데이터를 다루는 기법 중에 하나로 데이터의 검색과 저장이 아주 빠르게 진행된다

키(KEY)에 데이터(Value)를 매핑할 수 있는 데이터 구조

해시 함수를 통해 키의 데이터를 배열에 저장할 수 있는 주소(인덱스 번호)를 계산

키를 통해서 저장된 데이터를 빠르게 찾고, 저장 및 탐색 속도가 획기적으로 빨라짐



(소스 코드)

CPP

```

1 #include<iostream>
2 #include<unordered_map>
3 #include<string>
4 using namespace std;
5 unordered_map<string, int> pokemon;
6 string pname[100001];
7 int main(){
8     ios::sync_with_stdio(0);
9     cin.tie(0);
10    int N, M;
11    cin >> N >> M;
12    for (int i = 1; i <= N; i++) {
13        cin >> pname[i];
14        pokemon[pname[i]] = i;
15    }
16
17    for (int i = 0; i < M; i++) {
18        string tmp;
19        cin >> tmp;
20        if(tmp[0] >= '1' && tmp[0] <= '9') {
21            cout << pname[stoi(tmp)] << '\n';
22        }
23        else {
24            cout << pokemon[tmp] << '\n';
25        }
26    }
27
28    return 0;
29 }

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀 점)

해시 알고리즘을 활용한 STL인 <unordered_map>을 사용하여 문제를 해결했다. String 을 인덱스로 하고 int 를 value 값으로 하는 map 인 pokemon 을 선언해주고, 다양한 포켓몬 이름을 담을 수 있는 string 배열 pname 을 선언하고 시작했다. 처음에는 배열로 할 생각을 못 하고 그냥 string pname; 으로 하나씩 받고 map 처리? 를 했었는데, 문제에서 번호를 통한 포켓몬 이름을 알아내야 할 필요성이 있다는 걸 알게 됐고 배열로 선언하는 방식을 사용하였다.

아까 선언한 map 에 포켓몬 이름 넣어주어서, 그걸 hash 함수를 통해 index 로 바꿔주고 거기에 포켓몬 이름을 받은 순서의 번호를 값으로 넣어주면서 계속 받았다.

그 후 포켓몬의 이름이나 번호로 답을 낼 때, 숫자면 0번째 인덱스가 숫자일 거라는 판단을 이용해 if else 문을 만들었고, 해시 함수를 이용하여 시간초과 없이 잘 풀 수 있었다.

유용한 자료구조를 알게 되었고, 앞으로 더 잘 써먹어봐야겠다

(제대로 써 내려간 건지 잘 모르겠다)

문제 풀이 (BOJ 11404 플로이드)

(문제 분석 및 프로그램 설계)

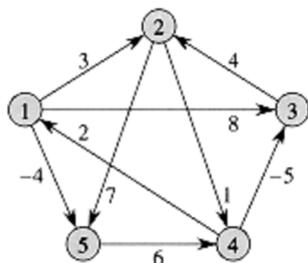
첫째 줄에 도시의 개수 n 이 주어지고 한 도시에서 출발하여 다른 도시에 도착하는 버스의 개수 m 이 주어진다. 각 버스는 한 번 사용할 때 필요한 비용이 있고, 셋째 줄부터 $m+2$ 줄까지 다음과 같은 버스의 정보가 주어진다. 먼저 처음에는 그 버스의 출발 도시의 번호가 주어진다. 버스의 정보는 버스의 시작 도시 a , 도착 도시 b , 한 번 타는데 필요한 비용 c 로 이루어져 있다. 시작 도시와 도착 도시가 같은 경우는 없다.

모든 도시의 쌍 (A, B) 에 대해서 도시 A 에서 B 로 가는데 필요한 비용의 최솟값을 구하는 프로그램을 작성해야 한다.

(문제를 해결하는데 필요한 개념)

● 플로이드-워셜 알고리즘(Floyd-Warshall)

모든 최단 경로를 구하는 알고리즘이다. 한 번 실행하여 모든 노드 간 최단 경로를 구할 수 있다. 모든 노드 간의 최단거리를 구해야 하므로 2차원 인접 행렬을 구성한다. 알고리즘은 여러 라운드로 구성된다. 라운드마다 각 경로에서 새로운 중간 노드로 사용할 수 있는 노드를 선택하고, 더 짧은 길이를 선택하여 줄이는 과정을 반복한다.



$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad D^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

(소스 코드)

CPP

```
1 #include<iostream>
2 #include<algorithm>
3 #define INF (100 * 100000 + 5) //infimum
4 using namespace std;
5
6 int adj[101][101]; //adjacent matrix
7 int main() {
8     ios::sync_with_stdio(0);
9     cin.tie(0);
10
11     int n, m;
12     cin >> n;
13     cin >> m;
14
15     for (int i = 1; i <= n; i++) {
16         for (int j = 1; j <= n; j++) {
17             if (j == i) continue; //INF로 채웠을 때 0이 안 되는 문제가 발생했었음
18             adj[i][j] = INF; //일단 diagonal 빼고 다 채워주기
19         }
20     }
21
22     for (int i = 0; i < m; i++) {
23         int st, en, co; //start, end, cost
24         cin >> st >> en >> co;
25         adj[st][en] = min(adj[st][en], co); //출발지와 도착지가 구분됨! directed
26     }
27
28     for (int k = 1; k <= n; k++) {
29         for (int i = 1; i <= n; i++) {
30             for (int j = 1; j <= n; j++) {
31                 adj[i][j] = min(adj[i][j], adj[i][k] + adj[k][j]);
32             }
33         }
34     }
35
36     for (int i = 1; i <= n; i++) {
37         for (int j = 1; j <= n; j++) {
38             if (adj[i][j] == INF) cout << 0 << ' ';
39             else cout << adj[i][j] << ' ';
40         }
41         cout << '\n';
42     }
43
44     return 0;
45 }
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

알고리즘 스터디에서 플로이드-워셜 알고리즘 을 알게 된 후 푼 문제이다. 일단 인접 행렬을 전역적으로 선언한 후, $n \times n$ 행렬의 주대각선을 제외한 부분에 INF(두 정점 간의 관계가 없다는 뜻으로, 가장 큰 수를 넣어주는 것)로 채웠다. 그리고 이어져 있는 두 정점을 입력받고, 이를 이용해서 플로이드 - 워셜 알고리즘을 실행하면 되었다.

정확한 개념 원리는 모르지만, $k \rightarrow i \rightarrow j$ 순서로 for 문을 작성해서 인접행렬의 각각의 성분에 가장 최소의 cost 가 입력되도록 하여 문제를 마무리 했다.

풀긴 풀었지만 정확한 개념을 모르는 게 약간 찝찝한 느낌이다. 일단 외우고 이용하겠지만 나중에는 알고리즘이 돌아가는 대강의 원리를 이해할 수 있으면 좋겠다.



스터디 2주차 보고서

(7/14 ~ 7/20)

작성자 : 원종호

작성일 : 2025.07.20

문제 풀이 (BOJ 11657 타임머신)

(문제 분석 및 프로그램 설계)

문제

N 개의 도시가 있다. 그리고 한 도시에서 출발하여 다른 도시에 도착하는 버스가 M 개 있다. 각 버스는 A, B, C 로 나타낼 수 있는데, A 는 시작도시, B 는 도착도시, C 는 버스를 타고 이동하는데 걸리는 시간이다. 시간 C 가 양수가 아닌 경우가 있다. $C = 0$ 인 경우는 순간 이동을 하는 경우, $C < 0$ 인 경우는 타임머신으로 시간을 되돌아가는 경우이다.

1 번 도시에서 출발해서 나머지 도시로 가는 가장 빠른 시간을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 도시의 개수 N ($1 \leq N \leq 500$), 버스 노선의 개수 M ($1 \leq M \leq 6,000$)이 주어진다. 둘째 줄부터 M 개의 줄에는 버스 노선의 정보 A, B, C ($1 \leq A, B \leq N$, $-10,000 \leq C \leq 10,000$)가 주어진다.

출력

만약 1 번 도시에서 출발해 어떤 도시로 가는 과정에서 시간을 무한히 오래 전으로 되돌릴 수 있다면 첫째 줄에 -1 을 출력한다. 그렇지 않다면 $N-1$ 개 줄에 걸쳐 각 줄에 1 번 도시에서 출발해 2 번 도시, 3 번 도시, ..., N 번 도시로 가는 가장 빠른 시간을 순서대로 출력한다. 만약 해당 도시로 가는 경로가 없다면 대신 -1 을 출력한다.

(문제를 해결하는데 필요한 개념)

벨만-포드 알고리즘

벨만 포드 알고리즘은 최단 거리를 구하는 알고리즘으로 다익스트라 알고리즘과는 다르게 음수 가중치가 있어도 수행할 수 있다

또한 최단거리 배열에서 업데이트 반복 횟수는 노드 개수-1개라는 점에서 최단 거리를 구한 후 한번 더 사이클을 돋아 최단거리가 변화한다면 음수 사이클이 있음을 알수있다.

위 문제도 이와 같은 방식으로 진행하여 음수 사이클 여부를 알 수 있었다.

(소스 코드)

CPP

```

#include <vector>
#include <iostream>
#include <tuple>
#include <limits.h>
using namespace std;
typedef tuple<int,int,int> edge;
static vector<edge> edges;
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int N,M;
    cin >> N >> M;
    for(int i=0;i<M;i++){
        int s,e,w;
        cin >> s >> e >> w;
        edges.push_back(make_tuple(s,e,w));
    }
    vector<long> D(N+1,LONG_MAX);
    D[1]=0;
    for(int i=1;i<N;i++){
        for(int j=0;j<M;j++){
            edge medge = edges[j];
            int start = get<0>(medge);
            int end = get<1>(medge);
            int time = get<2>(medge);

            if(D[start]!=LONG_MAX&&D[end]>D[start]+time){
                D[end]=D[start]+time;
            }
        }
    }
    bool mCycle=false;
    for(int i=0;i<M;i++){
        edge medge = edges[i];
        int start = get<0>(medge);
        int end = get<1>(medge);
        int time = get<2>(medge);
        if(D[start]!=LONG_MAX&&D[end]>D[start]+time) mCycle=true;
    }
    if(!mCycle){
        for(int i=2;i<=N;i++){
            if(D[i]==LONG_MAX)
                cout << -1 << "\n";
            else cout << D[i] << "\n";
        }
    }
}

```

```
 }else cout << -1;  
 return 0;  
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

개념만 알고있으면 풀 수 있는 문제였기에 어렵지 않게 풀 수 있었다.

문제 풀이 (BOJ 11404 플로이드)

(문제 분석 및 프로그램 설계)

문제

$n(2 \leq n \leq 100)$ 개의 도시가 있다. 그리고 한 도시에서 출발하여 다른 도시에 도착하는 $m(1 \leq m \leq 100,000)$ 개의 버스가 있다. 각 버스는 한 번 사용할 때 필요한 비용이 있다.

모든 도시의 쌍 (A, B)에 대해서 도시 A 에서 B 로 가는데 필요한 비용의 최솟값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 도시의 개수 n 이 주어지고 둘째 줄에는 버스의 개수 m 이 주어진다. 그리고 셋째 줄부터 $m+2$ 줄까지 다음과 같은 버스의 정보가 주어진다. 먼저 처음에는 그 버스의 출발 도시의 번호가 주어진다. 버스의 정보는 버스의 시작 도시 a , 도착 도시 b , 한 번 타는데 필요한 비용 c 로 이루어져 있다. 시작 도시와 도착 도시가 같은 경우는 없다. 비용은 100,000 보다 작거나 같은 자연수이다.

시작 도시와 도착 도시를 연결하는 노선은 하나가 아닐 수 있다.

출력

n 개의 줄을 출력해야 한다. i 번째 줄에 출력하는 j 번째 숫자는 도시 i 에서 j 로 가는데 필요한 최소 비용이다. 만약, i 에서 j 로 갈 수 없는 경우에는 그 자리에 0을 출력한다.

(문제를 해결하는데 필요한 개념)

플로이드-워셜 알고리즘을 이용하여 쉽게 풀 수 있는 문제다

플로이드 워셜 알고리즘의 시간복잡도가 $O(v^3)$ 인것을 고려하여 노드 수가 적고, 모든 도시의 쌍에 대해 비용을 찾아야 하므로 플로이드 워셜 알고리즘을 써야한다는것을 어렵지 않게 알 수 있다.

(소스 코드)

CPP

```

#include <vector>
#include <iostream>
#include <limits.h>
using namespace std;
static int N,M;
static long D[101][101];
#define INF INT_MAX

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> N >> M;
    for(int i=1;i<=N;i++){
        for(int j=1;j<=N;j++){
            if(i==j) D[i][j]=0;
            else D[i][j]=INF;
        }
    }
    for(int i=0;i<M;i++){
        int s,e,v;
        cin >> s >> e >> v;
        if(D[s][e]>v)
            D[s][e]=v;
    }
    for(int k=1;k<=N;k++){
        for(int i=1;i<=N;i++){
            for(int j=1;j<=N;j++){
                if(D[i][j]>D[i][k]+D[k][j])
                    D[i][j]=D[i][k]+D[k][j];
            }
        }
    }
    for(int i=1;i<=N;i++){
        for(int j=1;j<=N;j++){
            if(D[i][j]==INF) cout << "0 ";
            else cout << D[i][j] << " ";
        }
        cout << "\n";
    }
    return 0;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

시작 도시와 도착 도시를 연결하는 노선은 1개가 아닐 수 있다는 점을 읽지 못했고 그 때문에 계속 테스트케이스에서 실패를 했다. 그래서 D[s][e]에 대해 입력받을땐 최솟값만 입력받게 하니 정답을 출력했다.

그래프 알고리즘을 배우고 있는데 종류가 다양해서 숙달하는데 시간이 걸릴것 같다. 또 어떤 때에 어떤 알고리즘을 사용해야 하는지 잘 감이 안와서 어렵다.



스터디 3주차 보고서

(7/20 ~ 7/27)

작성자 : 원종호

작성일 : 2025.07.27

문제 풀이 (BOJ 1238 파티)

(문제 분석 및 프로그램 설계)

문제

N 개의 숫자로 구분된 각각의 마을에 한 명의 학생이 살고 있다.

어느 날 이 N 명의 학생이 $X (1 \leq X \leq N)$ 번 마을에 모여서 파티를 벌이기로 했다. 이 마을 사이에는 총 M 개의 단방향 도로들이 있고 i 번째 길을 지나는데 $T_i (1 \leq T_i \leq 100)$ 의 시간을 소비한다.

각각의 학생들은 파티에 참석하기 위해 걸어가서 다시 그들의 마을로 돌아와야 한다. 하지만 이 학생들은 워낙 게을러서 최단 시간에 오고 가기를 원한다.

이 도로들은 단방향이기 때문에 아마 그들이 오고 가는 길이 다를지도 모른다. N 명의 학생들 중 오고 가는데 가장 많은 시간을 소비하는 학생은 누구일지 구하여라.

입력

첫째 줄에 $N (1 \leq N \leq 1,000)$, $M (1 \leq M \leq 10,000)$, X 가 공백으로 구분되어 입력된다. 두 번째 줄부터 $M+1$ 번째 줄까지 i 번째 도로의 시작점, 끝점, 그리고 이 도로를 지나는데 필요한 소요시간 T_i 가 들어온다. 시작점과 끝점이 같은 도로는 없으며, 시작점과 한 도시 A 에서 다른 도시 B 로 가는 도로의 개수는 최대 1 개이다.

모든 학생들은 집에서 X 에 갈 수 있고, X 에서 집으로 돌아올 수 있는 데이터만 입력으로 주어진다.

출력

첫 번째 줄에 N 명의 학생들 중 오고 가는데 가장 오래 걸리는 학생의 소요시간을 출력한다.

(문제를 해결하는데 필요한 개념)

다익스트라 알고리즘을 통해 구할 수 있다

파티에 참석하기 위해 가는 길은 단방향 도로의 방향을 전부 뒤집어서 파티가 일어나는 장소를 시작점으로 다익스트라 알고리즘을 통해 최단거리를 구하고, 파티에서 마을로 돌아가는 길은 도로의 방향을 뒤집지 않은 상태로 다익스트라 알고리즘으로 최단거리를 구한 뒤 더하면 된다.

(소스 코드)

CPP

```

#include <algorithm>
#include <vector>
#include <iostream>
#include <queue>
using namespace std;
typedef pair<int,int> edge;
#define INF 210000000
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int N,M,X;
    cin >> N >> M >> X;
    vector<vector<edge>> A(N+1);
    vector<vector<edge>> rA(N+1);
    for(int i=0;i<M;i++){
        int s,e,v;
        cin >> s >> e >> v;
        A[e].push_back(make_pair(s,v));
        rA[s].push_back(make_pair(e,v));
    }
    vector<int> D(N+1,INF);
    vector<int> rD(N+1,INF);
    D[X]=0;
    rD[X]=0;
    priority_queue<edge, vector<edge>, greater<edge>> B;
    B.push(make_pair(0,X));
    while(!B.empty()){
        int dist = B.top().first;
        int cur = B.top().second;
        B.pop();

        for(edge i : A[cur]){
            if(D[i.first]>i.second+D[cur]){
                D[i.first]=i.second+D[cur];
                B.push(make_pair(D[i.first],i.first));
            }
        }
        B.push(make_pair(0,X));
    }
    while(!B.empty()){
        int dist = B.top().first;
        int cur = B.top().second;
        B.pop();

        for(edge i : rA[cur]){
            if(rD[i.first]>i.second+rD[cur]){
                rD[i.first]=i.second+rD[cur];
            }
        }
    }
}

```

```
        B.push(make_pair(rD[i.first],i.first));
    }
}
int max=0;
for(int i=1;i<=N;i++){
    if(max<D[i]+rD[i]) max = D[i]+rD[i];
}
cout << max;
return 0;
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

문제 풀이 (BOJ 1197 최소 스패닝 트리)

(문제 분석 및 프로그램 설계)

문제

그래프가 주어졌을 때, 그 그래프의 최소 스패닝 트리를 구하는 프로그램을 작성하시오.

최소 스패닝 트리는, 주어진 그래프의 모든 정점들을 연결하는 부분 그래프 중에서 그 가중치의 합이 최소인 트리를 말한다.

입력

첫째 줄에 정점의 개수 $V(1 \leq V \leq 10,000)$ 와 간선의 개수 $E(1 \leq E \leq 100,000)$ 가 주어진다. 다음 E 개의 줄에는 각 간선에 대한 정보를 나타내는 세 정수 A, B, C 가 주어진다. 이는 A 번 정점과 B 번 정점이 가중치 C 인 간선으로 연결되어 있다는 의미이다. C 는 음수일 수도 있으며, 절댓값이 1,000,000 을 넘지 않는다.

그래프의 정점은 1번부터 V 번까지 번호가 매겨져 있고, 임의의 두 정점 사이에 경로가 있다. 최소 스패닝 트리의 가중치가 -2,147,483,648 보다 크거나 같고, 2,147,483,647 보다 작거나 같은 데이터만 입력으로 주어진다.

출력

첫째 줄에 최소 스패닝 트리의 가중치를 출력한다.

(문제를 해결하는데 필요한 개념)

최소 스패닝 트리를 구하기 위해 크루스칼 알고리즘을 사용한다.

가장 가중치가 작은 간선만을 택하며 택한 간선의 개수가 $N-1$ 개 일때까지 연결하는 방식 단, 이때 사이클이 생기는 간선은 택하지 않는다.

최소 스패닝 트리의 간선은 $N-1$ 개임을 이용한 방식,

사이클 여부를 알기위해 유니온파인드를 같이 사용한다.

(소스 코드)

CPP

```

#include <queue>
#include <iostream>
using namespace std;
void Union(int a,int b);
int find(int a);
static vector<int> parent;

typedef struct Edge {
    int s,e,v;
    bool operator > (const Edge& temp) const{
        return v > temp.v;
    }
} Edge;

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

    int N,M;
    cin >> N >> M;
    priority_queue<Edge , vector<Edge>, greater<Edge>> pq; // 오름차순 정렬
    parent.resize(N+1);

    for(int i=0;i<=N;i++) parent[i]=i;
    for(int i=0;i<M;i++){
        int s,e,v;
        cin >> s >> e >> v;
        pq.push(Edge{s,e,v});
    }

    int useEdge = 0;
    int result =0;
    while(useEdge<N-1){
        Edge now = pq.top();
        pq.pop();

        if(find(now.s)!=find(now.e)){
            Union(now.s,now.e);
            result=result+now.v;
            useEdge++;
        }
    }
    cout << result;
    return 0;
}

```

```
void Union(int a,int b){  
    a = find(a);  
    b = find(b);  
  
    if(a!=b) parent[b]=a;  
}  
  
int find(int a){  
    if(a==parent[a]) return a;  
    else{  
        return parent[a]=find(parent[a]);  
    }  
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

Edge 구조체에서 간선 가중치를 기준으로 정렬하기 위해 가중치 v를 가장 앞에 위치하게 할까 했지만

구조체의 operator를 활용하는 방식이 있다는 것을 알게되어 그 방식을 채택했다.

아직 operator에 대한 이해가 부족하지만 앞으로 우선순위 큐를 쓸때마다 최대한 쓰는 연습을 해 숙달할 것이다.



스터디 4주차 보고서

(7/27 ~ 8/03)

작성자 : 원종호

작성일 : 2025.08.03

문제 풀이 (BOJ 20040 사이클 게임)

(문제 분석 및 프로그램 설계)

문제

사이클 게임은 두 명의 플레이어가 차례대로 돌아가며 진행하는 게임으로, 선 플레이어가 홀수 번째 차례를, 후 플레이어가 짝수 번째 차례를 진행한다. 게임 시작 시 0부터 $n - 1$ 까지 고유한 번호가 부여된 평면 상의 점 n 개가 주어지며, 이 중 어느 세 점도 일직선 위에 놓이지 않는다. 매 차례마다 플레이어는 두 점을 선택해서 이를 연결하는 선분을 긋는데, 이전에 그린 선분을 다시 그을 수는 없지만 이미 그린 다른 선분과 교차하는 것은 가능하다. 게임을 진행하다가 처음으로 사이클을 완성하는 순간 게임이 종료된다. 사이클 C는 플레이어가 그런 선분들의 부분집합으로, 다음 조건을 만족한다.

C에 속한 임의의 선분의 한 끝점에서 출발하여 모든 선분을 한 번씩만 지나서 출발점으로 되돌아올 수 있다.

문제는 선분을 여러 개 그리다 보면 사이클이 완성 되었는지의 여부를 판단하기 어려워 이미 사이클이 완성되었음에도 불구하고 게임을 계속 진행하게 될 수 있다는 것이다. 이 문제를 해결하기 위해서 게임의 진행 상황이 주어지면 몇 번째 차례에서 사이클이 완성되었는지, 혹은 아직 게임이 진행 중인지를 판단하는 프로그램을 작성하려 한다.

입력으로 점의 개수 n 과 m 번째 차례까지의 게임 진행 상황이 주어지면 사이클이 완성 되었는지를 판단하고, 완성되었다면 몇 번째 차례에서 처음으로 사이클이 완성된 것인지를 출력하는 프로그램을 작성하시오.

입력

입력은 표준입력을 사용한다. 입력의 첫 번째 줄에는 점의 개수를 나타내는 정수 $3 \leq n \leq 500,000$ 과 진행된 차례의 수를 나타내는 정수 $3 \leq m \leq 1,000,000$ 이 주어진다. 게임에서 사용하는 n 개의 점에는 0부터 $n - 1$ 까지 고유한 번호가 부여되어 있으며, 이 중 어느 세 점도 일직선 위에 놓이지 않는다. 이어지는 m 개의 입력 줄에는 각각 i 번째 차례에 해당 플레이어가 선택한 두 점의 번호가 주어진다 ($1 \leq i \leq m$).

출력

출력은 표준출력을 사용한다. 입력으로 주어진 케이스에 대해, m 번째 차례까지 게임을 진행한 상황에서 이미 게임이 종료되었다면 사이클이 처음으로 만들어진 차례의 번호를 양의 정수로 출력하고, m 번의 차례를 모두 처리한 이후에도 종료되지 않았다면 0을 출력한다.

(문제를 해결하는데 필요한 개념)

유니온 파인드

각 노드를 대표하는 노드를 담는 배열을 통해 유니온 연산은 두 노드를 하나로 연결, 파인드 연산은 해당 노드의 대표값을 찾는 연산이다.

(소스 코드)

CPP

```

#include <iostream>
#include <vector>
using namespace std;
static vector<int> parent;
int find(int a);
void Union(int a,int b);
int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    int N,M;
    cin >> N >> M;
    parent.resize(N);
    for(int i=0;i<N;i++){
        parent[i]=i;
    }
    for(int i=1;i<=M;i++){
        int a,b;
        cin >> a >> b;
        if(find(b)!=find(a)){
            Union(a,b);
        }else{
            cout << i;
            return 0;
        }
    }
    cout << 0;
    return 0;
}
int find(int a){
    if(a==parent[a])return a;
    else{
        return parent[a]=find(parent[a]);
    }
}
void Union(int a,int b){
    a = find(a);
    b = find(b);

    if(a!=b){
        parent[b]=a;
    }
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

이전에 크루스칼 알고리즘을 통해 최소 스패닝 트리를 찾을 때 사이클임을 확인하기 위해 유니온 파인드 연산을 썼던 점이 기억나 그대로 활용했다.

도서관에서 데스크 업무 보는데 할 일 없어서 몰래 했다 개꿀

문제 풀이 (BOJ 2448 별 찍기 - 11)

(문제 분석 및 프로그램 설계)

문제

예제를 보고 규칙을 유추한 뒤에 별을 찍어 보세요.

입력

첫째 줄에 N 이 주어진다. N 은 항상 $3 \times 2k$ 수이다. ($3, 6, 12, 24, 48, \dots$) ($0 \leq k \leq 10$, k 는 정수)

출력

첫째 줄부터 N 번째 줄까지 별을 출력한다.

예제 입력 1

24

예제 출력 1

*
* *

* * * *
***** *****
* * * *
***** *****
* * * * *
* * * * * *
***** ***** ***** *****
* * * *
* * * *
***** *****
* * * *
* * * *
***** *****
* * * * *
* * * * * *
***** *****
* * * * * *
* * * * * * *
***** ***** ***** *****
* * * * * * * *
***** ***** ***** *****

(문제를 해결하는데 필요한 개념)

재귀

(소스 코드)

CPP

```
#include <vector>
#include <iostream>
using namespace std;
static bool paint[3072][8000];
static int N;
void draw(int x,int y){
    paint[y][x]=true;
    paint[y+1][x+1]=paint[y+1][x-1]=true;
    for(int i=0;i<5;i++) paint[y+2][x+i-2]=true;
}

void bigdraw(int x,int y,int n){
    if(n==3) {
        draw(x,y);
        return;
    }
    else if(n<3) return;
    bigdraw(x,y,n/2);
    bigdraw(x+n/2,y+n/2,n/2);
    bigdraw(x-n/2,y+n/2,n/2);
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> N;
    bigdraw(N-1,0,N);
    for(int i=0;i<N;i++){
        for(int j=0;j<N*2-1;j++){
            if(paint[i][j]) cout << '*';
            else if(paint[i][j]==false) cout << ' ';
        }
        cout << "\n";
    }
    return 0;
}
```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

배열 크기가 너무 커서 메모리 초과가 날까봐 bool형 2차원 배열로 생성했다.

처음엔 작은 삼각형에서 큰 삼각형으로 가는 방식(바텀업?)을 생각했다.

가장 기본이 되는 삼각형을 생성하고 그 삼각형 3개로 다음 삼각형을 만든다 또 그렇게 만든 삼각형을 3개 이용해서 $k=i$ 인 삼각형으로 $k=i+1$ 인 삼각형을 만들어 $k=n/3$ 이 될 때 까지 만드는 방식으로 하려했지만 좌표 계산이 어렵고 자기자신을 복제하는 방식이 복잡해서 포기하고

큰 삼각형을 쪼개서 작은 삼각형으로 만드는 방식(탑다운?)을 채택했다.

가장 큰 삼각형을 쪼개서 가장 기본 삼각형으로만 이루어지게 계산을 하고 채우는 방식

$k=i$ 인 삼각형의 좌표를 계산해서 $k=i-1$ 의 삼각형 위치를 알아내는 방식으로 $k=0$ 이 되면 그 자리에 기본 삼각형을 채운다



스터디 6주차 보고서

(8/11 ~ 8/17)

작성자 : 원종호

작성일 : 2025.08.17

문제 풀이 (BOJ 15681 트리와 쿼리)

(문제 분석 및 프로그램 설계)

문제

간선에 가중치와 방향성이 없는 임의의 루트 있는 트리가 주어졌을 때, 아래의 쿼리에 답해보도록 하자.

- 정점 U 를 루트로 하는 서브트리에 속한 정점의 수를 출력한다.
- 만약 이 문제를 해결하는 데에 어려움이 있다면, 하단의 힌트에 첨부한 문서를 참고하자.

입력

트리의 정점의 수 N 과 루트의 번호 R , 쿼리의 수 Q 가 주어진다. ($2 \leq N \leq 105$, $1 \leq R \leq N$, $1 \leq Q \leq 105$)

이어 $N-1$ 줄에 걸쳐, UV 의 형태로 트리에 속한 간선의 정보가 주어진다. ($1 \leq U, V \leq N$, $U \neq V$)

이는 U 와 V 를 양 끝점으로 하는 간선이 트리에 속함을 의미한다.

이어 Q 줄에 걸쳐, 문제에 설명한 U 가 하나씩 주어진다. ($1 \leq U \leq N$)

입력으로 주어지는 트리는 항상 올바른 트리임이 보장된다.

출력

Q 줄에 걸쳐 각 쿼리의 답을 정수 하나로 출력한다.

(문제를 해결하는데 필요한 개념)

BFS, 우선순위큐

(소스 코드)

CPP

```

#include <vector>
#include <iostream>
#include <queue>
#include <algorithm>
using namespace std;

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int N,R,Q;
    cin >> N >> R >> Q;
    vector<int> parent(N+1,0);
    vector<int> depth(N+1,-1);
    parent[R]=-1; //root
    depth[R] = 0;
    vector<vector<int>> tree;
    tree.resize(N+1);
    vector<bool> visited(N+1,false);
    vector<int> leaf;
    for(int i=0;i<N-1;i++){
        int a,b;
        cin >> a >> b;
        tree[a].push_back(b);
        tree[b].push_back(a);
    }

    //bfs 를 진행해 parent 채워주기
    queue<int> q;
    q.push(R);
    visited[R]=true;

    while(!q.empty()){
        int now = q.front();
        q.pop();
        if(tree[now].size()==1&&tree[now][0]==parent[now]) leaf.push_back(now); //
        for(int next : tree[now]){
            if (depth[next] == -1) {
                depth[next] = depth[now] + 1;
                parent[next] = now;
                q.push(next);
            }
        }
    }
    priority_queue<pair<int,int>> q2;
    vector<int> subtree(N+1,1);
    fill(visited.begin(),visited.end(),false);
    for(int i : leaf){
        q2.push({depth[i],i});
    }
    while(!q2.empty()){
        pair<int,int> now = q2.top();
        q2.pop();

        if(parent[now.second]!=-1){
            subtree[parent[now.second]]+=subtree[now.second];
            if(!visited[parent[now.second]]) q2.push({depth[parent[now.second]],parent[now.second]});
            visited[parent[now.second]]=true;
        }
    }
    for(int i=0;i<Q;i++){
        int p;
        cin >> p;
        cout << subtree[p] << "\n";
    }
    return 0;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

처음 떠올린 아이디어는 세그먼트 트리에서 착안해, 각 subtree 배열을 만드는 것이다
subtree의 인덱스의 값은 해당 인덱스의 서브트리 값이며

어떤 p의 subtree값은 그 노드의 자식의 subtree를 합한거에 1을 더한 값이라는 식을
통해 구할 수 있었다.

이를 위해 BFS를 실행하여 각 노드의 깊이와 부모 노드의 번호를 저장하였고
한 노드의 subtree값을 계산하기 위해선 그 노드의 자식 노드의 subtree값이 먼저 계
산되어야하기 때문에

깊이값을 기준으로 우선순위큐에 넣어 깊이가 작은 노드의 subtree값을 우선 계산하도
록 하였다

이 방식의 장점은 최초 한번 배열을 계산하면 그 다음은 O(1)의 시간복잡도로 쿼리를
수행할 수 있어 쿼리가 많아 질때 이점이 있다.

문제 풀이 (BOJ 17404 RGB거리 2)

(문제 분석 및 프로그램 설계)

문제

RGB 거리에는 집이 N 개 있다. 거리는 선분으로 나타낼 수 있고, 1 번 집부터 N 번
집이 순서대로 있다.

집은 빨강, 초록, 파랑 중 하나의 색으로 칠해야 한다. 각각의 집을 빨강, 초록,
파랑으로 칠하는 비용이 주어졌을 때, 아래 규칙을 만족하면서 모든 집을 칠하는
비용의 최솟값을 구해보자.

- 1 번 집의 색은 2 번, N 번 집의 색과 같지 않아야 한다.
- N 번 집의 색은 N-1 번, 1 번 집의 색과 같지 않아야 한다.
- $i(2 \leq i \leq N-1)$ 번 집의 색은 $i-1, i+1$ 번 집의 색과 같지 않아야 한다.

입력

첫째 줄에 집의 수 $N(2 \leq N \leq 1,000)$ 이 주어진다. 둘째 줄부터 N 개의 줄에는 각
집을 빨강, 초록, 파랑으로 칠하는 비용이 1 번 집부터 한 줄에 하나씩 주어진다.
집을 칠하는 비용은 1,000 보다 작거나 같은 자연수이다.

출력

첫째 줄에 모든 집을 칠하는 비용의 최솟값을 출력한다.

(문제를 해결하는데 필요한 개념)

DP

(소스 코드)

CPP

```

#include <vector>
#include <iostream>
using namespace std;
#define MAX 100000000
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int N;
    cin >> N;
    int A[1001][3];
    for(int i=1;i<=N;i++){
        for(int j=0;j<3;j++){
            cin >> A[i][j];
        }
    }
    int D[1001][3];
    int result = MAX+1000000;
    for(int p=0;p<3;p++){
        //p: 처음에 고를 수
        for(int i=0;i<3;i++){
            D[1][i] = (i==p)?A[1][i]:MAX;
        }
        for(int i=2;i<=N;i++){
            for(int j=0;j<3;j++){
                D[i][j]=min(D[i-1][(j+1)%3],D[i-1][(j+2)%3])+A[i][j];
            }
        }
        for(int i=0;i<3;i++){
            if(i==p) continue;
            result = min(result,D[N][i]);
        }
    }
    cout << result;
    return 0;
}

```

(문제를 풀기 위해 생각하고 해결했던 과정 및 느낀점)

언뜻보면 RGB거리 1과 비슷해 보이지만 1번집과 N번집의 조건이 다르다

RGB거리 1은 1번집은 2번집하고만, N번집은 N-1번집과만 다르면 되었지만

2에선 1:N,2 N:N-1,1 끼리 달라야 한다

1번집에서 무엇을 선택했느냐에 따라 마지막 N번집의 결과에 제한이 올수도 있다는것

그래서 1번집을 무엇을 선택했는지 3가지 경우를 나눠서 각각 DP를 실행하였다

일주일두백준

1주차

작성자 : 조민정

제출일 : 2025.05.07

1. 2742번 문제

: 자연수 N이 주어졌을 때, N부터 1까지 한 줄에 하나씩 출력하는 프로그램을 작성하시오.

1) 코드

```
#include <stdio.h>

int main(void) {
    int N;
    scanf("%d", &N);
    for (int i = N; i >= 1; i--) {
        printf("%d\n", i);
    }

    return 0;
}
```

- #include <stdio.h>: 표준 입출력을 위한 헤더
- int N; → 사용자로부터 자연수를 입력받을 변수
- scanf("%d", &N); → 입력받아서 N에 저장
- for (int i = N; i >= 1; i--) → for 반복문을 통해 N회 반복
- printf("%d\n", i); → 하나씩 줄바꿈하여 출력
- return 0; → 프로그램 정상 종료

2. 10807번 문제

: 총 N개의 정수가 주어졌을 때, 정수 v가 몇 개인지 구하는 프로그램을 작성하시오.

1) 코드

```
#include <stdio.h>
```

```

int main(void) {
    int N, v;
    int numbers[100];
    int count = 0;
    scanf("%d", &N);

    for (int i = 0; i < N; i++) {
        scanf("%d", &numbers[i]);
    }

    scanf("%d", &v);

    for (int i = 0; i < N; i++) {
        if (numbers[i] == v) {
            count++;
        }
    }

    printf("%d\n", count);
    return 0;
}

```

- 정수 개수 N과 N개의 정수를 입력 받아 배열에 저장한다.
- 찾고자 하는 정수 v를 입력 받는다.
- 배열을 순회하며 각 값이 v와 같은지 확인한다.
- 같을 경우 count 값을 증가시킨다.
- 최종적으로 count를 출력한다.



스터디 5주차 보고서

(8/12 ~ 8/18)

작성자 : 임주원

작성일 : 2025.07.13

문제 풀이 (백준 2798번)

주어진 수들이 팰린드롬수인지 아닌지 판별하는 문제였다.

10으로 나눈 나머지를 이용해 뒤집힌 수를 임시로 만들어 비교하는 알고리즘으로 문제를 풀었다.

원래 수 길이의 중간까지만 반복하면 팰린드롬수인지 아닌지 판별할 수 있지만 귀찮아서 그냥 끝까지 하는 것으로 했다.

```
1 #include <stdio.h>
2 int p(int n){
3     int tmp =n;
4     int ev =0;
5     while(1){
6         if(tmp ==0){
7             break;
8         }
9         ev = ev*10;
10        ev = ev + tmp%10;
11        tmp = tmp/10;
12    }
13    if(ev==n){
14        return 1;
15    }
16    else{
17        return 0;
18    }
19 }
20 int main(){
21     int n;
22     while(1){
23         scanf("%d",&n);
24
25         if(n==0){
26             return 0;
27         }
28
29         if(p(n)){
30             printf("yes\n");
31         }
32         else{
33             printf("no\n");
34         }
35     }
36 }
```

백준 - 2869번 달팽이는 올라가고 싶다

쉬워보여서 했다... 그리고 문제 이름이 좀 귀엽다.

수학적으로 문제가 없다고 생각하고 코드를 짰는데 오류가 계속 나서 살펴보니 값 조정 코드 뒤에 값 선언 코드가 있었다. 이런 자잘한 오류들을 줄이려면 순서 상관 없이 작동되게 모든 값 선언을 값 조정으로 바꾸는 습관을 들여야겠다. 값 선언은 다소 위험한 코드일수도...(아님말고)

```
Back > C:\005.c > main()
● 1 #include<stdio.h>
  2 int main(){
  3     int a,b,z;
  4     int n = 0;
  5     scanf("%d %d %d",&a,&b,&z);
  6     if((z-a)<=0){
  7         printf("%d",1);
  8         return 0;
  9     }
 10    z = z-a;
 11    a = a-b;
 12    n = z/a +1;
 13    if((z%a) != 0){
 14        n++;
 15    }
 16    printf("%d",n);
 17    return 0;
 18
 19 }
```

백준 스터디 2주차

2025112418 전정환

10926번 제출 맞힌 사람 솔코딩 재체점 결과 채점 현황 내 제출 난이도 기여 ☰ 질문 게시판

??! 성공

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	336858	168247	143688	50.360%

문제

준하는 사이트에 회원가입을 하다가 joonas라는 아이디가 이미 존재하는 것을 보고 놀랐다. 준하는 놀람을 ??!로 표현한다. 준하가 가입하려고 하는 사이트에 이미 존재하는 아이디가 주어졌을 때, 놀람을 표현하는 프로그램을 작성하시오.

입력

첫째 줄에 준하가 가입하려고 하는 사이트에 이미 존재하는 아이디가 주어진다. 아이디는 알파벳 소문자로만 이루어져 있으며, 길이는 50자를 넘지 않는다.

출력

첫째 줄에 준하의 놀람을 출력한다. 놀람은 아이디 뒤에 ??!를 붙여서 나타낸다.

예제 입력 1 복사

joonas

예제 출력 1 복사

joonas??!

예제 입력 2 복사

baekjoon

예제 출력 2 복사

baekjoon??!

10926번 제출 맞힌 사람 솔코딩 재체점 결과 채점 현황 내 제출 난이도 기여 ☰ 질문 게시판

??!

언어 C++17 언어 설정

소스 코드 공개 공개 비공개 맛있을 때만 공개

소스 코드

```
1 #include<stdio.h>
2 int main()
3 {
4     char a;
5     scanf("%s", a);
6     printf("%s??!", a);
7
8     return 0;
9 }
```

제출

위 사진이 내가 처음 작성한 코드이다. 하지만 계속되는 오류에 문자열 선언이 틀렸나, 문자열을 입력 받고 출력하는 부분이 이게 아닌가 계속 바꿔보고 했지만, 좀처럼 잘 해결되지 않았다. 원인은 바로 char a; 가 아니라 char a[51]; 이라고 써야하는 것이었다. 이를 질문 게시판에서 찾아보면서 알게 되었는데, 왜그런지 알기 위해 구글링을 해보았다.

여기서 [51]은 배열의 크기를 지정해주는 부분이다. 문자열은 문자 배열로 저장되며, 마지막 문자에는 반드시 null character('0')가 포함되어야 문자열의 끝을 알 수 있다. 이러한 배열의 크기를 지정하지 않으면, 컴파일러는 a 배열에 저장할 공간을 할당할 수 없기 때문에 오류가 발생하는 것이었다. 숫자가 51인 이유는 배열의 크기를 충분히 크게 설정해줘야 하기 때문이다. 51이면 50글자까지 입력 받을 수 있고, 나머지 한자리는 '\0'을 저장하는 것이다.

10926번 제출 맞힌 사람 솔코딩 재체점 결과 채점 현황 내 제출 난이도 기여 ☰ 질문 게시판

??!

언어 C++17 언어 설정

소스 코드 공개 공개 비공개 맛있을 때만 공개

```
1 #include<stdio.h>
2 int main()
3 {
4     char a[51];
5     scanf("%s", a);
6     printf("%s??!", a);
7
8     return 0;
9 }
10
```

제출

이를 통해 오늘 알게된 것은

1. 문자열을 선언할 때에는, char이라고 선언하되, 변수 이름 뒤에 문자열의 크기를 적어줘야한다.
2. scanf() 함수로 문자열을 받아야할때, 변수 지정해줄때 다른 문자형과는 다르게, &를 쓰지 않는다.
3. 문자 하나를 받을때는 %c, 문자열을 받을때는 %s로.

백준 스터디 3주차 보고서

39기 전정환

18108번

제출 및힌 사람 솟코딩 재체점 결과 체점 현황 내 제출 난이도 기여 질문 게시판

1998년생인 내가 태국에서는 2541년생?! 성공



시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초 (추가 시간 없음)	1024 MB	242238	153373	135956	63.405%

문제

ICPC Bangkok Regional에 참가하기 위해 수완나품 국제공항에 막 도착한 팀 레드시프트 일행은 눈을 믿을 수 없었다. 공항의 대형 스크린에 올해가 2562년이라고 적혀 있던 것이다.

불교 국가인 태국은 불멸기원(佛滅紀元), 즉 석가모니가 열반한 해를 기준으로 연도를 세는 불기를 사용한다. 반면, 우리나라는 서기 연도를 사용하고 있다. 불기 연도가 주어질 때 이를 서기 연도로 바꿔 주는 프로그램을 작성하시오.

입력

서기 연도를 알아보고 싶은 불기 연도 y 가 주어진다. ($1000 \leq y \leq 3000$)

출력

불기 연도를 서기 연도로 변환한 결과를 출력한다.

예제 입력 1 복사

2541

예제 출력 1 복사

1998

18108번

제출

맞힌 사람

솟코딩

재체점 결과

체점 현황

내 제출

난이도 기여

질문 게시판

1998년생인 내가 태국에서는 2541년생?!

언어

C++17

언어 설정

소스 코드 공개

공개

비공개

맞았을 때만 공개

소스 코드

```
1 #include<stdio.h>
2 int main()
3 {
4     int year;
5     scanf("%d", &year);
6     printf("%d", year - 543);
7
8     return 0;
9 }
```

제출

이번 문제는 전 주차 문제보다 훨씬 할만 했다. 문제 예시를 통해 태국 연도와 우리나라 연도 차이가 543년이라는 것을 알고, 알고싶은 연도를 scanf를 통해 입력 받은 뒤, 그에 543년 전 연도를 printf를 통해 출력하면 되었다.

백준 스터디 4주차 보고서

2025112418 전정환

10818번 제출 맞힌 사람 솟코딩 재채점 결과 채점 현황 내 제출 난이도 기여 ☰ 질문 게시판

최소, 최대 성공 ☆

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	446180	204022	153150	44.500%

문제

N개의 정수가 주어진다. 이때, 최솟값과 최댓값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 정수의 개수 N ($1 \leq N \leq 1,000,000$)이 주어진다. 둘째 줄에는 N개의 정수를 공백으로 구분해서 주어진다. 모든 정수는 -1,000,000보다 크거나 같고, 1,000,000보다 작거나 같은 정수이다.

출력

첫째 줄에 주어진 정수 N개의 최솟값과 최댓값을 공백으로 구분해 출력한다.

오늘은 10818번 배열 문제 중 최소, 최대를 찾는 문제를 풀이했습니다.

```
#include<stdio.h>
int main()
{
    int num;
    scanf("%d", &num);
    int list[num];
    for(int i=0; i<num; i++){
        scanf("%d", &list[i]);
    }
    int max, min;
    max = list[0];
    min = list[0];
    for(int j=1; j<num; j++){
        if(max<list[j])
            max = list[j];
        if(min>list[j])
            min = list[j];
    }
    printf("%d %d \n", min, max);
    return 0;
}
```

위 코드가 위 문제의 답이었습니다. 마침 기프에서 현재 배열을 배우고 있었기 때문에, 오류없이 한번에 풀이할 수 있었습니다. 코드를 풀이하자면, 먼저 리스트의 크기를 사용자로부터 입력받고, 그 크기만큼의 배열을 입력받아 만듭니다. 그 후에, 반복문을 통해 최댓값과 최솟값을 구하여 출력하는 방식입니다.

이 문제를 통해서 배열을 만들때 반복문이 굉장히 많이 쓰이고, 반복문을 사용해야지 효율적으로 배열을 다룰 수 있다는 것을 알 수 있었습니다. 앞으로도 기프 수업에서도, 그리고 평소 코딩 공부를 할때에도 배열이란 개념이 굉장히 많이 써야할 것 같은데, 이를 위해 반복문을 잘 활용하는 능력을 길러서 배열을 쉽게 사용할 수 있게 되도록 노력해야겠습니다.