코딩테스트 필수 알고리즘 10개

1. BFS
2. DFS
3. 백트래킹
4. 시뮬레이션
5. 이진탐색
6. 그리디
7. DP
8. MST
9. 다익스트라
10. 플로이드

핵심문제 + 10번 풀고 시험보기

푸는 방법

● 풀기 전에 최대한 구체적인 계획을 세우기
● 다음 세가지 주석으로 써보고 문제 풀기

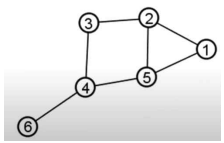
1. 아이디어 : 문제를 어떻게 풀것인지 여기서 대부분 설계하고 진행
2. 시간복잡도 : 내가 설계한 방법이 오래걸리는지 확인
3. 자료구조 : 내가 자료구조를 어떻게 사용할지 미리 계획
 숫자의 경우 최대자리에 따라 데이터 타입 예상

문제 페이지

- 백준 / 프로그래머스

요즘 코테 추세 : IDE에서 작성 후 복붙이기 불가
Debugger 사용불가하므로, Console 출력으로 연습

==========



BFS(Breadth-first Search) - 1 2 5 3 4 6

그래프 탐색 : 어떤것들이 연속해서 이어질 때, 모두 확인하는 방법

-〉 Graph : Vertex(어떤 것) + Edge(이어지는 것)

그래프 탐색 종류

-〉 BFS(Breadth-first search) : 너비우선탐색 = 자식을 먼저 탐색 = Queue를 사용 / 형제노드 우선

-〉 DFS(Depth-first search) : 깊이우선탐색 = 자식의 자식을 먼저 탐색 = Stack을 사용 / 자식노드 우선

아이디어

- 시작점에 연결된 Vertex 찾기
- 찾은 Vertex를 큐에 저장
- Queue의 가장 먼저 것 뽑아서 반복

시간복잡도

- 시간복잡도 : 알고리즘이 얼마나 오래 걸리는지
- BFS : O(V + E)

자료구조

- 검색할 그래프
- 방문여부 확인(재방문 금지)
- Queue : BFS 실행

기본문제 : 백준 1926 -〉 1이 연속(BFS)된 개수와 최댓값

2중포문 -〉 값이 1, 방문 X =〉 BFS

입력 https://www.acmicpc.net/problem/1926

```
6 5
1 1 0 1 1
0 1 1 0 0
0 0 0 0 0
1 0 1 1 1
0 0 1 1 1
0 0 1 1 1
```

코드

"""

1. 아이디어

- 2중 for =〉 값 1 && 방문 X =〉 BFS
- BFS 돌면서 그림 개수 +1, 최대값을 갱신

2. 시간복잡도

- BFS : O(V+E) = O(V + 4V = 5V = 5m * 5n = 5(500 * 500))
- V : m * n
- E : v * 4
- V + E : 5 * 250000 = 100만 〈 2억 〉〉 가능

3. 자료구조

- 그래프 전체 지도 : int[][]
- 방문 : bool[][]
- Queue(BFS)

"""

```
from collections import deque

import sys
input = sys.stdin.readline

n, m = map(int, input().split())
map = [list(map(int, input().split())) for _ in range(n)]
chk = [[False] * m for _ in range(n)]

dy = [0, 1, 0, -1] # 우, 하, 좌, 위 순
dx = [1, 0, -1, 0]

# BFS 함수
def bfs(y, x):
    # 그림의 사이즈 구하기
    rs = 1 # result

    q = deque()
    # Queue에 넣기
    q.append((y, x))
    # Queue에 새로 들어가지 않을 때 까지 반복
    while q:
        # 각각의 Queue를 새로운 변수에 추가
        ey, ex = q.popleft()
        # 4방향(동, 서, 남, 북)으로 이동하면서 새로운 1이 존재하는지 확인
        for k in range(4):
            # 하나씩 추가하면서 확인
            ny = ey + dy[k]
            nx = ex + dx[k]
            # 그림의 사이즈 이상 넘어가는 것 방지
            if 0<=ny<n and 0<=nx<m:
                if map[ny][nx] == 1 and chk[ny][nx] == False:
                    rs += 1
                    chk[ny][nx] = True
                    q.append((ny, nx))
    return rs

cnt = 0 # 1이 연결되어 있는 수
maxv = 0 # 1로 연결된 것의 최대값

# 2중 for - y 후 x
for j in range(n):
    for i in range(m):
        if map[j][i] == 1 and chk[j][i] == False:
            # 방문을 True로 설정
            chk[j][i] = True
            # 전체 그림 갯수를 +1
            cnt += 1
            # BFS 〉 그림 크기를 구해주고
            # 최댓값 갱신
            maxv = max(maxv, bfs(j, i))

print(cnt) # Count 출력
print(maxv) # MaxValue 출력
```

C++

```
#include <iostream>
#include <algorithm>
#include <queue>
#include <vector>
#define MAX 500

using namespace std;

int n, m;
int map[MAX][MAX] = { 0, };
bool chk[MAX][MAX] = { 0, };

int dy[] = { 0, 1, 0, -1 };
int dx[] = { 1, 0, -1, 0 };
queue<pair<int, int>> q; // BFS 사용 큐
vector<int> v; // 그림 개수 저장 벡터
int s = 1;  // 그림 넓이
```

```
void BFS(int y, int x) {
    chk[y][x] = true;
    q.push(make_pair(y, x));

    while (!q.empty()) {
        y = q.front().first;
        x = q.front().second;
        q.pop();

        for (int i = 0; i < 4; i++)
        {
            int ny = y + dy[i];
            int nx = x + dx[i];
            if (ny < 0 || nx < 0 || ny >= n || nx >= m)
                continue;
            if (map[ny][nx] == 1 && chk[ny][nx] == 0) {
                chk[ny][nx] = true;
                s++;
                q.push(make_pair(ny, nx));
            }
        }
    }
}

bool Compare(int i, int j) {
    return i > j;
}

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

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

    int cnt = 0; // 영역 개수
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if (map[i][j] == 1 && chk[i][j] == 0) {
                BFS(i, j);
                v.push_back(s);
                cnt++;
                s = 1;
            }
        }
    }

    sort(v.begin(), v.end(), Compare); // 벡터 오름차순 정렬

    cout << cnt << endl;

    if (cnt == 0)
        cout << 0 << endl;
    else
        cout << v[0] << endl;
}
```

==========

DFS (Depth-first search) 깊이 우선 탐색 - 1 2 3 4 6 5

스택, 재귀함수랑 자료구조 동일

- 그래프 내부 모든 확인은 BFS로도 가능하나 DFS는 재귀함수를 사용하기 위해 하는 것이다 -> 백트래킹에서 가장 중요

재귀함수

- 자기 자신을 다시 호출하는 함수
- 주의할 점

▶ 재귀함수가 종료되는 시점 반드시 명시

▶ 재귀함수의 깊이가 너무 깊어지면 Stack Overflow

- DFS, 백트래킹에서 주로 사용

아이디어

- 시작점에 연결된 Vertex 찾기
- 연결된 Vertex를 계속해서 찾음(끝날 때 까지)
- 더 이상 연결된 Vertex 없을 경우 처음으로 돌아가 다시처음부터(중복x)

시간복잡도

- DFS : O(V+E)

자료구조

- 검색할 그래프 : 2차원 배열
- 방문여부 확인 : 2차원 배열(재방문 금지)

기본문제 : 백준 2667 그림

https://www.acmicpc.net/problem/2667

입력

7
0110100
0110101
1110101
0000111
0100000
0111110
0111000

1. 2중 for / 1 && 방문 x일 때 재귀(DFS)
2. 각 방향을 확인 후 DFS 호출 / 값을 리스트에 저장
3. 정렬 후 출력

코드

```
"""
1. 아이디어
- 2중 for, 값 1 && 방문X ==> DFS
- DFS를 통해 찾은 값을 저장 후 정렬해서 출력

2. 시간복잡도
- DFS : O(V + E)
- V, E : N^2, 4N^2
- V + E : 5N^2 ~= N^2 ~= 625 < 2억 >> 가능

3. 자료구조
- 그래프 저장 : int[][]
- 방문여부 : bool[][]
- 결과값 : int[]
"""
import sys
input = sys.stdin.readline

N = int(input())
map = [list(map(int, input().strip())) for _ in range(N)]
chk = [[False] * N for _ in range(N)]

result = []
each = 0 # 각각의 연산

dy = [0, 1, 0, -1] # 우, 하, 좌, 상
dx = [1, 0, -1, 0] # 시계 반대방향

def dfs(y, x):
    # 연속된 노드의 크기를 each를 사용
    global each
    each += 1
    # 각 노드에서 4방향을 전부 탐색
    for k in range(4):
        ny = y + dy[k]
        nx = x + dx[k]
        if 0<=ny<N and 0<=nx<N:
            if map[ny][nx] == 1 and chk[ny][nx] == False:
                chk[ny][nx] = True
                # 재귀호출 ->4방향확인후 있다면 있는 노드에서 다시 4방향 확인
                # 이미 방문한 경우 확인 X
                dfs(ny, nx)

for j in range(N):
    for i in range(N):
        # 1인 동시에 방문 x
        if map[j][i] == 1 and chk[j][i] == False:
            # 방문 체크 표시
            chk[j][i] = True
            # 새로운 노드를 탐색할 때마다 크기를 0으로 초기화
            each = 0
            # DFS 로 크기 구하기
            # BFS : 함수 호출, 리턴값으로 크기
            dfs(j,i)
            # 크기를 결과 리스트에 넣기
            result.append(each)
```

```
result.sort()
print(len(result))
for i in result:
    print(i)
```

C++

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

#define MAX 26
using namespace std;

int n, cnt = 0;

int map[MAX][MAX];
int chk[MAX][MAX];
vector<int> cntVec;

int dy[4] = { -1, 0, 1, 0 };
int dx[4] = { 0, 1, 0, -1 };

void DFS(int y, int x) {
    for (int i = 0; i < 4; i++)
    {
        int ny = y + dy[i];
        int nx = x + dx[i];

        if (ny >= n || ny < 0 || nx >= n || nx < 0) continue;

        if (chk[ny][nx] == 0 && map[ny][nx] == 1) { // 방문 안했고 집에 있으면
            chk[ny][nx] = 1; // 방문했다고 표시하고
            cnt += 1; // 집 개수 세기
            DFS(ny, nx);
        }
    }
}

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

    int res = 0;

    cin >> n;
    string str;

    for (int i = 0; i < n; i++) {
        cin >> str;
        for (int j = 0; j < str.length(); j++) { // 입력주의
            chk[i][j] = 0;

            if (str[j] == '1')
                map[i][j] = 1;
            else
                map[i][j] = 0;
        }
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (map[i][j] == 1 && chk[i][j] == 0)
            {
                chk[i][j] = 1;
                cnt = 1; // 처음은 시작점 포함하므로 1로 초기화
                DFS(i, j);
                cntVec.push_back(cnt);
                res++; // 단지 그룹 1개 탐색 끝남
            }
        }
    }

    sort(cntVec.begin(), cntVec.end());
    cout << res << "\n";

    for (int i = 0; i < cntVec.size(); i++)
        cout << cntVec[i] << '\n';
}
```

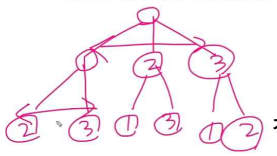
==========

백트래킹

개념

- 모든 경우의 수를 확인해야 할 때

▶ for로는 확인불가한 경우(깊이가 달라질 때)
▶ for는 깊이가 정해져 있음
▶ 범위가 정해지지 않을 때 사용
▶ 트리방식으로 사용



각각 for문으로 작동 -> 재귀함수를 사용

ex) 피보나치

기본문제

- 백준 15649 N과 M

https://www.acmicpc.net/problem/15649

입력 3 1

- 1 ~ N까지 1개 선택과 방문여부 확인-> 결과값에 추가와 방문여부체크 -> 재귀

아이디어

- 1부터 N중에 하나를 선택한 뒤
- 다음 1부터 N까지 선택할 때 이미 선택한 값이 아닌 경우 선택
- M개를 선택할 경우 프린트 -> 재귀함수를 구현할 때 끝에 종료하는 조건 추가 -> 출력

시간복잡도

- N^N : 중복이 가능, N = 8까지 가능 -> 2억이 넘지 않아야 된다
- N! : 중복이 불가, N = 10까지 가능 -> 2억이 넘지 않아야 된다

자료구조

- 방문 여부 확인 배열 : bool[]
- 선택한 값 입력 배열 : int[]

코드

"""

1. 아이디어
- 백트래킹 재귀함수 안에서, for 돌면서 숫자 선택(이때 방문여부 확인)
- 재귀함수에서 M개를 선택할 경우 print

2. 시간복잡도
- N! > 10까지 가능 문제 = 8 까지 >> 가능

3. 자료구조
- 결과값 저장 int[]
- 방문여부 체크 bool[]

"""

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

N, M = map(int, input().split())
rs = []
chk = [False] * (N+1) # 결과값 출력시 N - 1을 해주는 것 방지 => 1로 바로 인덱스 접근이 가능하게 하기 위해

def recur(num):
    if num == M:
        # 결과값 print시 string으로 변경 후 출력
        print(' '.join(map(str, rs)))
        return
    for i in range(1, N+1):
        if chk[i] == False:
            chk[i] = True
            rs.append(i)
            recur(num + 1)
            chk[i] = False
            rs.pop()

# 모든 경우를 확인하기 때문에 동작 X
# 트리구조로 생각했을 때 마지막을 빼주는 동작이 필요
    # 빼는 동시에 방문여부도 다시 돌려주어야 함
recur(0)
```

Tip

- 백트래킹 문제는 N이 작음 -> 10근처라면 백트래킹
- 재귀함수 사용할 때, 종료 시점 잊지 말기

C++ // 시간초과 문제 발생

```
#include <iostream>
#define MAX 9

using namespace std;

int n, m;
int rs[MAX] = { 0, };
bool chk[MAX] = { false, };
```

```
void recur(int num) {
    if (num == m) {
        for (int i = 0; i < m; i++)
        {
            cout << rs[i] << ' ';
        }
        cout << '\n';
        return;
    }
    for (int i = 1; i < n + 1; i++)
    {
        if (chk[i] == false) {
            chk[i] = true;
            rs[num] = i;
            recur(num + 1);
            chk[i] = false;
        }
    }
}

int main()
{
    cin >> n >> m;

    recur(0);
}
```

==========

시뮬레이션 - 꼭 알아두어야 하는 문제

개념

- 각 조건에 맞는 상황을 구현하는 문제
- ▶ 지도상에서 이동하면서 탐험하는 문제
- ▶ 배열안에서 이동하면서 탐험하는 문제
- 별도의 알고리즘 없이 풀수 있으나, 구현력 중요
- 매 시험마다 1문제 이상무조건 출제

기본문제

- 백준 14503 로봇청소기
- https://www.acmicpc.net/problem/14503
- 입력

3 3
1 1 0
1 1 1
1 0 1
1 1 1

아이디어

- 특정 조건 만족하는 한 계속 이동 > While
- 4방향 탐색 먼저 수행 > 빈칸 있을 경우 이동
- 4방향 탐색 안될 경우, 뒤로 한칸 가서 반복

시간복잡도

- While문 최대 : N X M (세로 X 가로)
- 각 칸에서 4방향 연산 수행

자료구조

- 전체 지도 : int[][]
- 내 위치, 방향 : int, int, int

코드

```
"""
1. 아이디어
- while 문으로 특정 조건 종료될 때까지 반복
- 4방향을 for문으로 탐색
- 더이상 탐색이 불가능할 경우, 뒤로 한칸 후진
- 후진이 불가능하면 종료

2. 시간복잡도
- O(NM) : 50^2 = 2500 < 2억 >> 가능

3. 자료구조
- map : int[][]
- 로봇청소기 위치, 방향, 전체 정소한 곳 수

"""

import sys
input = sys.stdin.readline

N, M = map(int, input().split()) # 가로 세로 크기
y, x, d = map(int, input().split()) # y, x, 방향
map = [list(map(int, input().split())) for _ in range(N)]
cnt = 0

dy = [-1, 0, 1, 0]
dx = [0, 1, 0, -1]

while 1:
    # 청소가 안된 경우만 청소하도록
    if map[y][x] == 0:
    # 현재 위치 청소
        map[y][x] = 2
        cnt += 1
    sw = False
    # 4방향 확인
    for i in range(1, 5):
        # 내가 바라보는 방향
        ny = y + dy[d - i]
        nx = x + dx[d - i]
        if 0 <= ny < N and 0 <= nx < M:
            if map[ny][nx] == 0:
                # 그 방향으로 회전한 다음 한 칸을 전진하고 1번부터 진행한다.

                # dy, dx 범위 초과로 인한 수정 -> 1 ~ 4까지의 숫자로 조정
                d = (d - i + 4 ) % 4
                y = ny
                x = nx
                sw = True
                break


    # 4방향 모두 있지 않은 경우
    if sw == False:
        # 바라보는 방향을 유지한 채로 한 칸 후진을 하고 2번으로 돌아간다.
        ny = y - dy[d]
        nx = x - dx[d]
        if 0 <= ny < N and 0 <= nx < M:
            if map[ny][nx] == 1:
                break
            else:
                y = ny
                x = nx
        else:
            break
print(cnt)
```

Tip

- 주어진 조건을 되도록 그대로 구현(나중에 매우 헷갈림)
- 되도록 쉽게 구현
- ▶ 최악 케이스 : 코드 복잡해 디버깅 안됨, 시간 대부분 소모
- Console에 Log 찍는 것 연습

C++

```
#include <iostream>

using namespace std;

#define MAX 50
int N, M;
int visited_count;
int map[MAX][MAX]; // 지도
int visited[MAX][MAX] = { 0, }; // 청소기 경로, 방문했으면 1

// 북, 동, 남, 서
int dx[4] = { -1, 0, 1, 0 };
int dy[4] = { 0, 1, 0, -1 };
int r, c, d;

void Input() {
	cin >> N >> M;
	cin >> r >> c >> d;
	for (int i = 0; i < N; i++)
		for (int j = 0; j < M; j++)
			cin >> map[i][j];

	visited[r][c] = 1;
	visited_count++;
}

void DFS() {
	// 네 방향 청소, 계속 왼쪽으로 회전
	for (int i = 0; i < 4; i++)
	{
		int next_d = (d + 3 - i) % 4; // next direction(왼쪽)
		int next_r = r + dx[next_d];
		int next_c = c + dy[next_d];

		// B. 왼쪽 방향에 청소할 공간이 없다면, 그 방향으로 회전하고 2번으로 돌아간다
		if (next_r < 0 || next_r >= N || next_c < 0 || next_c >= M ||
map[next_r][next_c] == 1)
			continue;

		// A. 왼쪽 방향에 아직 청소하지 않은 공간이 존재한다면, 그 방향으로 회전한 다음
한칸을 전진하고 1번부터 진행
		if (map[next_r][next_c] == 0 && visited[next_r][next_c] == 0)
		{
			visited[next_r][next_c] = 1;
```

```
                r = next_r;
                c = next_c;
                d = next_d;
                visited_count++;
                DFS();
            }
        }

        int back_idx = d > 1 ? d - 2 : d + 2;
        int back_r = r + dx[back_idx];
        int back_c = c + dy[back_idx];

        // C. 네 방향 모두 청소가 이미 되어있거나 벽인 경우에는,
        if (back_r >= 0 && back_r <= N || back_c >= 0 || back_c <= M) {
            // 바라보는 방향을 유지한 채로 한 칸 후진을 하고 2번으로 돌아간다
            if (map[back_r][back_c] == 0) {
                r = back_r;
                c = back_c;
                DFS();
            }
            // D. 뒤쪽 방향이 벽이라 후진도 할 수 없는 경우에는 작동을 멈춘다.
            else
            {
                cout << visited_count << endl;
                exit(0);
            }
        }
}

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

        Input();
        DFS();
}
```

==========

투 포인터

개념
- 각 원소마다 모든 값을 순회해야 할 때, O(N^2)
- 연속하다는 특성을 이용해서 처리 O(N)
- 두 개의 포인터(커서)가 움직이면서 계산
- 처음부터 생각하기 어려움, 쉬운 방법부터 생각

기본문제
- 백준 2559 수열
- https://www.acmicpc.net/problem/2559
- 입력
  - 10 2
  - 3 -2 -4 -9 0 3 7 13 8 -3

처음 아이디어
- for문으로 각 숫자의 위치에서 이후 K개의 수를 더함
- 이때마다 최대값으로 갱신

처음 시간 복잡도
- for문 : O(n)
- 각 위치에서 K개의 값 더함 : O(K)
- 총 : O(nK) -> 10만 * 10만 -> 2억이상 >> 불가능

아이디어
- 처음 K개의 값을 더함
- for문 : 다음 인덱스의 값을 더하고, 앞의 값을 뺌
- 이때 최대값을 갱신

시간복잡도
- 숫자 개수만큼 for => O(n) -> 원래는 O(2n)

자료구조 // 범위를 조심 :
- 전체 정수 배열 : int[]
  - 수 모두 -100 ~ 100 > int 가능
- 합한 수 : int
  - 100 * 1e5 = 1e7 > int 가능

코드

```
"""
1. 아이디어
- 투포인터 활용
- for문으로, 처음 K개 값을 저장
- 다음 인덱스 더해주고, 이전 인덱스 빼줌
- 이때마다 최대값을 갱신

2. 시간복잡도
- O(N) = 1e5 > 가능

3. 자료구조
- 각 숫자들 N개 저장 배열 : int[]
    - 숫자들 최대 100 > int 가능
- K개의 값을 저장 변수 : int
    - 최대 : K * 100 = 1e5 * 100 = 1e7 >> int 가능
- 최대값 : int

"""
import sys
input = sys.stdin.readline

N, K = map(int, input().split())
nums = list(map(int, input().split()))
# K개를 더해주는 변수
each = 0

# K 개를 더해주기
for i in range(K):
    each += nums[i]
maxv = each

# 다음 인덱스 더해주고, 이전 인덱스 빼주기
for i in range(K, N):
    each += nums[i]
    each -= nums[i - K]
    maxv = max(maxv, each)

print(maxv)
```

Tip
- 처음부터 생각하기 어려움, 쉬운방법부터 생각
  - O(N^2) 시간 복잡도가 초과한다면
  - 연속하다는 특징을 활용할 수 있는지 확인
- for 내부 투포인터 계산하는 값의 최댓값 확인 필수(Int 초과)
- 투포인터 문제 종류
  - 두 개 다 왼쪽에서 / 각각 왼쪽, 오른쪽 / 다른 배열
  - 일반 : O(N) / 정렬 후 투포인터 : O(NlgN)

==========

이진탐색

개념
- 어떤 값을 찾을 때 정렬의 특징을 이용해 빨리 찾음
- 정렬되어 있을 경우, 어떤 값 찾을 때 : O(N) -> O(lgN)
- 처음부터 생각하기 어려움, 쉬운 방법부터 생각
- 1~4 숫자 중 특정 숫자 찾아야할 때
  - 모두 탐색 : O(N)
  - 이진 탐색 : O(lgN)

핵심 코드 -> 암기 = 재귀함수 이용

```
def search(st, en, target): # 시작, 끝, 목표
    if st == en:
        # ~~ ex) print(st or en)
        return
    mid = (st + en) // 2 # 나머지를 버린다 -> 작은 값을 선택한다
    if nums[mid] < target:
        search(mid + 1, en, target)
    else:
        search(st, mid, target)
```

기본문제
- 백준 1920 수 찾기
- https://www.acmicpc.net/problem/1920
- 입력

```
5
4 1 5 2 3
5
1 3 7 9 5
```

처음 아이디어
- M개의 수마다 각각 어디 있는지 찾기
- for : M개의 수
- for : N개의 수 안에 있는지 확인

처음 시간 복잡도
- for : M개의 수 > O(N)
- for : N개의 수 안에 있는지 확인 > O(N)
- O(MN) = 1e10 > 시간초과

아이디어
- M개를 확인해야하는데, 연속하다는 특징 활용 가능 > 불가
- 정렬해서 이진탐색 가능
  - N개의 수 먼저 정렬
  - M개의 수 하나씩 이진탐색으로 확인

시간복잡도
- N개의 수 정렬 : O(N * lgN) => 정렬하는 경우 시간복잡도 동일

– M개의 수 이진탐색 : O(M * lgN)
– O((N + M)lgN) = 2e5 * 20 = 4e6 => 가능 [log2 N = 20]
자료구조
– 탐색 대상 수 : int[]
▶ 모든 수 범위 : –2^31 ~ 2^31 > int 가능
– 탐색 하려는 수 : int[]
▶ 모든 수 범위 : –2^31 ~ 2^31 > int 가능
코드
"""
1. 아이디어
– N개의 숫자를 정렬
– M개를 for 돌면서, 이진탐색
– 이진탐색 안에서 마지막에 데이터 찾으면, 1 출력, 아니면 0출력

2. 시간복잡도
– N개 입력값 정렬 = O(NlgN)
– M개를 N개 중에서 탐색 = O(M * lgN)
– 총합 : O((N + M) lgN) > 가능

3. 자료구조
– N개 숫자 : int[]
– M개 숫자 : int[]

"""

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

N = int(input())
nums = list(map(int, input().split()))
M = int(input())
target_list = list(map(int, input().split()))

nums.sort() # 정렬해야 이진탐색 가능

# 이진탐색 코드
def search(st, en, target):
    if st == en:
        # 같은 값인지 확인
        if nums[st] == target:
            print(1)
        else:
            print(0)
        return

    mid = (st + en) // 2
    if nums[mid] < target:
        search(mid + 1, en, target)
    else:
        search(st, mid, target)

# for문을 통한 탐색
for each_target in target_list:
    search(0, N-1, each_target)
```

Tip
– 처음부터 생각하기 어려움, 쉬운 방법부터 생각
▶ 어떤 값을 여러번 탐색해야 하는 경우
– 입력의 개수가 1e5인 경우라면 의심

C++

```
#include <iostream>
#include <algorithm>
#define MAX 100010

using namespace std;

int n, m;
int nums[MAX] = { };
int target[MAX] = { };

void Search_class(int st, int en, int target) {
	if (st == en) {
		if (nums[st] == target) {
			cout << 1 << endl;
		}
		else
		{
			cout << 0 << endl;
		}
		return;
	}

	int mid = (st + en) / 2;

	if (nums[mid] < target)
		Search_class(mid + 1, en, target);
	else
		Search_class(st, mid, target);
}

int main()
{
	ios_base::sync_with_stdio(0); cin.tie(0);
	cin >> n;
	for (int i = 0; i < n; i++)
		cin >> nums[i];
	sort(nums, nums + n);

	cin >> m;
	for (int i = 0; i < n; i++)
		cin >> target[i];

	for (int i = 0; i < n; i++)
	{
		Search_class(0, n - 1, target[i]);
	}
}
```

==========

그리디
개념
– 때로는 당장 눈앞의 최선이, 최고의 결과를 가져온다
▶ 현재 차례의 최고의 답을 찾는 문제
▶ 어려운 이유 : 왜 그런지 증명하기가 어려움 –> 이유를 찾는 연습 필요
▶ 예시 : 다른 금액의 동전이 여러 개 주어졌을 때 M원을 만드는 최소의 개수 –> 찾기 힘든 경우 반례를 찾는다
기본문제
– 백준 11047 동전 0
– https://www.acmicpc.net/problem/11047
– 입력
10 4200
1
5
10
50
100
500
1000
5000
10000
50000
아이디어
– 큰 금액의 동전부터 차감
– 반례? : 동전의 개수가 무한대라서 없애는 것으로 보임
– K를 동전 금액으로 나눈 뒤 남은 값으로 갱신
시간복잡도
– for : N > O(N)
자료구조
– 동전금액 : int[]
▶ 최대값 : 1e6 > Int 가능
– 현재 남은 금액 : int
▶ 최대값 : 1e6 > Int 가능
– 동전 개수 : int
▶ 최대값 : 1e6 > Int 가능
코드
"""
1. 아이디어
– 동전을 저장한 뒤, 반대로 뒤집음
– 동전 for >
  – 동전 사용개수 추가
  – 동전 사용한 만큼 K값 갱신
2. 시간 복잡도
– O(N)
3. 자료구조
– 동전 금액 : int[]
– 동전 사용 cnt : int
– 남은 금액 : int
"""

```
import sys
```

```
input = sys.stdin.readline

N, K = map(int, input().split())
coins = [int(input()) for _ in range(N)]
coins.reverse()
cnt = 0

for each_coin in coins:
    cnt += K // each_coin
    K  = K % each_coin

print(cnt)
```

Tip
- 실전 문제에서, 그리디로 푸는 문제임을 생각하기가 어려움
- 그리디 사용 이유 설명 or 반례 찾기 연습

C++
```
#include <iostream>

using namespace std;

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

	int n, k;
	int a[10];

	cin >> n >> k;
	for (int i = 0; i < n; i++)
		cin >> a[i];

	int count = 0;
	for (int i = n - 1; i >= 0; i--)
	{
		if (a[i] <= k) {
			count = count + k / a[i];
			k = k % a[i];
		}
		if (k == 0)
			break;
	}

	cout << count << endl;
}
```
==========

DP
개념
- DP : Dynamic Programming
- 이전의 값을 재활용하는 알고리즘 -> 점화식 $A_n = A_{n-1} + A_{n-2}$ 같은
- 처음부터 해보면서 패턴 파악하기
- 예시 : 1~10 숫자 중, 각각 이전값들을 합한 값 구하기
- 이전의 값을 활용해서 시간 복잡도 줄일 수 있음

기본문제
- 백준 11726 2xN 타일링
- https://www.acmicpc.net/problem/11726
- 입력
▶ 2

아이디어
- A1 : 1, A2 : 2, A3 : 1 + 2
- An = A(n-1) + A(n-2)
- for 문으로 3부터 N까지 돌면서
- 이전값과, 그 이전값을 더해서 저장(이때 10007로 나눈 나머지 값)

시간복잡도
- for : N 〉 O(N)

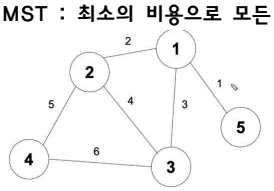
자료구조
- 방법의 수 배열(An) : int[]
▶ 최대값 : 10006 〉 int가능

코드
```
"""
1. 아이디어
- 점화식 : An = An-1 + An-2
- N값 구하기 위해, for문 3부터 N까지의 값을 구해주기
- 이전값, 이전이전값 더해서, 10007로 나눠 저장
2. 시간 복잡도
- O(N)
3. 자료구조
- DP값 저장하는 (경우의 수) 배열 : int[]
    - 최대값 : 10007보다 작음 〉 INT
"""
import sys

input = sys.stdin.readline

N = int(input())
rs = [0, 1, 2]

for i in range(3, N + 1):
    rs.append((rs[i - 1] + rs[i - 2] % 10007))

print(rs[N])
```

Tip
- 어떻게 할지 모르겠을 땐, 하나씩 그려보면서 규칙 찾기
- 점화식을 명확하게 세우고 코드 짜기

C++
```
#include <iostream>
#include <vector>

using namespace std;

void main() {
	ios_base::sync_with_stdio(0);
	cin.tie(0);

	int n;
	vector<int> rs = { 0, 1, 2 };

	cin >> n;

	for (int i = 3; i < n + 1; i++)
		rs.push_back((rs[i - 1] + rs[i - 2] % 10007));

	cout << rs[n];
}
```
==========

MST
개념
- MST : Minimum Spanning Tree
- Spanning Tree : 모든 노드가 연결된 트리
- MST : 최소의 비용으로 모든 노드가 연결된 트리



- MST 푸는 방법 : Kruskal or Prim
- Kruskal : 전체 간선 중 작은것부터 연결
- Prim : 현재 연결된 트리에 이어진 간선중 가장 작은 것을 추가
- heap
▶ 최대값, 최소값을 빠르게 계산하기 위한 자료구조
▶ 이진 트리 구조
▶ 처음에 저장할때부터 최대값 or 최소값 결정하도록

핵심코드
```
heap = [[0, 1]] // 비용, 노드번호

while heap:
    w, next_node = heapq.heappop(heap) // 힙 초기화
    if chk[next_node] == False: // 방문여부 확인
        chk[next_node] = True // 방문으로 변경
        rs += w // rs = 0
        for next_edge in edge[next_node]:
            if chk[next_edge[1]] == False: // 방문하지 않은 경우
                heapq.heappush(heap, next_edge) // 힙에 추가
```

기본문제
- 백준 1197 최소 스패닝 트리
- https://www.acmicpc.net/problem/1197
- 입력 // 정점의 수, 간선의 개수 // 연결 노드, 연결 노드, 가중치
3 3
1 2 1
2 3 2
1 3 3
-> 방향, 양방향, 무방향 그래프인지 확인

아이디어
- 최소 스패닝 트리 기본문제(암기)
- 간선을 인접 리스트 형태로 저장
- 시작점부터 힙에 넣기

- 힙이 빌때까지,
▶ 해당 노드 방문 안한곳일 경우
▶ 방문 체크, 비용 추가, 연결된 간선 새롭게 추가
시간 복잡도
- Edge 리스트에 저장 : O(E)
- Heap 안 모든 Edge에 연결된 간선 확인 : O(E + E)
- 모든 간선 힙에 삽입 : O(ElgE)
- O(E + 2E + ElgE) = O(3E + ElgE) = O(E(3+lgE)) = O(ElgE)
자료구조
- Edge 저장 리스트 (int, int)[] // 무게, 다음노드
▶ 무게 최대 : 1e6 〉 int 가능
▶ 정점 번호 최대 : 1e4 〉 int 가능
- 정점 방문 : bool[]
- MST 비용 : int(최대 2^31보다 이내)

코드
"""
1. 아이디어
- MST 기본문제, 외우기
- 간선을 인접리스트에 집어넣기
- 힙에 시작점 넣기
- 힙이 빌때까지 다음의 작업을 반복
  - 힙의 최소값 꺼내서, 해당 노드 방문 안했다면
    - 방문표시, 해당 비용 추가, 연결된 간선들 힙에 넣어주기

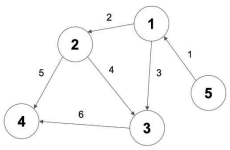
2. 시간복잡도
- MST : O(ElgE)

3. 자료구조
- 간전 저장 되는 인접리스트 : (무게, 노드번호)
- 힙 : (무게, 노드번호)
- 방문여부 : bool[]
- MST 결과값 : int
"""
```
import sys
import heapq
input = sys.stdin.readline

V, E = map(int, input().split())
edge =  [[] for _ in range(V + 1)] # 가중치, 노드 번호
chk = [False] * (V + 1)
rs = 0

for i in range(E):
    a, b, c = map(int, input().split())
    edge[a].append([c , b])
    edge[b].append([c , a])

heap = [[0, 1]]

while heap:
    w, each_node = heapq.heappop(heap)
    if chk[each_node] == False: # 방문 여부 확인
        chk[each_node] = True
        rs += w
        for next_edge in edge[each_node]:
            if chk[next_edge[1]] == False:
                heapq.heappush(heap, next_edge)

print(rs)
```

Tip
- 최소 스패닝 트리 코드는 그냥 외우기
- 중요한건, 해당 문제가 MST 문제인지 알아내는 능력
▶ 모든 노드가 연결되도록 한다거나
▶ 이미 연결된 노드를 최소의 비용으로 줄이기

C++

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

using namespace std;
typedef pair<int, pair<int, int>> T; // 배열생성 -> 백터<int <int,int>>

int V, E; // 노드, 간선
vector<T> v; // 백터 생성
int parent[10000 + 1]; // chk와 같은 역할
int ans;

int FindParent(int x) { // 본인인 경우 패스
	if (parent[x] == x) return x;
	return parent[x] = FindParent(parent[x]); //
}

void UnionParent(int a, int b) {
	a = FindParent(a);
	b = FindParent(b);
	parent[b] = a;
}

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

	cin >> V >> E;
	for (int i = 0; i < E; i++)
	{
		int a, b, c; // 연결된 노드, 노드, 가중치
		cin >> a >> b >> c;
		v.push_back({ c, {a, b} });
	}
	sort(v.begin(), v.end()); // 백터 정렬
	for (int i = 1; i <= V; i++)
		parent[i] = i;

	int cnt = 0;
	for (int i = 0; i < v.size(); i++)
	{
		T curEdge = v[i];
		int cost = curEdge.first;
		int now = curEdge.second.first;
		int next = curEdge.second.second;

		if (FindParent(now) == FindParent(next)) continue;

		UnionParent(now, next);
		ans += cost;

		if (++cnt == V - 1) break;
	}
	cout << ans << '\n';
}
```
==========
다익스트라
개념
- 한 노드에서 다른 모든 노드까지 가는데 최소비용

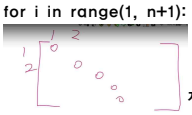
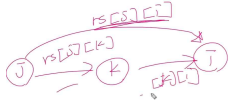


작동 원리
- 간선 : 인접 리스트, 거리배열 : 초기값 무한대로 설정, 힙 시작점 추가
- 힙에서 현재 노드 빼면서, 간선 통할 때 더 거리 짧아진다면
▶ 거리 갱신 및 힙에 추가
핵심코드
```
dist[K] = 0
heapq.heappush(heap, (0, K)) // K -> 시작점

while heap:
    w, v = heapq.heappop(heap)
    if w != dist[v] : continue
    for nw, nv in edge[v]:
        if dist[nv] > dist[v] + nw:
            dist[nv] = dist[v] + nw
            heapq.heappush(heap, (dist[nv], nv))
```
기본문제
- 백준 1753 최단경로
- https://www.acmicpc.net/problem/1753
- 입력 // 노드, 간선 // 시작노드
```
5 6
1
5 1 1
1 2 2
1 3 3
2 3 4
2 4 5
3 4 6
```

아이디어
- 한 점에서 다른 모든 점으로의 최단 경로 〉 다익스트라 사용
- 모든 점 거리 초기값 무한대로 증가
- 시작점 거리 0 설정 및 힙에 추가
- 힙에서 하나씩 빼면서 수행할 것
▶ 현재 거리가 새로운 간선 거칠때보다 크다면 갱신
▶ 새로운 거리 힙에 추가

시간복잡도
– 다익스트라 시간복잡도 : ElgV
▶ E : 3e5, lgV = 20
– O(ElgV) = 6e6 〉 가능
변수
– 다익스트라 사용 힙 : (비용(int), 다음노드(int))[]
▶ 비용 최대값 : 10 * 2e4 = 2e5 =〉 int가능
▶ 다음 노드 : 2e4 =〉 int 가능
– 거리 배열 : int[]
▶ 거리 최대값 : 10 * 2e4 = 2e5 =〉 int 가능
– 간선, 인접리스트 : (비용(int), 다음노드(int))[]

코드
"""
1. 아이디어
– 한점시작, 모든 거리 : 다익스트라
– 간선, 인접리스트 저장
– 거리배열 무한대 초기화
– 시작점 : 거리배열 0, 힙에 넣어주기
– 힙에서 빼면서 다음의 것들 수행
  – 최신값인지 먼저 확인
  – 간선을 타고 간 비용이 더 작으면 갱신
2. 시간복잡도
– 다익스트라 : O(ElgV)
  – E : 3e5
  – V : 2e4 , lgV ~= 20
  – ElgV = 6e6 〈 2억 〉〉 가능
3. 변수
– 힙 : (비용, 노드번호)
– 거리 배열 : 비용 : int[]
– 간선 저장, 인접 리스트 : (비용, 노드번호)[]
"""
```
import sys
import heapq
input = sys.stdin.readline
INF = sys.maxsize

V, E = map(int, input().split())
K = int(input())
edge = [[] for _ in range(V+1)] # 인덱스 1번부터 데이터가 저장되기 때문
dist = [INF] * (V + 1) # 거리

for i in range(E): # 간선
    u, v, w = map(int, input().split()) # 비용,다음노드순으로 인접리스트 저장
    edge[u].append([w, v])

# 시작점 초기화
dist[K] = 0
heap = [[0, K]]

while heap:
    ew, ev = heapq.heappop(heap) # 현재 가중치, 현재 노드
    # 최신값인지 확인
    if dist[ev] != ew: continue
    for nw, nv in edge[ev]: # 다음 가중치, 다음 노드
        if dist[nv] > ew + nw:
            dist[nv] = ew + nw
            heapq.heappush(heap, [dist[nv], nv])

for i in range(1, V + 1):
    if dist[i] == INF : print("INF")
    else:
        print(dist[i])
```
Tip
– 다익스트라 코드는 그냥 외우기 –〉 이진탐색, MST
– 코드가 복잡하므로 연습 필요
– 중요한건, 해당 문제가 다익스트라 문제인지 알아내는 능력
▶ 한 점에서 다른 점으로 가는 최소비용
▶ 코드변형이 거의 없으므로 알아채는 것이 중요
▶ 모든 점에서 모든 점 –〉 VElgV

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

using namespace std;

int INF = 98765432;
int dp[20003];
vector<pair<int, int>> v[20003];

void Dijstra(int st) { // 다익스트라
	memset(dp, INF, sizeof(dp)); // 초기값
	priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>>
pq; // 우선순위 큐
	pq.push({ 0, st });
	dp[st] = 0; // 시작 정점 거리

	while (!pq.empty())
	{
		int x = pq.top().second; // 현재 정점
		int cost = pq.top().first; // 현재 정점까지 거리
		pq.pop();
		for (int i = 0; i < v[x].size(); i++)
		{
			int nx = v[x][i].first; // 다음 정점
			int ncost = cost + v[x][i].second; // 다음정점까지 거리

			if (dp[nx] > ncost) { // 다음 정점에 기록된 거리와 비교
				pq.push({ ncost, nx });
				dp[nx] = ncost;
			}
		}
	}
}


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

	int V, E, K;
	cin >> V >> E >> K;
	int a1, a2, a3;
	for (int i = 0; i < E; i++)
	{
		cin >> a1 >> a2 >> a3;
		v[a1].push_back({ a2, a3 });
	}
	Dijstra(K);
	for (int i = 1; i <= V; i++)
	{
		if (dp[i] < INF) {
			cout << dp[i] << '\n';
		}
		else
			cout << "INF" << '\n';
	}
}
```
==========
플로이드
– 모든 노드에서 다른 모든 노드까지 가는데 최소비용, O(V^3)
– 다익스트라 : 한 노드 –〉 다른 모든 노드, O(ElgV)
▶ Elgv * V = VElgV = V^3lgV –〉 lgV만큼의 시간절약
작동원리
– 노드 j –〉 노드 i비용 배열[][] 만들기, 초기값 : INF
– 간선의 값을 비용 배열에 반영
– 모든 노드에 대해 해당 노드 거쳐서 가서 비용 작아질 경우 값 갱신
▶ 모두 INF로 저장되어 있으므로 값을 갱신 시 작아짐
– 거치는 노드 – for / 시작노드 – for / 끝 – for =〉 V^3
핵심코드
```
for i in range(1, n+1): rs[i][i] = 0 // 자기 자신
```

자기 자신한테 가는 것은 0으로 초기화
```
for i in range(m):
    a, b, c = map(int, input().split()) // 시작점, 끝점, 거리
    rs[a][b] = min(rs[a][b], c) // 최솟값을 위 배열에 대입

for k in range(1, n+1): // 거치는 값
    for j in range(1, n+1): // 시작점
        for i in range(1, n+1): // 끝점
            if rs[j][i] > rs[j][k] + rs[k][i]: // 시작에서 끝으로 가는 것이 더 큰 경우
```

```
                rs[j][i] = rs[j][k] + rs[k][i] // 변경해라
```
기본문제
– 백준 11404 플로이드
– https://www.acmicpc.net/problem/11404
– 입력 // 노드// 간선// 시작, 끝, 비용
```
5
14
1 2 2
```

```
1 3 3
1 4 1
1 5 10
2 4 2
3 4 1
3 5 1
4 5 3
3 5 10
3 1 8
1 4 2
5 1 7
3 4 2
5 2 4
```

아이디어
– 한점 –〉 모든 점 : 다익스트라
– 모든점 –〉 모든 점 : 플로이드 사용
– 거리 초기값 무한대로 설정, 자기 자신으로 가는 값 0 설정
– 노드 거쳐서 가서 비용 작아질 경우 값 갱신
시간복잡도
– 다익스트라 사용할 경우 : ElgV * V
▶ 1e5 * 10 * 1e2 = 1e8 =〉 시간초과 가능성
– 플로이드 사용할 경우 : V^3
▶ (1e2)^3 = 1e6 〉 가능
변수
– 거리배열 : int[][]
▶ 비용 최대 : 1e5 * 1e2 = 1e7 〉 int 가능

코드

```
"""
1. 아이디어
– 모든 점 –〉 모든 점 : 플로이드
– 비용배열 INF 초기화
– 간선의 비용을 대입
– 거쳐서 비용 줄어들경우, 갱신(for 3번)

2. 시간복잡도
– 플로이드 : O(V^3)
– 1e6 〉 가능

3. 변수
– 비용배열 , int [][]

"""

import sys
input = sys.stdin.readline
INF = sys.maxsize

n = int(input())
m = int(input())
rs = [[INF] * (n + 1) for _ in range(n+1)]

for i in range(1, n+1):
    rs[i][i] = 0

for i in range(m):
    a, b, c = map(int, input().split())
    rs[a][b] = min(rs[a][b], c)

# k –〉 i(다수) / j(다수) –〉 k / j –〉 k(다수) –〉 i
for k in range(1, n+1): # 거치는 값
    for j in range(1, n+1): # 시작 점
        for i in range(1, n+1): # 도착점
            # 기존의 값보다 거쳐서 가는 값이 더 작다면 갱신시키기
            if rs[j][i] 〉 rs[j][k] + rs[k][i]:
                rs[j][i] = rs[j][k] + rs[k][i]

for j in range(1, n+1):
    for i in range(1, n+1):
        if rs[j][i] == INF: print(0, end=' ')
        else: print(rs[j][i], end=' ')
    print()
```

Tip
– 그래프 거리 문제 나올 때
▶ 한 점 –〉 여러 점 : 다익스트라(ElgV)
▶ 여러 점 –〉 여러 점 : 플로이드(V^3)
▶ MST
– 코드가 복잡하므로 연습 필요

C++

```
#include 〈iostream〉
#include 〈algorithm〉
#define INF 987654321
#define ARR_SIZE 100 + 1

using namespace std;

int vertex, edge;
int arr[ARR_SIZE][ARR_SIZE];
int from, to, weight;

void FloydWarshall() { // i vertex를 거치는 경우
   for (int i = 1; i 〈= vertex; i++) // from vertex
      for (int j = 1; j 〈= vertex; j++) // to vertex
         for (int k = 1; k 〈= vertex; k++)
            if (arr[j][i] != INF && arr[i][k] != INF)
               arr[j][k] = min(arr[j][k], arr[j][i] + arr[i][k]);
}

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

   cin 〉〉 vertex 〉〉 edge;
   for (int i = 1; i 〈= vertex; i++) // vertex table 초기화
      for (int j = 1; j 〈= vertex; j++)
         arr[i][j] = INF;

   for (int i = 0; i 〈 edge; i++) { // from vertex에서 to vertex 입력, 가중치 입력
      cin 〉〉 from 〉〉 to 〉〉 weight;
      if (arr[from][to] 〉 weight)
         arr[from][to] = weight;
   }

   FloydWarshall();
   for (int i = 1; i 〈= vertex; i++) {
      for (int j = 1; j 〈= vertex; j++) {
         if (i == j || arr[i][j] == INF)
            cout 〈〈 0 〈〈 " ";
         else
            cout 〈〈 arr[i][j] 〈〈 " ";
      }
      cout 〈〈 endl;
   }
}
```

==========

오답노트 작성
– 알고리즘 별 정리
– 문제의 유형을 스스로 요약(자기만의 언어로)
– 틀린문제 주기적으로 풀기
– 중요!! : 면접과 코테가 반복될 때 유용
– 예시
– https://docs.google.com/document/d/1KFb_vR5HTIszkmQiyySfQy9mk2k7UQ–yE9KE4–XuWoE/edit

스터디
– 두시간에 4문제
– 난이도 : 실버 1 ~ 골드 1