완전탐색

0. 완전탐색이란?
1. 브루트 포스
2. 재귀
    재귀 활용시 주의할 점
    예제: 14891 톱니바퀴
3. 조합과 순열
    itertools 활용
    직접 구현해보기
    예제: 15686 치킨배달
4. dfs bfs
    예제: 14500 테트로미노
5. 비트마스크
문제 풀이
    공통 문제:
    개인 문제:

0. 완전탐색이란?

- **모든 경우의 수를 다 확인**하는 방법

  → 어떠한 문제에 접근하는 가장 기본적인 방법
- 알고리즘이라기보다는 일종의 문제 접근 방식
- 가장 정확하지만 **가장 느린 방법**

  → 아래 테크닉들 및 기타 알고리즘과 결합해서 최적화
- **코딩테스트 준비 시 다양한 유형의 문제 풀어서 숙달 필요!!**

1. 브루트 포스

- 반복문 사용해서 모든 경우의 수 탐색
- for, if, while….
- 사실상 **완전탐색의 기본** → 다양한 알고리즘 및 테크닉과 결합!!

2. 재귀

- 자기 자신(함수)를 연속적으로 호출하여 연산

재귀 활용시 주의할 점

- **기저 조건(base condition)** 잘 설정하기
- **탈출 조건** 잘 설정하기
- **현재 함수 상태 저장하는 변수** 설정!(ex - cnt)
    - global 변수 활용
- 문제 **변수 범위** 잘 확인하고 적용하기 (시간 복잡도)
- (파이썬)**재귀 깊이** 조정하기
    - `sys.setrecursionlimit(<depth:int>)`
    - pypy로 할 경우 메모리 초과에 주의!

예제: 14891 톱니바퀴

https://www.acmicpc.net/problem/14891

▼ 정답코드(브루트 포스)

```
# 220826 14891 톱니바퀴

from collections import deque
# import sys

# input = sys.stdin.readline


# rotate: 톱니바퀴 회전 함수 선언
def rotate(gear: list, gear_num: int, direction: int):
    # 1번 톱니바퀴일 때
    if gear_num == 1:
        # 1, 2, 3, 4 톱니바퀴가 맞물릴 때
        if (gear[0][2] != gear[1][6]) and (gear[1][2] != gear[2][6]) and (gear[2][2] != gear[3][6]):
            # print(1, 1)
            gear[0].rotate(direction)
            gear[1].rotate(-direction)
            gear[2].rotate(direction)
            gear[3].rotate(-direction)
        # 1, 2, 3 톱니바퀴가 맞물릴 때
        elif (gear[0][2] != gear[1][6]) and (gear[1][2] != gear[2][6]):
            # print(1, 2)
            gear[0].rotate(direction)
            gear[1].rotate(-direction)
            gear[2].rotate(direction)
        # 1, 2 톱니바퀴가 맞물릴 때
        elif gear[0][2] != gear[1][6]:
            # print(1, 3)
            gear[0].rotate(direction)
            gear[1].rotate(-direction)
        # 아무 톱니바퀴와도 맞물리지 않을 때
        else:
            # print(1, 4)
            gear[0].rotate(direction)

    # 2번 톱니바퀴
    elif gear_num == 2:
        # 1, 2, 3, 4 톱니바퀴가 맞물릴 때
        if (gear[0][2] != gear[1][6]) and (gear[1][2] != gear[2][6]) and (gear[2][2] != gear[3][6]):
            # print(2, 1)
            gear[0].rotate(-direction)
            gear[1].rotate(direction)
            gear[2].rotate(-direction)
            gear[3].rotate(direction)
        # 2, 3, 4 톱니바퀴가 맞물릴 때
        elif (gear[1][2] != gear[2][6]) and (gear[2][2] != gear[3][6]):
            # print(2, 2)
            gear[1].rotate(direction)
            gear[2].rotate(-direction)
            gear[3].rotate(direction)
        # 1, 2, 3 톱니바퀴가 맞물릴 때
        elif (gear[0][2] != gear[1][6]) and (gear[1][2] != gear[2][6]):
            # print(2, 4)
            gear[0].rotate(-direction)
            gear[1].rotate(direction)
            gear[2].rotate(-direction)
        # 2, 3 톱니바퀴가 맞물릴 때
        elif gear[1][2] != gear[2][6]:
            # print(2, 3)
            gear[1].rotate(direction)
            gear[2].rotate(-direction)
        # 1, 2 톱니바퀴가 맞물릴 때
        elif gear[0][2] != gear[1][6]:
            # print(2, 5)
            gear[0].rotate(-direction)
            gear[1].rotate(direction)
        # 아무 톱니바퀴와도 맞물리지 않을 때
        else:
            # print(2, 6)
            gear[1].rotate(direction)

    # 3번 톱니바퀴
    elif gear_num == 3:
        if (gear[0][2] != gear[1][6]) and (gear[1][2] != gear[2][6]) and (gear[2][2] != gear[3][6]):
            # print(3, 1)
            gear[0].rotate(direction)
            gear[1].rotate(-direction)
            gear[2].rotate(direction)
            gear[3].rotate(-direction)
        elif (gear[1][2] != gear[2][6]) and (gear[2][2] != gear[3][6]):
            # print(3, 2)
            gear[1].rotate(-direction)
            gear[2].rotate(direction)
            gear[3].rotate(-direction)
        elif (gear[0][2] != gear[1][6]) and (gear[1][2] != gear[2][6]):
```

```
            # print(3, 4)
            gear[0].rotate(direction)
            gear[1].rotate(-direction)
            gear[2].rotate(direction)
        elif gear[2][2] != gear[3][6]:
            # print(3, 3)
            gear[2].rotate(direction)
            gear[3].rotate(-direction)
        elif gear[1][2] != gear[2][6]:
            # print(3, 5)
            gear[1].rotate(-direction)
            gear[2].rotate(direction)
        else:
            # print(3, 6)
            gear[2].rotate(direction)

    # 4번 톱니바퀴
    else:
        if (gear[0][2] != gear[1][6]) and (gear[1][2] != gear[2][6]) and (gear[2][2] != gear[3][6]):
            # print(4, 1)
            gear[0].rotate(-direction)
            gear[1].rotate(direction)
            gear[2].rotate(-direction)
            gear[3].rotate(direction)
        elif (gear[1][2] != gear[2][6]) and (gear[2][2] != gear[3][6]):
            # print(4, 2)
            gear[1].rotate(direction)
            gear[2].rotate(-direction)
            gear[3].rotate(direction)
        elif gear[2][2] != gear[3][6]:
            # print(4, 3)
            gear[2].rotate(-direction)
            gear[3].rotate(direction)
        else:
            # print(4, 4)
            gear[3].rotate(direction)


# gear: 톱니바퀴
gear = []
for _ in range(4):
    gear.append(deque(map(int, input())))

# K: 회전 수
K = int(input())
# rotate_info[0]: 회전시킬 톱니바퀴 번호, rotate_info[1]: 회전 방향
rotate_info = []
for _ in range(K):
    rotate_info.append(list(map(int, input().split())))

# 주어진 input만큼 회전을 시행한다.
for i in range(K):
    rotate(gear, rotate_info[i][0], rotate_info[i][1])
    # print(gear)

# ans: 점수, 점수를 계산한다
ans = 0
if gear[0][0] == 1:
    ans += 1
if gear[1][0] == 1:
    ans += 2
if gear[2][0] == 1:
    ans += 4
if gear[3][0] == 1:
    ans += 8

print(ans)
```

- 톱니바퀴가 10개, 100개….라면??
  - **톱니바퀴의 움직임을 재귀로 구현**

▼ 정답코드(재귀)(출처: https://velog.io/@tyjk8997/백준-삼성기출-X-톱니바퀴python)

```
# 출처: https://velog.io/@tyjk8997/%EB%B0%B1%EC%A4%80-%EC%82%BC%EC%84%B1%EA%B8%B0%EC%B6%9C-X-%ED%86%B1%EB%8B%88%EB%B0%94%ED%80%

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

wheel = [0] + [ deque(map(int, input().rstrip())) for _ in range(4) ] # 인덱스 1부터 12시방향부터 시계방향, N극 : 0 S극 : 1
k = int(input()) # 회전시킨 횟수
rotate = [ list(map(int, input().split())) for _ in range(k) ] # [ 톱니바퀴 번호, 회전 방향(시계방향 : 1, 반시계방향 : -1) ]
```

```
def rotate_right(x, dr): # 시계 방향으로 회전
    if x > 4 or wheel[x-1][2] == wheel[x][6]: # 범위를 벗어나거나 왼쪽 톱니바퀴와 극이 같은 경우 return
        return

    if wheel[x-1][2] != wheel[x][6]: # 왼쪽 톱니바퀴와 극이 다른경우 재귀를 돌리고 시계방향으로 회전해준다.
        rotate_right(x+1, -dr)
        wheel[x].rotate(dr)


def rotate_left(x, dr): # 반시계 방향으로 회전
    if x < 1 or wheel[x][2] == wheel[x+1][6]: # 범위를 벗어나거나 오른쪽 톱니바퀴와 극이 같은 경우 return
        return

    if wheel[x][2] != wheel[x+1][6]: # 오른쪽 톱니바퀴와 극이 다른경우 재귀를 돌리고 반시계방향으로 회전해준다.
        rotate_left(x-1, -dr)
        wheel[x].rotate(dr)


for num, dr in rotate:

    # 함수를 호출할 때 '-'를 해주는 이유는 극이 다른 경우 방향이 반대가 되기 때문임.
    # deque의 rotate(x)는 x가 양수인 경우 시계방향으로, x가 음수인 경우 반시계방향으로 회전한다.
    rotate_right(num+1, -dr)
    rotate_left(num-1, -dr)
    wheel[num].rotate(dr)


answer = 0
for i in range(1, len(wheel)):
    if wheel[i][0] == 1: # 12시 방향이 S극이면
        answer += 2 ** (i-1)
print(answer)
```

3. 조합과 순열

순열	$_nP_r = \frac{n!}{(n-r)!}$
중복순열	$_n\Pi_r = n^r$
조합	$_nC_r = \frac{n!}{(n-r)!r!}$
중복조합	$_nH_r = {}_{n+r-1}C_r$

itertools 활용

▼ 코드 블럭

```
from itertools import permutations, combinations, combinations_with_replacement


numbers = [1, 2, 3, 4]


# 순열

# 주어진 iterable에서 3개의 원소를 뽑아 순열 만드는 경우의 수
numbers_permut = list(permutations(numbers, 3))

print(numbers_permut)
# [(1, 2, 3), (1, 2, 4), (1, 3, 2), (1, 3, 4), (1, 4, 2), (1, 4, 3), (2, 1, 3), (2, 1, 4), (2, 3, 1), (2, 3, 4), (2, 4, 1), (2, 4,


# 조합

# 주어진 iterable에서 3개의 원소를 뽑아 조합 만드는 경우의 수
numbers_comb = list(combinations(numbers, 3))

print(numbers_comb)
# [(1, 2, 3), (1, 2, 4), (1, 3, 4), (2, 3, 4)]


# 중복 조합
```

```
# 주어진 iterable에서 중복을 허용해 3개의 원소를 뽑아 조합 만드는 경우의 수
numbers_comb_replace = list(combinations_with_replacement(numbers, 3))

print(numbers_comb_replace)
# [(1, 1, 1), (1, 1, 2), (1, 1, 3), (1, 1, 4), (1, 2, 2), (1, 2, 3), (1, 2, 4), (1, 3, 3), (1, 3, 4), (1, 4, 4), (2, 2, 2), (2, 2,
```

직접 구현해보기

- 삼성 코테에서는 itertools가 안된다는 소문이…?

▼ 코드 블럭

```
# 순열
def my_permutaion(arr, n):
    result = []

    if n == 0:
        return [[]]

    for i in range(len(arr)):
        elem = arr[i]
        for j in my_permutaion(arr[:i] + arr[i+1:], n - 1):
            result.append([elem] + j)
            # print(result)

    return result


# 조합
def my_combination(arr, n):
    result = []

    if n == 0:
        return [[]]

    for i in range(len(arr)):
        elem = arr[i]
        for j in my_combination(arr[i + 1:], n - 1):
            result.append([elem] + j)
            # print(result)

    return result


# 중복 조합
def my_combination_with_replacement(arr, n):
    result = []

    if n == 0:
        return [[]]

    for i in range(len(arr)):
        elem = arr[i]
        for j in my_combination(arr[i:], n - 1):
            result.append([elem] + j)
            # print(result)

    return result


numbers = [1, 2, 3]

print(my_permutaion(numbers, 2))
# [[1, 2], [1, 3], [2, 1], [2, 3], [3, 1], [3, 2]]
print(my_combination(numbers, 2))
# [[1, 2], [1, 3], [2, 3]]
print(my_combination_with_replacement(numbers, 2))
# [[1, 1], [1, 2], [1, 3], [2, 2], [2, 3], [3, 3]]
```

예제: 15686 치킨배달

https://www.acmicpc.net/problem/15686

- **치킨집 M개에 대한 경우의 수(조합)** 구해서 반복문

▼ 정답코드

```
# 220904 15686 치킨 배달
```

```
import sys
from itertools import combinations
input = sys.stdin.readline

# N: 도시 배열의 크기, M: 남길 치킨 집 개수, arr: 도시의 배열
N, M = map(int, input().split())
arr = [list(map(int, input().split())) for _ in range(N)]

# chicken: 현재 있는 치킨집 위치, home: 도시 내 집들의 위치
chicken = []
home = []
# 현재 도시 내에 치킨집과 집들의 위치를 각각 리스트에 담아준다.
for r in range(N):
    for c in range(N):
        if arr[r][c] == 1:
            home.append((r, c))
        elif arr[r][c] == 2:
            chicken.append((r, c))

# ans: 정답을 담을 변수(충분히 큰 값으로 초기화)
ans = 99999
# 조합을 이용해 가능한 치킨집 경우의 수를 모두 구해 반복문
for left_chickens in combinations(chicken, M):
    # chicken_distance: 도시 전체 치킨 거리
    chicken_distance = 0

    # 각 집마다 치킨 거리를 구한다.
    for house in home:
        # min_distance: 한 집의 치킨 거리(충분히 큰 값으로 초기화)
        min_distance = 99999

        # 모든 치킨집의 치킨 거리를 구한 뒤 계속 비교해가며 최솟값을 구한다.
        for i in range(M):
            distance = abs(house[0] - left_chickens[i][0]) + abs(house[1] - left_chickens[i][1])
            if distance < min_distance:
                min_distance = distance

        # 도시 전체 치킨 거리에 더해주기
        chicken_distance += min_distance

    # 현재 최소 치킨 거리랑 비교해가면서 갱신
    if chicken_distance < ans:
        ans = chicken_distance

print(ans)
```

4. dfs bfs

- 우리가 잘 아는 그거…
- 시간복잡도
  - 그래프(인접 리스트): O(N + E) (N: 정점, E: 간선)
  - 이차원배열: O(NM)
- **백트래킹과 결합해서** 최적화!
- 상태공간 정의에 주의하자!

예제: 14500 테트로미노

▼ 정답코드(브루트포스)

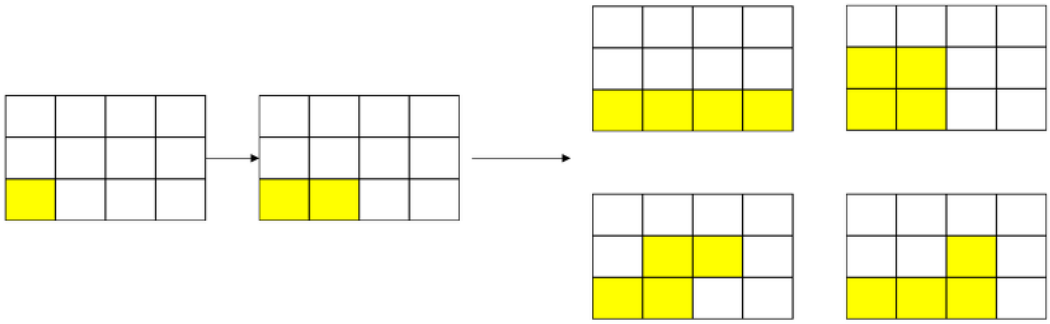
```
# 220903 14500 테트리미노

# 정답코드

import sys
input = sys.stdin.readline


# 단방향 테트로미노 검증 함수
# 기본 배열에 한번, 90도 회전 시킨 배열에 또 한번 적용해서 결과를 구한다.
def tetromino(arr, N, M, max_sum):

    # num_sum: 블럭 숫자의 합계

    # (ㅡ)자 블럭 검증
    for r in range(N):
        for c in range(M - 3):
            num_sum = sum(arr[r][c:c + 4])
```

```
            # 최댓값 비교해주면서 갱신
            if num_sum > max_sum:
                max_sum = num_sum

    # 사각형 블럭 검증
    for r in range(N - 1):
        for c in range(M - 1):
            num_sum = arr[r][c] + arr[r + 1][c] + arr[r][c + 1] + arr[r + 1][c + 1]
            if num_sum > max_sum:
                max_sum = num_sum

    # L, ㄴ, z자 블럭 검증
    for r in range(N - 1):
        for c in range(M - 2):
            # 6(2 * 3)칸짜리 배열 잘라내기
            mini_arr = [arr[r][c:c + 3], arr[r + 1][c:c + 3]]

            # L, ㄴ자 블럭 검증
            # temp: 수들의 합을 담을 임시 배열
            temp = []
            # 잘라낸 6칸짜리 배열에서 (밑/위)에 3칸에 (밑/위)에 한칸씩 각각 더한다.
            for i in range(3):
                num_sum = sum(mini_arr[0]) + mini_arr[1][i]
                temp.append(num_sum)
                num_sum = sum(mini_arr[1]) + mini_arr[0][i]
                temp.append(num_sum)
            # 최댓값만을 뽑아내서 현재 최댓값과 비교해서 갱신
            if max(temp) > max_sum:
                max_sum = max(temp)

            # z자 블럭 검증
            # temp: 수들의 합을 담을 임시 배열
            temp = []
            # (밑/위)행 0, 1번 원소 + (밑/위)행 1, 2번 원소
            num_sum = sum(mini_arr[0][0:2]) + sum(mini_arr[1][1:3])
            temp.append(num_sum)
            num_sum = sum(mini_arr[1][0:2]) + sum(mini_arr[0][1:3])
            temp.append(num_sum)
            # 최댓값만을 뽑아내서 현재 최댓값과 비교해서 갱신
            if max(temp) > max_sum:
                max_sum = max(temp)

    # 최댓값을 반환한다.
    return max_sum


# N, M: 배열의 크기, arr: 배열
N, M = map(int, input().split())
arr = [list(map(int, input().split())) for _ in range(N)]

# first_max: 배열 arr 테트로미노(단방향)의 최댓값
first_max = tetromino(arr, N, M, 0)

# arr_rotate: 배열 arr을 시계방향으로 90도 회전시킨 배열
arr_rotate = list(map(list, zip(*arr[::-1])))

# ans: first_max에서 이어서 검증한 arr_rotate 테트리미노의 최댓값
ans = tetromino(arr_rotate, M, N, first_max)

print(ans)
```

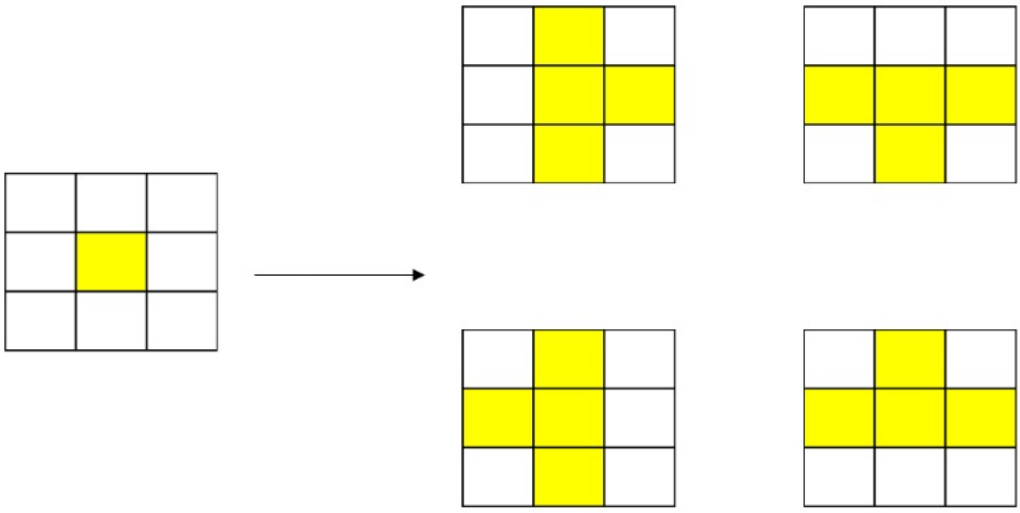
- 그런데 블럭들 모양이 뭔가 유사한데…?

▼ DFS 활용!

- dfs로 탐색! (*백트래킹 필수!)



- 凸자 블럭은 따로 계산해주자!



5. 비트마스크

- **비트 연산** 활용
- 주로 **집합 & 순열** 문제에서 활용
- 빠른 처리 속도와 간결한 코드
- 자세한 내용은 생략한다,,,

  https://justkode.kr/algorithm/bitmash

문제 풀이

공통 문제:

- 14502: 연구소: https://www.acmicpc.net/problem/14502

개인 문제:

https://www.acmicpc.net/workbook/view/7316

https://www.acmicpc.net/workbook/view/1152