**#소수 찾기**

**문제 설명**

한자리 숫자가 적힌 종이 조각이 흩어져있습니다. 흩어진 종이 조각을 붙여 소수를 몇 개 만들 수 있는지 알아내려 합니다.

각 종이 조각에 적힌 숫자가 적힌 문자열 numbers가 주어졌을 때, 종이 조각으로 만들 수 있는 소수가 몇 개인지 return 하도록 solution 함수를 완성해주세요.

제한사항

* numbers는 길이 1 이상 7 이하인 문자열입니다.
* numbers는 0~9까지 숫자만으로 이루어져 있습니다.
* 013은 0, 1, 3 숫자가 적힌 종이 조각이 흩어져있다는 의미입니다.

입출력 예

| **numbers** | **return** |
| --- | --- |
| “17” | 3 |
| “011” | 2 |

def **solution**(numbers):

    answer = 0

    A = []

    file = []

    ans = []

    for i in numbers:

        A.append(i)

*#itertool이라는 파이썬의 내장 함수를 이용하여서 순열을 구한다. 원래는 DFS를 이용해서 구했었는데, 그렇게 하게 되면 numbers의 자리수가 3개 이상일 경우에 모든 경우를 구할 수 없다. 예를 들면 양끜 말고 여러 수 사이에 숫자를 입력해서 경우를 만들 때*

    import itertools

    for i in **range**(1,**len**(A)+1):

        file.extend(list(**map**(''.join, itertools.permutations(A,i))))

    for i in file:

        ans.append(int(i))

    ans = list(set(ans))

    num = **max**(ans)

*#소수 찾기(에라토스테네스의 체를 이용하여 ans리스트에 있는 수들 중의 최댓값까지의 수들을 check리스트에 소수면 1, 아니면 0으로 체크 되도록 한다.*

    check = [0 for x in **range**(num+1)]

*#0과1은 소수가 아님 여기서 미리 조치를 취해 주어야 함*

    check[0] = 1

    check[1] = 1

    for i in **range**(2, num+1):

*#range(i, num, i)로 안한 이유는 그 이전의 수들, 즉 i보다 더 작은 수들의 배수 중에 없다면 i는 소수이고, 자기자신이 약수이자 배수인 것은 소수의 조건이 아니다.*

        for j in **range**(i\*2, num+1, i):

            check[j] = 1

    for i in ans:

        if check[i] == 0:

            answer += 1

*#함수 내에서 print로 하면 None이 발생된다. 그냥 return으로 하는게 맞다.*

    return answer

**#괄호 변환 (2020 KAKAO BLIND RECRUITMENT)**

###### 문제 설명

카카오에 신입 개발자로 입사한 **콘**은 선배 개발자로부터 개발역량 강화를 위해 다른 개발자가 작성한 소스 코드를 분석하여 문제점을 발견하고 수정하라는 업무 과제를 받았습니다. 소스를 컴파일하여 로그를 보니 대부분 소스 코드 내 작성된 괄호가 개수는 맞지만 짝이 맞지 않은 형태로 작성되어 오류가 나는 것을 알게 되었습니다.  
수정해야 할 소스 파일이 너무 많아서 고민하던 콘은 소스 코드에 작성된 모든 괄호를 뽑아서 올바른 순서대로 배치된 괄호 문자열을 알려주는 프로그램을 다음과 같이 개발하려고 합니다.

### **용어의 정의**

**'('** 와 **')'** 로만 이루어진 문자열이 있을 경우, '(' 의 개수와 ')' 의 개수가 같다면 이를 **균형잡힌 괄호 문자열**이라고 부릅니다.  
그리고 여기에 '('와 ')'의 괄호의 짝도 모두 맞을 경우에는 이를 **올바른 괄호 문자열**이라고 부릅니다.  
예를 들어, "(()))("와 같은 문자열은 균형잡힌 괄호 문자열 이지만 올바른 괄호 문자열은 아닙니다.  
반면에 "(())()"와 같은 문자열은 균형잡힌 괄호 문자열 이면서 동시에 올바른 괄호 문자열 입니다.

'(' 와 ')' 로만 이루어진 문자열 w가 균형잡힌 괄호 문자열 이라면 다음과 같은 과정을 통해 올바른 괄호 문자열로 변환할 수 있습니다.

1. 입력이 빈 문자열인 경우, 빈 문자열을 반환합니다.

2. 문자열 w를 두 "균형잡힌 괄호 문자열" u, v로 분리합니다. 단, u는 "균형잡힌 괄호 문자열"로 더 이상 분리할 수 없어야 하며, v는 빈 문자열이 될 수 있습니다.

3. 문자열 u가 "올바른 괄호 문자열" 이라면 문자열 v에 대해 1단계부터 다시 수행합니다.

3-1. 수행한 결과 문자열을 u에 이어 붙인 후 반환합니다.

4. 문자열 u가 "올바른 괄호 문자열"이 아니라면 아래 과정을 수행합니다.

4-1. 빈 문자열에 첫 번째 문자로 '('를 붙입니다.

4-2. 문자열 v에 대해 1단계부터 재귀적으로 수행한 결과 문자열을 이어 붙입니다.

4-3. ')'를 다시 붙입니다.

4-4. u의 첫 번째와 마지막 문자를 제거하고, 나머지 문자열의 괄호 방향을 뒤집어서 뒤에 붙입니다.

4-5. 생성된 문자열을 반환합니다.

**균형잡힌 괄호 문자열** p가 매개변수로 주어질 때, 주어진 알고리즘을 수행해 **올바른 괄호 문자열**로 변환한 결과를 return 하도록 solution 함수를 완성해 주세요.

### **매개변수 설명**

* p는 '(' 와 ')' 로만 이루어진 문자열이며 길이는 2 이상 1,000 이하인 짝수입니다.
* 문자열 p를 이루는 '(' 와 ')' 의 개수는 항상 같습니다.
* 만약 p가 이미 올바른 괄호 문자열이라면 그대로 return 하면 됩니다

### **입출력 예**

| **p** | **result** |
| --- | --- |
| "(()())()" | "(()())()" |
| ")(" | "()" |
| "()))((()" | "()(())()" |

*#방법들이나 판단할 것들이 많기 떄문에 함수 설정 해주기*

*#균형잡힌 문자열의 길이 확인*

def **balanced**(p):

*#계속 이렇게 해서 무한루프가 돌고 함수로 계산할 문자열이 없었다. 그래서 아래 코드로 바꾸어 주었다.*

*#a,b = 0,0*

*#while a != b:*

*#for i in p:*

*#if i == '(':*

*#a += 1*

*#else:*

*#b += 1*

*#return a\*2*

    a = 0

    for idx, value in **enumerate**(p):

        if value == '(':

            a += 1

        if value == ')':

            a -= 1

        if a == 0:

            return idx+1

*#올바른 괄호 문자열인지 확인*

def **isRight**(string):

    ans = []

    for i in string:

        if i == '(':

            ans.append(i)

        else:

            if **len**(ans)== 0:

                return False

            ans.pop()

    if **len**(ans)!=0:

        return False

    return True

*#출력할 문자열을 만드는 역할을 한다*

def **solution**(p):

*#빈문자열이나 자체로 올바른 문자열일때 그대로 반환*

    if p =='' or isRight(p):

        return p

    u,v = p[:balanced(p)], p[balanced(p):]

    if isRight(u):

        return u+solution(v)

    else:

        answer = ''

*#여기서 u[0:-1]이라고 해서 slicing실수를 했었음*

        u = list(u[1:-1])

        for i in **range**(**len**(u)):

            if u[i] == '(':

                u[i] = ')'

            else:

                u[i] = '('

        answer += '('+solution(v)+')'

        for i in u:

            answer += i

        return answer

**#가장 큰 수**

**문제 설명**

0 또는 양의 정수가 주어졌을 때, 정수를 이어 붙여 만들 수 있는 가장 큰 수를 알아내 주세요.

예를 들어, 주어진 정수가 [6, 10, 2]라면 [6102, 6210, 1062, 1026, 2610, 2106]를 만들 수 있고, 이중 가장 큰 수는 6210입니다.

0 또는 양의 정수가 담긴 배열 numbers가 매개변수로 주어질 때, 순서를 재배치하여 만들 수 있는 가장 큰 수를 문자열로 바꾸어 return 하도록 solution 함수를 작성해주세요.

제한 사항

* numbers의 길이는 1 이상 100,000 이하입니다.
* numbers의 원소는 0 이상 1,000 이하입니다.
* 정답이 너무 클 수 있으니 문자열로 바꾸어 return 합니다.

입출력 예

| **numbers** | **return** |
| --- | --- |
| [6, 10, 2] | 6210 |
| [3, 30, 34, 5, 9] |  |

처음에는 그냥 DFS를 이용하여

numbers = [6,10,2]

numbers = [str(x) for x in numbers]

ch = [0]\***len**(numbers)

res = 0

def **DFS**(v,ans):

    global res

    if v == **len**(numbers):

        res = **max**(res, int(ans))

    else:

        for x in **range**(**len**(numbers)):

            if ch[x] == 0:

                ch[x] = 1

                DFS(v+1, ans+numbers[x])

                ch[x] = 0

이런 식으로 하려 했었는데, 프로그래머스의 채점 특성상 def solution(numbers)를 이용해야 하기 때문에 계속 없는 변수라고 나왔었다. 또한, 모든 경우를 다 해보아야 했기 때문에 시간 초과가 날 수밖에 없었다. 물론 처음에 주어진 numbers라는 리스트를 reverse=True로 정렬을 한 뒤에 넣으려 했지만, 그럴 때는 첫 자리 수가 같은 경우를 피할 수 없었다. 그래서 numbers의 리스트에 있는 모든 수가 1000이하임을 이용해서 각 숫자를 3번 반복하여 그 상태에서 내림차순 정렬을 한 뒤에 for i, value in enumerate numbers를 한 뒤에 다른 리스트에 [str\*3,i]를 input했다. 그렇게 하면 내림차순으로 정렬을 해도 numbers리스트에서의 index는 알 수 있기 때문에 answer = ‘’에 순서대로 문자열을 더한다.

answer = ''

file = []

for i, value in **enumerate**(numbers):

    st = str(value)

    st1 = st\*3

    file.append([st1,i])

file.sort(reverse = True)

for i, value in **enumerate**(file):

    index = value[1]

    answer += str(numbers[index])

for i in answer:

    if i != '0':

        print(answer)

print('0')

**#더 맵게**

**문제 설명**

매운 것을 좋아하는 Leo는 모든 음식의 스코빌 지수를 K 이상으로 만들고 싶습니다. 모든 음식의 스코빌 지수를 K 이상으로 만들기 위해 Leo는 스코빌 지수가 가장 낮은 두 개의 음식을 아래와 같이 특별한 방법으로 섞어 새로운 음식을 만듭니다.

섞은 음식의 스코빌 지수 = 가장 맵지 않은 음식의 스코빌 지수 + (두 번째로 맵지 않은 음식의 스코빌 지수 \* 2)

Leo는 모든 음식의 스코빌 지수가 K 이상이 될 때까지 반복하여 섞습니다.  
Leo가 가진 음식의 스코빌 지수를 담은 배열 scoville과 원하는 스코빌 지수 K가 주어질 때, 모든 음식의 스코빌 지수를 K 이상으로 만들기 위해 섞어야 하는 최소 횟수를 return 하도록 solution 함수를 작성해주세요.

제한 사항

* scoville의 길이는 2 이상 1,000,000 이하입니다.
* K는 0 이상 1,000,000,000 이하입니다.
* scoville의 원소는 각각 0 이상 1,000,000 이하입니다.
* 모든 음식의 스코빌 지수를 K 이상으로 만들 수 없는 경우에는 -1을 return 합니다.

입출력 예

| **scoville** | **K** | **return** |
| --- | --- | --- |
| [1, 2, 3, 9, 10, 12] | 7 | 2 |

일단 heap이라는 우선순위 트리 자료 구조를 사용하는 문제이다. 기본 구조, 즉 heapq모듈을 사용하지 않는다면 리스트가 기본적인 구조이다. 이 자료구조는 파이썬에서 그냥 사용할 수 있는 모듈로, import heapq라고 하면 이용이 가능하다. 이 구조를 이용하면 좋은 이유가 끊임없이 sort()를 이용하지 않더라도 오름차순으로 정렬된 상태를 유지하기 때문이다.

조심해야 할 것은 리스트의 인덱스를 이용해서 다루려 할 때에는 heapq[3]이런식이 절대 아니라 리스트명[3]이런식으로 출력해야 한다.

기존의 리스트를 heapq로 바꾸기: heapq.heapify(list명)

제일 작은 요소 출력: heapq.heappop(리스트 명)

요소 입력: heapq.heappush(리스트명, 입력요소)

*#그냥 리스트 자료구조로 for문을 이용해 정렬 유지(시간초과)*

scoville.sort()

while True:

    if scoville[0] >= k:

        break

    else:

        num = scoville.pop(0)+scoville.pop(0)\*2

        count+=1

        scoville.insert(0,num)

        for x in **range**(**len**(scoville)-1):

            if scoville[x] > scoville[x+1]:

                scoville[x],scoville[x+1] = scoville[x+1],scoville[x]

            else:

                break

print(count)

*#heapq모듈을 이용해서 정렬*

import heapq

heapq.heapify(scoville)

while True:

    if scoville[0] >= k:

        break

    else:

        if **len**(scoville) == 1:

            count = -1

            break

        else:

            num = heapq.heappop(scoville)+heapq.heappop(scoville)\*2

            count += 1

            heapq.heappush(scoville, num)

print(count)

**#숫자 야구**

**문제 설명**

숫자 야구 게임이란 2명이 서로가 생각한 숫자를 맞추는 게임입니다.

각자 서로 다른 1~9까지 3자리 임의의 숫자를 정한 뒤 서로에게 3자리의 숫자를 불러서 결과를 확인합니다. 그리고 그 결과를 토대로 상대가 정한 숫자를 예상한 뒤 맞힙니다.

\* 숫자는 맞지만, 위치가 틀렸을 때는 볼

\* 숫자와 위치가 모두 맞을 때는 스트라이크

\* 숫자와 위치가 모두 틀렸을 때는 아웃

이렇게 할 때 처음에는 순서대로 모든 조건을 따져 가면서 계속 체로 거르는 것처럼 조건에 맞게 리스트를 만들어 가려고 했다. 그러나 생긴 문제점이 우선 베이스가 되는 세 자리 수들이 없다 보니 제대로 추려 낼 수 없다는 것이었다. 그래서 예전에 숫자들을 이용해 순열의 개수의 숫자 조합을 만드는 모듈이 파이썬에 있음을 기억해 내고 from itertools import permutation을 시도해 보았다. 그 결과 답이 나왔다.

그 순서를 다 살펴 보자면 우선

permutation의 기본이 되는 조합이 될 숫자들의 리스트는 현재 대소비교/덧셈 뺄셈등이 아니므로 문자열 조합으로 해야 한다. 그리고 permutation을 실행 하면 각각의 배치가 하나의 수로 나오는 것이 아니라 튜플의 형태로 숫자 조합이 묶여서 나오기 때문에 예를 들면 (‘1’,’2’,’3’) 이런 상태로 묶여 나오기 때문에 만약에 여기서 0번째를 출력 하면 ‘ 이게 출력된다. 그러므로 ‘’.join(file[0])이런 식으로 해서 문자열의 형태로 만들어야 한다.

결국 여기 까지만 간다면 마지막에는 주어진 리스트 속의 요소들과 비교 하는 것만 남기 때문에 이중 for문을 이용해서 답을 구하면 된다.

from itertools import permutations

baseball = [[123,1,1],[356,1,0],[327,2,0],[489,0,1]]

def **baseballgame**(test, answer):

    s,b = 0,0

    for i in **range**(3):

        if test[i] == answer[i]:

            s += 1

        elif test[i] == answer[(1+i)%3] or test[i] == answer[(i+2)%3]:

            b += 1

    return [s,b]

def **solution**(baseball):

    num = ['1','2','3','4','5','6','7','8','9']

    file = list(permutations(num,3))

    for i in **range**(**len**(file)):

        file[i] = ''.join(file[i])

    res = file[:]

    for i in **range**(**len**(baseball)):

        for j in **range**(**len**(file)):

            if baseballgame(str(baseball[i][0]),file[j]) != baseball[i][1:]:

                if file[j] in res:

                    res.remove(file[j])

    return res

#라면 공장

문제는 우선순위 큐를 적용하라고 했으나 왜 그렇게 해야하는지 감이 안 와서 처음에는 DFS를 적용해서 풀어보았으나 테스트 케이스를 제외하고 나머지는 모두 틀렸다고 나왔다. 그래서 큐를 적용할 방법을 생각해 보았다.

우선 큐는 앞서 언급 한 것처럼 항상 정렬된 자료로서 큐에서 pop()하는 삾은 항상 정렬된 가장 첫번째 요소이다. 이 문제에서 제일 중요한 것은 밀가루를 추가할지 말지를 결정하는 것이 아닌, 밀가루가 부족해지는 시점에 밀가루를 추가하는 방식이다. 밀가루를 공급 받을 수 있는 날이 되면 우선 우선순위 큐에 공급 받을 수 있는 밀가루를 넣는다. 그리고 밀가루가 필요한 시점에 공급 가능한 밀가루중 가장 큰 밀가루(.rightpop())를 공급 받음으로써 밀가루를 골급받는 횟수를 최소로 만들어야 한다. 우선순위 큐에 있는 밀가루들은 처음에 기본으로 다 넣는 것이 아닌, 밀가루 공급 시점에 추가하는 것이다. 밀가루는 오늘, 즉 0일부터 시작하며 k-1일에 사용할 수 있는 양까지만 검사하면 되기 때문에 for 조건문을 잘 설정해야 한다.

1. 최대 힙 기반의 우선순위 큐를 작성한다.
2. Stock 가 k 보다 작을 때 까지 다음을 반복한다.
3. Idx~dates의 끝까지 반복한다.
4. stock < dates[i]라면 반복을 멈춘다
5. 그 외에는 우선순위 큐에 supplies[i]를 넣는다.
6. Idx를 I 바로 다음 인덱스로 옮긴다.
7. 우선 순위 큐에서 데이터를 꺼내 stock에 더한다.
8. Answer에 1을 더한다.

위의 방법음 우선 sotck에 있는 값, 즉 남은 밀가루의 양은 제일 마지막 날까지 계속 매일 1톤씩 빠져도 지장이 없을 순간에는 그냥 반복문을 빠져 나가도 좋다. 그러나 만약에 k보다 작을 때에는 그러한 과정/비교를 dates리스트에 있는 날짜에도 그대로 적용하면 된다. 밀가루를 더 공급하는 날보다 많으면 break, 아니면 우선순위 큐에 해당 날에 공급하는 밀가루를 넣으면 된다. 단, 힙은 최소힙 출력만 되므로 heappush를 할 때에 -1을 곱하고 넣은 뒤 빼낼 때 다시 -1을 곱하되, 이때는 heappop이 이용 가능하다.

이후 stock에 큐에 입력된 값들 중 제일 큰 값을 넣고, answer에 1을 더한다.

그리고 마지막에 answer를 출력하면 된다. 내가 주의하지 못한 점은 ‘밀가루가 부족할 때‘를 단순히 남은 밀가루가 -1일때라고 생각했는데, 해당 밀가루 공급 날짜와 비교를 하는게 더 빨랐었다.

import heapq

def **solution**(stock, dates, supplies, k):

    answer, idx = 0,0

    file = []

    while stock < k:

        for i in **range**(idx, **len**(dates)):

            if stock < dates[i]:

                break

            else:

                heapq.heappush(file, -supplies[i])

                idx = i+1

        stock += (heapq.heappop(file)\*-1)

        answer += 1

    print(answer)

이 경우는 k번까지 순서대로 계산해 보면서 했는데, 일단 아얘 모든 답이 틀릴 때에는 stock의 값에 계속 1을 빼주면서도 부가적으로 공급 날짜와 현재 날짜 사이의 차이를 더해주려 했다. 그러나 이렇게 하면 예상보다 stack == -1일 때가 자주 나오므로 안되고 그것을 수정했을 때에는 다 맞는데 효율성 테스트에서 실패를 했기 때문인데, stock >= k일 때break를 하면 전반적으로 휠씬 빨라지지만 그래도 여전히 효율성은 실패 했다. 그래서 결국 큐를 이용하는 것의 핵심은 BFS도 결합하여 반복적으로 idx += 1을 해주는 것이다.

import heapq

def **solution**(stock, dates, supplies,k):

    answer = 0

    file = []

    for i in **range**(1,k):

        if i in dates:

            a = heapq.heappush(file, -supplies.pop(0))

        stock -= 1

        if stock == 0:

            a = heapq.heappop(file)\*-1

            stock += a

            answer += 1

    return answer

**#가장 큰 정사각형 찾기**

**문제 설명**

1와 0로 채워진 표(board)가 있습니다. 표 1칸은 1 x 1 의 정사각형으로 이루어져 있습니다. 표에서 1로 이루어진 가장 큰 정사각형을 찾아 넓이를 return 하는 solution 함수를 완성해 주세요. (단, 정사각형이란 축에 평행한 정사각형을 말합니다.)

이 문제가 사실 제일 풀기도 어렵고 여러가지 방법을 시도 했던 문제이다. 처음에는 그냥 순서대로 1가 있으면 더하고 각 행 별로 최댓값을 도출하려 했으나. 그렇게 되면 정사각형이 제대로 만들어지기가 어렵기 때문에 당연히 답은 나오지 않았다. 그리고 매번 함수를 호출해 해당 지점으로부터에 있는 정사각형의 길이를 찾아 계속 ans변수와 max()를 이용해 갱신해 주려 했으나 이렇게 하니까 0이 나올 때 처리가 막막했고 너무 효율성이 떨어졌다.

결국에는 n으로 정사각형의 길이를 설정해 놓고 순차적으로 n이 커지게 하면서 그 정사각형에 맞게

for i in range(sixe-n+1):

for j in range(size-n+1):

이런 식으로 해서 board 리스트를 슬라이싱 해서 그 리스트에 0이 없으면 ans = n\*\*2로 갱신하려고 했으나, 이상하게 계속 갱신이 안되고 0이 리스트에 있는지 없는지 여부가 계속 제대로 측정이 안되었다. 그래서 바꿔서 마지막으로 동적 계획법을 사용하였다.

from itertools import chain

def **solution**(board):

    height, width = **len**(board), **len**(board[0])

    for i in **range**(1, height):

        for j in **range**(1, width):

            mins = **min**(board[i-1][j], board[i][j-1],board[i-1][j-1])

            if board[i][j]==0 or mins == 0:

                continue

            else:

                board[i][j] = **max**(board[i][j], mins+1)

    return **max**(list(chain(\*board)))\*\*2

이렇게 해결하였는데, 우선 나는 주어지는 board도 정사각형 일 것이라고 너무 당연하게 생각하여 가로 세로 길이를 따로 구했어야 했는데 그러지 않았다는 점에서 오류가 발생했다.

이 경우에는 그냥 board리스트에다가 대놓고 dp의 값을 갱신했는데, board[i-1][j],board[i][j-1], board[i-1][j-1]이 모두 1이라면 board[i][j]까지 포함해서 길이가 2인 정사각형인 것이다. 그래서 min()을 이용하는 것이고, 그렇게 순차적으로 갱신하기 위해 셋중의 최소+1과 현재 값중 최댓값을 board[i][j]의 값으로 바꾸어 주면 된다.

마지막에 이중 리스트를 병합하는데 애를 먹었지만, 가능한 방법은 3가지나 있었다.

1. **sum**(리스트명, [])

2. list(chain,from\_iterable(리스트명))

3. list(chain(\*리스트명))

2, 3번은 import itertools를 해야 이용이 가능하다.

**#튜플**

**문제 설명**

셀수있는 수량의 순서있는 열거 또는 어떤 순서를 따르는 요소들의 모음을 튜플(tuple)이라고 합니다. n개의 요소를 가진 튜플을 n-튜플(n-tuple)이라고 하며, 다음과 같이 표현할 수 있습니다.

예를 들어 튜플이 (2, 1, 3, 4)인 경우 이는 {{2}, {2, 1}, {2, 1, 3}, {2, 1, 3, 4}} 와 같이 표현할 수 있습니다.

특정 튜플을 표현하는 집합이 담긴 문자열 s가 매개변수로 주어질 때, s가 표현하는 튜플을 배열에 담아 return 하도록 solution 함수를 완성해주세요.

def **solution**(s):

    ans= []

    file = s.split('},{')

    file[0] = file[0].strip('{')

    file[-1] = file[-1].strip('}')

    for i in **range**(**len**(file)):

        ans.append(list(**map**(int, file[i].split(','))))

    ans = **sorted**(ans, key = lambda x: **len**(x))

    res = []

    for i in **range**(**len**(ans)):

        for j in **range**(**len**(ans[i])):

            if ans[i][j] not in res:

                res.append(ans[i][j])

    return res

처음에는 이 문제를 풀 때에 문자열의 형태로 주어질 때에 리스트로 바꿔주는 방법을 너무 헷갈려서 이번 기회에 문자열 정렬 방법을 정리해 보고자 한다.

.lstrip() *#문자열 중 가장 왼쪽에 있는 한 칸 이상의 연속된 공백을 지운다.*

*#()안에 다른 값을 넣으면 그 것을 지워줌*

.rstrip() *#.lstrip()과 같은 역할이지만 오른쪽에 관여*

.strip()  *#오른쪽 왼쪽 상관 없이 관여*

.split()  *#()로 하면 공백을 기준으로 문자열을 나누어 준다*

*#()안에 값이 들어가면 그 것을 기준으로 문자열을 나누어 준다*

사전형 자료의 작동법

discard(값) *#괄호 안의 값을 삭제(없을때 무시)*

add(값)     *#괄호 안의 값을 중복이 아닐경우에만 추가*

pop()       *#임의의 값 리턴 후 삭제*

clear()     *#집합의 모든 값 삭제*

copy()      *#집합을 복제하여 리턴*

문자열 -> 리스트

','.join(리스트명) *#문자열의 각 요소를 ,로 나누어서 리스트 형태로 바꿈*

*#알아두면 좋을 진수 변환(파이썬 내장함수 이용)*

b = **format**(value, 'b') *#2진수*

o = **format**(value, '0') *#8진수*

h = **format**(value, 'h') *#16진수*

b = int('0b111100', 2) *#2진수 -> 10진수*

o = int('0o74', 8)     *#8진수 -> 10진수*

h = int('0x3c', 16)    *#16진수 -> 10진수*

*#진수 변환할때 내장 함수가 생각이 안나는 경우(10진수 -> 2진수)*

*#십진수 n을 몫이 0이 될 때까지 2로 계속 나눈다*

*#나머지들을 나중에 나온 순서대로, 즉 역순으로 읽으면 이진수가 된다.*

*#8진수, 16진수 등 어떤 진수 여도 해당 수로 나누면 된다.*

answer = ''

while n:

    answer = str(n%2) + answer

    n = n//2

**#N으로 표현**

**문제 설명**

아래와 같이 5와 사칙연산만으로 12를 표현할 수 있습니다.

12 = 5 + 5 + (5 / 5) + (5 / 5)  
12 = 55 / 5 + 5 / 5  
12 = (55 + 5) / 5

5를 사용한 횟수는 각각 6,5,4 입니다. 그리고 이중 가장 작은 경우는 4입니다.  
이처럼 숫자 N과 number가 주어질 때, N과 사칙연산만 사용해서 표현 할 수 있는 방법 중 N 사용횟수의 최솟값을 return 하도록 solution 함수를 작성하세요.

제한사항

* N은 1 이상 9 이하입니다.
* number는 1 이상 32,000 이하입니다.
* 수식에는 괄호와 사칙연산만 가능하며 나누기 연산에서 나머지는 무시합니다.
* 최솟값이 8보다 크면 -1을 return 합니다.

이 문제의 원래 유형이 dp, 즉 동적 계획법을 이용하는 것이라고 해서 동적 계획법을 계속 고민해 봤는데, 아무리 생각해도 너무 어려웠다. 그래서 그냥 완전 탐색이나 모든 경우를 다 해보는 백트래킹을 DFS로 시도해 보았다.

뭔가 규칙성을 띄게 반복적인 작업을 연속적으로 수행해야 할 때는 DFS/BFS를 이용하자.

def **DFS**(count, value):

*#count는 전체 이용한 N의 개수 value는 numbers와 비교할 계산 값*

    global N, number, answer

    if value == number:

        answer = **min**(answer, count)

    if count == 8:

        return

    else:

        for i in **range**(1, 9-count):

*#8자리가 넘으면 탈락이기 때문에 range를 해당 길이를 넘기 전까지로 설정*

            DFS(count+i, value+int(str(N)\*i))

            DFS(count+i, value-int(str(N)\*i))

            DFS(count+i, value\*int(str(N)\*i))

            if value%int(str(N)\*i) == 0:

                DFS(count+i, value/int(str(N)\*i))

이 문제의 원래 의도인 dynamic programming으로 다시 풀어본다면

**#네트워크(BFS)**

사실 처음에는 DFS문제 인줄 알고 DFS로 풀려고 했었다. 그러나 네크워크는 연결만 되어 있으면 되는 것이므로 깊이를 탐색 할 필요가 없었던 것이다. 그래서 BFS로 바꾸어서 풀었다. BFS문제를 푸는 일반적인 루틴은

1. 방문여부를 판단하는 리스트, 부모 노드를 넣고 뺄 큐 생성
2. 탐색을 시작할 노드를 큐에 넣기+시작 노드의 방문 여부 변경
3. 큐가 빌 때까지 반복문 수행
4. 큐의 앞에서부터 노드 하나씩 pop(0)
5. 꺼낸 노드에 인접한 노드들을 방문하는 반복문 수행
6. 이전에 방문한 적이 없다면 큐에 노드 넣기
7. 방문 여부 변경

이 문제는 여기서 더 나아가서 computer이라는 리스트에서 연결 여부도 확인해야 하며, 큐가 빌 때까지 반복문을 한번만 돌리면 네트워크를 하나만 검색하게 되는 꼴이다. 그래서 모든 노드를 방문할 때까지 반복해서 네트워크를 찾아야 한다. 그렇기 때문에 while 0 in ch이후에 dQ.append(0)를 바꾸지 않으면 계속 같은 노드를 탐색 할 것이므로 계속 답이 나오지 않았다. 이는 아직 방문하지 않은 노드를 방문하기 시작해야 하므로 x = ch.index(0)로 해서 방문 안 한 index를 찾아준다.

def **solution**(n, computers):

*#방문 여부를 판단해주는 리스트*

    ch = [0]\*n

*#네트워크의 개수를 측정해주는 변수*

    dis = 0

*#이동중인 노드들을 넣어줌(부모 노드를 저장해두는 리스트)*

    dQ = []

*#모든 노드를 방문할 때까지*

    while 0 in ch:

*#아직 방문하지 않은 노드를 찾아간다*

        x = ch.index(0)

        dQ.append(x)

        ch[x] = 1

*#순차적으로 하나의 네트워크를 완성시킨다*

        while dQ:

            now = dQ.pop(0)

*#computer[now][i] == 1이라는 사실보다 방문했느냐가 더 중요하다*

*#for i in range(now+1, n)으로 하면 복잡하게 연결된 것들은 해결할 수 없으며, 계속 오류가 났던 이유이다.*

            for i in **range**(n):

*#아직 방문하지 않았고 부모 노드와 연결되어 있을 때*

                if ch[i] == 0 and computers[now][i] == 1:

*#i를 다음 부모 노드로 설정한다*

                    dQ.append(i)

                    ch[i] = 1

*#네트워크 변수의 개수에 1을 더해준다*

        dis += 1

    return(dis)

DFS와 BFS중 어떤 방법을 사용해야 풀리는 문제인지 알아보기 위해..일단 둘 다 이진트리를 이용한다. 그러나 DFS의 경우 트리의 노드들의 깊이가 매우 중요한 요소이다. 계속 이어 나가면서 재귀적으로 함수를 호출 해야 하기 때문이다.

반면, BFS의 경우에는 깊이가 아닌 그냥 연결만 되어 있으면 되는 것이므로 1차원 자료구조인 큐에 값을 넣어서 노드끼리의 연결 여부, 즉 부모 노드-자녀 노드의 관계만 파악 해 주면 된다. 그러기 위해서는 큐가 빈 리스트 일 때까지 탐색을 진행하게 된다.

몇 ‘번째’인지가 중요한 요소라면, 즉 index를 매번 갱신 해 주는 것이 중요한 상황이라면, 그리고 모든 경우를 다 계속 탐색 하는 것이 중요하다면, 그리고 최대/최소 등의 상황이라면 DFS이용하자.

**#백준 (미로 찾기)**

N×M크기의 배열로 표현되는 미로가 있다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 |

미로에서 1은 이동할 수 있는 칸을 나타내고, 0은 이동할 수 없는 칸을 나타낸다. 이러한 미로가 주어졌을 때, (1, 1)에서 출발하여 (N, M)의 위치로 이동할 때 지나야 하는 최소의 칸 수를 구하는 프로그램을 작성하시오. 한 칸에서 다른 칸으로 이동할 때, 서로 인접한 칸으로만 이동할 수 있다.

위의 예에서는 15칸을 지나야 (N, M)의 위치로 이동할 수 있다. 칸을 셀 때에는 시작 위치와 도착 위치도 포함한다.

이 문제는 DFS, BFS 두가지 방법 모두로 푸는 것이 가능하다. 다만 DFS를 이용했을 때는 시간 초과가 발생 했고, BFS를 이용했을 때에는 정답으로 처리가 되었다. 그래서 아래에는 두가지 코드를 모두 적어보고자 한다.

이렇게 풀기 전에 입력 값을 리스트, 즉 원하는 자료 구조의 형태로 바꾸는데에 시간이 오래걸렸었는데, 이는

100111001

이런 문자열을 [1,0,1,1,1,0,0,1]

이런 리스트의 형태로 바꾸는 것이었다.

n,m = **map**(int, **input**().split())

file = [[0]\*m for x in **range**(n)]

for i in **range**(n):

    a = **input**()

    for j in **range**(m):

        file[i][j] = (int(a[j]))

이런 식으로 하는게 제일 맞는 방법인 것 같다.

그리고 함수 진행을 멈추고 싶을 때에 사용하는 sys.exit(0)는 사용하기 전 import sys로 sys모듈을 미리 호출해야 한다.

*#BFS*

from collections import deque

dQ = [(0,0)]

ch = [[0]\*m for x in **range**(n)]

ch[0][0] = 1

dx, dy = [-1,1,0,0],[0,0,-1,1]

while dQ:

    now = dQ.pop(0)

    if now[0] == n-1 and now[1] == m-1:

        break

    for i in **range**(4):

        a,b = now[0]+dx[i], now[1]+dy[i]

        if 0<=a<n and 0<=b<m and file[a][b] == 1 and ch[a][b] == 0:

            file[a][b] = file[now[0]][now[1]] + 1

            ch[a][b] = 1

            dQ.append((a,b))

print(file[-1][-1])

*#DFS*

ans = 500000

dx, dy = [-1,1,0,0],[0,0,-1,1]

def **DFS**(x,y,count):

    global ans

    if x == n-1 and y == m-1:

        ans = **min**(ans, count)

        return

        sys.exit()

    else:

        for i in **range**(4):

            a,b = x+dx[i], y+dy[i]

            if 0 <= a < n and 0<=b<m and file[a][b] == 1:

                file[a][b] = 0

                DFS(a,b,count+1)

                file[a][b] = 1

file[0][0] = 0

DFS(0,0,1)

print(ans)

**#3차원 토마토(백준 7569)**

이 문제는 두가지를 얻을 수 있었다. 일단 차원만 추가된 BFS문제 연습이었기 때문에 BFS를 구현하는 과정에서 어려움이 있었던 것은 전혀 아니었다. 그러나 3차원 리스트의 형태를 입력 받는 과정이 어려웠다.

[]\*10 이나 [] for x in range(10) 이렇게 하면 아무것도 안되고 겉을 또 리스트로 묶어 주어야 리스트가 된다.

그리고 마지막에 변화한 file 리스트를 이용해서 답을 도출하는 과정에서는 아마도 -1로만 이루어진 file이 문제가 되었을 것같다. 그럴 경우에는 0을 출력해야 하는데 -1이 출력 되었을 것이기 때문이다.

import sys

from collections import deque

m,n,h = **map**(int, **input**().split())

dQ = deque()

file = [[[]for x in **range**(n)]for x in **range**(h)]

for i in **range**(h):

    for x in **range**(n):

        file[i][x] = list(**map**(int, sys.stdin.readline().split()))

        for y in **range**(m):

            if file[i][x][y] == 1:

                dQ.append((i,x,y))

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

while dQ:

    a,b,c = dQ.popleft()

    for i in **range**(6):

        x,y,z = b+dx[i], c+dy[i], a+dz[i]

        if 0 <= x<n and 0<=y<m and 0<= z<h and file[z][x][y] == 0:

            file[z][x][y] = file[a][b][c] + 1

            dQ.append((z,x,y))

def **solve**():

    ans = 0

    for i in **range**(h):

        for j in **range**(n):

            if 0 in file[i][j]:

                return -1

            ans = **max**(ans, **max**(file[i][j])-1)

    return ans

print(solve())

**#숨바꼭질(백준 1697)**

문제

수빈이는 동생과 숨바꼭질을 하고 있다. 수빈이는 현재 점 N(0 ≤ N ≤ 100,000)에 있고, 동생은 점 K(0 ≤ K ≤ 100,000)에 있다. 수빈이는 걷거나 순간이동을 할 수 있다. 만약, 수빈이의 위치가 X일 때 걷는다면 1초 후에 X-1 또는 X+1로 이동하게 된다. 순간이동을 하는 경우에는 1초 후에 2\*X의 위치로 이동하게 된다.

수빈이와 동생의 위치가 주어졌을 때, 수빈이가 동생을 찾을 수 있는 가장 빠른 시간이 몇 초 후인지 구하는 프로그램을 작성하시오.

DFS와 BFS로 풀어 보려고 했었는데 DFS의 경우에는 답이 절대 나올 수 없었고 아무 메모이제이션 없이 BFS로 풀 때에는 시간 초과가 발생했었고 결국에는 일종의 메모이제이션을 이용한 백트래킹+BFS를 이용해서 문제를 풀게 되었다.

그런데 내가 계속 헷갈렸던 점은 ch리스트, 즉 가지치기를 위해 해당 지점을 지나갔는지 안갔는지 확인하는 것을 다시 ch의 값을 0으로 바꿔 놓아야 한다고 생각했었다. 그런데 그게 아니었고, 직접 그림을 그려보면 더 명확히 알 수 있었다. 각 노드들의 ‘높이’가 ch리스트에 저장이 되는 것이었고, 따라서 최단 거리면 가장 먼저 도착할 테니 break를 붙여 주면 훨씬 시간에 있어서 효율적일 것이다. Ch의 값이 0인 경우만 지나가는 이유는 그렇지 않으면 당연히 최단 거리는 될 수 없기 때문이다.

ch = [0]\*100001

ch[n] = 1

dQ.append(n)

while dQ:

    now = dQ.popleft()

    if now == k:

        print(ch[now]-1)

    for j in (now+1, now-1, now\*2):

        if 0<= j < 100001 and (ch[j] == 0 or ch[now] + 1 < ch[j]):

            ch[j] = ch[now]+1

            dQ.append(j)

**#벽 부수고 이동하기(백준 2206)**

문제

N×M의 행렬로 표현되는 맵이 있다. 맵에서 0은 이동할 수 있는 곳을 나타내고, 1은 이동할 수 없는 벽이 있는 곳을 나타낸다. 당신은 (1, 1)에서 (N, M)의 위치까지 이동하려 하는데, 이때 최단 경로로 이동하려 한다. 최단경로는 맵에서 가장 적은 개수의 칸을 지나는 경로를 말하는데, 이때 시작하는 칸과 끝나는 칸도 포함해서 센다.

만약에 이동하는 도중에 한 개의 벽을 부수고 이동하는 것이 좀 더 경로가 짧아진다면, 벽을 한 개 까지 부수고 이동하여도 된다.

맵이 주어졌을 때, 최단 경로를 구해 내는 프로그램을 작성하시오.

처음에는 단순하게 한번만 BFS를 이용하면 답이 나오는 단순한 문제인 줄 알았다. 그러나 그렇게 하게 되면 벽이 있는 지점에서는 어쩔 수 없이 벽을 반드시 쳐서, 벽을 부수지 않는 상황과의 대소 비교를 할 수 없게 된다. 따라서 코드를 바꿔주었는데, 마지막에 도착 불가하면 -1을 출력해야 하기 때문에 이럴 때 함수의 return 기능이 편하기에 함수로 BFS를 정의해 주었다.

내가 항상 다루기 어려워하는게 리스트의 구조인데, 3차원 리스트를 만드는데 애를 먹었다.

[[0,0],[0,0]]이런 형태를 만들려면 [[0,0]\*2] 가 아니라 [[[0]\*2]for x in range(2)]이런 식으로 해야 한다.

자료구조 오류 해결이 끝난 상황에서는 큐와 방문 여부를 확인하는 vist()리스트를 잘 고려해 주어야 하는데, 일단은 dQ에 넣을 값은 [a,b,c]인데 a,b는 좌표이고 c는 벽의 부셔짐 여부이다. 그리고 visit리스트는 3차원 리스트인데 각각의 [i,j]로 입력이 되어 있다면 i는 해당 노드/죄표에서 벽을 뚫어야 할 때 기록 저장, j는 뚫지 않는 경우에의 기록 저장이다.

따라서 두 경우, 뚫고 가는 경우와 안 뚫고 가는 경우 모두를 탐색 할 수 있지만, 결국에 더 먼저 도착하는 경우가 최단 거리이므로 그 값을 return하게 된다.

file = []

for i in **range**(n):

    file.append(list(**map**(int, list(**input**().strip()))))

dx,dy = [-1,1,0,0],[0,0,-1,1]

visit = [[[0]\*2 for i in **range**(m)]for x in **range**(n)]

visit[0][0][1] = 1

dQ = deque()

dQ.append([0,0,1])

def **BFS**():

    while dQ:

        a,b,c = dQ.popleft()

        if a == n-1 and b == m-1:

            return visit[a][b][c]

        for i in **range**(4):

            x,y = a+dx[i], b+dy[i]

            if 0<= x<n and 0<=y<m:

                if file[x][y] == 0 and visit[x][y][c] == 0:

                    visit[x][y][c] = visit[a][b][c]+1

                    dQ.append([x,y,c])

                elif file[x][y] == 1 and c == 1:

                    visit[x][y][0] = visit[a][b][1] + 1

                    dQ.append([x,y,0])

    return -1

**#구슬 탈출 2 (BOJ 13460)**

스타트링크에서 판매하는 어린이용 장난감 중에서 가장 인기가 많은 제품은 구슬 탈출이다. 구슬 탈출은 직사각형 보드에 빨간 구슬과 파란 구슬을 하나씩 넣은 다음, 빨간 구슬을 구멍을 통해 빼내는 게임이다.

보드의 세로 크기는 N, 가로 크기는 M이고, 편의상 1×1크기의 칸으로 나누어져 있다. 가장 바깥 행과 열은 모두 막혀져 있고, 보드에는 구멍이 하나 있다. 이때, 구슬을 손으로 건드릴 수는 없고, 중력을 이용해서 이리 저리 굴려야 한다. 왼쪽으로 기울이기, 오른쪽으로 기울이기, 위쪽으로 기울이기, 아래쪽으로 기울이기와 같은 네 가지 동작이 가능하다. 각각의 동작에서 공은 동시에 움직인다. 빨간 구슬이 구멍에 빠지면 성공이고 기울이는 동작을 그만하는 것은 더 이상 구슬이 움직이지 않을 때 까지이다. 보드의 상태가 주어졌을 때, 최소 몇 번 만에 빨간 구슬을 구멍을 통해 빼낼 수 있는지 구하는 프로그램을 작성하시오.

이 문제는 브루트 포스 알고리즘과 BFS를 따른다. 모든 경우를 규칙을 찾아서 시도해보되, BFS의 규칙을 따라 while 문을 이용하여 너비 우선 탐색을 진행하는 것이다. 일단 문재 해결의 실마리가 되는 규칙들을 정리해 보자면

1. 파란 구슬과 빨간구슬이 일직선이 아닐 때
2. 빨간 구슬의 위치에 구멍 -> 탈출
3. 파란 구슬의 위치에 구멍 -> 실패
4. 어디에도 구멍 없으면 이동할 수 있는데까지 이동 후 큐에 push
5. 빨간 구슬이 앞설 때
6. 구멍 0 -> 실패
7. 구멍 X -> 제일 멀리 갈 수 있는 만큼 이동 후 큐에 push
8. 파란 구슬이 앞설 때
9. 파란 구슬 뒤에 구멍 -> 실패
10. 파란 구슬 앞에 구멍 -> 탈출
11. 구멍X -> 빨간 구슬이 뒤로 한 칸 이후push

이렇게 해결하려는데 처음에는 큐를 만드는데에도 빨간 구슬, 파란 구슬 따로 만들었었기 때문에 while문을 사용함에 있어서 불편했고, 방문 확인 리스트인 visit도 겨유 3차원 리스트로 만들어서 마찬가지로 확인하기 어려웠다.

이 문제에서 두 구슬을 독립적인 요소로 나누어서 문제가 생겼는데, 둘은 동시에 움직이는 것이기 때문에 한번 이동한 후의 상황, 즉 두 구슬 모두의 좌표를 index로서 방문 여부를 저장 할 수 있어야 했다. 그래서 4차원 리스트를 만들어서 true와 false로 표현 해 주었다.

물론 이전에 틀린 코드에서도 #이 보일 때까지 이동한 좌표를 알려주는 함수를 만들었었지만, 고려 안한 점은 O,즉 구멍에 도달 할 때 까지라는 것도 고려해야 했기 때문이다.

제일 애 먹었던 것은 만약 두 좌표의 이동 후 index를 구할 때에 좌표가 완ㄷ전히 겹치는 상황에서의 대처인데, 더 먼저 나아간 것을 찾아야 했는데, 각각의 x,y좌표를 모두 나누며 구하기에는 복잡했다. 결국에는 이동거리를 이용해 비교를 하였는데, 더 이동 거리가 긴 구슬이 뒤에 있는 것이므로 뒤에 있던 구슬의 좌표를 한칸 물러나게 한다. 그리고 여기서 x,y구분의 필요가 없는건 dx,dy의 for문 안에서 구하는 것이기 때문이다.

이후에는 방문 여부확인, 갱신, 그리고 큐에 넣었고 만약에 파란구슬은 구멍이 아니고 빨간 구슬이 구멍이라면 depth를 출력 해 주었다.

from collections import deque

dQ = deque()

lastx, lasty = 0,0

n,m = **map**(int, **input**().split())

file = []

for x in **range**(n):

    file.append(list(**map**(str, **input**().strip())))

    for j in **range**(m):

        if file[x][j] == 'R':

            a,b = x,j

        elif file[x][j] == 'B':

            c,d = x,j

        elif file[x][j] == 'O':

            lastx, lasty = x,j

visited = [[[[False] \* m for \_ in **range**(n)] for \_ in **range**(m)] for \_ in **range**(n)]

dQ.append((a,b,c,d,1))

visited[a][b][c][d] = True

dx,dy = [-1,1,0,0], [0,0,-1,1]

def **move**(x,y,i):

    count = 0

    while file[x+dx[i]][y+dy[i]] != '#' and file[x][y] != 'O':

        x,y = x+dx[i], y+dy[i]

        count += 1

    return x,y,count

def **BFS**():

    while dQ:

        a,b,c,d,depth = dQ.popleft()

        if depth > 10:

            break

        for i in **range**(4):

            aa,bb,Rcount = move(a,b,i)

            cc,dd,Bcount = move(c,d,i)

            if file[cc][dd] != 'O':

                if file[aa][bb] == 'O':

                    return depth

                if aa == cc and bb == dd:

                    if Rcount > Bcount:

                        aa -= dx[i]

                        bb -= dy[i]

                    else:

                        cc -= dx[i]

                        dd -= dy[i]

                if not visited[aa][bb][cc][dd]:

                    visited[aa][bb][cc][dd] = True

                    dQ.append((aa,bb,cc,dd,depth+1))

    return -1

print(BFS())

동적 알고리즘으로 풀 때에는 우선 냅색 알고리즘을 생각하자. 그리고 전화식 구하는 것에 초점을 맞출 것.

**#백준 10942 팰린드롬**

맞기 전까지는 계속해서 시간 초과가 발생했었는데, 단순히 input()를 sys.stdin.readline()으로 바꿔주었기 때문인지 while 문을 for이중문으로 바꿨기 떄문인지는 모르겠다. 그러나 지금까지의 경험으로 미루어 볼 때 for문의 시간 복잡도가 더 작은 것은 맞는 것으로 보인다.

일단 원래 문제의 유형이던 dynamic programing으로 풀기 전에는 file이라는 n\*n의 2차원 리스트에 file[i][j]가 num[i]와 num[j]가 같은지 다른지의 여부를 넣어서 만약 한부분이라도 다르다면 팰린드롬이 아니기 때문에 break를 하는 방식을 썼지만, 당연하게도 시간 초과가 발생했다.

그리고 마지막에 dp를 이용해서 관찰 부분의 거리에 따라서 해결 했는데, dp[i][j]가 num[i] == num[j] 이고 dp[i+1][j-1] == 1일 때 1 이 되도록 한 것이다. 그렇게 하니까 시간 초과가 발생하지 않았고, 특히 거리를 늘리는 부분을 for문을 사용했기 때문이라고 생각한다.

dp = [[0]\*n for x in **range**(n)]

for i in **range**(n):

    dp[i][i] = 1

for i in **range**(n-1):

    if num[i] == num[i+1]:

        dp[i][i+1] = 1

for count in rane(2, n):

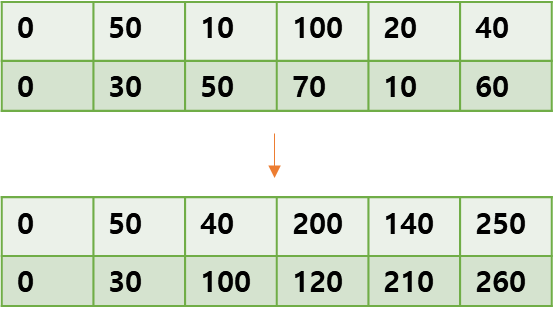
    for i in **range**(n-count):

        if num[i] == num[i+count] and dp[i+1][i-1+count] == 1:

            dp[i][i+count] = 1

**#백준 9465 스티커**

이 문제를 풀 때 처음에는 그냥 대각성으로 지그재그로 이동하면서 더해 두 값 중 최댓값을 출력하면 되는 것 인줄 알았다. 그런데 다시 문제를 읽어 보니 무조건 대각선에 바로 있는 것을 더할 필요가 없고 한 칸 건너 뛸 수도 있었다.

이런 식으로 처음에 입력 받는 file 을 그냥 dp리스트로서 이용하고

Dp[i][j] += max( dp[1-i][j-1], dp[1-i][j-2])

를 이용해서 한줄 띌 때와 그냥 연속으로 대각선에 있는 것을 더할 때중에 더 큰 값을 정했다.

그리고 처음에는 for문의 점화식에 j-2가 들어갔기 때문에 일반적인 1부터 불가능 했기에 원래의 리스트의 앞에 0을 각각 넣어서 열을 하나 더 만들었다.

for i in **range**(2):

        dp.append(list(**map**(int, **input**().split())))

        dp[i].insert(0,0)

    for j in **range**(2, n+1):

        for i in **range**(2):

            dp[i][j] += **max**(dp[1-i][j-2], dp[1-i][j-1])

    print(**max**(dp[0][-1], dp[1][-1]))

코드중에 for문 두개의 순서를 바꾸었었기 때문에 제대로 답이 안 나왔었다. 무조건 x좌표를 먼저 정하는 것이 아님을 명심하자.

**#JadenCase 문자열 만들기**

일단 이 문제를 통해서 알페벳이 아닌 문자에 .upper()이나 .lower()을 적용해도 오류가 나지 않는다는 것을 발견했다.

그리고 이 문제를 처음에 풀었을 때는 정확도 검사에서 43밖에 안나왔는데, 처음부터 .split()를 해서 공백을 없애주었기 때문에 문자 사이에 연속된 공백이 있을 경우 결과값을 그대로 출력해 내지 못하기 떄문이다.

그래서 문자열의 index까지 모두 고려해 가면서 구해야 하는데, 이때 이용하는 것이 enumerate함수이다.

고려할 것은 첫번째 문자열의 경우 .upper(), 해당 문자열 이전에 공백이면 .upper(), 그리고 나머지는 그냥 .lower()로 공백까지도 추가해 준다.

‘’.join()함수가 다시 헷갈렸는데,

만약 어떤 문자열에 이어서 값을 join하는 거라면 ()안에 ([문자열 이름, 추가하려는 값]) 이렇게 하고 아니면 그냥(추가하려는 값) 으로 하면 된다.

굳이 코드를 첨부할 필요는 없어 보인다.

**#프렌즈 4 블록**

블라인드 공채를 통과한 신입 사원 라이언은 신규 게임 개발 업무를 맡게 되었다. 이번에 출시할 게임 제목은 프렌즈4블록.  
같은 모양의 카카오프렌즈 블록이 2×2 형태로 4개가 붙어있을 경우 사라지면서 점수를 얻는 게임이다.

만약 판이 위와 같이 주어질 경우, 라이언이 2×2로 배치된 7개 블록과 콘이 2×2로 배치된 4개 블록이 지워진다. 같은 블록은 여러 2×2에 포함될 수 있으며, 지워지는 조건에 만족하는 2×2 모양이 여러 개 있다면 한꺼번에 지워진다.

블록이 지워진 후에 위에 있는 블록이 아래로 떨어져 빈 공간을 채우게 된다.

만약 빈 공간을 채운 후에 다시 2×2 형태로 같은 모양의 블록이 모이면 다시 지워지고 떨어지고를 반복하게 된다.  
입력으로 주어진 판 정보를 가지고 몇 개의 블록이 지워질지 출력하라.

이 문제를 풀 때 당연히 제일 먼저 든 생각은 함수 3개 설정이었다. 게임판에서 연속으로 정사각형이 되는 모양 찾기, 칸 밀어서 빈 칸 매꿔주기, 그리고 solution()함수에서 정사각형을 만드는 좌표를 이용해서 그 부분을 공백으로 만들어 주는 것이었다.

일단 연속으로 이어지는 형태를 찾는 findsquare(m,n,board)에서는 오른쪽, 아래, 대각선 아래 이 세방향과 같으며 공백이 아닌 것을 만드는 좌표를 append한 list를 출력하게 된다.

이후에 solution함수에서 그 좌표들에 공백을 만들어 주었으며, 이후에 공백을 매우기 위해 밀어주는 작업을 진행하는 함수를 만들었다. 이게 제일 어려웠는데, 처음에는 for i in range(n), for j in range(m)이런 식으로 공백이 안나올떄까지 하나씩 밀어주려했지만, 몇번을 반복해야하는지 명확하지 않은 상태에서는 불가능 했다.

그래서 m\*n의 board를 n\*m의 board로 바꿔준 뒤에 첫번째 줄부터 여백을 없앤 상태로 문자열로 만들어서 .ljust(m)이라는 내장 함수를 이용해서 문자열의 길이가 m이 될수 있게 여백을 만들어 오른쪽으로 문자들을 민 것이다. 이 상태에서 다시 n\*m이었던 board를 m\*n의 board로 바꿔 주면 밀린 상태의 board가 완성 될 수 있었다.

Cf. 리스트 안의 요소들이 문자열일 때 이를 각각 다른 요소로 바꿔서 리스트로 만들기 위해

이 두가지 모두 가능하다.

A = list(**map**(lambda x : list(x), board))

B = []

for i in **range**(**len**(board)):

    B.append(list(**map**(str, board[i].strip())))

def **change**(m,n,board):

    newboard = []

    newstr = ''

    for i in **range**(n):

        for j in **range**(m):

            newstr += board[j][i]

        newboard.append(newstr)

        newstr = ''

    board = []

    for i in newboard:

        i = i.replace(' ', '')

        i = i.rjust(m)

        board.append(i)

    newboard = []

    newstr = []

    for i in **range**(m):

        for j in **range**(n):

            newstr.append(board[j][i])

        newboard.append(newstr)

 newstr = []