백준 문제 풀이 - 단계별로 풀어보기 2

상태	완료 🙌
작성일시	@2023년 6월 5일 오후 4:37
유형	**기본 알고리즘 문제들과 재귀, 백트래킹+dfs, 동적계획법(dp), 누적합, 그리디 알고리즘에 대한 백준 문제풀이 노트. (ver. 파이썬)**
텍스트	백준을 풀어보자!! https://www.acmicpc.net/user/yuyunjae03
dp	0-1 배낭 문제, 가장 긴 증가 수열 문제(LIS), 연쇄 행렬 곱셈 문제, 그래프 상의 최단거리 문제(밸먼-포드 알고리즘, 플로이드-워셜 알고리즘, 다익스트라 알고리즘)

백준 10870번 피보나치 수 5 https://www.acmicpc.net/problem/10870
매우 쉬운 문제다. 첫 번째는 그냥 재귀로 풀었고, 두 번째는 dp를 활용해 연산의 속도를 더 빠르게 높였다. dynamic programming에 대해 더 공부해보자.

```
#my code 44m/s
from sys import stdin
```

```
def fibonacci(x: int):
    if x <= 1:
        return x
    else:
        return fibonacci(x-1) + fibonacci(x - 2)


n = int(stdin.readline().rstrip())
print(fibonacci(n))
```

```
#my other code 40m/s
#dynamic programming
from sys import stdin
n = int(stdin.readline().rstrip())
fibo = [0] * (n+1)
if len(fibo) > 1:
    fibo[1] = 1


def fibonacci(x: int):
    if x <= 1:
        return x
    if not fibo[x]:
        fibo[x] = fibonacci(x - 1) + fibonacci(x - 2)
    return fibo[x]


print(fibonacci(n))
```

백준 24060번 알고리즘 수업 - 병합 정렬 1 https://www.acmicpc.net/problem/24060
문제를 푸는데 너무 오래걸렸다. merge_sort 개념과 global(전역변수)를 이용하는 방법에 대해 다시 공부해봐야할 것 같다.

```
#my code 1820m/s pypy-> 604m/s
from sys import stdin
```

```
def merge_sort(li: list):
    length = len(li)
    if length <= 1:
        return li
    group1 = merge_sort(li[:(length+1)//2])
    group2 = merge_sort(li[(length+1)//2:])
    return merge(group1, group2)


def merge(group1, group2):
    global count
    global K
    result = list()
    ans = index1 = index2 = 0
    while index1 < len(group1) and index2 < len(group2):
        if group1[index1] < group2[index2]:
            result.append(group1[index1])
            index1 += 1
            count += 1
            if count == K:
                ans = result[-1]
        else:
            result.append(group2[index2])
            index2 += 1
            count += 1
            if count == K:
                ans = result[-1]
    while index1 < len(group1):
        result.append(group1[index1])
        index1 += 1
        count += 1
        if count == K:
            ans = result[-1]
    while index2 < len(group2):
        result.append(group2[index2])
        index2 += 1
```

```
        count += 1
        if count == K:
            ans = result[-1]
    if ans:
        print(ans)
        exit(0)
    return result


count = 0
N, K = map(int, stdin.readline().split())
A = list(map(int, stdin.readline().split()))
merge_sort(A)
if count < K:
    print(-1)
```

```
#other good code pypy-> 496m/s
import sys
input = sys.stdin.readline

def merge_sort(List, left, right):
  if(left < right and cnt <= k):
    mid = (left + right) // 2
    merge_sort(List, left, mid)
    merge_sort(List, mid+1, right)
    merge(List, left, mid, right)

def merge(List, left, mid, right):
  global cnt, res
  i, j = left, mid+1
  tmp = []

  while i <= mid and j <= right:
    if(List[i] <= List[j]):
      tmp.append(List[i])
      i += 1
    else:
      tmp.append(List[j])
```

```
      j += 1

  while i <= mid:
    tmp.append(List[i])
    i += 1

  while j <= right:
    tmp.append(List[j])
    j += 1

  i, t = left, 0
  while i <= right:
    List[i] = tmp[t]
    cnt += 1
    if cnt == k:
      res = List[i]
      break;
    i += 1
    t += 1

n, k = map(int, input().split())
List = list(map(int, input().split()))
cnt = 0
res = -1
merge_sort(List, 0, n-1)
print(res)
```

백준 4779번 칸토어 집합 https://www.acmicpc.net/problem/4779
첫 번째는 재귀를 쓰지 않고 풀었고, 두 번째는 재귀를 이용해 풀었다. 마지막은 다른 분의 코드인데 bottom-up 방식으로 멋있게 푸셨다.!

```
#my code 788m/s
from sys import stdin

while True:
    try:
        N = int(stdin.readline().rstrip())
```

```
        li_count = 3 ** N
        cantor_set = ['-'] * li_count
        for i in range(N):
            point = 0
            for j in range(3 ** i):
                bl = 3 ** (N - i - 1)
                point += bl
                cantor_set[point:point + 3 ** (N - i - 1)] =
                point += 2 * bl

        print(*cantor_set, sep='')
    except (EOFError, ValueError):
        break
```

```
#my other code 420m/s
from sys import stdin


def cantor_set(li: list, x: int):
    if len(li) <= 1:
        return li
    split3 = len(li)//3
    return cantor_set(li[:split3], x-1) + [' ']*split3 + cant


while True:
    try:
        N = int(stdin.readline().rstrip())
        li_count = 3 ** N
        cant_li = ['-'] * li_count
        print(*cantor_set(cant_li, N), sep='')
    except (EOFError, ValueError):
        break
```

```
#other good code 40m/s
def cantor(i,s,n):
    if i == n:
```

```
        print(s)
    else:
        cantor(i + 1, s + ' ' * len(s) + s,n)

while True:
    try:
        n=int(input())
        cantor(0,'-',n)
    except EOFError:
        break
```

백준 11729번 하노이 탑 이동 순서 https://www.acmicpc.net/problem/11729
마지막 코드는 dynamic programming의 방식으로, cache에 반복되는 구간을 저장해두어 매우 빠르게 처리했다… 진짜 이해하기 쉽지 않은 코드인 것 같다. dp에 대해 더 공부해야 할 것 같다.

```
#m code 1460m/s
from sys import stdin


def hanoi(fr: int, to: int, mid: int, obj: int):
    ans = list()
    if obj == 1:
        return [fr, to]
    ans += (hanoi(fr, mid, to, obj-1))
    ans += [fr, to]
    ans += (hanoi(mid, to, fr, obj-1))
    return ans


N = int(stdin.readline().rstrip())
a = hanoi(1, 3, 2, N)
print(len(a)//2)
for i in range(0, len(a), 2):
    print(*a[i:i+2])
```

```
#my other code 864m/s
from sys import stdin


def hanoi(fr: int, to: int, mid: int, obj: int):
    if obj == 1:
        print(fr, to)
    else:
        hanoi(fr, mid, to, obj-1)
        print(fr, to)
        hanoi(mid, to, fr, obj-1)


N = int(stdin.readline().rstrip())
print(2**N-1)
hanoi(1, 3, 2, N)
```

```
#other good code 76m/s
cache = {}


def hanoi(num, start, end):
    if num == 1:
        return f'{start} {end}\n'
    if (num, start, end) in cache:
        return cache[(num, start, end)]
    p = 6 - start - end
    cur = [hanoi(num-1, start, p), f'{start} {end}\n', hanoi(
    cache[(num, start, end)] = ''.join(cur)
    return cache[(num, start, end)]


n = int(input())
print(2 ** n - 1)
print(hanoi(n, 1, 3))
```

백준 2447번 별찍기 - 10 https://www.acmicpc.net/problem/2447
프랙탈 구조에서 bottom-up 방식이 매우 빠르고 유용하다는 것을 윗 문제와 이번 문제를 통해 알 수 있었다.
풀이 방식은 비슷하나 시간차이가 수십배가 난다… list 뿐 만 아니라 문자열도 파이썬에서는 곱셈이 가능하다는 것을 생각하지 못했다. 문자열을 한글짜 한글짜 따로 분리하지않고, 이를 문자열로 쭉 더했으면 아래코드처럼 간단하게 나왔을 것 같다.

```
#my code 1392m/s
from sys import stdin


def point_star(i: int, li: list, n: int):
    if i == n:
        return li
    ps_li = list()
    for j in range(i):
        ps_li.append(li[j]*3)
    for j in range(i):
        ps_li.append(li[j] + [' ']*i + li[j])
    for j in range(i):
        ps_li.append(li[j]*3)
    return point_star(i*3, ps_li, n)


N = int(stdin.readline().rstrip())
first = [['*', '*', '*'], ['*', ' ', '*'], ['*', '*', '*']]
first = point_star(3, first, N)
for k in range(N):
    print(*first[k], sep='')
```

```
#other good code 48m/s
n = int(input())

def star(n):
  if n==3:
    return ['***','* *','***']
  arr=star(n//3)
  stars = []
```

```
  for i in arr:
    stars.append(i*3)
  for i in arr:
    stars.append(i+" "*(n//3)+i)
  for i in arr:
    stars.append(i*3)
  return stars

print('\n'.join(star(n)))
```

백준 15649번 N과 M (1) https://www.acmicpc.net/problem/15649

https://docs.python.org/ko/3.8/library/itertools.html → itertool 모듈(확통)에 대한 설명

백트래킹을 이용해서 순열을 구하는 문제였다. 다른 분의 코드는 permutations(순열)과 combinations(조합)이 있는 itertools 모듈을 사용해서 풀었다. 속도가 매우 빠른 것을 봐서는 효율적으로 코딩이 되어있는 것 같다.

```
#my code 204m/s
from sys import stdin


    def back_tracking(n: int, m: int, ans: list):
        if len(ans) == m:
            print(*ans)
            return 0
        for i in range(1, n+1):
            if i not in ans:
                ans.append(i)
                back_tracking(n, m, ans)
                ans.pop()


    N, M = map(int, stdin.readline().split())
    back_tracking(N, M, [])
```

```
#other good code 52m/s
from itertools import permutations

N, M = map(int, input().split())
nums = list(map(str, range(1, N + 1)))
combs = permutations(nums, M)
print('\n'.join(' '.join(comb) for comb in combs))
```

백준 15651번 N과 M (3) https://www.acmicpc.net/problem/15651

itertools 모듈에 순열과 조합만 있는 줄 알았는데, 중복순열(π)에 해당하는 product()함수도 있었다.
순열 : permutations(n, m) = nPm 순서 상관O, 중복X 백준 15649번
조합 : combinations(n, m) = nCm 순서 상관X, 중복X 백준 15650번
중복 순열 : product(n, repeat=m) = nπm 순서 상관O, 중복O 백준 15651번
중복 조합 : combinations_with_replacement(n, m) = nHm 순서 상관X, 중복O 백준 15652번
해당 4문제는 진짜 외워야 할 것 같다.!!!

n 은 해당문제에서 1~n까지 숫자가 들어있는 list고 m은 그중 몇 개의 수를 뽑을 것인지를 나타내는 int형이다.

```
#my code 2000m/s
from sys import stdin


def back_tracking(n: int, m: int, ans: list):
    if len(ans) == m:
        print(*ans)
        return 0
    for i in range(1, n + 1):
        ans.append(i)
        back_tracking(n, m, ans)
        ans.pop()
```

```
N, M = map(int, stdin.readline().split())
back_tracking(N, M, [])
```

```
#other good code 232m/s
import itertools
N,M = map(int,input().split())
list1 = list(map(str,range(1,N+1)))

print("\n".join(list(map(" ".join,itertools.product(list1,rep
```

백준 9663번 N-Queen https://www.acmicpc.net/problem/9663
제한 시간이 10초였고 어찌저찌 통과는 했지만, pypy3로도 30초가 넘는 시간이 걸렸다… 아무리 생각해도 이런 방식이 맞는 것 같아 gpt를 이용해 c++로 바꿔본 결과 제한시간의 절반인 5초 정도 나오는 것을 확인 할 수 있었다. 확실히 파이썬보다는 c++이 빠르긴 한 것 같다.. 변환하는 과정속에서 vector가 스택과 비슷한 알고리즘이라는 것을 알 수 있었다.
이 문제는 정말 4시간 이상의 시간을 쓴 것 같다. 바로 위에서 ***BACK_TRACKING***을 공부한 다음인데도 제한 시간 때문에 정말 애를 많이 먹었다. 특히 행렬에서 대각선을 처리할 때 어떻게 해야할 지 고민을 많이 했는데 ***abs(li[ele][0] - (li_l+1)) == abs(li[ele][1] - i)*** *이방법은 정말 획기적인 것 같다.* 대각선을 지날 경우 기울기가 1이거나 -1 이기 때문에, 이를 이용해서 정말 편하게 예외처리를 할 수 있었다.!!!

```
#my code 30000m/s
from sys import stdin
N = int(stdin.readline().rstrip())
count = 0


def back_tracking(li: list):
    global count
    li_l = len(li)
    if li_l == N:
        count += 1
        return 0
    for i in range(1, N+1):
        flag = True
        for ele in range(li_l):
```

```
            if li[ele][1] == i:
                flag = False
                break
            elif abs(li[ele][0] - (li_l+1)) == abs(li[ele][1]
                flag = False
                break
        if flag:
            li.append([li_l + 1, i])
            back_tracking(li)
            li.pop()


back_tracking([])
print(count)
# li[[column1, row1], [column2, row2], ...]
```

```
#other good code timeover(over 30s)
from sys import stdin

N = int(stdin.readline().rstrip())

result = 0
row = [0] * N


def is_place(x):
    for i in range(x):
        if row[x] == row[i] or abs(row[x] - row[i]) == abs(x
            return False
    return True


def n_queens(x):
    global result
    if x == N:
        result += 1
        return
    else:
```

```
        for n in range(N):
            row[x] = n
            if is_place(x):
                n_queens(x + 1)


n_queens(0)
print(result)
```

```
//내 알고리즘을 c++로 바꿨을 때: 50000m/s
#include <iostream>
#include <vector>
using namespace std;

int N;
int count = 0;

void back_tracking(vector<vector<int>>& li) {
    int li_l = li.size();
    if (li_l == N) {
        count++;
        return;
    }
    for (int i = 1; i <= N; i++) {
        bool flag = true;
        for (int ele = 0; ele < li_l; ele++) {
            if (li[ele][1] == i) {
                flag = false;
                break;
            }
            else if (abs(li[ele][0] - (li_l+1)) == abs(li[ele
                flag = false;
                break;
            }
        }
        if (flag) {
            li.push_back({li_l + 1, i});
            back_tracking(li);
```

```
            li.pop_back();
        }
    }
}

int main() {
    cin >> N;
    vector<vector<int>> li;
    back_tracking(li);
    cout << count << endl;
    return 0;
}
```

백준 2580번 스도쿠 https://www.acmicpc.net/problem/2580
백트래킹으로 풀긴 했으나 3000m/s가 나오면서 pypy3일 때는 통과. python3로 돌릴때는 시간초과가 발생했다. 내 코드의 문제는 li[row][column] != 0 일때도 재귀를 통해 탐색할 때마다 계속 비교연산을 실행하기 때문인 것 같다. 이때문에 백트래킹 할 때도 시간이 더 걸린다.
→ 이를 해결해봤는데 500m/s정도 줄었지만, 큰 차이는 없는 것같다… 역시 c++로 공부를 해야하려나….
빨리 푸신분들은 2진수와 비트연산으로 푸신 것 같다.

```
#my code 3100m/s
from sys import stdin
li = [list(map(int, stdin.readline().split())) for _ in range


def is_place(row: int, column: int, num: int):
    for j in range(9):
        if li[row][j] == num or li[j][column] == num:
            return False
    row_box = row // 3 * 3
    column_box = column // 3 * 3
    for i in range(3):
        for j in range(3):
            if li[row_box + i][column_box + j] == num:
                return False
```

```
    return True


def sudoku(n: int):
    if n >= 81:
        for i in range(9):
            print(' '.join(map(str, li[i])))
        exit(0)
    row = n // 9
    column = n % 9
    if li[row][column] != 0:
        sudoku(n+1)
    else:
        for i in range(10):
            if is_place(row, column, i):
                li[row][column] = i
                sudoku(n+1)
        li[row][column] = 0


sudoku(0)
```

```
#other good code 488m/s
import sys


def set_bit(i, j, n):
    m = 1 << n
    row[i] |= m
    col[j] |= m
    sqr[i//3*3 + j//3] |= m


def clear_bit(i, j, n):
    m = ~(1 << n)
    row[i] &= m
    col[j] &= m
    sqr[i//3*3 + j//3] &= m
```

```
def is_set(i, j, n):
    return (1 << n) & row[i] & col[j] & sqr[i//3*3 + j//3]


def sudoku(ep):
    if ep >= len(empty):
        print('\n'.join(' '.join(map(str, x)) for x in brd))
        sys.exit(0)

    x, y = empty[ep]
    for n in range(1, 10):
        if is_set(x, y, n):
            brd[x][y] = n
            clear_bit(x, y, n)
            sudoku(ep + 1)
            set_bit(x, y, n)


b = 0b1111111111
row, col, sqr = [b] * 9, [b] * 9, [b] * 9
brd, empty = [], []
for i in range(9):
    brd.append([int(x) for x in input().split()])
    for j in range(9):
        if brd[i][j]:
            clear_bit(i, j, brd[i][j])
        else:
            empty.append((i, j))

sudoku(0)
```

```
#my other code 2580m/s
from sys import stdin
li = list()
que = list()
for q in range(9):
```

```
    li.append(list(map(int, stdin.readline().split())))
    for p in range(9):
        if li[q][p] == 0:
            que.append((q, p))


def is_place(row: int, column: int, num: int):
    for j in range(9):
        if li[row][j] == num or li[j][column] == num:
            return False
    row_box = row // 3 * 3
    column_box = column // 3 * 3
    for i in range(3):
        for j in range(3):
            if li[row_box + i][column_box + j] == num:
                return False
    return True


def sudoku(n: int):
    if n == len(que):
        for i in range(9):
            print(' '.join(map(str, li[i])))
        exit(0)
    row = que[n][0]
    column = que[n][1]
    for i in range(10):
        if is_place(row, column, i):
            li[row][column] = i
            sudoku(n+1)
    li[row][column] = 0


sudoku(0)
```

백준 14888번 연산자 끼워넣기 https://www.acmicpc.net/problem/14888
아니 이정도나 차이난다고??? 내가 진짜 뭔가 잘못 생각하고 있는 것 같다. 실버1 문제를 푸

는데 1시간이 넘게 걸리기도 했고 (에러가 계속 떠서 대략 1시간 반걸림.. ㅋㅋ) 시간 차이가 비슷한 알고리즘이라고 나름 생각했는데 100배가 차이난다는 것은, 정말 어마어마한 수치다. 백트래킹과 dfs를 열심히 공부하고 있지만, 아직 진짜 갈길이 먼 것 같다….
def dfs(depth, total, plus, minus, multiply, divide): 이런 방식을 생각해낼 줄 알아야 할 것 같다..

```
#my code 5568m/s
from sys import stdin


N = int(stdin.readline().rstrip())
num = list(map(int, stdin.readline().split()))
operator_num = list(map(int, stdin.readline().split()))
operator = ['+']*operator_num[0] + ['-']*operator_num[1] + ['
check = [True] * (N-1)
boundary = 1000000000   #1billion
maxi = -boundary
mini = boundary


def back_tracking(count: int, li_sum: int):
    global maxi, mini
    if count == N-1:
        if maxi < li_sum:
            maxi = li_sum
        if mini > li_sum:
            mini = li_sum
        return 0
    else:
        for i in range(N-1):
            if check[i]:
                check[i] = False
                if operator[i] == '+':
                    back_tracking(count + 1, li_sum + num[cou
                elif operator[i] == '-':
                    back_tracking(count + 1, (li_sum - num[co
                elif operator[i] == '*':
                    back_tracking(count + 1, li_sum * num[cou
```

```
                elif operator[i] == '//':
                    back_tracking(count + 1, int(li_sum / num
                check[i] = True


back_tracking(0, num[0])
print(maxi, mini, sep='\n')
```

```
#other good code 68m/s
import sys

input = sys.stdin.readline
N = int(input())
num = list(map(int, input().split()))
op = list(map(int, input().split()))  # +, -, *, //

maximum = -1e9
minimum = 1e9


def dfs(depth, total, plus, minus, multiply, divide):
    global maximum, minimum
    if depth == N:
        maximum = max(total, maximum)
        minimum = min(total, minimum)
        return

    if plus:
        dfs(depth + 1, total + num[depth], plus - 1, minus, m
    if minus:
        dfs(depth + 1, total - num[depth], plus, minus - 1, m
    if multiply:
        dfs(depth + 1, total * num[depth], plus, minus, multi
    if divide:
        dfs(depth + 1, int(total / num[depth]), plus, minus,


dfs(1, num[0], op[0], op[1], op[2], op[3])
```

```
print(maximum)
print(minimum)
```

백준 14889번 스타트와 링크 https://www.acmicpc.net/problem/14889

순열, 조합이 필요한 경우 → 재귀를 이용한 back_tracking….
조합인데 이번에 순열로 풀어버리면서 시간초과가 계속 떴다… 이를 고치고, li팀에 0번 플레이어를 고정시켜서 경우의 수를 1/2로 줄였다.(어처피 li팀에 있든, st팀에 있는 점수의 차는 동일하게 나오기 때문 ex: li(0, 1, 2), st(3, 4, 5)의 점수 차이는 li(3, 4, 5), st(0, 1, 2)의 점수 차이와 같다.)
맨 아래 다른분의 코드는 각 맴버별로 S의 가로세로의 모든 값을 더하고 이를 combination을 통해 절반의 맴버의 가로세로 점수를 전체 S의 점수에서 빼면서 비교했다. (진짜 너무 어렵다…ㅋㅋ) 그냥 정석대로 풀어야겠다.

```
#my code 1128m/s
from sys import stdin
N = int(stdin.readline().rstrip())
S = [list(map(int, stdin.readline().split())) for _ in range(
check = [True]*N
ans = 1e9


def best_case():
    st = list()
    li = list()
    li_sum = st_sum = 0
    for i in range(N):
        if check[i]:
            st.append(i)
        else:
            li.append(i)
    for i in li:
        for j in li:
            li_sum += S[i][j]
    for i in st:
        for j in st:
            st_sum += S[i][j]
```

```
    return abs(li_sum - st_sum)


def back_tracking(n: int, j: int):
    global ans
    if n == N//2:
        ans = min(best_case(), ans)
        return ans
    for i in range(j, N):
        if check[i]:
            check[i] = False
            back_tracking(n+1, i+1)
            check[i] = True
    return ans


check[0] = False
print(back_tracking(1, 1))
```

```
#other good code 136m/s
import sys
import itertools
input = sys.stdin.readline

n = int(input())
s = [list(map(int, input().split())) for _ in range(n)]

s_row_sums = [sum(i) for i in s]
s_col_sums = [sum(i) for i in zip(*s)]
s_member_sums = [sum(i) for i in zip(s_row_sums, s_col_sums)]
s_sum = sum(sum(s, []))

team_scores = [abs(s_sum - sum(s_member_sum)) for s_member_su

print(min(team_scores))
```

백준 2961번 도영이가 만든 맛있는 음식 https://www.acmicpc.net/problem/2961
조합을 이용하면 쉽게 풀 수 있는 문제.. dfs, back_tracking연습을 위해 구현해서 풀었다.!

```
#my code 40m/s
from sys import stdin
N = int(stdin.readline().rstrip())
element = [list(map(int, stdin.readline().split())) for _ in
minimum = 1e9


def back_tracking(n: int, idx: int, sour: int, bitter: int):
    global minimum
    if n == N:
        return 0
    for i in range(idx, N):
        minimum = min(minimum, abs((sour * element[i][0] - (b
        back_tracking(n + 1, i + 1, sour * element[i][0], (bi


back_tracking(0, 0, 1, 0)
print(minimum)
```

```
#my other code 40m/s
from sys import stdin
N = int(stdin.readline().rstrip())
element = [list(map(int, stdin.readline().split())) for _ in
check = [True]*N
minimum = 1e9


def back_tracking(n: int, idx: int, sour: int, bitter: int):
    global minimum
    if n == N:
        return 0
    for i in range(idx, N):
        if check[i]:
            check[i] = False
            minimum = min(minimum, abs((sour * element[i][0]
```

```
            back_tracking(n+1, i, sour * element[i][0], (bitt
        check[i] = True


back_tracking(0, 0, 1, 0)
print(minimum)
```

백준 24416번 알고리즘 수업 - 피보나치 수1 https://www.acmicpc.net/problem/24416

dp알고리즘에 대해 알아보자!

```
#my code 40m/s
from sys import stdin
N = int(stdin.readline().rstrip())
f = [0] * (N+1)


def fibo(i: int):
    if i <= 1:
        f[i] = i
        return f[i]
    if f[i]:
        return f[i]
    else:
        f[i] = fibo(i-1) + fibo(i-2)
        return f[i]


print(fibo(N), N-2)
```

백준 9184번 신나는 함수 실행 https://www.acmicpc.net/problem/9184
dp로 푸는 문제다.
elif value[a][b][c]:
return value[a][b][c] 이부분을 계속 까먹고 있어서 시간이 너무 오래 소요되었다. 주석처리한 부분은 우연히 실행하다 알게된 사실인데, 이것이 실행시간을 조금 더 줄여주지 않을

까 생각해서 넣어도 보고 빼도 봤는데 동일하게 나왔다.. ㅋㅋ
밑의 코드도 거의 동일한 방식이다.

```
#my code 72m/s
from sys import stdin
value = [[[0] * 21 for _ in range(21)] for _ in range(21)]


def w(a: int, b: int, c: int):
    if a <= 0 or b <= 0 or c <= 0:
        return 1
    elif a > 20 or b > 20 or c > 20:
        return w(20, 20, 20)
    elif value[a][b][c]:
        return value[a][b][c]
    elif a < b < c:
        value[a][b][c] = w(a, b, c-1) + w(a, b-1, c-1) - w(a,
        return value[a][b][c]
    else:
        value[a][b][c] = w(a-1, b, c) + w(a-1, b-1, c) + w(a-
        return value[a][b][c]


# for i in range(1, 21):
#     value[i][i][i] = 2**i
while True:
    A, B, C = map(int, stdin.readline().split())
    if A == -1 and B == -1 and C == -1:
        break
    print(f'w({A}, {B}, {C}) =', w(A, B, C))
```

```
#other good code
import sys

l=list([[0]*21 for _ in range(21)]for _ in range(21))

def w(a, b, c):
    if a<=0 or b<=0 or c<=0: return 1
```

```
    elif a>20 or b>20 or c>20: return 1048576
    elif l[a][b][c]: return l[a][b][c]
    elif a<b and b<c:
        l[a][b][c]=w(a, b, c-1) + w(a, b-1, c-1) - w(a, b-1, 
        return l[a][b][c]
    else:
        l[a][b][c] = w(a-1, b, c) + w(a-1, b-1, c) + w(a-1, b
        return l[a][b][c]

while True:
    a, b, c= map(int, sys.stdin.readline().rstrip().split())
    if a==-1 and b==-1 and c==-1: break
    else: print('w({}, {}, {}) = {}'.format(a,b,c,w(a,b,c)))
```

백준 1912번 연속합 https://www.acmicpc.net/problem/1912
dp로 푸는 문제라고 했는데, 그렇게 풀고 싶지도 않았고, 그냥 풀 수 있을 것 같아서 내 방식대로 풀다보니, 계속 오류뜨고 정말 힘들었다.. 반복문 안에서 if elif로 했는데, 그러니까 maximum보다 더 큰 음수 (-3 < -1)가 들어왔을 때 value ≤ 0:이라는 조건문이 실행되지 않아서 -5 -4 -3 -2 -1 이 들어오는 문제에서 답이 -2로 나오는 문제가 있었다. 추가로 if value < 0: 조건문이 없으면 -1 42 74 -100 이런 입력이 들어오면 답이 116이 아닌 115가 나오기에 추가 했다..
아래가 정석 풀이다. ret가 n번까지의 최대값이 저장되고, dp에는 매 수를 더해가며 dp[i-1]이 음수가 되면 dp[i]가 arr[i]가 되고, 양수나 0이면 dp[i-1] + arr[i]가 된다.

```
#my code 76m/s
from sys import stdin
n = int(stdin.readline().rstrip())
n_li = list(map(int, stdin.readline().split()))
maximum = value = n_li[0]
if value < 0:
    value = 0

for i in range(1, n):
    value += n_li[i]
    if maximum < value:
        maximum = value
    if value <= 0:
```

```
        value = 0
print(maximum)
```

```
#other good code
#include <stdio.h>

#define MAX_ARR 100000
#define max(x,y) x > y ? x : y

int main() {
    int n;
    int arr[MAX_ARR];
    int dp[MAX_ARR];
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        scanf("%d ", &arr[i]);
    }

    int ret = arr[0];
    dp[0] = arr[0];

    for (int i = 1; i < n; ++i) {
        dp[i] = max(dp[i - 1] + arr[i], arr[i]);
        ret = max(dp[i], ret);
    }

    printf("%d\n", ret);

    return 0;
}
```

백준 1149번 RGB거리 https://www.acmicpc.net/problem/1149
실버1 문제인데도 벽을 느꼈다… 그냥 내가 dp문제를 너무 못푼다는 것을 알 수 있었다.. ㅠㅠ
첫 번째 코드는 내가 하다가 결국 틀린 코드고, 두 번째 코드는 다른분 코드를 조금 힌트를 얻고 다시 풀었다..

두 번째 코드를 꼭 보면서 dp복습을 하자!

```
#my code (wrong answer)
from sys import stdin
N = int(stdin.readline().rstrip())
a = [list(map(int, stdin.readline().split())) for _ in range(
dp_cost = [0]*N

dp_cost[0] = min(a[0])
dp_index = a[0].index(dp_cost[0])
second_minimum = 1000
for i in range(3):
    if dp_index != i:
        if second_minimum > a[0][i]:
            second_minimum = a[0][i]
second_choice = [second_minimum, a[0].index(second_minimum)]


for i in range(1, N):
    minimum = min(a[i])
    minimum_index = a[i].index(minimum)
    if minimum_index != dp_index:
        dp_cost[i] = dp_cost[i-1] + minimum
        sec_index = 3 - minimum_index - dp_index
        second_choice = [dp_cost[i-1]+a[i][sec_index], sec_in
        dp_index = minimum_index
    else:
        second_minimum = 1000
        second_index = 2
        for k in range(3):
            if dp_index != k:
                if second_minimum >= a[i][k]:
                    second_minimum = a[i][k]
                    second_index = k

        if second_choice[0] + minimum <= dp_cost[i-1] + secon
            dp_cost[i] = second_choice[0] + minimum
```

```
            dp_index = minimum_index
            second_choice = [dp_cost[i-1] + second_minimum, s
        else:
            dp_cost[i] = dp_cost[i-1] + second_minimum
            dp_index = second_index
            last_index = 3 - second_index - minimum_index
            if second_choice[0] + minimum <= dp_cost[i-1] + a
                second_choice = [second_choice[0] + minimum, 
            else:
                second_choice = [dp_cost[i-1] + a[i][last_ind
print(dp_cost[-1])
```

```
#my other code (Learned from other people's code) 44m/s
from sys import stdin
N = int(stdin.readline().rstrip())
a = [list(map(int, stdin.readline().split())) for _ in range(
dp = [[0]*3 for _ in range(N)]
dp[0] = a[0]
for i in range(1, N):
    dp[i][0] = min(dp[i-1][1], dp[i-1][2]) + a[i][0]
    dp[i][1] = min(dp[i-1][0], dp[i-1][2]) + a[i][1]
    dp[i][2] = min(dp[i-1][0], dp[i-1][1]) + a[i][2]
print(min(dp[-1]))
```

백준 1932번 정수 삼각형 https://www.acmicpc.net/problem/1932
dp문제 풀면서 정말 멘탈도 나가고 자신감도 떨어지고 힘들었는데, 이번 문제는 거의 예상했던 방향대로 딱 풀려서 좋았다..! 아래 코드는 내 코드에서 dp를 만들지 않고, triangle 배열만을 이용해서 풀었다. 어처피 dp = triangle[-1]도 얕은 복사기에 메모리 차이에 큰 변화는 없었을 것이다. (배열의 주솟값만 받은 셈)

```
#my code 112m/s
from sys import stdin
n = int(stdin.readline().rstrip())
triangle = [list(map(int, stdin.readline().split())) for _ in
dp = triangle[-1]
for i in range(n-1, 0, -1):
```

```
    for j in range(len(triangle[i]) - 1):
        dp[j] = max(dp[j], dp[j+1]) + triangle[i-1][j]
print(dp[0])
```

```
#other good code 92m/s
import sys


def solution():
    n = int(sys.stdin.readline())
    dp = [list(map(int, sys.stdin.readline().split())) for _ 

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

    print(dp[0][0])


solution()
```

백준 2579번 계단 오르기 https://www.acmicpc.net/problem/2579
무난하게 풀었다. 어려울 줄 알았는데, 생각보다 쉬웠다. 내 코드는 한 칸을 밟았으면 dp[i][0], 두 칸을 밟았으면 dp[i][1] 로 분리해서 구했고, 다른 사람의 코드는 3칸을 1칸, 2칸으로 오르냐 2칸, 1칸으로 오르냐를 나누어서 구했다.

```
#my code
from sys import stdin
n = int(stdin.readline().rstrip())
st_point = [int(stdin.readline().rstrip()) for _ in range(n)]
dp = [[0]*2 for _ in range(n)]
dp[0] = [st_point[0], st_point[0]]
if n > 1:
    dp[1][0] = dp[0][0] + st_point[1]
    dp[1][1] = st_point[1]
```

```
for i in range(2, n):
    dp[i][0] = dp[i-1][1] + st_point[i]    # 1칸 올랐을 때
    dp[i][1] = max(dp[i-2][0], dp[i-2][1]) + st_point[i]    #

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

```
#other good code
n = int(input())
s = [0 for i in range(301)]
dp = [0 for i in range(301)]
for i in range(n):
    s[i] = int(input())
dp[0] = s[0]
dp[1] = s[0] + s[1]
dp[2] = max(s[1] + s[2], s[0] + s[2])
for i in range(3, n):
    dp[i] = max(dp[i - 3] + s[i - 1] + s[i], dp[i - 2] + s[i]
print(dp[n - 1])
```

백준 1463번 1로 만들기 https://www.acmicpc.net/problem/1463
첨엔 그냥 3으로 나눠지면 3으로, 3으로 나눴을 때 나머지가 1이면 -1, 2로 나눠지면 2로, 그게 아니면 -1을 하도록 했는데, N = 16일때 4가 아닌 5가 나오면서 그냥 dp로 풀어야겠다고 생각했다. 나는 모든 경우를 다 확인해가는 코드인데, 이건 생각보다 재귀 깊이가 깊지 않아서 밑의 코드가 훨씬 효율적인 것 같다. dp는 정말 많이 풀어봐야겠다… 증명 불가능한 야매로는 풀지 말자…

```
#my code 716m/s
from sys import stdin
N = int(stdin.readline().rstrip())
dp = [0] * (N+1)
maximum = 10 ** 6
for i in range(2, N+1):
    a = [maximum] * 2
    if i % 3 == 0:
        a[0] = dp[i//3] + 1
    if i % 2 == 0:
```

```
        a[1] = dp[i//2] + 1
    dp[i] = min(a[0], a[1], dp[i-1] + 1)
print(dp[-1])
```

```
#other good code 36m/s
from sys import stdin

def reculsion(n):
    global dp

    if n in dp.keys():
        return dp[n]

    if n % 2 == 0 and n % 3 == 0:
        dp[n] = min(reculsion(n // 2), reculsion(n // 3)) + 1
    elif n % 3 == 0:
        dp[n] = min(reculsion(n // 3), reculsion(n - 1)) + 1
    elif n % 2 == 0:
        dp[n] = min(reculsion(n // 2), reculsion(n - 1)) + 1
    else:
        dp[n] = reculsion(n - 1) + 1

    return dp[n]


n = int(stdin.readline())
dp = {1 : 0, 2 : 1, 3 : 1}
print(reculsion(n))
```

백준 10844번 쉬운 계단 수 https://www.acmicpc.net/problem/10844
실버1 치고는 비교적 쉽고 빠르게 풀었다. 항상 코드짜기 전에 어떤식으로 해야 쉽게 풀릴지 구상을 하고 짜자!

```
#my code 40m/s
from sys import stdin
N = int(stdin.readline().rstrip())
```

```
a = [[0]*10 for _ in range(N+1)]
a[1] = [0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
for i in range(2, N+1):
    a[i][0] = a[i-1][1]
    a[i][9] = a[i-1][8]
    for j in range(1, 9):
        a[i][j] = (a[i-1][j-1] + a[i-1][j+1]) % int(1e9)
print(sum(a[-1]) % int(1e9))
```

```
#other good code 36m/s
n= int(input())
pd=[[0 for i in range(12)] for j in range(n)]
pd[0]=[0,0,1,1,1,1,1,1,1,1,1,0]

if n>=1:
    for i in range(1,n):
        for j in range(1,11):
            pd[i][j] = int(pd[i-1][j-1])+int(pd[i-1][j+1])

print(sum(pd[n-1])%1000000000)
```

백준 2156번 포도주 시식 https://www.acmicpc.net/problem/2156

백준 2579번 계단 오르기 https://www.acmicpc.net/problem/2579 문제와 거의 동일한 문제! 하지만 조금 다르다.. 처음에 비슷한 문제인 줄 알고 했다가 엄청 틀렸다.
6\n100\n100\n1\n1\n100\n100이 입력될 경우 400이 정답이지만, 2579번 처럼 풀면 301이 나와버린다. 안먹고 그냥 넘어가는 것도 포함시켜야 할 듯 싶다.
+ 내가 누적되는? dp를 잘 못쓰는 것 같다. 내가 사용하는 방식은 약간 dp[i]를 i 번째를 먹었을 때로 체크하는데, 다른 분들 코드를 보면, 먹었을 때 뿐만아니라, 전의 것을 먹고 이번을 안 먹었을 때도 고려해서, 누적해서 표기한다. 이런 부분에서 시간도 오래 걸리고 자주 틀리거나, 문제가 생기는 것 같다. dp관련 문제를 이번에 많이 풀었지만, 아직도 부족하다는 생각이 든다.

max(dp[i-1],dp[i-2]+wine[i],dp[i-3]+wine[i-1]+wine[i]) 이 부분을 좀더 이해해보자.

```
#my wrong code
from sys import stdin
N = int(stdin.readline().rstrip())
a = [int(stdin.readline().rstrip()) for _ in range(N)]
dp = [[0, 0, 0] for _ in range(N)]  # 1칸 전에 먹었을 때, 2칸 전어
dp[0] = [a[0], a[0], a[0]]
if N > 1:
    dp[1] = [a[0] + a[1], a[1], a[1]]
if N > 2:
    dp[2] = [a[1] + a[2], a[0] + a[2], a[2]]
for i in range(3, N):
    dp[i][0] = max(dp[i-1][1], dp[i-1][2]) + a[i]
    dp[i][1] = max(dp[i-2][0], dp[i-2][2]) + a[i]
    dp[i][2] = max(dp[i-3][0], dp[i-3][1]) + a[i]
if N < 2:
    print(max(dp[-1]))
else:
    print(max(max(dp[-1]), max(dp[-2])))
```

```
#my correct code 60m/s
from sys import stdin
N = int(stdin.readline().rstrip())
a = [int(stdin.readline().rstrip()) for _ in range(N)]
dp = [[0, 0, 0] for _ in range(N)]  # 1칸 전에 먹었을 때, 2칸 전어
dp[0] = [a[0], a[0], a[0]]
if N > 1:
    dp[1] = [a[0] + a[1], a[1], a[1]]
if N > 2:
    dp[2] = [a[1] + a[2], a[0] + a[2], a[2]]
for i in range(3, N):
    dp[i][0] = max(dp[i-1][1], dp[i-1][2]) + a[i]
    dp[i][1] = max(dp[i-2]) + a[i]
    dp[i][2] = max(dp[i-3]) + a[i]
if N < 2:
    print(max(dp[-1]))
```

```
else:
    print(max(max(dp[-1]), max(dp[-2])))
```

```
#other good code 44m/s
import sys
input = sys.stdin.readline

n = int(input())

wine = [0]
for i in range(n):
    wine.append(int(input()))
dp = [0]
dp.append(wine[1])

if n>1:
    dp.append(wine[1]+wine[2])

for i in range(3,n+1):
    dp.append(max(dp[i-1],dp[i-2]+wine[i],dp[i-3]+wine[i-1]+w
# dp[i-1]: 안먹고 지나갈 때, dp[i - 2] + wine[i - 1]: 2칸 전에 먹고
# dp[i - 3] + wine[i - 1] + wine[i]: 3칸 전에 먹고 1칸 전에 먹고 
print(dp[n])
```

백준 11053번 가장 긴 증가하는 부분 수열 https://www.acmicpc.net/problem/11053
dp는 정말 풀면 풀수록 바보가 되는 기분이다… 애초에 수열 A의 최대 길이가 1000이하 이므로 2번째 코드처럼 정말 간단하게 풀어도 됐었다… O(N^2) —>심지어 다른 분의 아이디어를 참고해서 풀음…
하지만, 시간복잡도를 좀 더 좋게 만들고 싶다면 마지막 코드처럼 이분탐색을 이용해 풀었으면 더 좋았을 것이다. 맨 아래 코드의 방식은 다음과 같다.
ex>input
8
1 1000 20 70 5 8 7 80
→ 1 → 1 1000 → 1 20 → 1 20 70 → 1 5 70 → 1 5 8 → 1 5 7→ 1 5 7 80
output: 4

ex>input
7
1 1000 20 70 7 5 80
→ 1 → 1 1000 → 1 20 → 1 20 70 → 1 7 70 → 1 5 70 → 1 5 70 80
output: 4
두 번째 예시를 보면 1 5 70 80 은 존재할 수 없는 수열이지만, 답은 올바르게 나온다. 이것이 나의 첫 번째 코드와 다른 점이다.. 입력값이 증가하는 부분 수열의 마지막 값(가장 큰값)보다 크면 계속 이어서 붙이고, 작으면 이분탐색으로 해당값보다 처음으로 큰 값이 나온 index의 값을 수정하여 계속 나열하는 방식이다.
LIS(Longest Increasing Subsequence) 에 대한 공부가 더 필요할 듯 싶다.

```
#my wrong code
# 16
# 277 730 790 994 242 185 633 545 830 557 194 994 44 28 755 6
# correct ans: 5
# my output: 4
from sys import stdin
N = int(stdin.readline().rstrip())
A = list(map(int, stdin.readline().split()))
dp = [A[0]]
recent = [A[0]] # recent와 dp로 나누어 가장 긴 값은 dp로 recent는 
for i in range(1, len(A)):
    if recent[-1] < A[i]:
        recent.append(A[i])
        if dp[-1] < A[i]:
            dp.append(A[i])
    else:
        if len(dp) <= len(recent):
            dp = recent[:]
        index = 0
        for j in range(len(recent)):    # 이분 탐색을 이용 했으면
            if recent[-j-1] < A[i]:
                index = len(recent) -j
                break
        recent = recent[:index]
        recent.append(A[i])
    print(dp, recent)
print(max(len(dp), len(recent)))
```

```
# recent를 감소하는 값이 나왔을 때, 그보다 큰 뒷 부분을 다 짤라 버리는게
# len(recent)는 변하지 않으므로 정말 획기적인 방법으로 구할 수 있었다. 
```

```
#my other code
from sys import stdin
N = int(stdin.readline().rstrip())
A = list(map(int, stdin.readline().split()))
dp = [1] * N
for i in range(N):
    for j in range(i):
        if A[j] < A[i]:
            dp[i] = max(dp[j] + 1, dp[i])
print(max(dp))
```

```
#other good code  best!!
import sys
input = sys.stdin.readline

n = int(input())
cases = list(map(int, input().split()))
lis = [0]

for case in cases:
    if lis[-1]<case:
        lis.append(case)
    else:
        left = 0
        right = len(lis)

        while left<right:
            mid = (right+left)//2
            if lis[mid]<case:
                left = mid+1
            else:
                right = mid
        lis[right] = case
```

```
print(len(lis)-1)
```

백준 11054번 가장 긴 바이토닉 부분 수열 https://www.acmicpc.net/problem/11054
이분 탐색을 사용하는 방식은 정말 생각이 나질 않아서, 윗 문제의 두 번째 코드를 응용해서 풀었다.
dp[i][0]은 i번째 요소가 증가하는 부분 수열일 때의 길이를 구했다. (윗 문제의 두 번째 코드)
dp[i][1]은 i번째 요소가 증가 후 감소하는 부분 수열일 때의 길이를 구했다.
dp[i][2]은 i번째 요소가 포함된 가장 긴 바이토닉 부분 수열의 길이를 구했다.

아래 방식은 bisect 라이브러리를 이용해 이분 탐색을 했고, 입력된 문자열을 뒤집어서 다시 증가하는 수열의 길이를 구하는 방식으로 i+1번째부터 감소하는 수열의 길이를 구했다. 이를 이용해, max((i번째 요소까지의 증가하는 부분 수열 길이) + (i+1)번째 요소부터 감소하는 부분 수열의 길이))로 구했다.
따라 코딩해보면서 이런 발상을 익혀야겠다.

```
#my code 248m/s
from sys import stdin
N = int(stdin.readline().rstrip())
A = list(map(int, stdin.readline().split()))
dp = [[1, 1, 0] for _ in range(N)]  # [증가 하는 부분 수열 길이, 증
for i in range(N):
    for j in range(i):
        if A[j] < A[i]:  # 증가 하는 부분 + 1
            dp[i][0] = max(dp[j][0]+1, dp[i][0])
        elif A[j] > A[i]:   # 증가 -> 감소시작  or 증가 -> 감소에서
            dp[i][1] = max(dp[j][2]+1, dp[i][1])
    dp[i][2] = max(dp[i][0], dp[i][1])
print(max(dp, key=lambda x: x[2])[2])
```

```
#other good code 44m/s
import bisect

n = int(input())
nums = list(map(int,input().split()))
dp = [0]*(n)
```

```
dp2 = [0]

for i in range(n):
    if nums[i] > dp2[-1]:
        dp2.append(nums[i])
        dp[i] = len(dp2)-1
    else:
        dp[i] = bisect.bisect_left(dp2,nums[i])
        dp2[dp[i]] = nums[i]

dp3 = [0]*(n)
nums=nums[::-1]
dp2 = [0]

for i in range(n):
    if nums[i] > dp2[-1]:
        dp2.append(nums[i])
        dp3[i] = len(dp2)-1
    else:
        dp3[i] = bisect.bisect_left(dp2,nums[i])
        dp2[dp3[i]] = nums[i]

print(max([dp[i]+dp3[::-1][i] for i in range(n)])-1)
```

```
#my other code 48m/s
from sys import stdin
N = int(stdin.readline().rstrip())
A = list(map(int, stdin.readline().split()))
dp1 = [0] * N
inc_a = [0]


def bisect_left(arr: list, num: int):
    left = 0
    right = len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
```

```
        if arr[mid] < num:
            left = mid + 1
        else:
            right = mid - 1
    return left


# def bisect_right(arr: list, num: int):
#     left = 0
#     right = len(arr) - 1
#     while left <= right:
#         mid = (left + right) // 2
#         if arr[mid] > num:
#             right = mid - 1
#         else:
#             left = mid + 1
#     return right


def check(dp: list, in_de: list):
    for i in range(N):
        if A[i] > in_de[-1]:
            in_de.append(A[i])
        elif A[i] < in_de[-1]:
            in_de[bisect_left(in_de, A[i])] = A[i]
        dp[i] = len(in_de) - 1


check(dp1, inc_a)
A = A[::-1]
inc_b = [0]
dp2 = [0] * N
check(dp2, inc_b)
ans = 0
for k in range(N):
    if ans < (dp1[k] + dp2[-k-1]) - 1:  # 겹치는 부분을 제외 해야
        ans = dp1[k] + dp2[-k-1] - 1
print(ans)
```

백준 2565번 전깃줄 https://www.acmicpc.net/problem/2565
이 문제에 거의 6시간 이상을 고민해왔지만 정말 모르겠어서 다른 분들의 코드를 참고해봤다. 그런데 이게
가장 긴 증가하는 부분 수열LIS와 관련있다는 건 알고 있었지만, 어떻게 구현할지 도저히 방도를 몰랐다. 처음에는 그리디 알고리즘으로 가장 많이 겹치는 전깃줄을 자르는 방식으로 했으나, 이런 방식으로는 구할수 없다는 것을 알게 되었다. 그런데, 전깃줄을 입력 방식만 순서대로 입력되도록 정렬해서보니, 전깃줄이 도착하는 부분인 B의 숫자를 **LIS**처럼 풀면 해결된다는 것을 알았다. **(잘라야 하는 전깃줄) = (총 전깃줄) - (도착부분 숫자의 LIS길이)**

진짜 이걸 문제를 보자마자 생각해내는 사람들은 뭐하는 사람들인지 싶다..ㄷㄷ

```
#my code 40m/s lis로 해결
from sys import stdin

c = int(stdin.readline().rstrip())
line = [list(map(int, stdin.readline().split())) for _ in ran
line.sort(key=lambda x: x[0])
ans = [0]
for i in range(c):
    if ans[-1] < line[i][1]:
        ans.append(line[i][1])
    else:
        left = 0
        right = len(ans) - 1
        while left < right:
            mid = (left + right) // 2
            if ans[mid] < line[i][1]:
                left = mid + 1
            else:
                right = mid
        ans[right] = line[i][1]
print(c - len(ans) + 1)
```

```
#other good code 36m/s dp로 해결 -> https://hongcoding.tistory.
# 전깃줄
import sys
```

```
input = sys.stdin.readline

n = int(input())
lines = []
dp = [1] * n

for _ in range(n):
    lines.append(list(map(int, input().split())))
lines.sort(key=lambda x: x[0])

for i in range(n):
    for j in range(i):
        if lines[i][1] > lines[j][1]:
            dp[i] = max(dp[i], dp[j]+1)

print(n-max(dp))
```

백준 9251번 LCS https://www.acmicpc.net/problem/9251
진짜 라파신 이후 처음으로 푸는 문제이기도 했고, 파이썬과 dp개념을 많이 까먹은 것 같다. ㅋㅋ…
문제 이해를 너무 못한다..
처음에는 누적해서 풀면 될 줄 알았는데, 첫 코드에 작성해둔 예시처럼 저런 경우에는 계속 숫자를 늘려버린다.
진짜 고민을 많이했는데, 결국 사이트를 참고했고, pos이라는 변수로 지금까지의 최대값을 기록해둬서 만약 같은 경우 pos +1로 dp에 저장했다. 저기서 if 와 elif의 조건문 순서를 바꿔버리면 틀린다. 왜냐하면, 최대값이 갱신된 경우는 같든 틀리든 이전에 이미 체크하고 지나갔다는 것이기 때문(부분수열의 앞부분에 이미 포함되어 있다). 또한, 최대값을 먼저 갱신해줘야 그 이후에 같은 경우가 나오면, 최대값 + 1을 해서 dp에 기록을 해야하기 때문이다.

https://myjamong.tistory.com/317 → 푸는 방법에 대한 사이트이다.

```
#my wrong code
#ACAA
#AAACBA
#대입시 답은 3이나 4가 나온다..
from sys import stdin
```

```
a = stdin.readline().rstrip()
b = stdin.readline().rstrip()
dp = [0] * len(b)
ans = 0

for k in range(len(a)):
    pos = 0
    for i in range(len(b)):
        if a[k] == b[i] and dp[i] < i + 1:
            dp[i] = dp[i] + 1
        elif i >= 1:
            dp[i] = max(dp[i], dp[i - 1])
    ans = max(dp)
    #print(dp)
print(ans)
```

```
#other good code
from sys import stdin

a = stdin.readline().rstrip()
b = stdin.readline().rstrip()
dp = [0] * len(b)

for k in range(len(a)):
    pos = 0
    for i in range(len(b)):
        if pos < dp[i]:
            pos = dp[i]
        elif a[k] == b[i]:
            dp[i] = pos + 1
    #print(dp)
print(max(dp))
```

백준 11659번 구간 합 구하기4 https://www.acmicpc.net/problem/11659
기초 문제여서 그런지 쉽다. 조건을 보면 입력이 들어올 때마다 구간합을 구하는 것보다 누적

합을 구해놓고 해당하는 구간이 입력되면, dp[j] - dp[i - 1]을 하는 것이 훨씬 빠르다는 것을 알 수 있다.

```
#my code 264m/s
from sys import stdin

N, M = map(int, stdin.readline().split())
num = list(map(int, stdin.readline().split()))
dp = [0] * (N + 1)
for i in range(1, N + 1):
    dp[i] = dp[i - 1] + num[i - 1]
#print(dp)
for _ in range(M):
    i, j = map(int, stdin.readline().split())
    print(dp[j] - dp[i - 1])
```

```
#other good code 200m/s 비슷한 방법
rom sys import stdin, stdout
input = stdin.readline
n, m = map(int, input().split())
ls = list(map(int, input().split()))
accum_ls = [0]
accum = 0
for i in ls:
    accum_ls.append(accum+i)
    accum += i
for _ in range(m):
    a, b = map(int, input().split())
    stdout.write(str(accum_ls[b] - accum_ls[a-1]) + "\n")
```

백준 2559번 수열 https://www.acmicpc.net/problem/2559
윗문제와 비슷한 방법으로 해결했다. 누적합을 이용하지 않으면, 조금 시간이 걸릴것 같다. c의 경우, int로 하면 오버플로우가 날것 같았는데, 다른 답들을 보니 괜찮은 것 같다.
밑의 코드는 정말 신박한 방법으로 구했다. 구간 크기인 k만큼 구간 합을 구한 다음, 앞부분 하나를 빼고 뒷부분에 하나를 더하면서 구간 합을 구했다. 해당 방식을 이용하면 내 코드에서 dp만큼의 메모리 공간과 dp를 구하는 만큼의 시간을 아낄 수 있다.

```
#my code 116m/s
from sys import stdin

N, M = map(int, stdin.readline().split())
temp = list(map(int, stdin.readline().split()))
dp = [0] * (N + 1)
for i in range(1, N + 1):
    dp[i] = dp[i - 1] + temp[i - 1]
ans = [0] * (N - M + 1)
for i in range(N - M + 1):
    ans[i] = dp[i + M] - dp[i]
#print(dp, ans)
print(max(ans))
```

```
#other good code 68m/s
import sys

def accum(n, k, temperatures):
    max_sum = sum(temperatures[:k])
    temp = max_sum
    for i in range(n - k):
        temp = temp - temperatures[i] + temperatures[k + i]
        if temp > max_sum:
            max_sum = temp
    return max_sum


N, K = map(int, sys.stdin.readline().split())
temps = list(map(int, sys.stdin.readline().split()))

print(accum(N, K, temps))
```

백준 16139번 인간-컴퓨터 상호작용 https://www.acmicpc.net/problem/16139
ord를 이용해 아스키코드값의 차이로 인덱스에 접근했다. c는 문자끼리 사칙연산이 가능하지만, 파이썬은 불가능하기 때문. 100점이긴하지만, s가 매우 길어지면, 시간이 오래 걸리긴

한다.
dp[len(s)][26]이 아닌 dp[26][len(s)]를 했다면 시간이 더 짧게 걸렸을 것 같기도 하다.

```
#my code 868m/s ..list로 구현. ord를 이용해 아스키코드값으로 인덱스를
from sys import stdin

s = stdin.readline().rstrip()
dp = [[0] * 26]
for i in range(len(s)):
    li = dp[-1][:]
    li[ord(s[i]) - 97] += 1
    dp.append(li)
#print(dp)
count = int(stdin.readline().rstrip())
for _ in range(count):
    a, b, c = map(str, stdin.readline().split())
    print(dp[int(c) + 1][ord(a) - 97] - dp[int(b)][ord(a) - 9
```

```
#other good code ..dictionary로 구현
import sys
input = sys.stdin.readline

s = input().strip()
n = int(input())
dic = {}
dic[-1] = {}
for i in range(97,123):
    dic[-1][chr(i)] = 0

for i in range(len(s)):
    dic[i] = dic[i-1].copy()
    dic[i][s[i]] += 1

for i in range(n):
    alpha, start, end = input().split()

    if start == 0:
        print(dic[int(end)][alpha])
```

```
    else:
        print(dic[int(end)][alpha] - dic[int(start)-1][alpha]
```

백준 10986번 나머지 합 https://www.acmicpc.net/problem/10986
너무 복잡하게 생각했다. 누적합을 이용한 다음, m으로 나누었을때 나머지가 같은 경우를 찾아 (해당 나머지의 경우의 수)C2(조합)로 계산하면 되는 것이었다. 단, 나머지가 0이 되는 경우는 누적합이 그 자체만으로도 m으로 나누어지므로 그 경우만 한 번 더 더했다.

```
#my code 624m/s
from sys import stdin

n, m = map(int, stdin.readline().split())
num = list(map(int, stdin.readline().split()))
ans = 0
dp = [0] * m
sum_num = 0
for i in range(n):
    sum_num = (sum_num + num[i]) % m
    if sum_num == 0:
        ans += 1
    dp[sum_num] += 1
for i in dp:
    ans += int(i * (i - 1) / 2)
print(ans)
```

```
#other good code 392m/s
input = __import__("sys").stdin.readline
N, M = map(int, input().split())
sums, remains = 0, [0] * M
for i, n in enumerate(input().split()):
    sums = (sums + int(n)) % M
    remains[sums] += 1
ans = remains[0]
for r in remains:
    ans += r * (r - 1) // 2
print(ans)
```

백준 11660번 구간 합 구하기 5 https://www.acmicpc.net/problem/11660
난이도는 실버1인데 이걸 바로 풀지 못했다는 것이 너무 아쉽다. 단순히 구간합을 이용만 했지, 이를 응용해서 푸는 것을 못해 시간초과가 나왔다.
어떻게 하면, 직사각형 부분의 구간합을 구할까 고민하다가 결국 다른 분의 아이디어를 참고하게 되었는데, 구간합을 각 행별로 구한 후, 그 이후에 각 열별로 구간합을 하면, 딱 직사각형 부분까지만 구간합이 구해진다는 것을 알 수 있었다. 역시 문제를 많이 풀어봐야 하는 것 같다.

```
#my wrong code timeover..
from sys import stdin

n, m = map(int, stdin.readline().split())
map_sum = [[0] * (n + 1)]
for _ in range(n):
    li = list(map(int, stdin.readline().split()))
    li[0] += map_sum[-1][-1]
    for i in range(1, n):
        li[i] += li[i - 1]
    map_sum.append(map_sum[-1][-1:] + li)

for i in range(m):
    x1, y1, x2, y2 = map(int, stdin.readline().split())
    ans = 0
    for j in range(x1, x2 + 1):
        ans += map_sum[j][y2] - map_sum[j][y1 - 1]
    print(ans)
```

```
#my good code pypy3(444m/s) python3(1000m/s)
from sys import stdin

n, m = map(int, stdin.readline().split())
map_sum = [[0] * (n + 1)]
for _ in range(n):
    li = list(map(int, stdin.readline().split()))
    for i in range(1, n):
        li[i] += li[i - 1]
```

```
    map_sum.append([0] + li)

for i in range(1, n + 1):
    for j in range(1, n + 1):
        map_sum[j][i] += map_sum[j - 1][i]

for i in range(m):
    x1, y1, x2, y2 = map(int, stdin.readline().split())
    ans = map_sum[x2][y2] - map_sum[x1 - 1][y2] - map_sum[x2]
    print(ans)
```

백준 25682번 체스판 다시 칠하기 2 https://www.acmicpc.net/problem/25682
오랫만에 1트에 통과했다. 이중반복문을 자주 사용하기도 했고, 리스트도 많이 할당해서 메모리는 좀 많이 쓰기는 했다.
윗 문제에서 이중 반복문을 두번 사용해서 직사각형 영역의 구간합을 구했는데, 이번 문제의 다른 분의 풀이를 보면, 이중 반복문을 한번만 활용해서 깔끔하게 같은 결과를 도출해 낸것 같다. 이런 부분은 배워야 할 것 같다. 또한, 같은 시간복잡도라도 어떤 코드가 더 빠른지 파악하는 능력을 조금 더 길러야 할 것 같다.

```
#my code pypy3(620m/s)
from sys import stdin

n, m, k = map(int, stdin.readline().split())
map_w = [[0] * m for _ in range(n)] #top이 w일 때
ans = [[0] * m for _ in range(n)]
max = [[0] * m for _ in range(n)]

for i in range(n):
    li = stdin.readline().rstrip()
    for j in range(len(li)):
        if (i + j) % 2 == 0:
            if li[j] == 'W':
                map_w[i][j] = 0
            else:
                map_w[i][j] = 1
        else:
            if li[j] == 'B':
```

```
                map_w[i][j] = 0
            else:
                map_w[i][j] = 1

for i in range(n):
    max[i][k - 1] = sum(map_w[i][:k])
    for j in range(k, m):
       max[i][j] += max[i][j - 1] + map_w[i][j] - map_w[i][j
#print(*map_w,'',*max, sep='\n')
#print()
for i in range(m):
    for j in range(k):
        ans[k - 1][i] += max[j][i]
    for j in range(k, n):
        ans[j][i] += ans[j - 1][i] + max[j][i] - max[j - k][i
mini = k * k
maxi = 0
for i in range(k - 1, n):
    for j in range(k - 1, m):
        if ans[i][j] < mini:
            mini = ans[i][j]
        if ans[i][j] > maxi:
            maxi = ans[i][j]
#print(*ans, sep='\n')
print(min(mini, k * k - maxi))
```

```
#other good code pypy3(388m/s)
from sys import stdin

input = stdin.readline

n, m, k = map(int, input().split())
board = [input().strip() for _ in range(n)]
my_sum = [[0] * (m + 1) for _ in range(n + 1)]
default = ['W', 'B']
for i in range(n):
    for j in range(m):
        ex = int(default[(i + j) % 2] == board[i][j])
```

```
        my_sum[i][j] = my_sum[i - 1][j] + my_sum[i][j - 1] - 
ans = k * k
for i in range(n - k + 1):
    for j in range(m - k + 1):
        temp = my_sum[i + k - 1][j + k - 1] + my_sum[i - 1][j
        ans = min(ans, temp, k * k - temp)
print(ans)
```

백준 1931번 회의실 배정 https://www.acmicpc.net/problem/1931
처음에 냅색 배낭문제처럼 접근을 해봤는데, 시간이 2^31 - 1까지 인지라 리스트 자체가 터져버렸다. 그래서 고민하다가 정말 단순하게 회의가 끝나는 시간 > 회의가 시작하는 시간으로 리스트를 정렬한 후, 풀었더니 풀렸다.. ㅋㅋ 문제를 빠르게 파악하는 능력이 매우 중요한 것 같다.
c++에서는 vector를 list처럼 사용했고, algorithm 라이브러리의 sort를 이용해 동일한 방식으로 해결했다.

```
#my code 272m/s
from sys import stdin

n = int(stdin.readline().rstrip())
meeting = [list(map(int, stdin.readline().split())) for _ in
meeting.sort(key=lambda x: (x[1], x[0]))
ans = 1
m_time = meeting[0][1]
#print(meeting)
for i in range(1, n):
    if m_time <= meeting[i][0]:
        ans += 1
        m_time = meeting[i][1]
print(ans)
```

```
//other good code 80m/s
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
```

```
int main()
{
    int n;
    cin >> n;
    vector<pair<int, int>> list;

    for (int i = 0; i < n; i++)
    {
        int start, end;
        cin >> end >> start;
        list.push_back(make_pair(start, end));
    }
    sort(list.begin(), list.end());

    int time = list[0].first;
    int ans = 1;
    int j = 0;
    for (int i = 1; i <= n; i++)
    {
        if (list[i].second >= time)
        {
            ans++;
            time = list[i].first;
        }
    }
    cout << ans;
}
```

백준 1541번 잃어버린 괄호 https://www.acmicpc.net/problem/1541
처음에 문제를 잘못읽어서 그냥 + - 를 넣어서 계산만 하면 되는 것인 줄 알았다. 뒤늦게 깨달음을 얻어서 문제를 해결 할 수 있었다. 사실 쭉 문장을 보고 '-'가 없으면 '+'로 split한 후, 다 더하면 해결되고, '-'가 있다면 처음 '-'가 나오는 부분을 기준으로 앞부분은 모두 더하고, 뒷부분부터는 모든 숫자를 다 빼주면 쉽게 구할 수 있다.
ex) 20 + 10 + 3 - 10 + 30 - 40 - 10 + 20 = (20+10+3) - 10 - 30 - 40 - 10 - 20

```
#my code 44m/s
from sys import stdin

cal_str = stdin.readline().rstrip()
count = 0
for i in cal_str:
    if i == '-':
        count += 1

li = cal_str.split('-')
ans_li = list()
for i in li:
    num = list(map(int, (i.split('+'))))
    ans_li.append(sum(num))
ans = 2 * ans_li[0]
for i in ans_li:
    ans -= i
print(ans)
```

```
//other good code
#include <cstdio>

int main() {
    bool negative = false;
    char ch;
    int sum, next;

    // input
    scanf("%d", &sum);
    while(true) {
        scanf("%c", &ch);
        scanf("%d", &next);

        if(ch == '-') negative = true;
        else if(ch != '+') break;

        if(negative) sum -= next;
```

```
        else sum += next;
    }

    // output
    printf("%d", sum);

    return 0;
}
```

백준 13305번 주유소 https://www.acmicpc.net/problem/13305
확실히 그리디 알고리즘이나 쉬운 실버문제는 파이썬으로 해결해도 실행시간 차이가 작기 때문에, 그냥 파이썬으로 풀어도 좋을 것 같다.

```
#my code 120m/s
from sys import stdin

n = int(stdin.readline().rstrip())
length = list(map(int, stdin.readline().split()))
oil = list(map(int, stdin.readline().split()))

ans = 0
oil_price = oil[0]
go_len = length[0]
for i in range(1, n - 1):
    if oil_price < oil[i]:
        go_len += length[i]
    else:
        ans += oil_price * go_len
        oil_price = oil[i]
        go_len = length[i]
ans += oil_price * go_len
print(ans)
```

```
//other good code 20m/s
#include <iostream>
using namespace std;
```

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

    int n, i;
    long long t = 0, j, k = 1000000001;
    cin >> n;
    int fn[--n];
    i = n;
    while (i--)
        cin >> fn[i];

    while (n--) {
        cin >> j;
        k = min(k, j);
        t += k * fn[n];
    } cin >> n;

    cout << t;
}
```