알튜비튜
브루트포스

오늘은 '무식하게 풀기' 라고도 불리는 브루트 포스 알고리즘에 대해 배웁니다.
때로는 가장 쉬운 길이 정답일 때가 있죠.

이런 문제가 있다면

#알튜비튜

"4자리 수의 암호로 된 자물쇠를 풀어보자."

-> 0000 ~ 9999 까지 모두 해보면 된다.

완전탐색

#알튜비튜

브루트 포스

- 단순히 가능한 모든 경우를 찾는 기법
- 테스트 케이스의 크기에 따라 소요 시간이 엄청나게 커질 수 있음
- 하지만, 떠올리기 가장 쉬운 방법이므로 문제를 풀 때 가장 먼저 고려해야 하는 방법이기도 함
- 입력 범위와 시간복잡도를 잘 고려하여 선택하는 것이 중요

브루트 포스 특징

#알튜비튜

"100,000,000"

- 총 연산수가 약 1억 회 이하인 경우, 충분히 브루트 포스로 접근 가능
- 연산수 = 모든 경우의 수 * 각 경우의 연산 횟수
- 실제와는 거리가 있지만, 대략적인 브루트 포스 가능 기준으로 잡으면 유용함

-> 반복문, 비트마스크, 순열, 재귀함수 등으로 접근

#알튜비튜

2231번 : 분해합 – Bronze 2

문제

- n의 분해합 = n과 n을 이루는 각 자리수의 합
- n의 생성자 = 분해합이 n인 어떤 자연수 m
- (예) 245의 분해합 = 245+2+4+5 = 256 -> 245는 256의 생성자
- n의 가장 작은 생성자 구하는 문제

제한 사항

- n의 범위는 1 <= n <= 1,000,000

-> n이 최대 1,000,000 이므로 분해합을 구하는 연산 횟수의 최대는 대략
1,000,000 = 1,000,000 + 1 + 0 + 0 + 0 + 0 + 0 + 0 = 7 번

#알튜비튜

2231번 : 분해합 – Bronze 2

문제

- n의 분해합 = n과 n을 이루는 각 자리수의 합
- n의 생성자 = 분해합이 n인 어떤 자연수 m
- (예) 245의 분해합 = 245+2+4+5 = 256 -> 245는 256의 생성자
- n의 가장 작은 생성자 구하는 문제

제한 사항

- n의 범위는 1 <= n <= 1,000,000

-> n까지 돌리며 분해합을 모두 계산해봐도, 최대 1,000,000 * 7 < 1억이므로
브루트 포스 가능!

예제

#알튜비튜

예제 입력1

```
216
```

예제 출력1

```
198
```

1436번 : 영화감독 숌 - Silver 5

문제

- 종말의 숫자: 어떤 수에 6이 적어도 3개 이상 연속으로 들어가는 수
- (예) 666, 1666, 2666, 3666, …, 6661,… 등
- N번째 종말의 숫자를 구하는 문제 (사전순)

제한 사항

- n의 범위는 1 <= n <= 10,000

예제

#알튜비튜

예제 입력1	예제 입력2	예제 입력3	예제 입력4	예제 입력5
2	3	6	187	500

예제 출력1	예제 출력2	예제 출력3	예제 출력4	예제 출력5
1666	2666	5666	66666	166699

기본 문제 - 영화감독 숌

#알튜비튜

제한 사항

- n의 범위는 1 <= n <= 10,000

접근

- 666의 앞과 뒤에 숫자를 계속 붙이고, 오름차순 정렬

-> 충분히 가능! 다른 방법은?

기본 문제 - 영화감독 숌

#알튜비튜

제한 사항

- n의 범위는 1 <= n <= 10,000

접근

- 666의 앞과 뒤에 숫자를 계속 붙이고, 오름차순 정렬
-> 충분히 가능! 다른 방법은?

- 666부터 차례로 666이 수에 있는지 판단 (시간초과 안나나?)
-> 10,000번째 종말의 숫자는 6,669,999 보다는 작다 (6,660,000 ~ 6,669,999: 1만 개)
-> 6,669,999 < 1억 이므로 충분히 브루트 포스 접근 가능!

기본 문제 - 영화감독 숌

#알튜비튜

제한 사항

- n의 범위는 1 <= n <= 10,000

접근

- 666의 앞과 뒤에 숫자를 계속 붙이고, 오름차순 정렬
-> 충분히 가능! 다른 방법은?

- 666부터 차례로 666이 수에 있는지 판단 (시간초과 안나나?)
-> 10,000번째 종말의 숫자는 6,669,999 보다는 작다 (6,660,000 ~ 6,669,999: 1만 개)
-> 6,669,999 < 1억 이므로 충분히 브루트 포스 접근 가능!

- 더 효율적인 풀이가 존재할 수 있지만 시험장에서 단시간에 떠올리기 쉽지 않음
-> 브루트 포스 풀이는 비교적 떠올리기 쉬움
-> 일단 브루트 포스 생각해 보자! -> 시간 괜찮은지 확인 -> 바로 시도

/<> 14889번 : 스타트와 링크 - Silver 2

문제

- 각 사람이 어떤 사람과 팀이 되었을 때 팀에 더해지는 능력치가 주어진다.
- N명을 두 팀으로 나누어 두 팀의 능력치 차이의 최솟값을 구하는 문제
- 이때, 각 팀은 인원은 N / 2 명

제한 사항

- 입력 범위는 4 <= n <= 20

응용 문제 - 스타트와 링크

#알튜비튜

예제 입력1

```
4
0 1 2 3
4 0 5 6
7 1 0 2
3 4 5 0
```

i/j	1	2	3	4
1	0	1	2	3
2	4	0	5	6
3	7	1	0	2
4	3	4	5	0

teamA: (2, 3)
능력치 합 = 1 + 5

teamB: (1, 4)
능력치 합 = 3 + 3

예제 출력1

```
0
```

몰래 보세요

#알튜비튜

Hint

1. n의 범위가 작네요!
2. 팀을 어떻게 둘로 나눌 수 있을까요?
    a. 순열을 구해주는 함수가 있는 거, 아시나요?

std::next_permutation

std::next_permutation

<algorithm>

```
                template <class BidirectionalIterator>
default (1)       bool next_permutation (BidirectionalIterator first,
                                         BidirectionalIterator last);
                template <class BidirectionalIterator, class Compare>
custom (2)        bool next_permutation (BidirectionalIterator first,
                                         BidirectionalIterator last, Compare comp);
```

Transform range to next permutation

Rearranges the elements in the range `[first,last)` into the next *lexicographically greater* permutation.

A *permutation* is each one of the `N!` possible arrangements the elements can take (where *N* is the number of elements in the range). Different permutations can be ordered according to how they compare *lexicographicaly* to each other; The first such-sorted possible permutation (the one that would compare *lexicographically smaller* to all other permutations) is the one which has all its elements sorted in ascending order, and the largest has all its elements sorted in descending order.

The comparisons of individual elements are performed using either `operator<` for the first version, or *comp* for the second.

If the function can determine the next higher permutation, it rearranges the elements as such and returns `true`. If that was not possible (because it is already at the largest possible permutation), it rearranges the elements according to the first permutation (sorted in ascending order) and returns `false`.

조합을 어떻게 구하죠…?

#알튜비튜

std::next_permutation

- 정확히는, 다음 순열을 구해주는 함수
- 사전적으로 다음 순열이 되도록 원본 벡터(배열)을 변경하고, true를 리턴
- 이전 순열을 돌려주는 prev_permutation도 존재
- 다음/이전 순열이 존재하지 않으면 false를 리턴

- 순열은, 순서가 다르면 다른 것으로 취급
  -> 조합을 구하기 위해서는 임시 배열을 사용

파이썬은요…?

#알튜비튜

itertools.**combinations**(*iterable*, *r*)

Return *r* length subsequences of elements from the input *iterable*.

The combination tuples are emitted in lexicographic ordering according to the order of the input *iterable*. So, if the input *iterable* is sorted, the combination tuples will be produced in sorted order.

Elements are treated as unique based on their position, not on their value. So if the input elements are unique, there will be no repeat values in each combination.

itertools.**permutations**(*iterable*, *r=None*)

Return successive *r* length permutations of elements in the *iterable*.

If *r* is not specified or is `None`, then *r* defaults to the length of the *iterable* and all possible full-length permutations are generated.

The permutation tuples are emitted in lexicographic ordering according to the order of the input *iterable*. So, if the input *iterable* is sorted, the combination tuples will be produced in sorted order.

Elements are treated as unique based on their position, not on their value. So if the input elements are unique, there will be no repeat values in each permutation.

마무리

정리

- 모든 경우의 수를 다 해보는 브루트 포스
- 입력 범위가 커지는 만큼 시간초과 위험이 있음 (즉, 효율적인 방법은 아님)
- 그래도 비교적 떠올리기 쉽다
- 따라서, 연산의 횟수가 100,000,000 이하면 일단 시도하자!
- 설마가 사람 잡는다..

이것도 알아보세요!

- 브루트포스를 재귀함수로 구현해 보아요 (종료 조건 중요)
- 비트마스크에 대해서도 알아보아요 (백트래킹 문제에서 많이 사용)

과제

#알튜비튜

필수

- 2503번 : 숫자 야구 – Silver 4
- 10757번 : 큰 수 A+B – Bronze 5

Python: 그냥 print(a+b)로 푸시면 안됩니다.
더하기를 직접 구현해주세요.

3문제 이상 선택

- 2798번 : 블랙잭 – Bronze 2
- 1018번 : 체스판 다시 칠하기 – Silver 5
- 1759번 : 암호 만들기 – Gold 5
- 2858번 : 기숙사 바닥 – Bronze 2
- 17626번 : Four Squares – Silver 4

과제 마감일

#알튜비튜

코드리뷰 O 마감 ~ 3월 24일 목요일 낮 12시

코드리뷰 X 마감 ~ 3월 24일 목요일 밤 12시 (24에서 25일로 넘어가는 자정)

추가제출 마감 ~ 3월 25일 금요일 밤 12시 (25일에서 26일로 넘어가는 자정)