1 체육복

문제 설명
점심시간에 도둑이 들어, 일부 학생이 체육복을 도난당했습니다. 다행히 여벌 체육복이 있는 학생이 이들에게 체육복을 빌려주려 합니다. 학생들의 번호는 체격 순으로 매겨져 있어, 바로 앞번호의 학생이나 바로 뒷번호의 학생에게만 체육복을 빌려줄 수 있습니다. 예를 들어, 4 번 학생은 3 번 학생이나 5 번 학생에게만 체육복을 빌려줄 수 있습니다. 체육복이 없으면 수업을 들을 수 없기 때문에 체육복을 적절히 빌려 최대한 많은 학생이 체육수업을 들어야 합니다.
전체 학생의 수 n, 체육복을 도난당한 학생들의 번호가 담긴 배열 lost, 여벌의 체육복을 가져온 학생들의 번호가 담긴 배열 reserve 가 매개변수로 주어질 때, 체육수업을 들을 수 있는 학생의 최댓값을 return 하도록 solution 함수를 작성해주세요.
제한사항
전체 학생의 수는 2 명 이상 30 명 이하입니다.
체육복을 도난당한 학생의 수는 1 명 이상 n 명 이하이고 중복되는 번호는 없습니다.
여벌의 체육복을 가져온 학생의 수는 1 명 이상 n 명 이하이고 중복되는 번호는 없습니다.
여벌 체육복이 있는 학생만 다른 학생에게 체육복을 빌려줄 수 있습니다.
여벌 체육복을 가져온 학생이 체육복을 도난당했을 수 있습니다. 이때 이 학생은 체육복을 하나만 도난당했다고 가정하며, 남은 체육복이 하나이기에 다른 학생에게는 체육복을 빌려줄 수 없습니다.
입출력 예

n	lost	reserve	return
5	[2, 4]	[1, 3, 5]	5
5	[2, 4]	[3]	4
3	[3]	[1]	2

입출력 예 설명
예제 #1
1 번 학생이 2 번 학생에게 체육복을 빌려주고, 3 번 학생이나 5 번 학생이 4 번 학생에게 체육복을 빌려주면 학생 5 명이 체육수업을 들을 수 있습니다.
예제 #2
3 번 학생이 2 번 학생이나 4 번 학생에게 체육복을 빌려주면 학생 4 명이 체육수업을 들을 수 있습니다.
출처
※ 공지 - 2019 년 2 월 18 일 지문이 리뉴얼되었습니다.
※ 공지 - 2019 년 2 월 27 일, 28 일 테스트케이스가 추가되었습니다.

```
public int solution(int n, int[] lost, int[] reserve) {
	int[] all = new int[n];

	for (int i : reserve)
		all[i - 1]++;
```

```
        for (int i : lost)
                all[i - 1]--;

        for (int i = 0; i < all.length; i++)
                if (all[i] < 0)
                        if (i != all.length - 1 && all[i + 1] > 0) {
                                all[i]++;
                                all[i + 1]--;
                        } else if (i != 0 && all[i - 1] > 0) {
                                all[i]++;
                                all[i - 1]--;
                        }

        int answer = 0;

        for (int i = 0; i < all.length; i++)
                if (!(all[i] < 0))
                        answer++;

        return answer;
}
```

https://velog.io/@delay/JAVA-%EC%B2%B4%EC%9C%A1%EB%B3%B5-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%A8%B8%EC%8A%A4

2. 큰 수 만들기
문제 설명
어떤 숫자에서 k 개의 수를 제거했을 때 얻을 수 있는 가장 큰 숫자를 구하려 합니다.
예를 들어, 숫자 1924 에서 수 두 개를 제거하면 [19, 12, 14, 92, 94, 24] 를 만들 수 있습니다. 이 중 가장 큰 숫자는 94 입니다.
문자열 형식으로 숫자 number 와 제거할 수의 개수 k 가 solution 함수의 매개변수로 주어집니다. number 에서 k 개의 수를 제거했을 때 만들 수 있는 수 중 가장 큰 숫자를 문자열 형태로 return 하도록 solution 함수를 완성하세요.
제한 조건
number 는 1 자리 이상, 1,000,000 자리 이하인 숫자입니다.
k 는 1 이상 number 의 자릿수 미만인 자연수입니다.
입출력 예

number	k	return
1924	2	94
1231234	3	3234
4177252841	4	775841

```
class Solution {
    public String solution(String number, int k) {
        StringBuilder sb = new StringBuilder();

                int cnt = number.length() - k;
        int left = 0;
        int right = number.length() - cnt;
        int max = -1;
        int idx = 0;

        while(cnt > 0) {
         max = -1;
            for(int j = left ; j <= right ; ++j){
                int num = number.charAt(j) - '0';
                if(num > max){
                    idx = j;
                    max = num;
                }
            }
            sb.append(number.charAt(idx));
            left = idx + 1;
            right = number.length() - --cnt;
        }

        return sb.toString();
    }
}
```

https://velog.io/@hyeon930/%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%A8%B8%EC%8A%A4-%ED%81%B0-%EC%88%98-%EB%A7%8C%EB%93%A4%EA%B8%B0-Java

3. 조이스틱

문제 설명
조이스틱으로 알파벳 이름을 완성하세요. 맨 처음엔 A 로만 이루어져 있습니다.
ex) 완성해야 하는 이름이 세 글자면 AAA, 네 글자면 AAAA
조이스틱을 각 방향으로 움직이면 아래와 같습니다.
▲ - 다음 알파벳
▼ - 이전 알파벳 (A 에서 아래쪽으로 이동하면 Z 로)
◀ - 커서를 왼쪽으로 이동 (첫 번째 위치에서 왼쪽으로 이동하면 마지막 문자에 커서)
▶ - 커서를 오른쪽으로 이동
예를 들어 아래의 방법으로 JAZ 를 만들 수 있습니다.
- 첫 번째 위치에서 조이스틱을 위로 9 번 조작하여 J 를 완성합니다.
- 조이스틱을 왼쪽으로 1 번 조작하여 커서를 마지막 문자 위치로 이동시킵니다.
- 마지막 위치에서 조이스틱을 아래로 1 번 조작하여 Z 를 완성합니다.
따라서 11 번 이동시켜 "JAZ"를 만들 수 있고, 이때가 최소 이동입니다.
만들고자 하는 이름 name 이 매개변수로 주어질 때, 이름에 대해 조이스틱 조작 횟수의 최솟값을 return 하도록 solution 함수를 만드세요.
제한 사항
name 은 알파벳 대문자로만 이루어져 있습니다.
name 의 길이는 1 이상 20 이하입니다.
입출력 예

name	return
JEROEN	56
JAN	23

출처
※ 공지 - 2019 년 2 월 28 일 테스트케이스가 추가되었습니다.

```
class Solution {
    public int solution(String name) {

        // 1. 위, 아래 최소 이동
        int ans = 0;
        for(int i = 0 ; i < name.length() ; i++) {
            if(name.charAt(i)!= 'A') {
                int up = name.charAt(i) - 'A';
                int down = 1 + 'Z' - name.charAt(i);
                ans += (up < down)? up : down;
            }
        }

        // 2. A 아닌 모든 문자를 들릴 수 있는 최소 좌우 이동
        // 모든 위치에서 역으로 돌아가는 경우 최소를 찾는다.
        int minMove = name.length() - 1;
        for(int i = 0 ; i < name.length() ; i++) {
```

```
            if(name.charAt(i) != 'A') {
                int next = i+1;
                while(next < name.length() && name.charAt(next)
== 'A') {
                    next++;
                }
                int move = 2 * i + name.length() - next;
                minMove = Math.min(move, minMove);
            }
        }

        return ans + minMove;
    }
}
```

https://keepgoing0328.tistory.com/71

4. 구명보트

문제 설명
무인도에 갇힌 사람들을 구명보트를 이용하여 구출하려고 합니다. 구명보트는 작아서 한 번에 최대 2 명씩 밖에 탈 수 없고, 무게 제한도 있습니다.
예를 들어, 사람들의 몸무게가 [70kg, 50kg, 80kg, 50kg]이고 구명보트의 무게 제한이 100kg 이라면 2 번째 사람과 4 번째 사람은 같이 탈 수 있지만 1 번째 사람과 3 번째 사람의 무게의 합은 150kg 이므로 구명보트의 무게 제한을 초과하여 같이 탈 수 없습니다.
구명보트를 최대한 적게 사용하여 모든 사람을 구출하려고 합니다.
사람들의 몸무게를 담은 배열 people 과 구명보트의 무게 제한 limit 가 매개변수로 주어질 때, 모든 사람을 구출하기 위해 필요한 구명보트 개수의 최솟값을 return 하도록 solution 함수를 작성해주세요.
제한사항
무인도에 갇힌 사람은 1 명 이상 50,000 명 이하입니다.
각 사람의 몸무게는 40kg 이상 240kg 이하입니다.
구명보트의 무게 제한은 40kg 이상 240kg 이하입니다.
구명보트의 무게 제한은 항상 사람들의 몸무게 중 최댓값보다 크게 주어지므로 사람들을 구출할 수 없는 경우는 없습니다.
입출력 예

people	limit	return
[70, 50, 80, 50]	100	3
[70, 80, 50]	100	3

```
public class Solution {

  public int solution(int[] people, int limit) {
    int answer = 0;

    Arrays.sort(people);

    int min = 0;

    for (int max = people.length - 1; min <= max; max--){
      if (people[min] + people[max] <= limit) min++;
      answer++;
    }

    return answer;
}

  @Test
  public void 정답() {
    Assert.assertEquals(1, solution(new int[]{50}, 50));
    Assert.assertEquals(2, solution(new int[]{20, 50, 50, 80},
100));
```

```
   Assert.assertEquals(3, solution(new int[]{70, 50, 80, 50},
100));
   Assert.assertEquals(3, solution(new int[]{50, 30, 20, 70, 10},
100));
   Assert.assertEquals(3, solution(new int[]{70, 80, 50}, 100));
   Assert.assertEquals(5, solution(new int[]{10, 20, 30, 40, 50,
60, 70, 80, 90}, 100));
  }
}
```

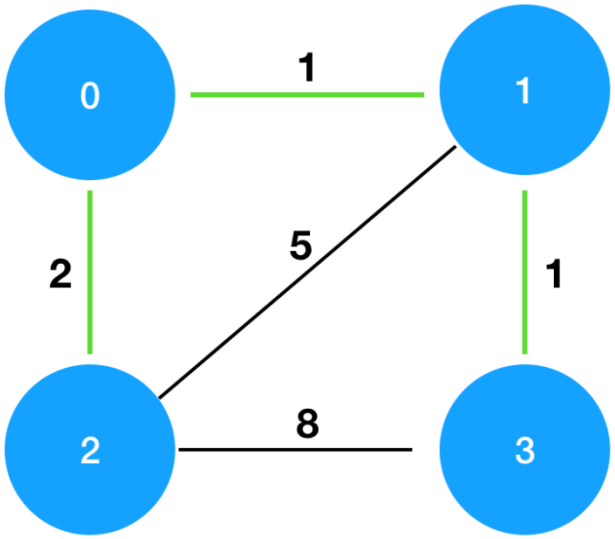
https://velog.io/@ajufresh/%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%A8%B8%EC%8A%A4-%EA%B5%AC%EB%AA%85%EB%B3%B4%ED%8A%B8-%EB%AC%B8%EC%A0%9C%ED%92%80%EC%9D%B4-Java

5. 섬 연결하기

문제 설명
n 개의 섬 사이에 다리를 건설하는 비용(costs)이 주어질 때, 최소의 비용으로 모든 섬이 서로 통행 가능하도록 만들 때 필요한 최소 비용을 return 하도록 solution 을 완성하세요.
다리를 여러 번 건너더라도, 도달할 수만 있으면 통행 가능하다고 봅니다. 예를 들어 A 섬과 B 섬 사이에 다리가 있고, B 섬과 C 섬 사이에 다리가 있으면 A 섬과 C 섬은 서로 통행 가능합니다.
제한사항
섬의 개수 n 은 1 이상 100 이하입니다.
costs 의 길이는 ((n-1) * n) / 2 이하입니다.
임의의 i 에 대해, costs[i][0] 와 costs[i] [1]에는 다리가 연결되는 두 섬의 번호가 들어있고, costs[i] [2]에는 이 두 섬을 연결하는 다리를 건설할 때 드는 비용입니다.
같은 연결은 두 번 주어지지 않습니다. 또한 순서가 바뀌더라도 같은 연결로 봅니다. 즉 0 과 1 사이를 연결하는 비용이 주어졌을 때, 1 과 0 의 비용이 주어지지 않습니다.
모든 섬 사이의 다리 건설 비용이 주어지지 않습니다. 이 경우, 두 섬 사이의 건설이 불가능한 것으로 봅니다.
연결할 수 없는 섬은 주어지지 않습니다.
입출력 예

n	costs	return
4	[[0,1,1],[0,2,2],[1,2,5],[1,3,1],[2,3,8]]	4

입출력 예 설명
costs 를 그림으로 표현하면 다음과 같으며, 이때 초록색 경로로 연결하는 것이 가장 적은 비용으로 모두를 통행할 수 있도록 만드는 방법입니다.



```java
import java.util.*;

class Solution {
    class Edge implements Comparable<Edge> {
        int from, to, cost;

        Edge(int from, int to, int cost){
            this.from = from;
            this.to = to;
            this.cost = cost;
        }

        @Override
        public int compareTo(Edge o){
            return this.cost - o.cost;
        }
    }

    static int[] parent;
    static PriorityQueue<Edge> adj;

    public int solution(int n, int[][] costs) {
        int answer = 0;
        parent = new int[n];
        adj = new PriorityQueue<>();

        for(int i = 0 ; i < costs.length ; ++i){
            Edge edge = new Edge(costs[i][0], costs[i][1],
costs[i][2]);
            adj.offer(edge);
        }

        for(int i = 0 ; i < n ; ++i) parent[i] = i;

        while(!adj.isEmpty()) {
        Edge edge = adj.poll();

            if(find(edge.from) == find(edge.to)) continue;
            else {
                union(edge.from, edge.to);
                answer += edge.cost;
            }
        }

        return answer;
    }

    public int find(int n){
        if(parent[n] == n) return n;
```

```
        return parent[n] = find(parent[n]);
    }

    public void union(int a, int b){
        int rootA = find(a);
        int rootB = find(b);

       if(rootA != rootB) parent[rootB] = rootA;
    }
}
```

https://velog.io/@hyeon930/%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%A8%B8%EC%8A%A4-%EC%84%AC-%EC%97%B0%EA%B2%B0%ED%95%98%EA%B8%B0-Java

6. 단속카메라

문제 설명

고속도로를 이동하는 모든 차량이 고속도로를 이용하면서 단속용 카메라를 한 번은 만나도록 카메라를 설치하려고 합니다.

고속도로를 이동하는 차량의 경로 routes 가 매개변수로 주어질 때, 모든 차량이 한 번은 단속용 카메라를 만나도록 하려면 최소 몇 대의 카메라를 설치해야 하는지를 return 하도록 solution 함수를 완성하세요.

제한사항

- 차량의 대수는 1 대 이상 10,000 대 이하입니다.
- routes 에는 차량의 이동 경로가 포함되어 있으며 routes[i][0]에는 i 번째 차량이 고속도로에 진입한 지점, routes[i][1]에는 i 번째 차량이 고속도로에서 나간 지점이 적혀 있습니다.
- 차량의 진입/진출 지점에 카메라가 설치되어 있어도 카메라를 만난것으로 간주합니다.
- 차량의 진입 지점, 진출 지점은 -30,000 이상 30,000 이하입니다.

입출력 예

routes	return
[[-20,15], [-14,-5], [-18,-13], [-5,-3]]	2

입출력 예 설명

-5 지점에 카메라를 설치하면 두 번째, 네 번째 차량이 카메라를 만납니다.

-15 지점에 카메라를 설치하면 첫 번째, 세 번째 차량이 카메라를 만납니다.

```
import java.util.*;

class Solution {
    public int solution(int[][] routes) {
        int answer = 1;

        Arrays.sort(routes, new Comparator<int[]>(){
            @Override
             public int compare(int[] a, int[] b){
                 return a[0] - b[0];
             }
        });
```

```
        int min = routes[0][0];
        int max = routes[0][1];
        for(int i = 1 ; i < routes.length ; ++i){
            int in = routes[i][0];
            int out = routes[i][1];

            if(in > max || out < min) {
                answer++;
                min = in;
                max = out;
            } else {
                min = in > min ? in : min;
                max = max > out ? out : max;
            }
        }

        return answer;
    }
}
```

https://velog.io/@hyeon930/%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%A8%B8%EC%8A%A4-%EB%8B%A8%EC%86%8D%EC%B9%B4%EB%A9%94%EB%9D%BC-Java