1149 - RGB거리

난이도	실버1

문제

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
0.5 초 (추가 시간 없음)	128 MB	71068	35342	26377	49.428%

문제

RGB거리에는 집이 N개 있다. 거리는 선분으로 나타낼 수 있고, 1번 집부터 N번 집이 순서대로 있다.

집은 빨강, 초록, 파랑 중 하나의 색으로 칠해야 한다. 각각의 집을 빨강, 초록, 파랑으로 칠하는 비용이 주어졌을 때, 아래 규칙을 만족하면서 모든 집을 칠하는 비용의 최솟값을 구해보자.

- 1번 집의 색은 2번 집의 색과 같지 않아야 한다.
- N번 집의 색은 N-1번 집의 색과 같지 않아야 한다.
- i(2 ≤ i ≤ N-1)번 집의 색은 i-1번, i+1번 집의 색과 같지 않아야 한다.

입력

첫째 줄에 집의 수 N(2 ≤ N ≤ 1,000)이 주어진다. 둘째 줄부터 N개의 줄에는 각 집을 빨강, 초록, 파랑으로 칠하는 비용이 1번 집부터 한 줄에 하나씩 주어진다. 집을 칠하는 비용은 1,000보다 작거나 같은 자연수이다.

출력

첫째 줄에 모든 집을 칠하는 비용의 최솟값을 출력한다.

예제 입력 1 복사

```
3
26 40 83
49 60 57
13 89 99
```

예제 출력 1 복사

```
96
```

예제 입력 2 복사

```
3
1 100 100
100 1 100
100 100 1
```

예제 출력 2 복사

```
3
```

예제 입력 3 복사

```
3
1 100 100
100 100 100
1 100 100
```

예제 출력 3 복사

```
102
```

예제 입력 4 복사

```
6
30 19 5
64 77 64
15 19 97
4 71 57
90 86 84
93 32 91
```

예제 출력 4 복사

```
208
```

예제 입력 5 복사

```
8
71 39 44
32 83 55
51 37 63
89 29 100
83 58 11
65 13 15
47 25 29
60 66 19
```

예제 출력 5 복사

```
253
```

풀이과정

처음에는

> 그냥 그때마다 가능한 선택지 중 최솟값 고르면 되는 거 아닌가?

라는 ~~완일한~~ 생각으로 구현해봤으나 당연히 틀렸다.

이번 선택에서 조금 손해보더라도 다음 선택에서 작은 값을 골라주는 게 최종적으로 더 작은 결과값을 도출해낼 수도 있다는 점을 간과했다.

그래서, 현재 집에서 특정 컬러를 선택했을 때, 이전 집에서의 가능한 최솟값을 더한 값을 누적하며 더해줬다. 그리고 마지막에서 세 컬러 중 최솟값을 선택하도록 하였다.

말로 풀어서 설명하니 조금 헷갈릴 것 같아, 간단한 예시를 들어서 설명해보겠다.

예시

```
6
30 19 5
64 77 64
15 19 97
4 71 57
90 86 84
93 32 91
```

다음과 같이 입력이 들어왔다고 해보자.

1번째	30	19	5
2번째	**69** (64+5)	**82** (77+5)	**83** (64+19)
3번째	**97** (15 + 82)	**88** (19 + 69)	**166** (97 + 69)
4번째	**92** (4 + 88)	**168** (71 + 97)	**145** (57 + 88)
5번째	**235** (90 + 145)	**178** (86 + 92)	**176** (84 + 92)
6번째	**269** (93 + 176)	**208** (32 + 176)	**269** (91 + 178)

위와 같이, 현재 집의 특정 컬러를 택했을 때 이전 집에서 고를 수 있는 최소값을 누적하여 더해가고, 가장 마지막 집에서는 세 컬러 중 최소 비용인 컬러를 선택해주면 된다.

점화식

N번째 집에서의 각 컬러당 비용을 저장하는 배열을 cost라 할 때, 점화식은 다음과 같다.

- cost[n][0] += min(cost[n-1][1], cost[n-1][2])
  cost[n][1] += min(cost[n-1][0], cost[n-1][2])
  cost[n][2] += min(cost[n-1][0], cost[n-1][1])

소스코드

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

/*
* Top-down 풀이 (재귀)
*
* 규칙: 인접한 집과 색이 달라야한다.
```

```
*
* 1. 그때그때의 최솟값을 선택하자. (이전에 선택한 컬러 제외) -> 오답
* 무작정 그때그때의 최솟값을 더하면 안된다!
* -> 이번 집에서 조금 손해보고 다음집에서의 최솟값을 고르는 선택이 더 나을 수도 있다.
*
* 2. 이번 집에서 각 컬러당 나올 수 있는 최솟값을 구하여 누적해서 더하고, 마지막에 최솟값을 선택하자 -> 성공
*
* 점화식:
*   cost[n][0] += min( cost[n-1][1], cost[n-1][2] )
*   cost[n][1] += min( cost[n-1][0], cost[n-1][2] )
*   cost[n][2] += min( cost[n-1][0], cost[n-1][1] )
*/

public class Main {

    static int N;
    static int R, G ,B;
    static int answer;
    static int[][] cost;

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StringTokenizer st;

        N = Integer.parseInt(br.readLine());
        cost = new int[N][3];

        for(int i=0 ; i<N ; i++){
            st = new StringTokenizer(br.readLine());
            R = Integer.parseInt(st.nextToken());
            G = Integer.parseInt(st.nextToken());
            B = Integer.parseInt(st.nextToken());

            cost[i][0] = R;
            cost[i][1] = G;
            cost[i][2] = B;
        }

        solution(1);
        System.out.println(answer);

    }

    public static void solution(int home) {
        cost[home][0] += Math.min(cost[home-1][1], cost[home-1][2]);
        cost[home][1] += Math.min(cost[home-1][0], cost[home-1][2]);
        cost[home][2] += Math.min(cost[home-1][0], cost[home-1][1]);

        if(home == N-1)
            answer = Math.min(cost[N-1][0], Math.min(cost[N-1][1], cost[N-1][2]));
        else
            solution(home+1);
    }
}
```