## Dota2 Senate

In the world of Dota2, there are two parties: the `Radiant` and the `Dire`.

The Dota2 senate consists of senators coming from two parties. Now the senate wants to make a decision about a change in the Dota2 game. The voting for this change is a round-based procedure. In each round, each senator can exercise `one` of the two rights:

1. `Ban one senator's right`:
   A senator can make another senator lose**all his rights** in this and all the following rounds.
2. `Announce the victory`:
   If this senator found the senators who still have rights to vote are all from**the same party**, he can announce the victory and make the decision about the change in the game.

Given a string representing each senator's party belonging. The character 'R' and 'D' represent the `Radiant` party and the `Dire` party respectively. Then if there are `n` senators, the size of the given string will be `n`.

The round-based procedure starts from the first senator to the last senator in the given order. This procedure will last until the end of voting. All the senators who have lost their rights will be skipped during the procedure.

Suppose every senator is smart enough and will play the best strategy for his own party, you need to predict which party will finally announce the victory and make the change in the Dota2 game. The output should be `Radiant` or `Dire`.

### Example 1:

```
Input: "RD"
Output: "Radiant"
Explanation: The first senator comes from Radiant and he can just ban the next senato
r's right in the round 1.
And the second senator can't exercise any rights any more since his right has been ba
nned.
And in the round 2, the first senator can just announce the victory since he is the o
nly guy in the senate who can vote.
```

### Example 2:

```
Input: "RDD"
Output: "Dire"
Explanation:
The first senator comes from Radiant and he can just ban the next senator's right in
the round 1.
And the second senator can't exercise any rights anymore since his right has been ban
ned.
And the third senator comes from Dire and he can ban the first senator's right in the
 round 1.
And in the round 2, the third senator can just announce the victory since he is the o
nly guy in the senate who can vote.
```

**Note:**

1. The length of the given string will in the range [1, 10,000].

## Solution 1

This is obliviously a greedy algorithm problem. Each senate `R` must ban its next **closest** senate `D` who is from another party, or else `D` will ban its next senate from `R's` party.

The idea is to use two queues to save the index of each senate from `R's` and `D's` parties, respectively. During each round, we delete the banned senates' indices; and plus each remainning senate's index with `n` (the length of the input string `senate`), then move it to the back of its respective queue.

The following code is with an `O(n)` space and an `O(n)` time complexity.

Java version:

```
    public String predictPartyVictory(String senate) {
        Queue<Integer> q1 = new LinkedList<Integer>(), q2 = new LinkedList<Integer>(
);
        int n = senate.length();
        for(int i = 0; i<n; i++){
            if(senate.charAt(i) == 'R')q1.add(i);
            else q2.add(i);
        }
        while(!q1.isEmpty() && !q2.isEmpty()){
            int r_index = q1.poll(), d_index = q2.poll();
            if(r_index < d_index)q1.add(r_index + n);
            else q2.add(d_index + n);
        }
        return (q1.size() > q2.size())? "Radiant" : "Dire";
    }
```

C++ version:

```
string predictPartyVictory(string senate) {
        queue<int> q1, q2;
        int n = senate.length();
        for(int i = 0; i<n; i++)
            (senate[i] == 'R')?q1.push(i):q2.push(i);
        while(!q1.empty() && !q2.empty()){
            int r_index = q1.front(), d_index = q2.front();
            q1.pop(), q2.pop();
            (r_index < d_index)?q1.push(r_index + n):q2.push(d_index + n);
        }
        return (q1.size() > q2.size())? "Radiant" : "Dire";
    }
```

written by Hao Cai original link here

## Solution 2

Simulate the process. We don't have to resolve bans until later - we can just let them "float".

We have `people = [int, int]` representing how many people are in the queue, and `bans = [int, int]` representing how many "floating" bans there are. When we meet a person, if there is a floating ban waiting for them, then they are banned and we skip them. Eventually, the queue will just have one type of person, which is when we break.

```
def predictPartyVictory(self, senate):
    A = collections.deque()
    people = [0, 0]
    bans = [0, 0]

    for person in senate:
        x = person == 'R'
        people[x] += 1
        A.append(x)

    while all(people):
        x = A.popleft()
        people[x] -= 1
        if bans[x]:
            bans[x] -= 1
        else:
            bans[x^1] += 1
            A.append(x)
            people[x] += 1

    return "Radiant" if people[1] else "Dire"
```

written by awice original link here

## Solution 3

The idea is to have two arrays nextRound and eliminate.
nextRound keeps track of number of elements in D/R who are still valid for the nextRound
eliminate keeps track of number of future candidates of D/R who are already eliminated by previous members.

As we walk through the string, we build a new string that contains all the selected senators for nextRound, and in the end of the round, we reset the nextround, but carry over the eliminate array, as the counts in this array are still valid for the nextround.

As we walk through each element in the string:

1. If am already eliminated based on the values in the eliminate array, reduce the elimination count and proceed without adding myself to the next round.
2. If am not eliminated, then I include myself in the nextRound, and chose to increase the elimination count of the other party.

```
public static String predictPartyVictory(String senate) {
        HashMap<Character, Integer> map = new HashMap<>();
        map.put('D', 0);
        map.put('R', 1);
        int[] nextRound = new int[2], eliminate = new int[2];
        while(true) {
            StringBuilder sb = new StringBuilder();
            for(int i = 0; i < senate.length(); i++) {
                if (eliminate[map.get(senate.charAt(i))] > 0) {
                    eliminate[map.get(senate.charAt(i))]--;
                } else {
                    nextRound[map.get(senate.charAt(i))]++;
                    eliminate[map.get(senate.charAt(i)) ^ 1]++;
                    sb.append(senate.charAt(i));
                }
            }

            if (nextRound[0] - eliminate[0] > 0 && nextRound[1] - eliminate[1] > 0)
{
                nextRound = new int[2];
                senate = sb.toString();
            } else {
                return nextRound[0] - eliminate[0] > 0 ? "Dire" : "Radiant";
            }
        }
    }
```

written by johnyrufus16 original link here