Dota2 Senate Problem - Solution Notes

Problem Statement

In the world of Dota2, there are two parties: **Radiant (R)** and **Dire (D)**. The Dota2 senate consists of senators coming from these two parties. The senators follow a round-based voting procedure where:

1. Each senator can exercise one of the two rights:
   - **Ban** another senator's right: A senator can make another senator lose all his rights in this and future rounds.
   - **Announce victory**: If all remaining senators belong to the same party, that party wins.
2. Given a string senate, where each character is either 'R' or 'D', representing the party of each senator, we need to **predict which party will finally win**.

Example 1:

Input: senate = "RD"
Output: "Radiant"
Explanation: The first senator ('R') bans the right of the second senator ('D'). In the next round, 'R' is the only senator left, so Radiant wins.

Example 2:

Input: senate = "RDD"
Output: "Dire"
Explanation: 'R' bans the first 'D'. The last 'D' bans 'R' in the same round. Now, only one 'D' is left, so Dire wins.

Thinking Process

1. We need to **simulate the banning process** in rounds.
2. Use **three queues**:
   - **Queue q** stores indices of all senators.
   - **Queue r** stores indices of 'R' senators.
   - **Queue d** stores indices of 'D' senators.
3. We process senators in the order they appear and eliminate the first available senator of the opposite party.
4. The senators cycle back in the queue if they survive the round.
5. The process continues until one party is eliminated.

Code Implementation

```
class Solution {
public:
    string predictPartyVictory(string s) {
        queue<int>q;
        queue<int>r;
        queue<int>d;

        for(int i=0; i<s.size(); i++){
            q.push(i);
            if(s[i]=='R'){
                r.push(i);
            }
            else {
                d.push(i);
            }
        }

        while(q.size()>=1){
            while(s[q.front()]=='X') q.pop();
            if(s[q.front()] == 'R'){
                if(d.size()==0){
                    return "Radiant";
                }
                else{
                    s[d.front()]='X';
                    d.pop();
                    q.push(q.front());
                    q.pop();
                    r.push(r.front());
                    r.pop();
                }
            }
            else { // 'D' senator's turn
                if(r.size()==0){
                    return "Dire";
                }
                else{
                    s[r.front()]='X';
                    r.pop();
                    q.push(q.front());
                    q.pop();
                    d.push(d.front());
                    d.pop();
                }
            }
        }
        if(s[q.front()]=='R') return "Radiant";
        else return "Dire";
    }
};
```

Detailed Code Breakdown

1. **Queue Initialization:**
    - q: Stores all senators’ indices.
    - r: Stores indices of 'R' senators.
    - d: Stores indices of 'D' senators.
2. **Round-based Voting Simulation:**
    - Process senators one by one.
    - If the senator at the front is already banned ('X'), pop it.
    - If an 'R' senator is encountered:
        - If no 'D' remains, return **"Radiant"**.
        - Otherwise, ban the first available 'D', move 'R' to the next round.

    - If a 'D' senator is encountered:
        - If no 'R' remains, return **"Dire"**.
        - Otherwise, ban the first available 'R', move 'D' to the next round.

3. **Returning the Result:**
    - The loop continues until one party is completely banned.
    - The remaining party is declared the winner.

Conclusion

- The approach efficiently simulates the banning process using **three queues**.
- **Time Complexity:** O(n), since each senator is processed once per round.
- **Space Complexity:** O(n), due to the queues used.
- This ensures that the problem is solved optimally while maintaining clarity.

Final Thought: This approach smartly maintains fairness in banning while ensuring the strongest senators survive till the end. 🔥