Experiment–3: Stable Attendance Window

Problem Statement

A university maintains an automated log of attendance activities recorded in order.

Each record represents a student being **Present (P)** or **Absent (A)**.

A **stable attendance window** is defined as a continuous period in which:

- The number of **Present** and **Absent** records are equal.

Input Format

1. Integer N — number of attendance records
2. N entries representing attendance status
   P → Present

   A → Absent

Output Format

- Print a single integer: the **maximum length of a stable attendance window**
- If no such window exists, print **0**

Algorithm

1. Read integer **N**.
2. Read **N** attendance records (P or A).
3. Initialize:
   - sum = 0
   - maxLen = 0
   - A map (or array) to store the **first occurrence** of each prefix sum.
4. Store sum = 0 at index -1 in the map.
5. For each index i from 0 to N-1:
   - If record is P, increase sum by 1.
   - If record is A, decrease sum by 1.
   - If sum already exists in the map:
     1. Update maxLen = max(maxLen, i - map[sum]).

        - Else:
            1. Store sum with index i in the map.

6. Print maxLen.
7. If no such window exists, maxLen will be 0.

CODE:

```
#include <iostream>
#include <unordered_map>
#include <vector>
using namespace std;

int main() {
  int n;
  cin >> n;

  vector<char> arr(n);
  for (int i = 0; i < n; i++) {
    cin >> arr[i];
  }

  unordered_map<int, int> mp;
  int sum = 0;
  int ans = 0;

  mp[0] = -1;

  for (int i = 0; i < n; i++) {
    if (arr[i] == 'P')
      sum++;
    else
      sum--;
```

```
        if (mp.find(sum) != mp.end()) {
            ans = max(ans, i - mp[sum]);
        } else {
            mp[sum] = i;
        }
    }

    cout << ans;
    return 0;
}
```

Testcase Input:

```
7
P P A A P A A
```

Output:

```
4
```