

# I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.

# I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's.
  1. The leftmost character of each string is a 'B'.
  2. In each string, there can be no two 'R' characters adjacent to one another.

# I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's.
  1. The leftmost character of each string is a 'B'.
  2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.

# I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's'.
  1. The leftmost character of each string is a 'B'.
  2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.
- After the first check, we go letter by letter, keeping track of the last 'B' instance, which starts at 0.

# I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's'.
  1. The leftmost character of each string is a 'B'.
  2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.
- After the first check, we go letter by letter, keeping track of the last 'B' instance, which starts at 0.
- When we encounter a 'R' character and the last 'B' is not right behind it, we increment the counter and update the last 'B' to the current location

# I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's'.
  1. The leftmost character of each string is a 'B'.
  2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.
- After the first check, we go letter by letter, keeping track of the last 'B' instance, which starts at 0.
- When we encounter a 'R' character and the last 'B' is not right behind it, we increment the counter and update the last 'B' to the current location
- Beware of repeated expensive string operations (e.g., substring) to avoid  $\mathcal{O}(n^2)$  time complexity.
  3. The number of 'B' characters in each string needs to be the same as the number of 'B' characters in every other string.

# I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's'.
  1. The leftmost character of each string is a 'B'.
  2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.
- After the first check, we go letter by letter, keeping track of the last 'B' instance, which starts at 0.
- When we encounter a 'R' character and the last 'B' is not right behind it, we increment the counter and update the last 'B' to the current location
- Beware of repeated expensive string operations (e.g., substring) to avoid  $\mathcal{O}(n^2)$  time complexity.
  3. The number of 'B' characters in each string needs to be the same as the number of 'B' characters in every other string.
- For the final requirement, for each word in the list: If the length of the word is smaller than the amount of 'B's, then it is impossible.

# I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's'.
  1. The leftmost character of each string is a 'B'.
  2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.
- After the first check, we go letter by letter, keeping track of the last 'B' instance, which starts at 0.
- When we encounter a 'R' character and the last 'B' is not right behind it, we increment the counter and update the last 'B' to the current location
- Beware of repeated expensive string operations (e.g., substring) to avoid  $\mathcal{O}(n^2)$  time complexity.
  3. The number of 'B' characters in each string needs to be the same as the number of 'B' characters in every other string.
- For the final requirement, for each word in the list: If the length of the word is smaller than the amount of 'B's, then it is impossible.
- Otherwise, we increment the counter with the difference of the max 'B's minus the 'B's in the word.



# I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's.
  1. The leftmost character of each string is a 'B'.
  2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.
- After the first check, we go letter by letter, keeping track of the last 'B' instance, which starts at 0.
- When we encounter a 'R' character and the last 'B' is not right behind it, we increment the counter and update the last 'B' to the current location
- Beware of repeated expensive string operations (e.g., substring) to avoid  $\mathcal{O}(n^2)$  time complexity.
  3. The number of 'B' characters in each string needs to be the same as the number of 'B' characters in every other string.
- For the final requirement, for each word in the list: If the length of the word is smaller than the amount of 'B's, then it is impossible.
- Otherwise, we increment the counter with the difference of the max 'B's minus the 'B's in the word.
- Lastly, return the counter. The overall complexity should be around  $\mathcal{O}(n)$

# I: Intricate Idioms

Problem author: Jeroen Op de Beek



- **Problem:** Check how many changes a string needs to go through to respect three properties.
- We first need to initialize a counter, along with an array keeping track of the number of 'B's in each word, as well as the maximum number of 'B's.
  1. The leftmost character of each string is a 'B'.
  2. In each string, there can be no two 'R' characters adjacent to one another.
- For the first two properties, we will go through of each word. We first check if character is a 'B'.
- After the first check, we go letter by letter, keeping track of the last 'B' instance, which starts at 0.
- When we encounter a 'R' character and the last 'B' is not right behind it, we increment the counter and update the last 'B' to the current location
- Beware of repeated expensive string operations (e.g., substring) to avoid  $\mathcal{O}(n^2)$  time complexity.
  3. The number of 'B' characters in each string needs to be the same as the number of 'B' characters in every other string.
- For the final requirement, for each word in the list: If the length of the word is smaller than the amount of 'B's, then it is impossible.
- Otherwise, we increment the counter with the difference of the max 'B's minus the 'B's in the word.
- Lastly, return the counter. The overall complexity should be around  $\mathcal{O}(n)$

Statistics: 30 submissions, 13 accepted, 14 unknown