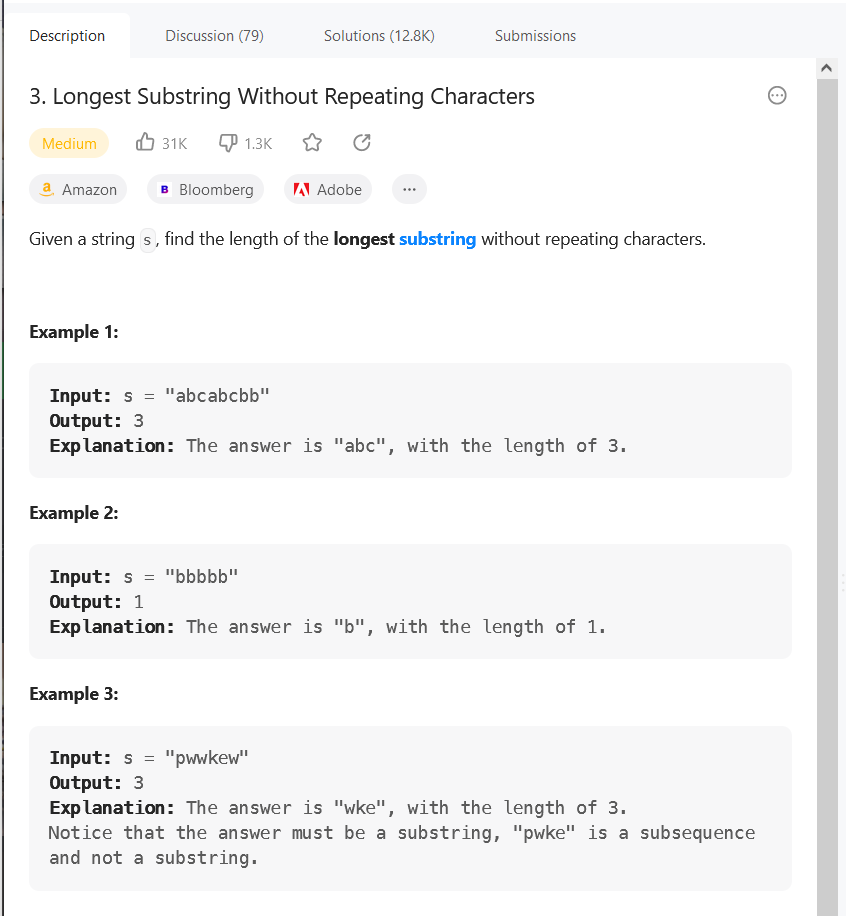
# L3. Longest Substring Without Repeating Characters



Ideas:

a.Find the longest substring without any repeating characters.

b.One step: Checkout the repeating characters

c.Need to use hash table to setup the key and value

class Solution {

public int lengthOfLongestSubstring(String s) {

if ( s==null || s.equals("")){

return 0;

}

//initial two pointer

//abcabcbb

//^

//^

//move the right pointer first

int leftPointer = 0;

int rightPointer = 0;

int max = 0;

// create one new hashset to store the Character

HashSet<Character> hash\_set = new HashSet();

//当右指针移动时小于s的长度时

while(rightPointer < s.length()){

if(!hash\_set.contains(s.charAt(rightPointer))){

hash\_set.add(s.charAt(rightPointer));

rightPointer++;

// compare the max and update all the time

max = Math.max(hash\_set.size(),max);

}else{

//duplicate condition

//remove the leftpointer and move forward

hash\_set.remove(s.charAt(leftPointer));

leftPointer++;

}

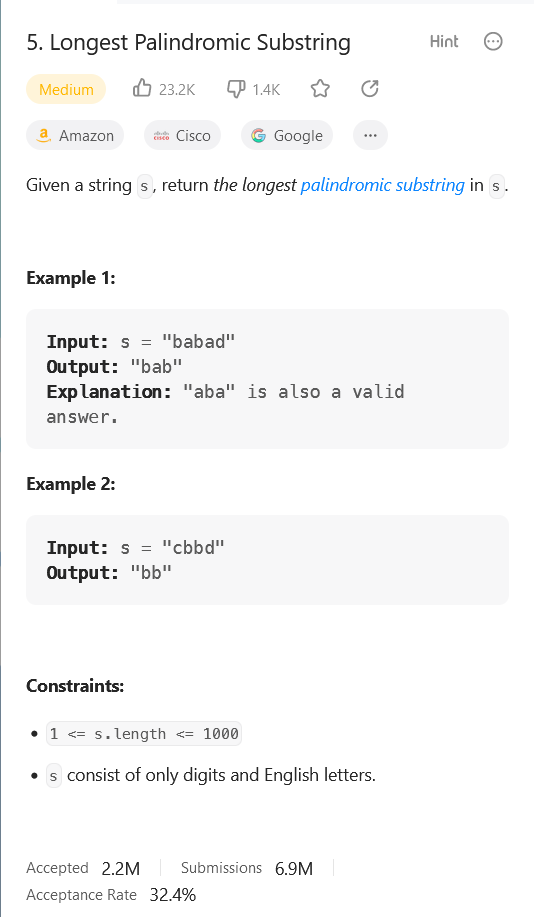
}

return max;

}

}

# L5. Longest Palindromic Substring



Idea: A string is said to be a palindrome if the string read from left to right is equal to the string read from right to left. Given the string s. Using dynamic programming. Iterate from the left to right.

Codes:

class Solution {

int maxLen = 0;

int left = 0; // The beginning of the left

public String longestPalindrome(String s) {

if(s == null || s.length() <=1) return s;

for(int i = 0; i < s.length();i++){

//case1:From one letter to find its left and right

helper(s,i,i);

helper(s,i,i+1);

}

//substring

return s.substring(left, left+maxLen);

}

public void helper(String s, int l, int r){

while(l >= 0 && r < s.length() && s.charAt(l)==s.charAt(r)){

l--;

r++;

}

if(maxLen < r-l-1){

maxLen = r-l-1;

left = l+1;

}

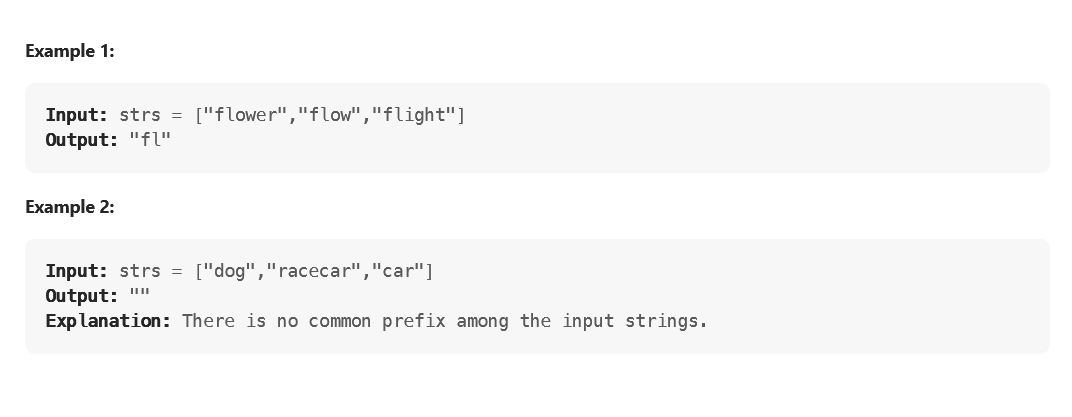
}

}

# L14 Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string



Target: Finding the most common string after searching three or more strings.

Solution:

class Solution {

public String longestCommonPrefix(String[] strs) {

// new string builder

StringBuilder result = new StringBuilder();

// exception

if (strs==null ||strs.length==0){

return "";

}

// sort the string alphabetically

Arrays.sort(strs);

// toCharArray convert the given string into a character

char [] first = strs[0].toCharArray();

char [] last = strs[strs.length-1].toCharArray();

for (int i=0;i<first.length;i++){

// If the lastl

if (last.length> i && last[i]==first[i]){

result.append(last[i]);

}

else{

return result.toString();

}

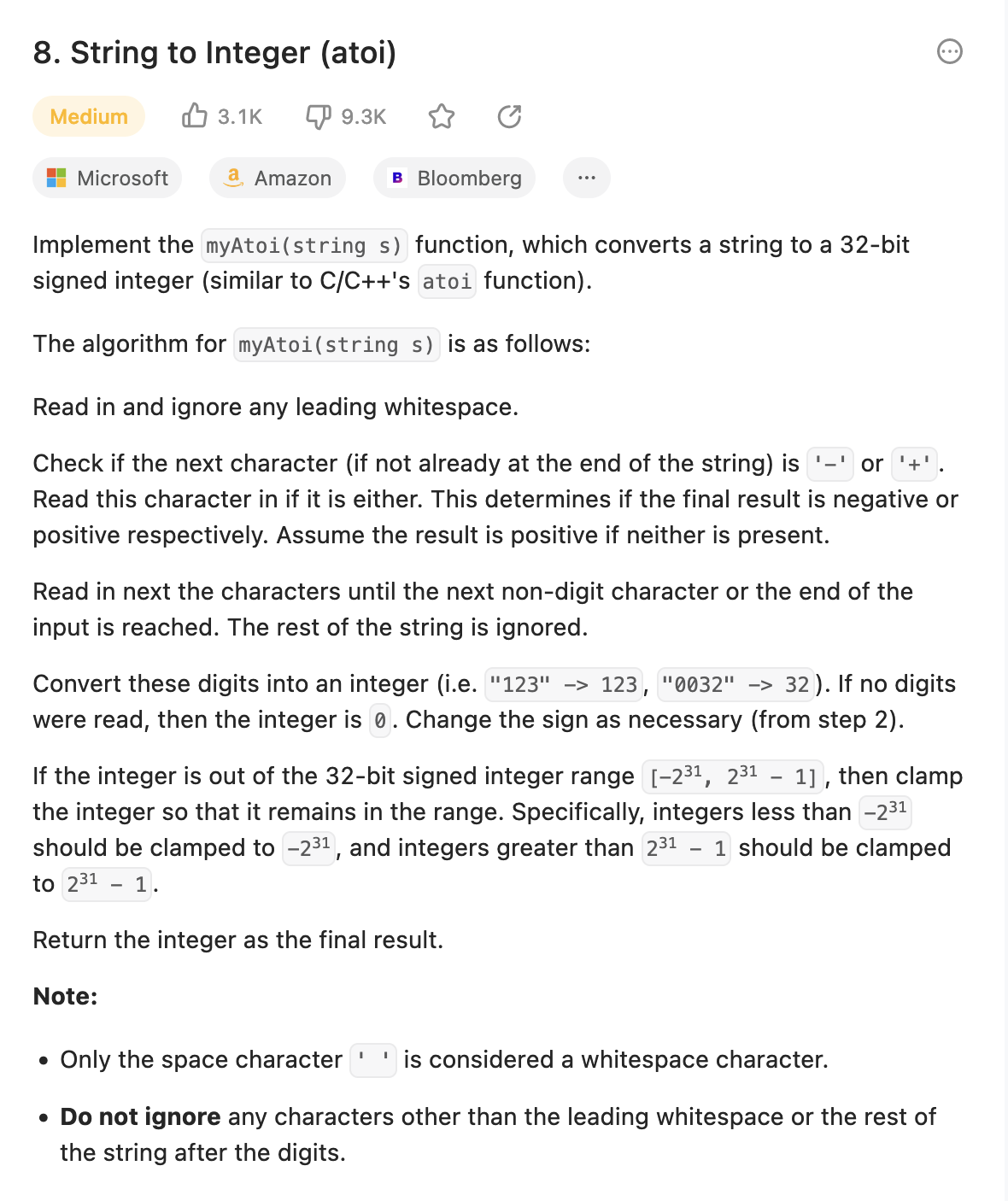
}

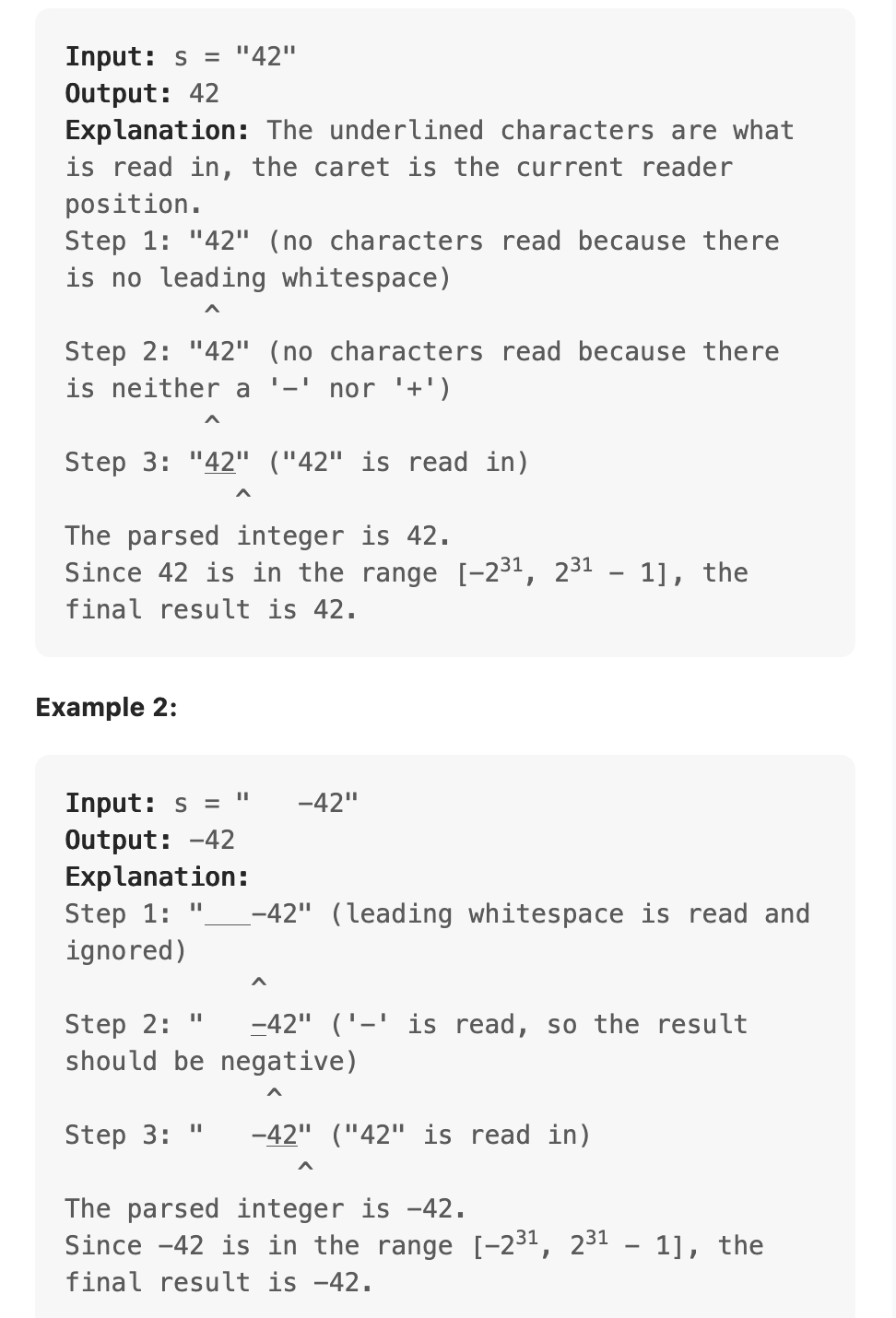
return result.toString();

}

}

# L8 String to Integer(atoi)





Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer (similar to C/C++'s atoi function).

The algorithm for myAtoi(string s) is as follows:

1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
3. Read in next the characters until the next non-digit character or the end of the input is reached. The rest of the string is ignored.
4. Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2).
5. If the integer is out of the 32-bit signed integer range [-231, 231 - 1], then clamp the integer so that it remains in the range. Specifically, integers less than -231 should be clamped to -231, and integers greater than 231 - 1 should be clamped to 231 - 1.
6. Return the integer as the final result.

**Note:**

* Only the space character ' ' is considered a whitespace character.
* **Do not ignore** any characters other than the leading whitespace or the rest of the string after the digits.

class Solution {

public int myAtoi(String s) {

// trim the trailing spaces

s = s.trim();

if (s.length() == 0) return 0;

//store the final answer in long so it cannot across the integer bounds

long answer = 0;

// "-" and "+" is read and decide the result should be negative or positive

for (int i = 0; i<s.length();i++){

char ch = s.charAt(i);

if (i == 0 && (ch == '-' || ch == '+')){

continue;

} else if(ch > '9' || ch <'0'){ // not an integer

break;

} else if (ch ==' '){ // case of white space

continue;

} else {

// out of bounds

if (answer < Integer.MIN\_VALUE || answer > Integer.MAX\_VALUE){

break;

}

// adding the answer

answer = (answer \* 10) + (ch - '0');

}

}

if (s.charAt(0) == '-'){

answer = -answer;

}

if (answer < Integer.MIN\_VALUE || answer > Integer.MAX\_VALUE){

return answer < Integer.MIN\_VALUE ? Integer.MIN\_VALUE: Integer.MAX\_VALUE;

}

return (int) answer;

}

}

# L10 Regular Expression Matching

Given an input string s and a pattern p, implement regular expression matching with support for '.' and '\*' where:

* '.' Matches any single character.​​​​
* '\*' Matches zero or more of the preceding element.

The matching should cover the **entire** input string (not partial).

<https://www.youtube.com/watch?v=KN22ZEpRTFY>

class Solution {

public boolean isMatch(String s, String p) {

boolean[][] dp = new boolean[s.length()+1][p.length()+1];

dp[0][0] = true;

for (int i = 1; i <= p.length(); i++){

if(p.charAt(i-1) == '\*' && dp[0][i-2]){

dp[0][i] = true;

}

}

for (int i =1; i <= s.length(); i++){

for (int j =1 ; j<= p.length(); j++){

if(s.charAt(i-1) == p.charAt(j-1) || p.charAt(j-1) == '.'){

dp[i][j] = dp[i-1][j-1];

} else if (p.charAt(j-1) == '\*'){

if (p.charAt(j-2) != s.charAt(i-1) && p.charAt(j-2) != '.'){

dp[i][j] = dp[i][j-2];

}else{

dp[i][j] = (dp[i][j-2] || dp[i][j-1] || dp[i-1][j]);

}

}

}

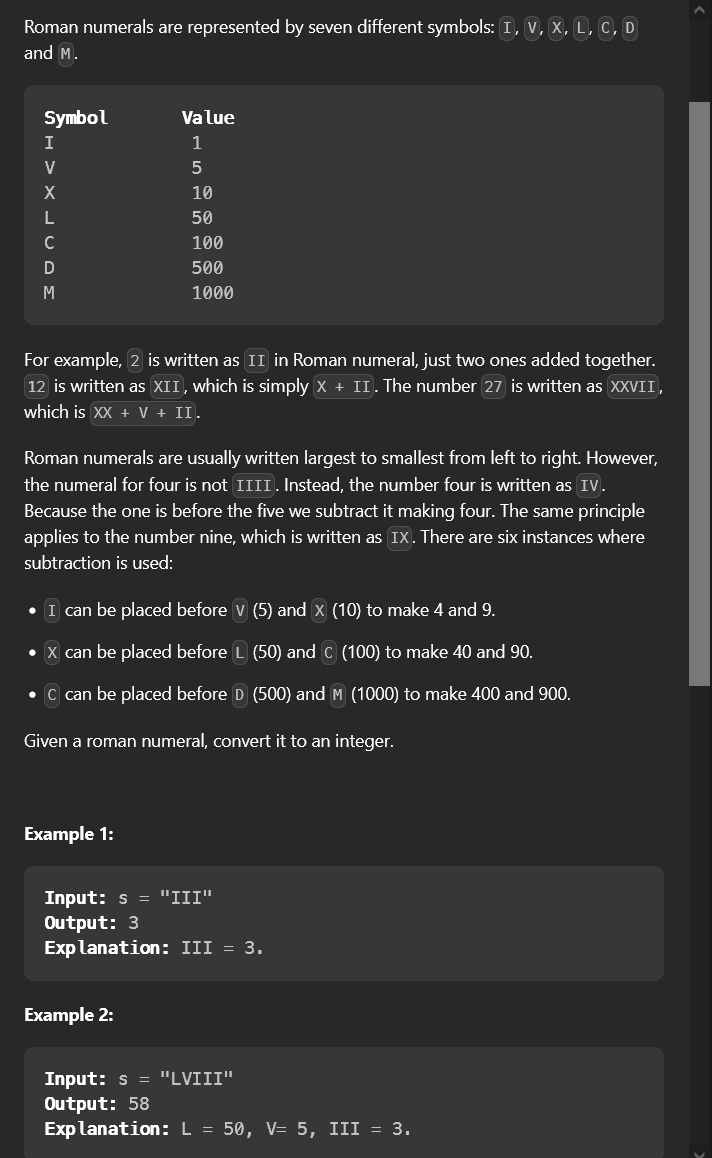
}

return dp[s.length()][p.length()];

}

}

# L13. Roman to Integer



Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

For example, 2 is written as II in Roman numerals, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

* I can be placed before V (5) and X (10) to make 4 and 9.
* X can be placed before L (50) and C (100) to make 40 and 90.
* C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Solution: