**[Minimum Window Substring](https://leetcode.com/problems/minimum-window-substring/)**

**import** java.util.HashMap;

**public** **class** MinimumWindowSubstring {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.***out***.println(*minWindow*("ADOBECODEBANC","ABC"));

}

**public** **static** String minWindow(String s, String t) {

**if**(s == **null** || t == **null**) {

**return** "";

}

HashMap<Character, Integer> dictionary = **new** HashMap<>(); //to track no of character in t required to search in s

**for**(**char** c : t.toCharArray()) {

dictionary.put(c, dictionary.getOrDefault(c, 0)+1); //map character in t with its occurrence

}

**int** required = dictionary.size();

**int** left = 0; //tracks left side of window

**int** right = 0; //tracks right side of window

**int**[] result = {0 , -1}; //tracks left boundary of window and minimum length of window at any point

HashMap<Character , Integer> map = **new** HashMap<>(); //to track no of character in s at any time

**int** formed = 0; //no of characters found in s that is same as in t

**while**(right < s.length()) {

**char** windowChar = s.charAt(right++);//starts checking and moving the window towards right

map.put(windowChar, map.getOrDefault(windowChar, 0)+1);

**if**(dictionary.containsKey(windowChar) && dictionary.get(windowChar).intValue() == map.get(windowChar).intValue()) {

formed++; //if no of character in s is same as that in t then one pattern is formed

}

**while**(left <= right && formed == required) { //to shrink the window from left towards right such that formed is same as required

**if**(result[1] == -1 || right - left < result[1]) { //if no minimum window fetched yet or any other min length window found

result[1] = right - left; //length of window

result[0] = left; //left boundary of window

}

**char** temp = s.charAt(left++);

map.put(temp, map.get(temp)-1); //start shrinking from left

**if**(dictionary.containsKey(temp) && dictionary.get(temp) > map.get(temp)) {

formed--; //reduce formed if while shrinking any required character has been removed

}

}

}

**return** result[1] == -1 ? "" : s.substring(result[0], result[0]+result[1]);

}

}

Time Complexity : O(n), n is length of s

Space Complexity : O(n+l), n is length of s and l is length of t