Lecture 23

Doyoun Kim

Overview

A goal of this exercise is to practice implementing specific functions in C++ to solve algorithmic problems.

In this exercise, you will implement three functions using priority queue and dynamic programming.

- **Minimum Edit Distance**: Calculate the minimum operations required to convert one string to another.
- **Merge K Sorted Linked Lists**: Combine multiple sorted lists into one sorted list.
- **Find Kth Largest Element**: Identify the Kth largest number in an unsorted array.

Edit distance

- **Goal**
  - Find the minimum number of operations required to convert one string (word1) to another string (word2).

```
Input: word1 = “horse”, word2 = “ros”
Output: 3
Explanation:
horse -> rorse (replace ‘h’ with ‘r’)
rorse -> rose (remove ‘r’)
rose -> ros (remove ‘e’)
```

Edit distance

- **Allowed Operations**:
  - Insert a character
  - Delete a character
  - Replace a character

- **Constraints:**
  - 0 <= word1.length, word2.length <= 500
  - word1 and word2 consist of lowercase English letters only.

Merge K sorted linked lists

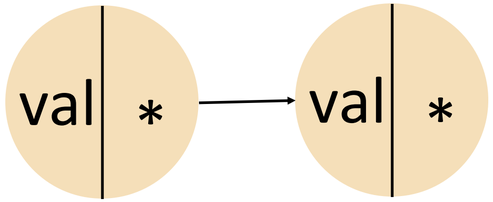
- **Goal**
  - Given an array of k linked lists, where each list is sorted in descending order, merge all lists into a single sorted linked list.

```
Input: lists = [[1,4,5], [1,3,4], [2,6]]
Output: [6,5,4,4,3,2,1,1]
Explanation: The linked-lists are:
[
   1->4->5,
   1->3->4,
   2, 6
]
Merging them into one sorted list:
6->5->4->4->3->2->1->1
```

```
Input: lists = []
Output: []
```

```
Input: lists = [[]]
Output: []
```

Merge K sorted linked lists




```
struct ListNode {
int val;
ListNode *next;
ListNode() : val(0), next(nullptr) {}
ListNode(int x) : val(x), next(nullptr) {}
ListNode(int x, ListNode *next) : val(x), next(next) {}
};
```

Find Kth largest

- **Goal**
  - Given an integer array nums and an integer k, return the kth largest element in the array. You should not use sort!

```
Input: nums = [3,2,1,5,6,4], k = 2
Output: 5
```

```
Input: nums = [3,2,3,1,2,4,5,5,6], k = 4
Output: 4
```

std::priority_queue

- A **priority queue** is a data structure that stores elements such that the **largest** element can be accessed immediately. (`#include <queue>`)

- **Key Features of std::priority_queue**
  - **Automatic Sorting**: As elements are added, the priority queue keeps the largest element at the top.
  - **Efficient Access**: Allows quick access to the largest (or smallest) element in constant time O(1).

Run exercise.cpp

- Find the TODO sections in exercise.cpp files and implement them correctly based on the instructions.
- There are 3 TODOs.
- $ g++ exercise.cpp -o exercise -std=c++11
- $ ./exercise

```
Minimum edit distance: 3
Minimum edit distance: 5
Merged list for [[1,4,5],[1,3,4],[2,6]]: [6 5 4 4 3 2 1 1]
Merged list for []:
Merged list for [[]]:
Merged list for [[1],[],[2,3]]: [3 2 1]
Output: 5
Output: 3
Output: 20
```

Levenshtein distance – Dynamic programming

- The minimum number of edits to transform a string to another.
- Insertion, deletion and replacement

string1 → string2

case 1: letter in string1 **==** letter in string2:

```
distance[r][c] = distance[r-1][c-1]
```

case 2: letter in string1 **!=** letter in string2:

```
distance[r][c] = 1 + min(distance[r][c-1], # deletion
                         distance[r-1][c-1], # replacement
                         distance[r-1][c]) # insertion
```

Levenshtein distance – Dynamic programming

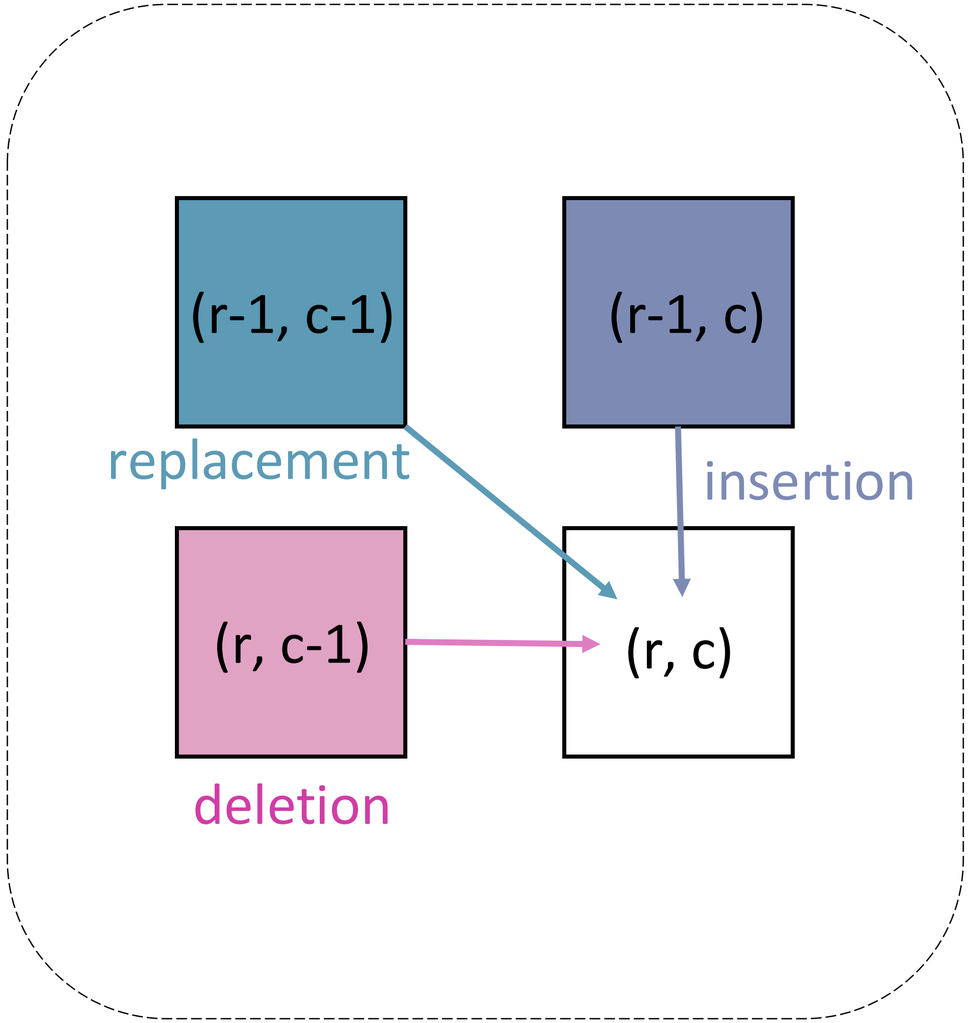
Base case:
String 1 is empty

Base case:
String 2 is empty

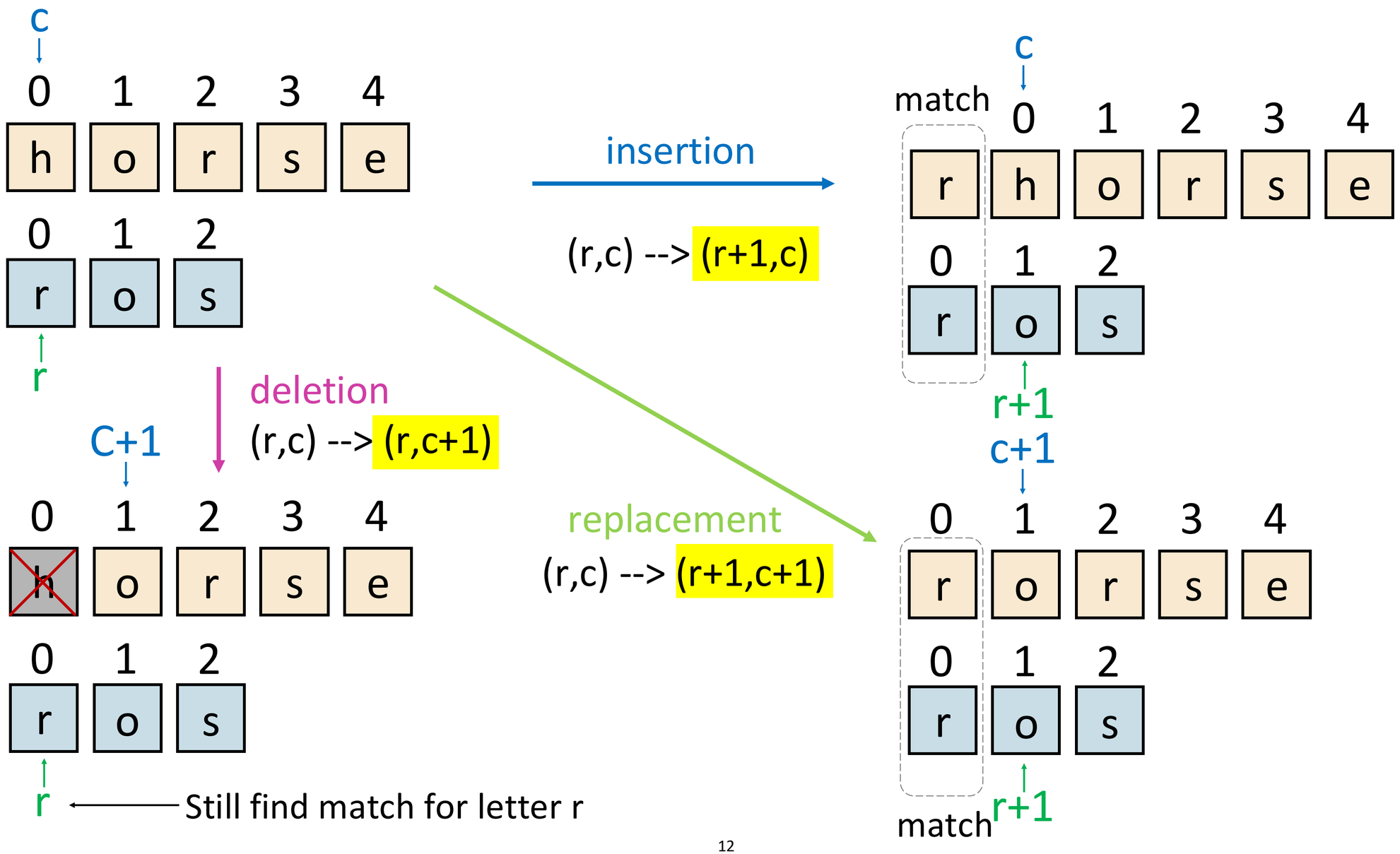
Distance matrix

		0	1	2	3	4	5
		“”	h	o	r	s	e
0	“”	0	1	2	3	4	5
1	r	1					
2	o	2					
3	s	3					

Resulting Levenshtein distance

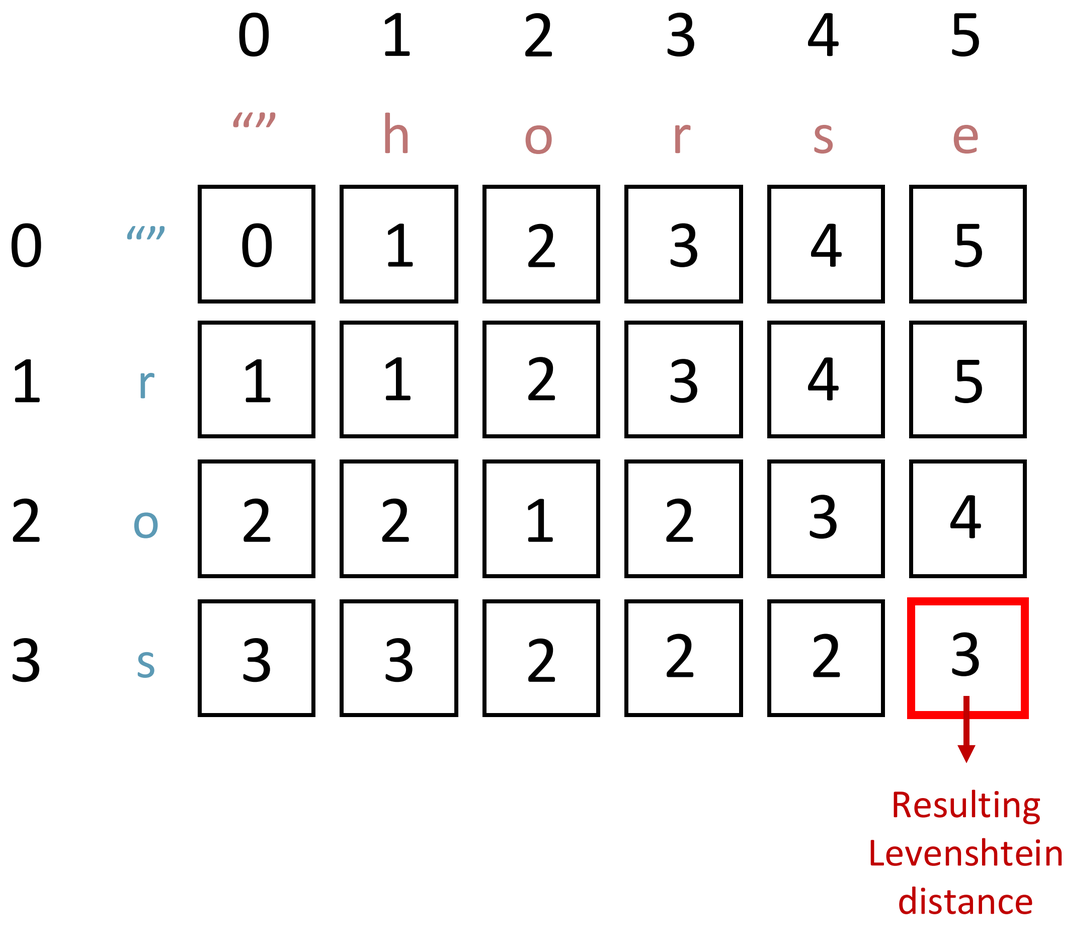


Levenshtein distance – Dynamic programming



Levenshtein distance – Dynamic programming

Distance matrix



Solutions – TODO 1

```cpp
int minDistance(const std::string& word1, const std::string& word2) {
    int m = word1.size();
    int n = word2.size();

    std::vector<std::vector<int>> dp(m + 1, std::vector<int>(n + 1, 0));

    for (int i = 0; i <= m; i++) {
        dp[i][0] = i;
    }
    for (int j = 0; j <= n; j++) {
        dp[0][j] = j;
    }

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (word1[i - 1] == word2[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1];
            } else {
                dp[i][j] = std::min({dp[i - 1][j] + 1, dp[i][j - 1] + 1, dp[i - 1][j - 1] + 1});
            }
        }
    }

    return dp[m][n];
}
```

Solutions – TODO 2

```
if(lists.empty()) {
    return nullptr;
}
std::priority_queue<int> q;

bool allNull = true;
for(ListNode* node: lists) {
    if(node != nullptr) {
        allNull = false;
        ListNode* head = node;
        while(head != nullptr) {
            q.push(head->val);
            head = head->next;
        }
    }

}
```

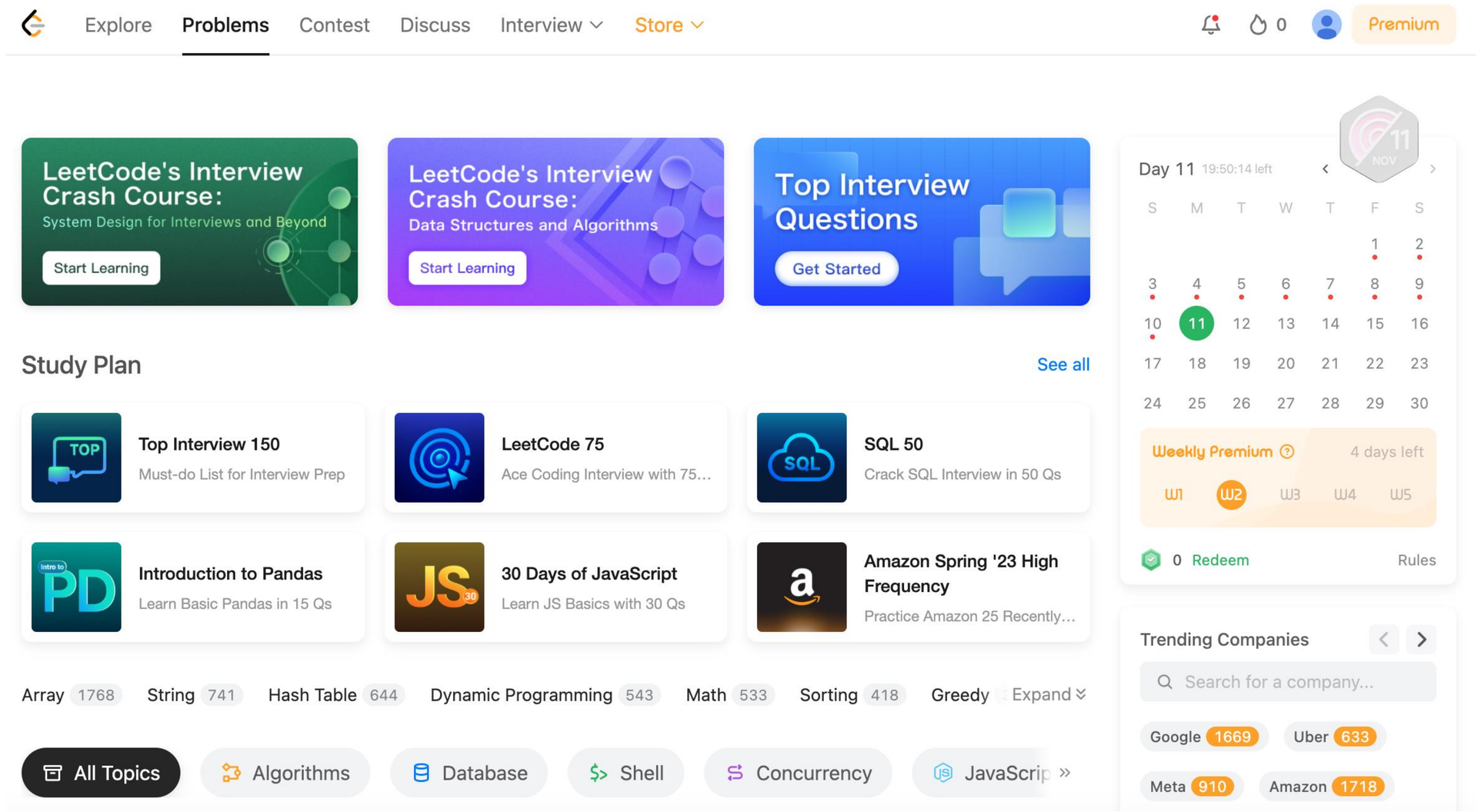
```
if(allNull) {
    return nullptr;
}

ListNode* ans;
int headVal = q.top();
q.pop();
ListNode* head = new ListNode(headVal);
ans = head;
while(!q.empty()) {
    ListNode* nextNode = new ListNode(q.top());
    q.pop();
    head->next = nextNode;
    head = head->next;
}
return ans;
```

Solutions – TODO 3

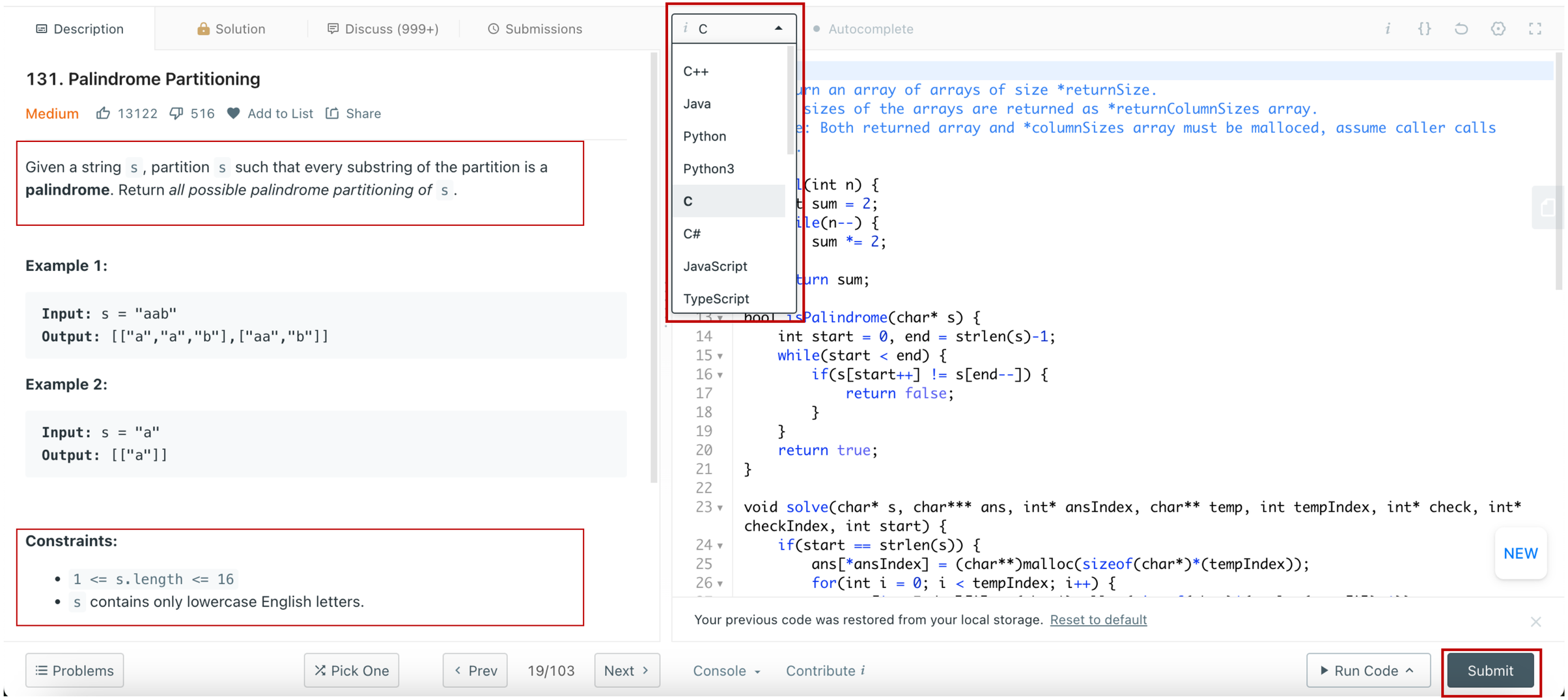
```cpp
int findKthLargest(std::vector<int>& nums, int k) {
    std::priority_queue<int> q;
    for(int num: nums) {
        q.push(num);
    }
    k--;
    while(k--) {
        q.pop();
    }
    return q.top();
}
```

Leetcode

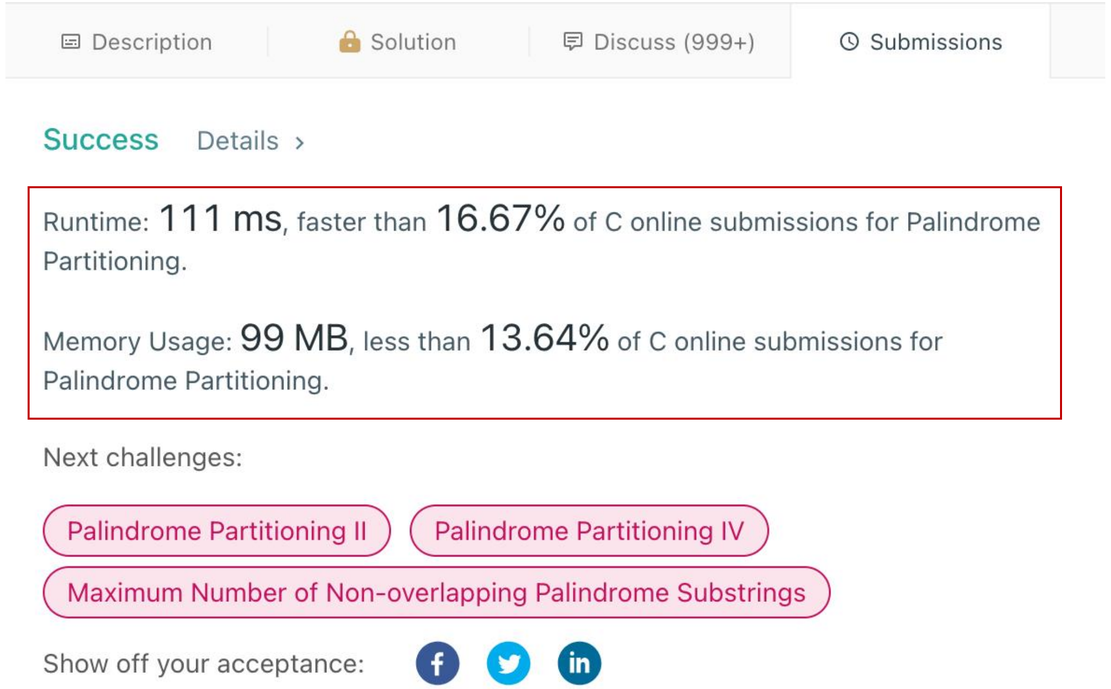


https://leetcode.com/problemset/

Leetcode


Description
Solution
Discuss (999+)
Submissions
C
Autocomplete
131. Palindrome Partitioning
Medium
13122
516
Add to List
Share
Given a string s , partition s such that every substring of the partition is a palindrome. Return all possible palindrome partitioning of s .
Example 1:
Input: s = "aab"
Output: [["a","a","b"],["aa","b"]]
Example 2:
Input: s = "a"
Output: [["a"]]
Constraints:
1 <= s.length <= 16
s contains only lowercase English letters.
C++
Java
Python
Python3
C
C#
JavaScript
TypeScript
rn an array of arrays of size *returnSize.
sizes of the arrays are returned as *returnColumnSizes array.
: Both returned array and *columnSizes array must be malloced, assume caller calls
(int n) {
sum = 2;
ile(n--) {
sum *= 2;
turn sum;
bool isPalindrome(char* s) {
int start = 0, end = strlen(s)-1;
while(start < end) {
if(s[start++] != s[end--]) {
return false;
}
}
return true;
}
void solve(char* s, char*** ans, int* ansIndex, char** temp, int tempIndex, int* check, int* checkIndex, int start) {
if(start == strlen(s)) {
ans[*ansIndex] = (char**)malloc(sizeof(char*)*(tempIndex));
for(int i = 0; i < tempIndex; i++) {
NEW
Your previous code was restored from your local storage. Reset to default
Problems
Pick One
Prev
19/103
Next
Console
Contribute
Run Code
Submit

Leetcode



Time Submitted	Status	Runtime	Memory	Language
11/11/2024 13:14	Accepted	111 ms	99 MB	c
03/30/2024 10:22	Accepted	404 ms	94.2 MB	c

Thank you