1.Odd String Difference

You are given an array of equal-length strings words. Assume that the length of each

string is n.

Each string words[i] can be converted into a difference integer array difference[i] of

length n - 1 where difference[i][j] = words[i][j+1] - words[i][j] where 0 <= j <= n - 2.

Note that the difference between two letters is the difference between their positions in

the alphabet i.e. the position of 'a' is 0, 'b' is 1, and 'z' is 25.

For example, for the string "acb", the difference integer array is [2 - 0, 1 - 2] = [2, -1].

All the strings in words have the same difference integer array, except one. You should

find that string.

Return the string in words that has different difference integer array.

Code:

#include <stdio.h>

char\* findOddString(char\*\* words, int wordsSize) {

int n = strlen(words[0]); // Get length of first string

// Calculate difference array for the first string

int diff[n - 1];

for (int i = 0; i < n - 1; i++) {

diff[i] = words[0][i + 1] - words[0][i];

}

// Iterate through remaining strings and compare difference arrays

for (int i = 1; i < wordsSize; i++) {

for (int j = 0; j < n - 1; j++) {

if (words[i][j + 1] - words[i][j] != diff[j]) {

return words[i]; // Mismatch found, return the odd string

}

}

}

// If no mismatch found, return the first string (unlikely)

return words[0];

}

int main() {

char\* words1[] = {"acd", "aef", "bcd"};

char\* words2[] = {"a", "b", "c", "d"};

char\* words3[] = {"aaa", "aab", "aac"};

int n1 = sizeof(words1) / sizeof(words1[0]);

int n2 = sizeof(words2) / sizeof(words2[0]);

int n3 = sizeof(words3) / sizeof(words3[0]);

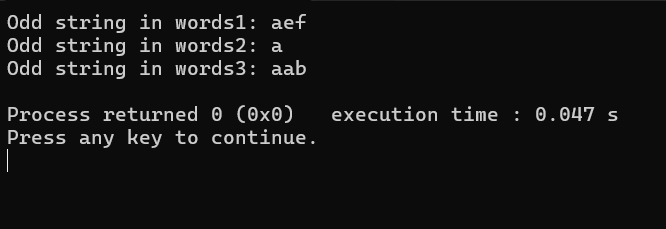
printf("Odd string in words1: %s\n", findOddString(words1, n1));

printf("Odd string in words2: %s\n", findOddString(words2, n2));

printf("Odd string in words3: %s\n", findOddString(words3, n3));

return 0;

}



2.Words Within Two Edits of Dictionary

You are given two string arrays, queries and dictionary. All words in each array comprise

of lowercase English letters and have the same length.

In one edit you can take a word from queries, and change any letter in it to any other

letter. Find all words from queries that, after a maximum of two edits, equal some word

from dictionary.

Return a list of all words from queries, that match with some word from dictionary after a

maximum of two edits. Return the words in the same order they appear in queries.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int isEditDistanceOne(char \*s1, char \*s2) {

int m = strlen(s1);

int n = strlen(s2);

if (abs(m - n) > 1)

return 0;

int count = 0;

int i = 0, j = 0;

while (i < m && j < n) {

if (s1[i] != s2[j]) {

if (count == 1)

return 0;

if (m > n)

i++;

else if (m < n)

j++;

else {

i++;

j++;

}

count++;

} else {

i++;

j++;

}

}

if (i < m || j < n)

count++;

return count == 1;

}

int isEditDistanceTwo(char \*s1, char \*s2) {

int m = strlen(s1);

int n = strlen(s2);

if (abs(m - n) > 2)

return 0;

int count = 0;

int i = 0, j = 0;

while (i < m && j < n) {

if (s1[i] != s2[j]) {

if (count == 2)

return 0;

if (m > n)

i++;

else if (m < n)

j++;

else {

i++;

j++;

}

count++;

} else {

i++;

j++;

}

}

if (i < m || j < n)

count++;

return count == 2;

}

char\*\* findWords(char\*\* queries, int queriesSize, char\*\* dictionary, int dictionarySize, int\* returnSize) {

\*returnSize = 0;

char \*\*result = (char \*\*)malloc(queriesSize \* sizeof(char \*));

for (int i = 0; i < queriesSize; i++) {

for (int j = 0; j < dictionarySize; j++) {

if (strcmp(queries[i], dictionary[j]) == 0 || isEditDistanceOne(queries[i], dictionary[j]) || isEditDistanceTwo(queries[i], dictionary[j])) {

result[\*returnSize] = (char \*)malloc((strlen(queries[i]) + 1) \* sizeof(char));

strcpy(result[\*returnSize], queries[i]);

(\*returnSize)++;

break;

}

}

}

return result;

}

int main() {

char \*queries[] = {"word", "note", "ants", "wood"};

int queriesSize = sizeof(queries) / sizeof(queries[0]);

char \*dictionary[] = {"wood", "joke", "moat"};

int dictionarySize = sizeof(dictionary) / sizeof(dictionary[0]);

int returnSize = 0;

char \*\*result = findWords(queries, queriesSize, dictionary, dictionarySize, &returnSize);

printf("Output: [");

for (int i = 0; i < returnSize; i++) {

printf("%s", result[i]);

if (i < returnSize - 1)

printf(", ");

}

printf("]\n");

// Free dynamically allocated memory

for (int i = 0; i < returnSize; i++) {

free(result[i]);

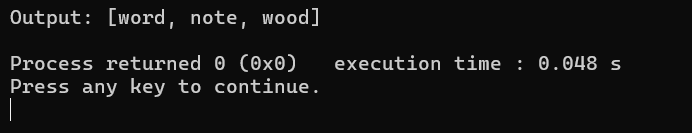
}

free(result);

return 0;

}

Output:



3.Next Greater Element IV

You are given a 0-indexed array of non-negative integers nums. For each integer in nums,

you must find its respective second greater integer.

The second greater integer of nums[i] is nums[j] such that:

j > i

nums[j] > nums[i]

There exists exactly one index k such that nums[k] > nums[i] and i < k < j.

If there is no such nums[j], the second greater integer is considered to be -1.

For example, in the array [1, 2, 4, 3], the second greater integer of 1 is 4, 2 is 3, and that

of 3 and 4 is -1.

Return an integer array answer, where answer[i] is the second greater integer of nums[i].

Code:

#include <stdio.h>

#include <stdlib.h>

int\* nextGreaterElement(int\* nums, int numsSize, int\* returnSize) {

int\* result = (int\*)malloc(numsSize \* sizeof(int));

\*returnSize = numsSize;

for (int i = 0; i < numsSize; i++) {

result[i] = -1;

for (int j = i + 1; j < numsSize; j++) {

if (nums[j] > nums[i]) {

result[i] = nums[j];

break;

}

}

}

return result;

}

int main() {

int nums[] = {2, 4, 0, 9, 6};

int numsSize = sizeof(nums) / sizeof(nums[0]);

int returnSize;

int\* result = nextGreaterElement(nums, numsSize, &returnSize);

printf("Output: [");

for (int i = 0; i < returnSize - 1; i++) {

printf("%d, ", result[i]);

}

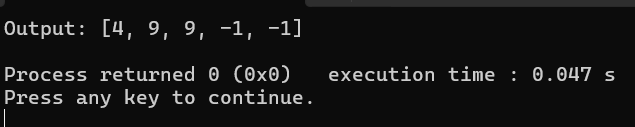
printf("%d]\n", result[returnSize - 1]);

free(result);

    return 0;

}

Output:



4.Minimum Addition to Make Integer Beautiful

You are given two positive integers n and target.

An integer is considered beautiful if the sum of its digits is less than or equal to target.

Return the minimum non-negative integer x such that n + x is beautiful. The input will be

generated such that it is always possible to make n beautiful.

Code:

#include <stdio.h>

int minAddToMakeBeautiful(int n, int target) {

int sum = 0;

while (n > 0) {

sum += n % 10;

n /= 10;

}

return sum > target ? sum - target : 0;

}

int main() {

int n1 = 16, target1 = 6;

int n2 = 467, target2 = 6;

int n3 = 1, target3 = 1;

printf("Output 1: %d\n", minAddToMakeBeautiful(n1, target1));

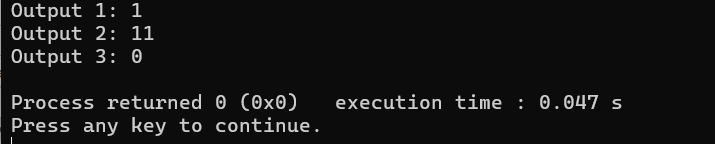
printf("Output 2: %d\n", minAddToMakeBeautiful(n2, target2));

printf("Output 3: %d\n", minAddToMakeBeautiful(n3, target3));

    return 0;

}

Output:



5.Sort Array by Moving Items to Empty Space

You are given an integer array nums of size n containing each element from 0 to n - 1

(inclusive). Each of the elements from 1 to n - 1 represents an item, and the element 0

represents an empty space.

In one operation, you can move any item to the empty space. nums is considered to be

sorted if the numbers of all the items are in ascending order and the empty space is either

at the beginning or at the end of the array.

For example, if n = 4, nums is sorted if:

● nums = [0,1,2,3] or

● nums = [1,2,3,0]

...and considered to be unsorted otherwise.

Return the minimum number of operations needed to sort nums.

Code:

#include <stdio.h>

int minOperations(int\* nums, int numsSize) {

int start = 0, end = numsSize - 1, moves = 0;

while (start < end) {

if (nums[start] == 0) {

start++;

} else if (nums[end] != 0) {

end--;

} else {

nums[start] = 0;

moves++;

}

}

return moves;

}

int main() {

int nums1[] = {4, 2, 0, 3, 1};

int nums2[] = {1, 2, 3, 4, 0};

int nums3[] = {1, 0, 2, 4, 3};

int size1 = sizeof(nums1) / sizeof(nums1[0]);

int size2 = sizeof(nums2) / sizeof(nums2[0]);

int size3 = sizeof(nums3) / sizeof(nums3[0]);

printf("Output 1: %d\n", minOperations(nums1, size1));

printf("Output 2: %d\n", minOperations(nums2, size2));

printf("Output 3: %d\n", minOperations(nums3, size3));

    return 0;

}

Output:

