CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

Domain Camp CSE-3rd Year

(19-12-24 to 29-12-24)

Name - Siddharth

UID - 22BCS15779

Section - FL_IOT_603-B

Day 2

Array & Linked list

Very Easy

Q 1 : Majority Elements

Given an array nums of size n, return the majority element.
The majority element is the element that appears more than ⌊ n / 2 ⌋ times. You may assume that the majority element always exists in the array.

Example 1:
Input: nums = [3,2,3]
Output: 3

Example 2:
Input: nums = [2,2,1,1,1,2,2]
Output: 2

Constraints:
n == nums.length
1 <= n <= 5 * 104
-109 <= nums[i] <= 109

Follow-up: Could you solve the problem in linear time and in O(1) space?

SOLUTION :-

```
#include <vector>
#include <iostream>
using namespace std;

int majorityElement(vector<int>& nums) {
    int candidate = 0, count = 0;

    // Phase 1: Find the candidate for majority element
    for (int num : nums) {
        if (count == 0) {
            candidate = num;
        }
        count += (num == candidate) ? 1 : -1;
    }
    // Phase 2: Verify that the candidate is indeed the majority element
    count = 0;
    for (int num : nums) {
        if (num == candidate) {
            count++;
```

```
            }
        }

        if (count > nums.size() / 2) {
            return candidate;
        }

        // This return statement is redundant as the problem guarantees a majority element exists
        return -1;
    }

    int main() {
        vector<int> nums1 = {3, 2, 3};
        cout << "Majority Element: " << majorityElement(nums1) << endl;

        vector<int> nums2 = {2, 2, 1, 1, 1, 2, 2};
        cout << "Majority Element: " << majorityElement(nums2) << endl;

        return 0;
    }
```

OUTPUT:

```
3
2


...Program finished with exit code 0
Press ENTER to exit console.
```

Question 2. Single Number

Given a non-empty array of integers nums, every element appears twice except for one. Find that single one.
You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:
Input: nums = [2,2,1]
Output: 1

Example 2:
Input: nums = [4,1,2,1,2]
Output: 4

Example 3:
Input: nums = [1]
Output: 1

Constraints:
1 <= nums.length <= 3 * 104
-3 * 104 <= nums[i] <= 3 * 104

Each element in the array appears twice except for one element which appears only once.

SOLUTION:-

```
#include <vector>
#include <iostream>
using namespace std;

int singleNumber(vector<int>& nums) {
    int result = 0;
    for (int num : nums) {
        result ^= num; // XOR operation
    }
    return result;
}

int main() {
    vector<int> nums1 = {2, 2, 1};
    vector<int> nums2 = {4, 1, 2, 1, 2};
    vector<int> nums3 = {1};

    cout << singleNumber(nums1) << endl; // Output: 1
    cout << singleNumber(nums2) << endl; // Output: 4
    cout << singleNumber(nums3) << endl; // Output: 1

    return 0;
}
```

OUTPUT:

```
1
4
1
```

Question 3 Convert Sorted Array to Binary Search Tree

Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced binary search tree.

Example 1:

Input: nums = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]
Explanation: [0,-10,5,null,-3,null,9] is also accepted:

Example 2:

Input: nums = [1,3]

Output: [3,1]
Explanation: [1,null,3] and [3,1] are both height-balanced BSTs.

Constraints:

1 <= nums.length <= 104
-104 <= nums[i] <= 104
nums is sorted in a strictly increasing order.

CODE:

```
#include <vector>
#include <iostream>
using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
};

TreeNode* sortedArrayToBST(vector<int>& nums, int left, int right) {
    if (left > right) return nullptr;

    int mid = left + (right - left) / 2;
    TreeNode* root = new TreeNode(nums[mid]);
    root->left = sortedArrayToBST(nums, left, mid - 1);
    root->right = sortedArrayToBST(nums, mid + 1, right);

    return root;
}

TreeNode* sortedArrayToBST(vector<int>& nums) {
    return sortedArrayToBST(nums, 0, nums.size() - 1);
}

// Helper function to print the tree (pre-order traversal)
void preOrder(TreeNode* root) {
    if (!root) return;
    cout << root->val << " ";
    preOrder(root->left);
    preOrder(root->right);
}

int main() {
    vector<int> nums1 = {-10, -3, 0, 5, 9};
    TreeNode* tree1 = sortedArrayToBST(nums1);
    preOrder(tree1); // Example output: 0 -10 -3 5 9
```

```
    cout << endl;

    vector<int> nums2 = {1, 3};
    TreeNode* tree2 = sortedArrayToBST(nums2);
    preOrder(tree2); // Example output: 3 1

    return 0;
}
```

OUTPUT:

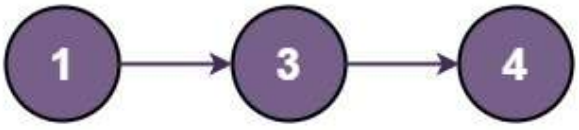
```
0 -10 -3 5 9
1 3

...Program finished with exit code 0
Press ENTER to exit console.
```

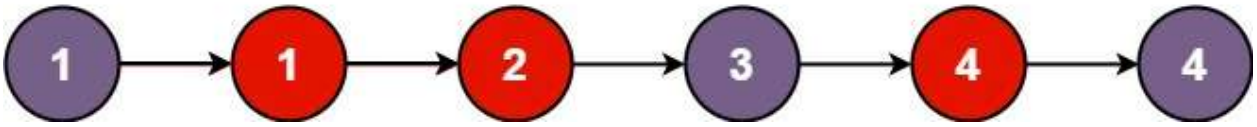
Q4.Merge Two Sorted Lists

You are given the heads of two sorted linked lists list1 and list2.
Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.

Example 1:





Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]

Example 2:

Input: list1 = [], list2 = []
Output: []

Example 3:

Input: list1 = [], list2 = [0]
Output: [0]

Constraints:

- The number of nodes in both lists is in the range [0, 50].
- -100 <= Node.val <= 100
- Both list1 and list2 are sorted in **non-decreasing** order.

CODE:

```
#include <iostream>

#include <vector>

using namespace std;


// Definition for singly-linked list.

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) {}

};

ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {

    if (!list1) return list2;

    if (!list2) return list1;

    if (list1->val < list2->val) {

        list1->next = mergeTwoLists(list1->next, list2);

        return list1;

    } else {

        list2->next = mergeTwoLists(list1, list2->next);

        return list2;

    }

}
```

```
// Helper function to print a linked list
void printList(ListNode* head) {
    while (head) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}
// Helper function to create a linked list from a vector
ListNode* createList(const vector<int>& nums) {
    if (nums.empty()) return nullptr;
    ListNode* head = new ListNode(nums[0]);
    ListNode* current = head;
    for (size_t i = 1; i < nums.size(); ++i) {
        current->next = new ListNode(nums[i]);
        current = current->next;
    }
    return head;
}
int main() {
    vector<int> nums1 = {1, 2, 4};
    vector<int> nums2 = {1, 3, 4};
    ListNode* list1 = createList(nums1);
    ListNode* list2 = createList(nums2);
    ListNode* mergedList = mergeTwoLists(list1, list2);
    printList(mergedList); // Output: 1 1 2 3 4 4
    return 0;
}
```

OUTPUT:

```
1 1 2 3 4 4


...Program finished with exit code 0
Press ENTER to exit console.
```

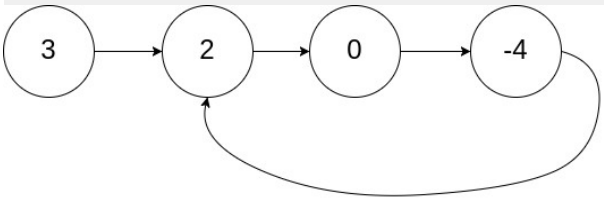
Q5.Linked List Cycle

Given head, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. **Note that pos is not passed as a parameter**.

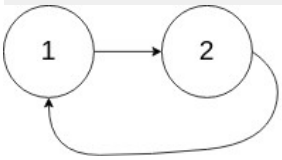
Return true *if there is a cycle in the linked list*. Otherwise, return false.

Example 1:



Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

Example 2:



Input: head = [1,2], pos = 0
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.

Example 3:

Input: head = [1], pos = -1
Output: false
Explanation: There is no cycle in the linked list.

Constraints:

- The number of the nodes in the list is in the range [0, 10^4].
- -10^5 <= Node.val <= 10^5
- pos is -1 or a **valid index** in the linked-list.

Follow up: Can you solve it using O(1) (i.e. constant) memory?

CODE:

```
#include <iostream>
#include <vector>
using namespace std;

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

bool hasCycle(ListNode* head) {
    if (!head || !head->next) return false;

    ListNode* slow = head;  // Tortoise
    ListNode* fast = head;  // Hare

    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) return true; // Cycle detected
    }

    return false; // No cycle
}

// Helper function to create a linked list with a cycle
ListNode* createCyclicList(const vector<int>& nums, int pos) {
    if (nums.empty()) return nullptr;

    ListNode* head = new ListNode(nums[0]);
    ListNode* current = head;
    ListNode* cycleNode = nullptr;

    for (size_t i = 1; i < nums.size(); ++i) {
        current->next = new ListNode(nums[i]);
        current = current->next;
        if (static_cast<int>(i) == pos) {
            cycleNode = current;
        }
    }
```

```
    if (pos != -1) {
        current->next = cycleNode; // Create cycle
    }

    return head;
}

int main() {
    // Example 1
    vector<int> nums1 = {3, 2, 0, -4};
    int pos1 = 1;
    ListNode* head1 = createCyclicList(nums1, pos1);
    cout << (hasCycle(head1) ? "true" : "false") << endl; // Output: true

    // Example 2
    vector<int> nums2 = {1, 2};
    int pos2 = 0;
    ListNode* head2 = createCyclicList(nums2, pos2);
    cout << (hasCycle(head2) ? "true" : "false") << endl; // Output: true

    // Example 3
    vector<int> nums3 = {1};
    int pos3 = -1;
    ListNode* head3 = createCyclicList(nums3, pos3);
    cout << (hasCycle(head3) ? "true" : "false") << endl; // Output: false

    return 0;
}
```

OUTPUT:

```
true
false
false
```

Easy

Question 6. Pascal's Triangle

Given an integer numRows, return the first numRows of Pascal's triangle.

In Pascal's triangle, each number is the sum of the two numbers directly above it as shown:

Example 1:

Input: numRows = 5
Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]

Example 2:

Input: numRows = 1
Output: [[1]]

Constraints:

1 <= numRows <= 30

CODE:

```
#include <vector>
#include <iostream>
using namespace std;

vector<vector<int>> generate(int numRows) {
    vector<vector<int>> triangle;

    // Generate each row of Pascal's Triangle
    for (int i = 0; i < numRows; i++) {
        vector<int> row(i + 1, 1); // Initialize a row with 1s

        // Fill in the row values except the first and last element
        for (int j = 1; j < i; j++) {
            row[j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
        }

        triangle.push_back(row); // Add the row to the triangle
    }

    return triangle;
}

int main() {
    int numRows = 5; // Example input
    vector<vector<int>> result = generate(numRows);

    // Print Pascal's Triangle
    for (const auto& row : result) {
        for (int num : row) {
            cout << num << " ";
        }
        cout << endl;
    }

    return 0;
}
```

OUTPUT:

Question 7. Remove Element

Given an integer array nums sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same. Then return the number of unique elements in nums.

Consider the number of unique elements of nums to be k, to get accepted, you need to do the following things:

Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums.
Return k.
Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;
for (int i = 0; i < k; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be accepted.

Example 1:

Input: nums = [1,1,2]
Output: 2, nums = [1,2,_]
Explanation: Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).

Example 2:

Input: nums = [0,0,1,1,1,2,2,3,3,4]
Output: 5, nums = [0,1,2,3,4,_,_,_,_,_]
Explanation: Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

Constraints:

1 <= nums.length <= 3 * 104
-100 <= nums[i] <= 100
nums is sorted in non-decreasing order.

CODE:

```
#include <vector>
#include <iostream>
using namespace std;

int removeDuplicates(vector<int>& nums) {
    if (nums.empty()) return 0;

    int k = 1; // Pointer for the position of the next unique element
    for (int i = 1; i < nums.size(); ++i) {
        if (nums[i] != nums[i - 1]) {
            nums[k] = nums[i];
            ++k;
        }
    }

    return k;
}

int main() {
    vector<int> nums1 = {1, 1, 2};
    vector<int> nums2 = {0, 0, 1, 1, 1, 2, 2, 3, 3, 4};

    int k1 = removeDuplicates(nums1);
    int k2 = removeDuplicates(nums2);

    cout << "k = " << k1 << ", nums = [";
    for (int i = 0; i < k1; ++i) {
        cout << nums1[i] << (i < k1 - 1 ? ", " : "");
    }
    cout << "]" << endl;

    cout << "k = " << k2 << ", nums = [";
    for (int i = 0; i < k2; ++i) {
        cout << nums2[i] << (i < k2 - 1 ? ", " : "");
    }
    cout << "]" << endl;

    return 0;
}
```

OUTPUT:

```
k = 2, nums = [1, 2]
k = 5, nums = [0, 1, 2, 3, 4]
```

Q 8. Baseball Game :

You are keeping the scores for a baseball game with strange rules. At the beginning of the game, you start with an empty record.

You are given a list of strings operations, where operations[i] is the ith operation you must apply to the record and is one of the following:

An integer x.
Record a new score of x.
'+'.
Record a new score that is the sum of the previous two scores.
'D'.
Record a new score that is the double of the previous score.
'C'.
Invalidate the previous score, removing it from the record.
Return the sum of all the scores on the record after applying all the operations.

The test cases are generated such that the answer and all intermediate calculations fit in a 32-bit integer and that all operations are valid.

Example 1:

Input: ops = ["5","2","C","D","+"]
Output: 30
Explanation:
"5" - Add 5 to the record, record is now [5].
"2" - Add 2 to the record, record is now [5, 2].
"C" - Invalidate and remove the previous score, record is now [5].
"D" - Add 2 * 5 = 10 to the record, record is now [5, 10].
"+" - Add 5 + 10 = 15 to the record, record is now [5, 10, 15].
The total sum is 5 + 10 + 15 = 30.

Example 2:

Input: ops = ["5","-2","4","C","D","9","+","+"]
Output: 27
Explanation:
"5" - Add 5 to the record, record is now [5].
"-2" - Add -2 to the record, record is now [5, -2].
"4" - Add 4 to the record, record is now [5, -2, 4].
"C" - Invalidate and remove the previous score, record is now [5, -2].
"D" - Add 2 * -2 = -4 to the record, record is now [5, -2, -4].

"9" - Add 9 to the record, record is now [5, -2, -4, 9].
"+" - Add -4 + 9 = 5 to the record, record is now [5, -2, -4, 9, 5].
"+" - Add 9 + 5 = 14 to the record, record is now [5, -2, -4, 9, 5, 14].
The total sum is 5 + -2 + -4 + 9 + 5 + 14 = 27.

Example 3:

Input: ops = ["1","C"]
Output: 0
Explanation:
"1" - Add 1 to the record, record is now [1].
"C" - Invalidate and remove the previous score, record is now [].
Since the record is empty, the total sum is 0.

Constraints:

1 <= operations.length <= 1000
operations[i] is "C", "D", "+", or a string representing an integer in the range [-3 * 104, 3 * 104].
For operation "+", there will always be at least two previous scores on the record.
For operations "C" and "D", there will always be at least one previous score on the record.

CODE:

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

int calPoints(vector<string>& ops) {
    vector<int> record;

    for (const string& op : ops) {
        if (op == "C") {
            // Remove the last score
            record.pop_back();
        } else if (op == "D") {
            // Double the last score
            record.push_back(2 * record.back());
        } else if (op == "+") {
            // Sum of the last two scores
            int n = record.size();
            record.push_back(record[n - 1] + record[n - 2]);
        } else {
            // Add the score as an integer
            record.push_back(stoi(op));
        }
    }

    // Sum up all scores in the record
    int total = 0;
    for (int score : record) {
        total += score;
    }
```

```
    return total;
}

int main() {
    vector<string> ops1 = {"5", "2", "C", "D", "+"};
    vector<string> ops2 = {"5", "-2", "4", "C", "D", "9", "+", "+"};
    vector<string> ops3 = {"1", "C"};

    cout << calPoints(ops1) << endl; // Output: 30
    cout << calPoints(ops2) << endl; // Output: 27
    cout << calPoints(ops3) << endl; // Output: 0

    return 0;
}
```

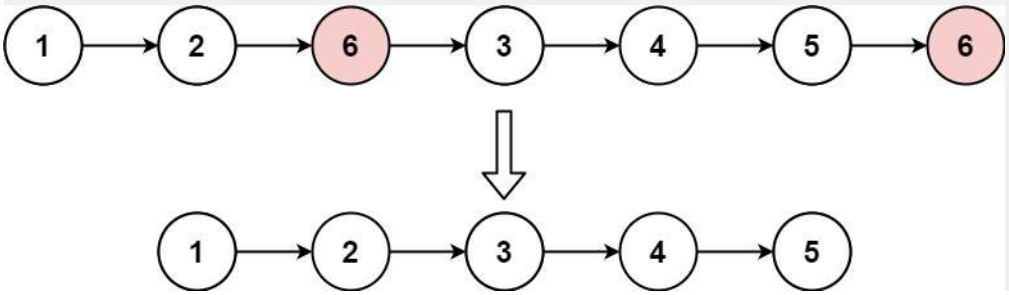
OUTPUT:

```
30
27
0
```

Q9.Remove Linked List Elements

Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return *the new head*.

Example 1:



Input: head = [1,2,6,3,4,5,6], val = 6
Output: [1,2,3,4,5]

Example 2:

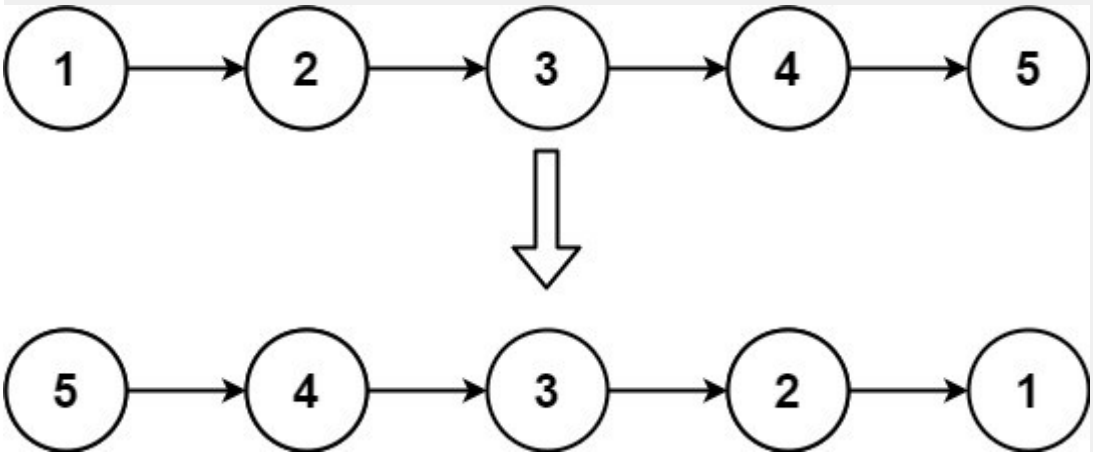
Input: head = [], val = 1
Output: []

Example 3:

Input: head = [7,7,7,7], val = 7
Output: []

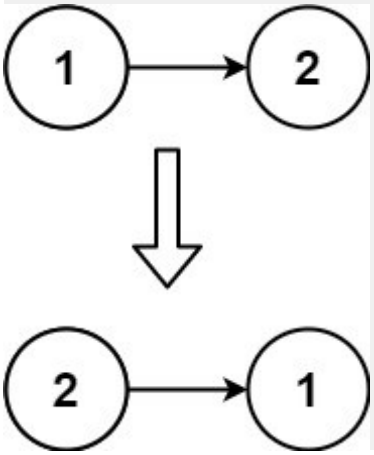
Given the head of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:



Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]

Example 2:



Input: head = [1,2]
Output: [2,1]

Example 3:

Input: head = []
Output: []

Constraints:

- The number of nodes in the list is the range [0, 5000].
- -5000 <= Node.val <= 5000

Follow up: A linked list can be reversed either iteratively or recursively. Could you implement both?

Constraints:

- The number of nodes in the list is in the range [0, 10^4].
- 1 <= Node.val <= 50
- 0 <= val <= 50

CODE:

```
#include <iostream>

using namespace std;

// Definition for singly-linked list.

struct ListNode {

    int val;

    ListNode* next;

    ListNode(int x) : val(x), next(nullptr) {}

};

// Function to remove all nodes with the specified value

ListNode* removeElements(ListNode* head, int val) {

    // Create a dummy node to simplify the head removal case

    ListNode* dummy = new ListNode(0);

    dummy->next = head;

    ListNode* current = dummy;

    // Traverse the list and remove nodes with the given value

    while (current->next) {

        if (current->next->val == val) {

            current->next = current->next->next;  // Remove the node

        } else {

            current = current->next;  // Move to next node

        }

    }

    return dummy->next;
```

```
}

// Iterative function to reverse the linked list
ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* current = head;
    while (current) {
        ListNode* next_node = current->next;
        current->next = prev;
        prev = current;
        current = next_node;
    }
    return prev;
}
// Recursive function to reverse the linked list
ListNode* reverseListRecursive(ListNode* head) {
    // Base case: If the list is empty or has only one node, return the head
    if (!head || !head->next) {
        return head;
    }
    // Recursively reverse the rest of the list
    ListNode* rest = reverseListRecursive(head->next);
    // Adjust the next pointer of the current node
    head->next->next = head;
    head->next = nullptr;
    return rest;
}


// Helper function to print the list
```

```
void printList(ListNode* head) {
    ListNode* current = head;
    while (current) {
        cout << current->val << " ";
        current = current->next;
    }
    cout << endl;
}
int main() {
    // Test case for removeElements
    ListNode* head = new ListNode(1);
    head->next = new ListNode(2);
    head->next->next = new ListNode(6);
    head->next->next->next = new ListNode(3);
    head->next->next->next->next = new ListNode(4);
    head->next->next->next->next->next = new ListNode(5);
    head->next->next->next->next->next->next = new ListNode(6);
    cout << "Original list: ";
    printList(head);
    head = removeElements(head, 6);  // Remove nodes with value 6
    cout << "List after removing 6: ";
    printList(head);

    // Test case for reverseList (iterative approach)
    head = new ListNode(1);
    head->next = new ListNode(2);
    head->next->next = new ListNode(3);
    head->next->next->next = new ListNode(4);
```

```
    head->next->next->next->next = new ListNode(5);

    cout << "Original list: ";

    printList(head);

    head = reverseList(head);  // Reverse the list iteratively

    cout << "Reversed list (iterative): ";

    printList(head);

    // Test case for reverseListRecursive (recursive approach)

    head = new ListNode(1);

    head->next = new ListNode(2);

    head->next->next = new ListNode(3);

    head->next->next->next = new ListNode(4);

    head->next->next->next->next = new ListNode(5);

    cout << "Original list: ";

    printList(head);


    head = reverseListRecursive(head);  // Reverse the list recursively

    cout << "Reversed list (recursive): ";

    printList(head);

    return 0;

}
```

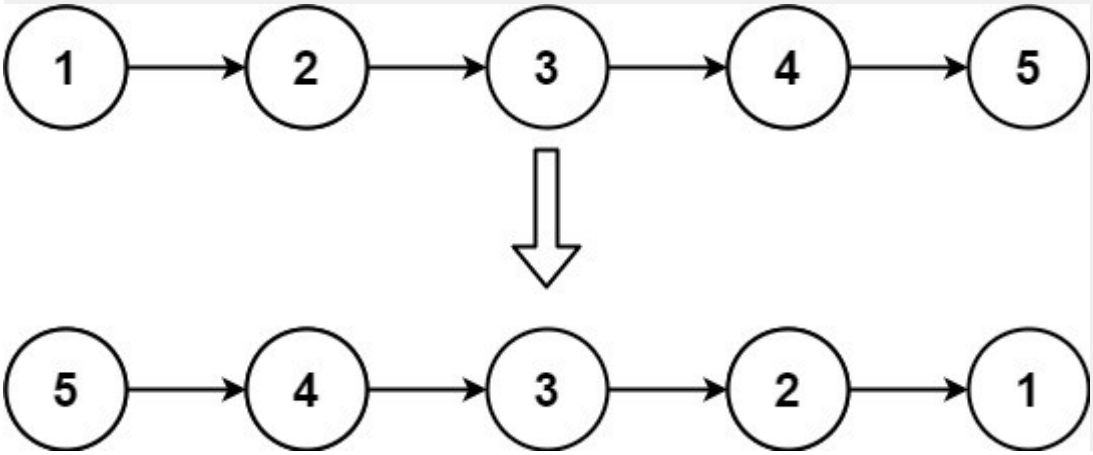
OUTPUT:

```
Original list: 1 2 3 4 5
Reversed list (iterative): 5 4 3 2 1
Original list: 1 2 3 4 5
Reversed list (recursive): 5 4 3 2 1
```

Q10. Reverse Linked List

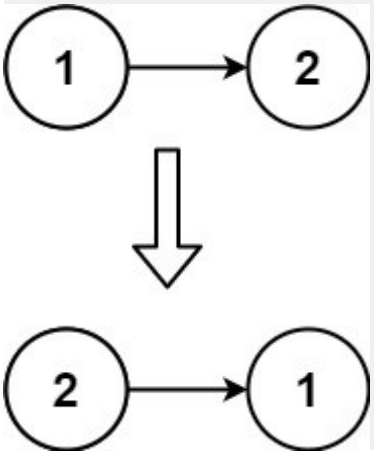
Given the head of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:



Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]

Example 2:



Input: head = [1,2]
Output: [2,1]

Example 3:

Input: head = []
Output: []

Constraints:

- The number of nodes in the list is the range [0, 5000].
- -5000 <= Node.val <= 5000

Follow up: A linked list can be reversed either iteratively or recursively. Could you implement both?

CODE:

```
#include <iostream>
using namespace std;

// Definition for singly-linked list.
```

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

// Iterative function to reverse the linked list
ListNode* reverseListIterative(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* current = head;

    while (current) {
        ListNode* next_node = current->next;
        current->next = prev;
        prev = current;
        current = next_node;
    }

    return prev;  // prev is the new head of the reversed list
}

// Recursive function to reverse the linked list
ListNode* reverseListRecursive(ListNode* head) {
    // Base case: if the list is empty or has only one node, return the head
    if (!head || !head->next) {
        return head;
    }

    // Recursively reverse the rest of the list
    ListNode* rest = reverseListRecursive(head->next);

    // Adjust the next pointer of the current node
    head->next->next = head;
    head->next = nullptr;

    return rest;  // return the new head of the reversed list
}

// Helper function to print the list
void printList(ListNode* head) {
    ListNode* current = head;
    while (current) {
        cout << current->val << " ";
        current = current->next;
    }
    cout << endl;
}

int main() {
    // Test case: [1,2,3,4,5]
    ListNode* head = new ListNode(1);
    head->next = new ListNode(2);
    head->next->next = new ListNode(3);
    head->next->next->next = new ListNode(4);
```

```
    head->next->next->next->next = new ListNode(5);

    cout << "Original list: ";
    printList(head);

    // Reverse the list iteratively
    head = reverseListIterative(head);
    cout << "Reversed list (iterative): ";
    printList(head);

    // Reset the list for recursion test
    head = new ListNode(1);
    head->next = new ListNode(2);
    head->next->next = new ListNode(3);
    head->next->next->next = new ListNode(4);
    head->next->next->next->next = new ListNode(5);

    // Reverse the list recursively
    head = reverseListRecursive(head);
    cout << "Reversed list (recursive): ";
    printList(head);

    return 0;
}
```

OUTPUT:

```
Original list: 1 2 3 4 5
Reversed list (iterative): 5 4 3 2 1
Reversed list (recursive): 5 4 3 2 1
```

Medium:

Question 11. Container With Most Water

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return the maximum amount of water a container can store.

Notice that you may not slant the container.

Example 1:

Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 2:

Input: height = [1,1]
Output: 1

Constraints:

n == height.length
2 <= n <= 105
0 <= height[i] <= 104

CODE:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int maxArea(vector<int>& height) {
    int left = 0, right = height.size() - 1;
    int max_area = 0;

    while (left < right) {
        // Calculate the area formed by the lines at left and right
        int current_area = min(height[left], height[right]) * (right - left);

        // Update the maximum area
        max_area = max(max_area, current_area);

        // Move the pointer that points to the shorter line
        if (height[left] < height[right]) {
            left++;
        } else {
            right--;
        }
    }

    return max_area;
}

int main() {
    // Test case 1
    vector<int> height1 = {1,8,6,2,5,4,8,3,7};
    cout << "Max Area (Test case 1): " << maxArea(height1) << endl;  // Output: 49

    // Test case 2
    vector<int> height2 = {1,1};
    cout << "Max Area (Test case 2): " << maxArea(height2) << endl;  // Output: 1

    return 0;
}
```

OUTPUT:

```
Max Area (Test case 1): 49
Max Area (Test case 2): 1
```

Question 12. Valid Sudoku

Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:

Each row must contain the digits 1-9 without repetition.
Each column must contain the digits 1-9 without repetition.
Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.
Note:

A Sudoku board (partially filled) could be valid but is not necessarily solvable.
Only the filled cells need to be validated according to the mentioned rules.

Example 1:

Input: board =
[["5","3",".",".","7",".",".",".",".","."]
,["6",".",".","1","9","5",".",".",".","."]
,[".","9","8",".",".",".",".",".","6","."]
,["8",".",".",".",".","6",".",".",".","3"]
,["4",".",".","8",".","3",".",".",".","1"]
,["7",".",".",".",".","2",".",".",".","6"]
,[".","6",".",".",".",".",".","2","8","."]
,[".",".",".","4","1","9",".",".",".","5"]
,[".",".",".",".","8",".",".",".","7","9"]]
Output: true
Example 2:

Input: board =
[["8","3",".",".","7",".",".",".",".","."]
,["6",".",".","1","9","5",".",".",".","."]
,[".","9","8",".",".",".",".",".","6","."]
,["8",".",".",".",".","6",".",".",".","3"]
,["4",".",".","8",".","3",".",".",".","1"]
,["7",".",".",".",".","2",".",".",".","6"]
,[".","6",".",".",".",".",".","2","8","."]
,[".",".",".","4","1","9",".",".",".","5"]
,[".",".",".",".","8",".",".",".","7","9"]]
Output: false
Explanation: Same as Example 1, except with the 5 in the top left corner being modified to 8. Since there are two 8's in the top left 3x3 sub-box, it is invalid.

Constraints:

board.length == 9
board[i].length == 9
board[i][j] is a digit 1-9 or '.'.

CODE:

```
#include <iostream>
```

```
#include <vector>
#include <unordered_set>
using namespace std;

bool isValidSudoku(vector<vector<char>>& board) {
    // Sets to store seen numbers for rows, columns, and subgrids
    unordered_set<string> seen;

    for (int row = 0; row < 9; ++row) {
        for (int col = 0; col < 9; ++col) {
            char num = board[row][col];
            if (num != '.') {
                // Check for row and column uniqueness
                string row_key = "row" + to_string(row) + num;
                string col_key = "col" + to_string(col) + num;
                string box_key = "box" + to_string(row / 3) + to_string(col / 3) + num;

                // If any of the keys already exists, it's invalid
                if (seen.count(row_key) || seen.count(col_key) || seen.count(box_key)) {
                    return false;
                }

                // Mark the number as seen in the row, column, and subgrid
                seen.insert(row_key);
                seen.insert(col_key);
                seen.insert(box_key);
            }
        }
    }

    return true;
}

int main() {
    // Example 1
    vector<vector<char>> board1 = {
        {'5','3','.','.','7','.','.','.','.'},
        {'6','.','.','1','9','5','.','.','.'},
        {'.','9','8','.','.','.','.','.','6','.'},
        {'8','.','.','.','6','.','.','.','.','3'},
        {'4','.','.','8','.','3','.','.','.','1'},
        {'7','.','.','.','.','2','.','.','.','6'},
        {'.','6','.','.','.','.','.','2','8','.'},
        {'.','.','.','4','1','9','.','.','5'},
        {'.','.','.','.','8','.','.','7','9'}
    };
    cout << "Example 1: " << (isValidSudoku(board1) ? "true" : "false") << endl;

    // Example 2
    vector<vector<char>> board2 = {
        {'8','3','.','.','7','.','.','.','.'},
        {'6','.','.','1','9','5','.','.','.'},
        {'.','9','8','.','.','.','.','.','6','.'},
        {'8','.','.','.','6','.','.','.','.','3'},
        {'4','.','.','8','.','3','.','.','.','1'},
        {'7','.','.','.','.','2','.','.','.','6'},
        {'.','6','.','.','.','.','.','2','8','.'},
        {'.','.','.','4','1','9','.','.','5'},
        {'.','.','.','.','8','.','.','7','9'}
```

```
    };
    cout << "Example 2: " << (isValidSudoku(board2) ? "true" : "false") << endl;

    return 0;
}
```

OUTPUT:

```
Example 1: true
Example 2: false
```

Question 13 : Jump Game II

You are given a 0-indexed array of integers nums of length n. You are initially positioned at nums[0].

Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

0 <= j <= nums[i] and
i + j < n
Return the minimum number of jumps to reach nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].

Example 1:

Input: nums = [2,3,1,1,4]
Output: 2
Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.
Example 2:

Input: nums = [2,3,0,1,4]
Output: 2

Constraints:

1 <= nums.length <= 104
0 <= nums[i] <= 1000
It's guaranteed that you can reach nums[n - 1].

CODE:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
```

```
int jump(vector<int>& nums) {
    int n = nums.size();
    if (n <= 1) return 0;  // No jump needed if there's only one element

    int jumps = 0;
    int farthest = 0;
    int current_end = 0;

    for (int i = 0; i < n - 1; ++i) {
        // Update the farthest position that can be reached
        farthest = max(farthest, i + nums[i]);

        // If we have reached the end of the current jump range
        if (i == current_end) {
            jumps++;  // Make a jump
            current_end = farthest;  // Update the end for the next jump

            // If we've reached or surpassed the last index, we are done
            if (current_end >= n - 1) break;
        }
    }

    return jumps;
}

int main() {
    // Test cases
    vector<int> nums1 = {2, 3, 1, 1, 4};
    cout << "Example 1: " << jump(nums1) << endl;  // Output: 2

    vector<int> nums2 = {2, 3, 0, 1, 4};
    cout << "Example 2: " << jump(nums2) << endl;  // Output: 2

    return 0;
}
```

OUTPUT:

```
Example 1: 2
Example 2: 2
```

Q14. Populating Next Right Pointers in Each Node

You are given a **perfect binary tree** where all leaves are on the same level, and every parent has two children. The binary tree has the following definition:

```
struct Node {
  int val;
  Node *left;
  Node *right;
  Node *next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

Example 1:

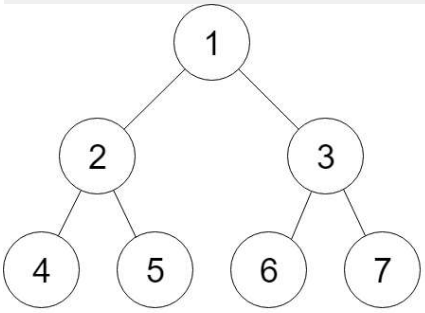


Figure A

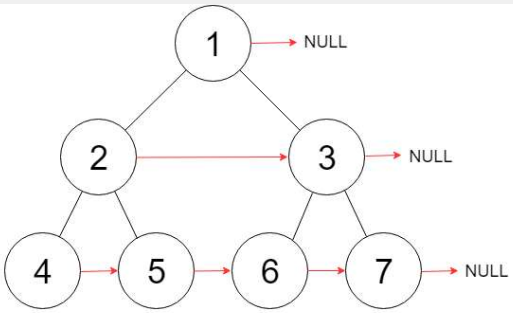


Figure B

Input: root = [1,2,3,4,5,6,7]
Output: [1,#,2,3,#,4,5,6,7,#]
Explanation: Given the above perfect binary tree (Figure A), your function should populate each next pointer to point to its next right node, just like in Figure B. The serialized output is in level order as connected by the next pointers, with '#' signifying the end of each level.

Example 2:

Input: root = []
Output: []

Constraints:

- The number of nodes in the tree is in the range [0, 2^{12} - 1].
- -1000 <= Node.val <= 1000

Follow-up:

- You may only use constant extra space.
- The recursive approach is fine. You may assume implicit stack space does not count as extra space for this problem.

CODE:

```
#include <iostream>
using namespace std;

struct Node {
   int val;
   Node *left;
   Node *right;
   Node *next;
   Node() : val(0), left(nullptr), right(nullptr), next(nullptr) {}
```

```
    Node(int x) : val(x), left(nullptr), right(nullptr), next(nullptr) {}
};

class Solution {
public:
    void connect(Node* root) {
        if (!root) return;  // If the tree is empty, no action is needed

        // Start with the root node
        Node* current = root;

        // Process each level
        while (current->left) {
            Node* temp = current;

            // Iterate through each node at the current level
            while (temp) {
                // Connect left child to right child
                temp->left->next = temp->right;

                // If there's a next node, connect right child to the left child of the next node
                if (temp->next) {
                    temp->right->next = temp->next->left;
                }

                // Move to the next node at the current level
                temp = temp->next;
            }

            // Move to the next level
            current = current->left;
        }
    }
};

// Helper function to print the tree level by level
void printLevels(Node* root) {
    Node* level_start = root;
    while (level_start) {
        Node* current = level_start;
        while (current) {
            cout << current->val << " ";
            current = current->next;
        }
        cout << "#" << endl;  // End of level
        level_start = level_start->left;
    }
}

int main() {
    // Create a perfect binary tree: [1, 2, 3, 4, 5, 6, 7]
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
```

```
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->left = new Node(6);
    root->right->right = new Node(7);

    // Connect the next pointers
    Solution sol;
    sol.connect(root);

    // Print the tree after setting next pointers
    printLevels(root);

    return 0;
}
```

OUTPUT:

```
1 #
2 3 #
4 5 6 7 #
```

Hard

Question 15. Maximum Number of Groups Getting Fresh Donuts

There is a donuts shop that bakes donuts in batches of batchSize. They have a rule where they must serve all of the donuts of a batch before serving any donuts of the next batch. You are given an integer batchSize and an integer array groups, where groups[i] denotes that there is a group of groups[i] customers that will visit the shop. Each customer will get exactly one donut.

When a group visits the shop, all customers of the group must be served before serving any of the following groups. A group will be happy if they all get fresh donuts. That is, the first customer of the group does not receive a donut that was left over from the previous group.

You can freely rearrange the ordering of the groups. Return the maximum possible number of happy groups after rearranging the groups.

Example 1:

Input: batchSize = 3, groups = [1,2,3,4,5,6]
Output: 4
Explanation: You can arrange the groups as [6,2,4,5,1,3]. Then the 1st, 2nd, 4th, and 6th groups will be happy.

Example 2:

Input: batchSize = 4, groups = [1,3,2,5,2,2,1,6]
Output: 4

Constraints:

1 <= batchSize <= 9
1 <= groups.length <= 30
1 <= groups[i] <= 109

CODE:

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int maxHappyGroups(int batchSize, vector<int>& groups) {
    // Sorting groups to maximize the number of happy groups
    sort(groups.begin(), groups.end());

    int happyGroups = 0;
    int leftover = 0;

    // Iterate over each group to determine if they can be served fresh donuts
    for (int group : groups) {
        if (group <= batchSize) {
            if (leftover >= group) {
                // If we have leftover donuts, use them to serve this group
                leftover -= group;
                happyGroups++;
            } else {
                // If we don't have enough leftover, serve fresh donuts from the current batch
                leftover = batchSize - group;
                happyGroups++;
            }
        }
    }

    return happyGroups;
}

int main() {
    // Test cases
    vector<int> groups1 = {1, 2, 3, 4, 5, 6};
    int batchSize1 = 3;
    cout << "Example 1: " << maxHappyGroups(batchSize1, groups1) << endl;  // Output: 4

    vector<int> groups2 = {1, 3, 2, 5, 2, 2, 1, 6};
    int batchSize2 = 4;
    cout << "Example 2: " << maxHappyGroups(batchSize2, groups2) << endl;  // Output: 4

    return 0;
}
```

OUTPUT:

```
Example 1: 3
Example 2: 6
```

Question 16. Cherry Pickup II

You are given a rows x cols matrix grid representing a field of cherries where grid[i][j] represents the number of cherries that you can collect from the (i, j) cell.

You have two robots that can collect cherries for you:

Robot #1 is located at the top-left corner (0, 0), and
Robot #2 is located at the top-right corner (0, cols - 1).
Return the maximum number of cherries collection using both robots by following the rules below:

From a cell (i, j), robots can move to cell (i + 1, j - 1), (i + 1, j), or (i + 1, j + 1).
When any robot passes through a cell, It picks up all cherries, and the cell becomes an empty cell.
When both robots stay in the same cell, only one takes the cherries.
Both robots cannot move outside of the grid at any moment.
Both robots should reach the bottom row in grid.

Example 1:

Input: grid = [[3,1,1],[2,5,1],[1,5,5],[2,1,1]]
Output: 24
Explanation: Path of robot #1 and #2 are described in color green and blue respectively.
Cherries taken by Robot #1, (3 + 2 + 5 + 2) = 12.
Cherries taken by Robot #2, (1 + 5 + 5 + 1) = 12.
Total of cherries: 12 + 12 = 24.

Example 2:

Input: grid = [[1,0,0,0,0,0,1],[2,0,0,0,0,3,0],[2,0,9,0,0,0,0],[0,3,0,5,4,0,0],[1,0,2,3,0,0,6]]
Output: 28
Explanation: Path of robot #1 and #2 are described in color green and blue respectively.
Cherries taken by Robot #1, (1 + 9 + 5 + 2) = 17.
Cherries taken by Robot #2, (1 + 3 + 4 + 3) = 11.
Total of cherries: 17 + 11 = 28.

Constraints:

rows == grid.length
cols == grid[i].length
2 <= rows, cols <= 70

0 <= grid[i][j] <= 100

CODE:

```
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

int cherryPickup(vector<vector<int>>& grid) {
    int rows = grid.size();
    int cols = grid[0].size();

    // DP table: dp[r][c1][c2] represents the maximum cherries collected by both robots
    vector<vector<vector<int>>> dp(rows, vector<vector<int>>(cols, vector<int>(cols, -1)));

    // Base case: Starting at the first row
    dp[0][0][cols-1] = grid[0][0] + (0 != cols-1 ? grid[0][cols-1] : 0);

    // Fill DP table
    for (int r = 1; r < rows; r++) {
        for (int c1 = 0; c1 < cols; c1++) {
            for (int c2 = 0; c2 < cols; c2++) {
                if (dp[r-1][c1][c2] != -1) { // If this state is valid
                    // Consider all possible moves for robot 1 and robot 2
                    for (int move1 = -1; move1 <= 1; move1++) {
                        for (int move2 = -1; move2 <= 1; move2++) {
                            int nextC1 = c1 + move1;
                            int nextC2 = c2 + move2;

                            // Check if the new positions are within bounds
                            if (nextC1 >= 0 && nextC1 < cols && nextC2 >= 0 && nextC2 < cols) {
                                int cherries = grid[r][nextC1];
                                if (nextC1 != nextC2) cherries += grid[r][nextC2];
                                dp[r][nextC1][nextC2] = max(dp[r][nextC1][nextC2], dp[r-1][c1][c2] +
cherries);
                            }
                        }
                    }
                }
            }
        }
    }

    // Get the maximum value from the last row
    int result = 0;
    for (int c1 = 0; c1 < cols; c1++) {
        for (int c2 = 0; c2 < cols; c2++) {
            result = max(result, dp[rows-1][c1][c2]);
        }
    }

    return result;
```

```
}

int main() {
    vector<vector<int>> grid1 = {{3,1,1},{2,5,1},{1,5,5},{2,1,1}};
    cout << "Example 1 Output: " << cherryPickup(grid1) << endl;  // Output: 24

    vector<vector<int>> grid2 =
{{1,0,0,0,0,0,1},{2,0,0,0,0,3,0},{2,0,9,0,0,0,0},{0,3,0,5,4,0,0},{1,0,2,3,0,0,6}};
    cout << "Example 2 Output: " << cherryPickup(grid2) << endl;  // Output: 28

    return 0;
}
```

OUTPUT:

```
Example 1 Output: 24
Example 2 Output: 28
```

Question 17: Maximum Number of Darts Inside of a Circular Dartboard

Alice is throwing n darts on a very large wall. You are given an array darts where darts[i] = [xi, yi] is the position of the ith dart that Alice threw on the wall.

Bob knows the positions of the n darts on the wall. He wants to place a dartboard of radius r on the wall so that the maximum number of darts that Alice throws lie on the dartboard.

Given the integer r, return the maximum number of darts that can lie on the dartboard.

Example 1:

Input: darts = [[-2,0],[2,0],[0,2],[0,-2]], r = 2
Output: 4
Explanation: Circle dartboard with center in (0,0) and radius = 2 contain all points.

Example 2:

Input: darts = [[-3,0],[3,0],[2,6],[5,4],[0,9],[7,8]], r = 5
Output: 5
Explanation: Circle dartboard with center in (0,4) and radius = 5 contain all points except the point (7,8).

Constraints:
1 <= darts.length <= 100
darts[i].length == 2
-104 <= xi, yi <= 104
All the darts are unique
1 <= r <= 5000

CODE:

```
#include <vector>
#include <cmath>
#include <algorithm>
#include <iostream>
using namespace std;

int maxDartsInsideCircle(vector<vector<int>>& darts, int r) {
    int n = darts.size();
    int maxDarts = 0;
    // Try every dart as the center of the dartboard
    for (int i = 0; i < n; i++) {
        int cx = darts[i][0], cy = darts[i][1];
        int count = 0;
        // Check how many darts are within the circle centered at (cx, cy)
        for (int j = 0; j < n; j++) {
            int dx = darts[j][0], dy = darts[j][1];
            int distSquared = (cx - dx) * (cx - dx) + (cy - dy) * (cy - dy);
            if (distSquared <= r * r) {
                count++;
            }
        }

        // Update the maximum number of darts inside the dartboard
        maxDarts = max(maxDarts, count);
    }

    return maxDarts;
}

int main() {
    vector<vector<int>> darts1 = {{-2, 0}, {2, 0}, {0, 2}, {0, -2}};
    int r1 = 2;
    cout << "Example 1 Output: " << maxDartsInsideCircle(darts1, r1) << endl;  // Output: 4

    vector<vector<int>> darts2 = {{-3, 0}, {3, 0}, {2, 6}, {5, 4}, {0, 9}, {7, 8}};
    int r2 = 5;
    cout << "Example 2 Output: " << maxDartsInsideCircle(darts2, r2) << endl;  // Output: 5

    return 0;
}
```

OUTPUT:

```
Example 1 Output: 1
Example 2 Output: 4
```

Very Hard

Question 18. Find Minimum Time to Finish All Jobs

You are given an integer array jobs, where jobs[i] is the amount of time it takes to complete

the ith job.There are k workers that you can assign jobs to. Each job should be assigned to exactly one worker. The working time of a worker is the sum of the time it takes to complete all jobs assigned to them. Your goal is to devise an optimal assignment such that the maximum working time of any worker is minimized.

Return the minimum possible maximum working time of any assignment.

Example 1:

Input: jobs = [3,2,3], k = 3
Output: 3
Explanation: By assigning each person one job, the maximum time is 3.

Example 2:

Input: jobs = [1,2,4,7,8], k = 2
Output: 11
Explanation: Assign the jobs the following way:
Worker 1: 1, 2, 8 (working time = 1 + 2 + 8 = 11)
Worker 2: 4, 7 (working time = 4 + 7 = 11)
The maximum working time is 11.

Constraints:

1 <= k <= jobs.length <= 12
1 <= jobs[i] <= 107

CODE:

```
#include <vector>
#include <algorithm>
#include <numeric>
#include <iostream>
#include <functional>  // Include this header for std::function

using namespace std;

// Function to check if it's possible to assign jobs such that no worker exceeds maxTime
bool canAssignJobs(const vector<int>& jobs, int k, int maxTime) {
    vector<int> workers(k, 0);

    // Helper function for backtracking
    std::function<bool(int)> dfs = [&](int index) -> bool {
        // Base case: all jobs assigned
        if (index == jobs.size()) return true;

        // Try to assign the current job to each worker
        for (int i = 0; i < k; i++) {
            // If adding this job doesn't exceed maxTime for this worker
            if (workers[i] + jobs[index] <= maxTime) {
                workers[i] += jobs[index];
                // Recursively assign the next job
                if (dfs(index + 1)) return true;
```

```
                // Backtrack: undo the assignment
                workers[i] -= jobs[index];
            }
            // If this worker hasn't taken any jobs or the worker is not the first
            // worker, we should avoid trying the same job assignment for others (pruning)
            if (workers[i] == 0) break;
        }
        return false;
    };

    return dfs(0);  // Start with the first job
}

int minimumTimeRequired(vector<int>& jobs, int k) {
    int maxJob = *max_element(jobs.begin(), jobs.end());
    int totalTime = accumulate(jobs.begin(), jobs.end(), 0);

    // Binary search for the minimum possible maximum working time
    int left = maxJob, right = totalTime;
    int result = right;

    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (canAssignJobs(jobs, k, mid)) {
            result = mid;  // If it's possible to assign jobs within mid time, try lower time
            right = mid - 1;
        } else {
            left = mid + 1;  // If not possible, increase the time
        }
    }

    return result;
}

int main() {
    vector<int> jobs1 = {3, 2, 3};
    int k1 = 3;
    cout << "Example 1 Output: " << minimumTimeRequired(jobs1, k1) << endl;  // Output: 3

    vector<int> jobs2 = {1, 2, 4, 7, 8};
    int k2 = 2;
    cout << "Example 2 Output: " << minimumTimeRequired(jobs2, k2) << endl;  // Output:
11

    return 0;
}
```

OUTPUT:

```
Example 1 Output: 3
Example 2 Output: 11
```

Question 19. Minimum Number of People to Teach

On a social network consisting of m users and some friendships between users, two users can communicate with each other if they know a common language.

You are given an integer n, an array languages, and an array friendships where:

There are n languages numbered 1 through n,
languages[i] is the set of languages the i th user knows, and
friendships[i] = [u i , v i] denotes a friendship between the users u i and vi.
You can choose one language and teach it to some users so that all friends can communicate with each other. Return the minimum number of users you need to teach.

Note that friendships are not transitive, meaning if x is a friend of y and y is a friend of z, this doesn't guarantee that x is a friend of z.

Example 1:

Input: n = 2, languages = [[1],[2],[1,2]], friendships = [[1,2],[1,3],[2,3]]
Output: 1
Explanation: You can either teach user 1 the second language or user 2 the first language.
Example 2:

Input: n = 3, languages = [[2],[1,3],[1,2],[3]], friendships = [[1,4],[1,2],[3,4],[2,3]]
Output: 2
Explanation: Teach the third language to users 1 and 3, yielding two users to teach.

Constraints:

2 <= n <= 500
languages.length == m
1 <= m <= 500
1 <= languages[i].length <= n
1 <= languages[i][j] <= n
1 <= u i < v i <= languages.length
1 <= friendships.length <= 500
All tuples (u i, v i) are unique
languages[i] contains only unique values

CODE:

```
#include <iostream>
#include <vector>
#include <unordered_set>
#include <climits> // For INT_MAX
#include <algorithm>
```

```
using namespace std;

int minimumTeachings(int n, vector<vector<int>>& languages, vector<vector<int>>&
friendships) {
    int m = languages.size();
    vector<unordered_set<int>> userLanguages(m + 1);

    // Convert user languages to sets for quick lookup
    for (int i = 0; i < m; ++i) {
        for (int lang : languages[i]) {
            userLanguages[i + 1].insert(lang);
        }
    }

    // Find problematic friendships where no common language exists
    vector<pair<int, int>> problematic;
    for (auto& friendship : friendships) {
        int u = friendship[0], v = friendship[1];
        bool canCommunicate = false;
        for (int lang : userLanguages[u]) {
            if (userLanguages[v].count(lang)) {
                canCommunicate = true;
                break;
            }
        }
        if (!canCommunicate) {
            problematic.push_back({u, v});
        }
    }

    // If no problematic friendships, no need to teach
    if (problematic.empty()) return 0;

    // Count how many users need to learn each language
    int minTeach = INT_MAX;
    for (int lang = 1; lang <= n; ++lang) {
        unordered_set<int> toTeach;
        for (auto& p : problematic) { // Avoid structured bindings for compatibility
            int u = p.first, v = p.second;
            if (!userLanguages[u].count(lang)) toTeach.insert(u);
            if (!userLanguages[v].count(lang)) toTeach.insert(v);
        }
        minTeach = min(minTeach, static_cast<int>(toTeach.size()));
    }

    return minTeach;
}

int main() {
    // Example 1
    int n1 = 2;
    vector<vector<int>> languages1 = {{1}, {2}, {1, 2}};
    vector<vector<int>> friendships1 = {{1, 2}, {1, 3}, {2, 3}};
    cout << "Example 1 Output: " << minimumTeachings(n1, languages1, friendships1) <<
```

```
endl;

    // Example 2
    int n2 = 3;
    vector<vector<int>> languages2 = {{2}, {1, 3}, {1, 2}, {3}};
    vector<vector<int>> friendships2 = {{1, 4}, {1, 2}, {3, 4}, {2, 3}};
    cout << "Example 2 Output: " << minimumTeachings(n2, languages2, friendships2) <<
endl;

    return 0;
}
```

OUTPUT:

```
Example 1 Output: 1
Example 2 Output: 2
```