DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

DOMAIN WINTER CAMP WORKSHEET

DAY-2 (20/12/2024)

Student Name :- Pratham Kapoor **University ID :- 22BCS10732**

Branch :- B.E. (C.S.E.) **Section/Group :- FL_ 603-A**

Problem-1 :- 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. **(Very Easy)**

Source Code

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

int majorityElement(vector<int>& nums) {
    int candidate = 0, count = 0;
    for (int num : nums) {
        if (count == 0) {
            candidate = num;
        }
        count += (num == candidate) ? 1 : -1;
    }
    return candidate;
}

int main() {
    int n;
    cout << "Enter the Size of the Array :- ";
    cin >> n;
```

```
    vector<int> nums(n);
    cout << "Enter the Elements of the Array :- ";
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }
    int result = majorityElement(nums);
    cout << "\nThe Majority Element is " << result << endl;
    return 0;
}
```

Output

```
Enter the Size of the Array :- 3
Enter the Elements of the Array :- 3 2 3

The Majority Element is 3
```

Problem-2 :- 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. **(Easy)**

Source Code

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

vector<vector<int>>generate(int numRows){
    vector<vector<int>>triangle;
    for(int i=0;i<numRows;i++){
        vector<int>row(i+1,1);
        for(int j=1;j<i;j++){
```

```
            row[j]=triangle[i-1][j-1]+triangle[i-1][j];
        }
        triangle.push_back(row);
    }
    return triangle;
}

int main(){
    int numRows;
    cout<<"Enter the Number of Rows :- ";
    cin>>numRows;
    cout<<endl;
    vector<vector<int>>triangle=generate(numRows);
    for(auto row:triangle){
        for(int num:row){
            cout<<num<<" ";
        }
        cout<<endl;
    }
    return 0;
}
```

Output

```
Enter the Number of Rows :- 5
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Problem-3 **:-** 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. **(Very Easy)**

Source Code

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

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

int main() {
    int n;
    cout<<"Enter the Number of Elements :- ";
    cin>>n;
    vector<int> nums(n);
    cout<<"Enter the Elements :- ";
    for(int i=0;i<n;i++) {
        cin>>nums[i];
    }

    int result=singleNumber(nums);
    cout<<"\nThe Single Number is "<<result<<endl;
```

```
    return 0;
}
```

Output

```
Enter the Number of Elements :- 3
Enter the Elements :- 2 2 1

The Single Number is 1
```

Problem-4 :- 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. **(Very Easy)**

Source Code

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

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

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;
    }
}

void printList(ListNode* head) {
    while (head != NULL) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}

ListNode* createList(int arr[], int size) {
    ListNode* head = new ListNode(arr[0]);
    ListNode* current = head;
    for (int i = 1; i < size; ++i) {
        current->next = new ListNode(arr[i]);
        current = current->next;
    }
    return head;
}

int main() {
    int n1, n2;

    cout << "Enter the Size of First List :- ";
    cin >> n1;
    int arr1[n1];
    cout << "Enter the Elements of First Sorted List :- ";
```

```
    for (int i = 0; i < n1; i++) {
        cin >> arr1[i];
    }

    cout << "Enter the Size of Second List :- ";
    cin >> n2;
    int arr2[n2];
    cout << "Enter the Elements of Second Sorted List :- ";
    for (int i = 0; i < n2; i++) {
        cin >> arr2[i];
    }

    ListNode* list1 = createList(arr1, n1);
    ListNode* list2 = createList(arr2, n2);
    ListNode* mergedList = mergeTwoLists(list1, list2);
    cout << "\nThe Merged Sorted List is ";
    printList(mergedList);
    return 0;
}
```

Output

```
Enter the Size of First List :- 3
Enter the Elements of First Sorted List :- 1 2 4
Enter the Size of Second List :- 3
Enter the Elements of Second Sorted List :- 1 3 4

The Merged Sorted List is 1 1 2 3 4 4
```

Problem-5 :- 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. **(Very Easy)**

Source Code

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

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

bool hasCycle(ListNode *head) {
    if(head == NULL) return false;
    ListNode *slow = head;
    ListNode *fast = head;

    while(fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;

        if(slow == fast) {
            return true;
        }
    }
    return false;
}

int main() {
    int n, pos;
```

```
cout << "Enter the Number of Nodes :- ";
cin >> n;

int val;
ListNode *head = NULL;
ListNode *temp = NULL;

for(int i = 0; i < n; i++) {
   cout << "Enter the Value for the Node " << i + 1 << " - ";
   cin >> val;
   ListNode *newNode = new ListNode(val);

   if(head == NULL) {
      head = newNode;
      temp = head;
   } else {
      temp->next = newNode;
      temp = temp->next;
   }
}

cout << "\nEnter the Position to Create Cycle :- ";
cin >> pos;

if(pos > 0) {
   temp = head;
   ListNode *cycleNode = NULL;

   for(int i = 1; i < pos; i++) {
      temp = temp->next;
   }
   cycleNode = temp;
```

```
        while(temp->next != NULL) {
            temp = temp->next;
        }

        temp->next = cycleNode;
    }

    if(hasCycle(head)) {
        cout << "\nTrue" << endl;
    } else {
        cout << "\nFalse" << endl;
    }

    return 0;
}
```

Output

```
Enter the Number of Nodes :- 4
Enter the Value for the Node 1 - 3
Enter the Value for the Node 2 - 2
Enter the Value for the Node 3 - 0
Enter the Value for the Node 4 - -4

Enter the Position to Create Cycle :- 1

True
```

Problem-6 **:-** 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. **(Easy)**

Source Code

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

int main() {
    int n;
    cout << "Enter the Number of Operations :- ";
    cin >> n;

    vector<string> ops(n);
    cout << "Enter the Operations - ";
    for(int i=0;i<n;i++) {
        cin >> ops[i];
    }

    vector<int> record;

    for(int i=0;i<n;i++) {
        if(ops[i]=="C") {
            if(!record.empty()) {
                record.pop_back();
            }
        } else if(ops[i]=="D") {
            if(!record.empty()) {
```

```
            record.push_back(2*record.back());
        }
    } else if(ops[i]=="+") {
        if(record.size()>=2) {
            record.push_back(record[record.size()-1]+record[record.size()-2]);
        }
    } else {
        record.push_back(stoi(ops[i]));
    }
}
int sum=0;
for(int i=0;i<record.size();i++) {
    sum+=record[i]; }
cout << "\nThe Sum of all Scores is " << sum << endl;
return 0;
}
```

Output

```
Enter the Number of Operations :- 5
Enter the Operations - 5 2 C D +

The Sum of all Scores is 30
```

Problem-7 :- 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. **(Easy)**

Source Code

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

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

ListNode* removeElements(ListNode* head, int val) {
    while(head!=NULL && head->val==val) {
        head=head->next;
    }
    ListNode* current=head;
    while(current!=NULL && current->next!=NULL) {
        if(current->next->val==val) {
            current->next=current->next->next;
        } else {
            current=current->next;
        }
    }
    return head; }

void printList(ListNode* head) {
    cout << "[";
    while(head!=NULL) {
        cout << head->val;
        if(head->next!=NULL) cout << ",";
        head=head->next;
    }
    cout << "]"; }
int main() {
    int n, val;
    cout << "Enter the Number of Elements in the Linked List :- ";
    cin >> n;
```

```
    cout << "Enter the Elements of the Linked List - ";
    ListNode* head=NULL;
    ListNode* tail=NULL;
    for(int i=0; i<n; i++) {
        int x;
        cin >> x;
        ListNode* newNode=new ListNode(x);
        if(head==NULL) {
            head=newNode;
            tail=newNode;
        } else {
            tail->next=newNode;
            tail=newNode;
        }
    }
    cout << "\nEnter the Value to Remove :- ";
    cin >> val;
    head=removeElements(head, val);
    cout << "\nThe Linked List after Removing the Element is ";
    printList(head);
    return 0;
}
```

Output

```
Enter the Number of Elements in the Linked List :- 7
Enter the Elements of the Linked List - 1 2 6 3 4 5 6

Enter the Value to Remove :- 6

The Linked List after Removing the Element is [1,2,3,4,5]
```

Problem-8 :- Given the head of a singly linked list, reverse the list and return the reversed list. **(Easy)**

Source Code

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

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

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;
}

ListNode* createLinkedList() {
 int n;
 cout<<"Enter the Number of Elements in the Linked List :- ";
 cin>>n;
 ListNode* head=nullptr;
```

```
 ListNode* tail=nullptr;
 for(int i=0;i<n;i++) {
  int val;
  cout<<"Enter the Value for Element - ";
  cin>>val;
  ListNode* new_node=new ListNode(val);
  if(!head) {
   head=new_node;
   tail=head;
  } else {
   tail->next=new_node;
   tail=new_node;
  }
 }
 return head;
}

void printLinkedList(ListNode* head) {
 while(head) {
  cout<<head->val<<" ";
  head=head->next;
 }
 cout<<endl;
}

int main() {
 ListNode* head=createLinkedList();
 head=reverseList(head);
 cout<<"\nReversed Linked List :- ";
 printLinkedList(head);
 return 0;
}
```

Output

```
Enter the Number of Elements in the Linked List :- 5
Enter the Value for Element - 1
Enter the Value for Element - 2
Enter the Value for Element - 3
Enter the Value for Element - 4
Enter the Value for Element - 5

Reversed Linked List :- 5 4 3 2 1
```

Problem-9 :- 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. **(Medium)**

Source Code

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

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

    while (left < right) {
        int area = min(height[left], height[right]) * (right - left);
        max_area = max(max_area, area);
        if (height[left] < height[right]) {
```

```
            left++;
        } else {
            right--;
        }
    }

    return max_area; }

int main() {
    int n;
    cout << "Enter the Number of Elements in the Height Array :- ";
    cin >> n;

    vector<int> height(n);
    cout << "Enter the Heights - ";
    for (int i = 0; i < n; i++) {
        cin >> height[i]; }

    cout << "\nThe Maximum Water a Container can Store is " << maxArea(height) << endl;
    return 0;
}
```

Output

```
Enter the Number of Elements in the Height Array :- 9
Enter the Heights - 1 8 6 2 5 4 8 3 7

The Maximum Water a Container can Store is 49
```

Problem-10 :- 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]. **(Medium)**

Source Code

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

int jump(vector<int>& nums) {
    int jumps = 0;
    int current_end = 0;
    int farthest = 0;
    int n = nums.size();

    for (int i = 0; i < n - 1; i++) {
        farthest = max(farthest, i + nums[i]);

        if (i == current_end) {
            jumps++;
            current_end = farthest;
        }
    }

    return jumps;
}

int main() {
    int n;
```

```
    cout << "Enter the Number of Elements in the Array :- ";
    cin >> n;

    vector<int> nums(n);
    cout << "Enter the Elements of the Array - ";
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }

    cout << "\nThe Minimum Number of Jumps to Reach the Last Index are "
<< jump(nums) << endl;
    return 0;
}
```

Output

```
Enter the Number of Elements in the Array :- 5
Enter the Elements of the Array - 2 3 1 1 4

The Minimum Number of Jumps to Reach the Last Index are 2
```

Problem-11 :- 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. **(Medium)**

Source Code

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

struct Node{
   int val;
   Node* left;
   Node* right;
   Node* next;
   Node(int x):val(x),left(NULL),right(NULL),next(NULL){}
};

void connect(Node* root){
   if(root==NULL)return;
   Node* current=root;
   while(current->left!=NULL){
      Node* temp=current;
      while(temp!=NULL){
         temp->left->next=temp->right;
         if(temp->next!=NULL){
            temp->right->next=temp->next->left;
         }
         temp=temp->next;
      }
      current=current->left;
   }
```

```
}

void printLevels(Node* root){
   while(root!=NULL){
      Node* temp=root;
      while(temp!=NULL){
         cout<<temp->val<<" ";
         temp=temp->next;
      }
      cout<<"# ";
      root=root->left;
   }
   cout<<endl;
}

Node* insertLevelOrder(int arr[],int n,int i){
   if(i>=n)return NULL;
   Node* temp=new Node(arr[i]);
   temp->left=insertLevelOrder(arr,n,2*i+1);
   temp->right=insertLevelOrder(arr,n,2*i+2);
   return temp;
}

int main(){
   int n;
   cout<<"Enter the Number of Nodes in the Tree :-  ";
   cin>>n;
   int* arr=new int[n];
   cout<<"Enter the Values of the Tree Nodes in Level Order - ";
   for(int i=0;i<n;i++){
      cin>>arr[i];
   }
```

```
    Node* root=insertLevelOrder(arr,n,0);
    connect(root);
    cout<<"\nThe Tree after Populating the Next Pointers :- ";
    printLevels(root);
    delete[] arr;
    return 0;
}
```

Output

```
Enter the Number of Nodes in the Tree :-  7
Enter the Values of the Tree Nodes in Level Order - 1 2 3 4 5 6 7

The Tree after Populating the Next Pointers :- 1 # 2 3 # 4 5 6 7 #
```

Problem-12 :- 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. **(Hard)**

Source Code

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

```
int maxHappyGroups(int batchSize, vector<int>& groups) {
    int n=groups.size();
    int happyGroups=0;
    vector<int> remainders(batchSize,0);

    for(int i=0;i<n;i++) {
        int rem=groups[i]%batchSize;
        if(rem==0) {
            happyGroups++;
        } else {
            remainders[rem]++;
        }
    }

    for(int i=1;i<batchSize;i++) {
        if(remainders[i]>0) {
            int complement=batchSize-i;
            if(remainders[complement]>0) {
                int minCount=min(remainders[i],remainders[complement]);
                happyGroups+=minCount;
                remainders[i]-=minCount;
                remainders[complement]-=minCount;
            }
        }
    }

    for(int i=1;i<batchSize;i++) {
        if(remainders[i]>0) {
            happyGroups++;
        }
    }
```

```
    return happyGroups;
}

int main() {
    int batchSize,n;
    cout<<"Enter the Batch Size :- ";
    cin>>batchSize;
    cout<<"Enter the Number of Groups :- ";
    cin>>n;

    vector<int> groups(n);
    cout<<"Enter the Group Sizes - ";
    for(int i=0;i<n;i++) {
        cin>>groups[i];
    }
    cout<<"\nThe Maximum Number of Happy Groups are "<<maxHappyGroups(batchSize,groups)<<endl;
    return 0;
}
```

Output

```
Enter the Batch Size :- 3
Enter the Number of Groups :- 6
Enter the Group Sizes - 1 2 3 4 5 6

The Maximum Number of Happy Groups are 4
```

Problem-13 :- 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. **(Hard)**

Source Code

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

int cherryPickup(vector<vector<int>>& grid) {
   int rows = grid.size(), cols = grid[0].size();
   vector<vector<vector<int>>> dp(rows, vector<vector<int>>(cols, vector<int>(cols, -1)));

   dp[0][0][cols-1] = grid[0][0] + grid[0][cols-1];

   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) continue;

            for (int dc1 = -1; dc1 <= 1; dc1++) {
               for (int dc2 = -1; dc2 <= 1; dc2++) {
                  int newC1 = c1 + dc1, newC2 = c2 + dc2;

                  if (newC1 < 0 || newC1 >= cols || newC2 < 0 || newC2 >= cols) continue;
```

```
                int cherries = grid[r][newC1] + grid[r][newC2];
                if (newC1 == newC2) cherries -= grid[r][newC1];

                dp[r][newC1][newC2] = max(dp[r][newC1][newC2], dp[r-1][c1][c2] + cherries);
            }
        }
    }
}

int result = -1;
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() {
    int rows, cols;
    cout << "Enter the Number of Rows :- ";
    cin >> rows;
    cout << "Enter the Number of Columns :- ";
    cin >> cols;
    cout<<endl;

    vector<vector<int>> grid(rows, vector<int>(cols));
```

```
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << "Enter the Grid Value at [" << i << "][" << j << "] :- ";
            cin >> grid[i][j];
        }
    }

    cout << "\nThe Maximum Cherries Collected are " << cherryPickup(grid) << endl;
    return 0;
}
```

Output

```
Enter the Number of Rows :- 4
Enter the Number of Columns :- 3

Enter the Grid Value at [0][0] :- 3
Enter the Grid Value at [0][1] :- 1
Enter the Grid Value at [0][2] :- 1
Enter the Grid Value at [1][0] :- 2
Enter the Grid Value at [1][1] :- 5
Enter the Grid Value at [1][2] :- 1
Enter the Grid Value at [2][0] :- 1
Enter the Grid Value at [2][1] :- 5
Enter the Grid Value at [2][2] :- 5
Enter the Grid Value at [3][0] :- 2
Enter the Grid Value at [3][1] :- 1
Enter the Grid Value at [3][2] :- 1

The Maximum Cherries Collected are 24
```

Problem-14 :- 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. **(Hard)**

Source Code

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

int maxDartsInDartboard(vector<vector<int>>& darts, int r) {
   int n = darts.size(), maxCount = 0;

   for (int i = 0; i < n; i++) {
      for (int j = i + 1; j < n; j++) {
         double cx = (darts[i][0] + darts[j][0]) / 2.0;
         double cy = (darts[i][1] + darts[j][1]) / 2.0;
         int count = 0;
         for (int k = 0; k < n; k++) {
            if (pow(darts[k][0] - cx, 2) + pow(darts[k][1] - cy, 2) <= pow(r, 2))
{
               count++;
            }
         }
         maxCount = max(maxCount, count);
      }
   }
   return maxCount;
}

int main() {
   int n, r;
```

```
    cout << "Enter the Number of Darts :- ";
    cin >> n;
    cout << "Enter the Radius of the Dartboard :- ";
    cin >> r;
    vector<vector<int>> darts(n, vector<int>(2));
    cout<<endl;
    for (int i = 0; i < n; i++) {
        cout << "Enter the Coordinates of Dart " << i + 1 << " - ";
        cin >> darts[i][0] >> darts[i][1];
    }

    cout << "\nThe Maximum Darts on Dartboard are " << maxDartsInDartboard(darts, r) << endl;
    return 0;
}
```

Output

```
Enter the Number of Darts :- 4
Enter the Radius of the Dartboard :- 2

Enter the Coordinates of Dart 1 - -2 0
Enter the Coordinates of Dart 2 - 2 0
Enter the Coordinates of Dart 3 - 0 2
Enter the Coordinates of Dart 4 - 0 -2

The Maximum Darts on Dartboard are 4
```

Problem-15 :- 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. **(Very Hard)**

Source Code

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

bool canAssignJobs(const vector<int>& jobs, int k, int mid) {
    vector<int> workers(k, 0);
    for (int i = 0; i < jobs.size(); i++) {
        bool assigned = false;
        for (int j = 0; j < k; j++) {
            if (workers[j] + jobs[i] <= mid) {
                workers[j] += jobs[i];
                assigned = true;
                break;
            }
        }
        if (!assigned) {
            return false;
        }
    }
    return true;
}

int minimumWorkingTime(vector<int>& jobs, int k) {
    int low = *max_element(jobs.begin(), jobs.end());
    int high = accumulate(jobs.begin(), jobs.end(), 0);
```

```
    int result = high;

    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (canAssignJobs(jobs, k, mid)) {
            result = mid;
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }
    return result;
}

int main() {
    int n, k;
    cout << "Enter the Number of Jobs :- ";
    cin >> n;
    cout << "Enter the Number of Workers :- ";
    cin >> k;

    vector<int> jobs(n);
    cout << "Enter the Job Times :- ";
    for (int i = 0; i < n; i++) {
        cin >> jobs[i];
    }

    cout << "\nThe Minimum Possible Maximum Working Time is " << minimumWorkingTime(jobs, k) << endl;
    return 0;
}
```

Output

```
Enter the Number of Jobs :- 3
Enter the Number of Workers :- 3
Enter the Job Times :- 3 2 3

The Minimum Possible Maximum Working Time is 3
```

Problem-16 :- 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 ith user knows and friendships[i] = [ui, vi] denotes a friendship between the users ui 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. **(Very Hard)**

Source Code

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

class UnionFind {
public:
   UnionFind(int n):parent(n),rank(n,0){for(int i=0;i<n;++i)parent[i]=i;}
   int find(int x){if(parent[x]!=x)parent[x]=find(parent[x]);return parent[x];}
   void unionSets(int x,int y){int
rootX=find(x),rootY=find(y);if(rootX!=rootY){if(rank[rootX]>rank[rootY])p
```

```
arent[rootY]=rootX;else
if(rank[rootX]<rank[rootY])parent[rootX]=rootY;else{parent[rootY]=rootX;r
ank[rootX]++;}}}
private:
    vector<int> parent,rank;
};

int minTeachings(int n, vector<vector<int>>& languages,
vector<vector<int>>& friendships) {
    int m = languages.size();
    UnionFind uf(m);
    for (const auto& friendship : friendships) {
        int u = friendship[0] - 1, v = friendship[1] - 1;
        uf.unionSets(u, v);
    }
    unordered_map<int,unordered_set<int>> components;
    for (int i = 0; i < m; ++i) {
        int root = uf.find(i);
        components[root].insert(i);
    }
    int totalTeachings = 0;
    for (const auto& component : components) {
        unordered_set<int> knownLanguages;
        for (int user : component.second) {
            for (int lang : languages[user]) {
                knownLanguages.insert(lang);
            }
        }
        bool canCommunicate = false;
        for (int user : component.second) {
            for (int lang : languages[user]) {
                if (knownLanguages.count(lang) == component.second.size()) {
```

```
                canCommunicate = true;
                break;
            }
        }
        if (canCommunicate) break;
    }
    if (!canCommunicate) {
        totalTeachings++;
    }
  }
  return totalTeachings;
}

int main() {
  int n,m,f;
  cout<<"Enter the Number of Languages :- ";
  cin>>n;
  cout<<"Enter the Number of Users :- ";
  cin>>m;
  cout<<endl;
  vector<vector<int>> languages(m);
  for (int i = 0; i < m; ++i) {
    int numLanguages;
    cout<<"Enter the Number of Languages Known by User "<<i+1<<" - ";
    cin>>numLanguages;
    languages[i].resize(numLanguages);
    cout<<"Enter the Languages Known by User "<<i+1<<" - ";
    for (int j = 0; j < numLanguages; ++j) {
      cin>>languages[i][j];
    }
  }
  cout<<endl;
```

```
    cout<<"Enter the Friendships :- "<<endl;
    vector<vector<int>> friendships;
    while (true) {
        int u,v;
        cout<<"Enter the Friendship Pair - ";
        if (!(cin>>u>>v)) break;
        friendships.push_back({u,v});
    }
    cout<<"\nThe Minimum Number of Users to Teach are "<<minTeachings(n,languages,friendships)<<endl;
    return 0;
}
```

Output

```
Enter the Number of Languages :- 2
Enter the Number of Users :- 3

Enter the Number of Languages Known by User 1 - 1
Enter the Languages Known by User 1 - 1
Enter the Number of Languages Known by User 2 - 1
Enter the Languages Known by User 2 - 2
Enter the Number of Languages Known by User 3 - 2
Enter the Languages Known by User 3 - 1 2

Enter the Friendships :-
Enter the Friendship Pair - 1 2
Enter the Friendship Pair - 1 3
Enter the Friendship Pair - 2 3
Enter the Friendship Pair - Done

The Minimum Number of Users to Teach are 1
```

Problem-17 :- In a linked list of size n, where n is even, the ith node (0-indexed) of the linked list is known as the twin of the (n-1-i)th node, if 0 <= i <= (n / 2) - 1. For example, if n = 4, then node 0 is the twin of node 3, and node 1 is the twin of node 2. These are the only nodes with twins for n = 4. The twin

sum is defined as the sum of a node and its twin. Given the head of a linked list with even length, return the maximum twin sum of the linked list. **(Very Hard)**

Source Code

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

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

class Solution {
public:
    int pairSum(ListNode* head) {
        ListNode *slow = head, *fast = head;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        ListNode *prev = nullptr, *curr = slow;
        while (curr) {
            ListNode *nextNode = curr->next;
            curr->next = prev;
            prev = curr;
            curr = nextNode;
        }
```

```
        ListNode *firstHalf = head, *secondHalf = prev;
        int maxSum = 0;
        while (secondHalf) {
            maxSum = max(maxSum, firstHalf->val + secondHalf->val);
            firstHalf = firstHalf->next;
            secondHalf = secondHalf->next;
        }
        return maxSum;
    }
};

ListNode* createLinkedList(const vector<int>& nums) {
    ListNode *head = new ListNode(nums[0]);
    ListNode *curr = head;
    for (int i = 1; i < nums.size(); ++i) {
        curr->next = new ListNode(nums[i]);
        curr = curr->next;
    }
    return head;
}

int main() {
    int n;
    cout << "Enter the Number of Nodes in the Linked List :- ";
    cin >> n;

    vector<int> input(n);
    cout << "Enter the Values for the Linked List - ";
    for (int i = 0; i < n; ++i) {
        cin >> input[i];
    }
```

```
    ListNode* head = createLinkedList(input);
    Solution;
    int result = solution.pairSum(head);
    cout << "\nThe Maximum Twin Sum is " << result << endl;
    return 0;
}
```

Output

```
Enter the Number of Nodes in the Linked List :- 4
Enter the Values for the Linked List - 5 4 2 1

The Maximum Twin Sum is 6
```

Problem-18 :- Given the head of a linked list head, in which each node contains an integer value. Between every pair of adjacent nodes, insert a new node with a value equal to the greatest common divisor of them. Return the linked list after insertion. The greatest common divisor of two numbers is the largest positive integer that evenly divides both numbers. **(Very Hard)**

Source Code

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

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

```
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

class Solution {
public:
    ListNode* insertGreatestCommonDivisors(ListNode* head) {
        if (!head || !head->next) return head;

        ListNode* current = head;
        while (current && current->next) {
            int gcdValue = gcd(current->val, current->next->val);
            ListNode* gcdNode = new ListNode(gcdValue);
            gcdNode->next = current->next;
            current->next = gcdNode;
            current = gcdNode->next;
        }
        return head;
    }
};

ListNode* createLinkedList(const vector<int>& nums) {
    if (nums.empty()) return nullptr;
    ListNode* head = new ListNode(nums[0]);
    ListNode* curr = head;
    for (int i = 1; i < nums.size(); ++i) {
```

```
        curr->next = new ListNode(nums[i]);
        curr = curr->next;
    }
    return head;
}

void printLinkedListFormatted(ListNode* head) {
    cout << "\nThe Modified Linked List is [";
    while (head) {
        cout << head->val;
        if (head->next) cout << ",";
        head = head->next;
    }
    cout << "]" << endl;
}

void freeLinkedList(ListNode* head) {
    while (head) {
        ListNode* temp = head;
        head = head->next;
        delete temp;
    }
}

int main() {
    int n;
    cout << "Enter the Number of Nodes in the Linked List :- ";
    cin >> n;

    vector<int> input(n);
    cout << "Enter the Values for the Linked List - ";
    for (int i = 0; i < n; ++i) {
```

```
        cin >> input[i];
    }
    ListNode* head = createLinkedList(input);
    Solution solution;
    head = solution.insertGreatestCommonDivisors(head);
    printLinkedListFormatted(head);
    freeLinkedList(head);
    return 0;
}
```

Output

```
Enter the Number of Nodes in the Linked List :- 4
Enter the Values for the Linked List - 18 6 10 3

The Modified Linked List is [18,6,6,2,10,1,3]
```

Problem-19 :- 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. **(Medium)**

Source Code

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

bool isValidSudoku(vector<vector<char>>& board) {
```

```
    unordered_set<string> seen;
    for (int i = 0; i < 9; ++i) {
        for (int j = 0; j < 9; ++j) {
            char num = board[i][j];
            if (num != '.') {
                string rowKey = "row" + to_string(i) + num;
                string colKey = "col" + to_string(j) + num;
                string boxKey = "box" + to_string(i / 3) + to_string(j / 3) + num;
                if (seen.count(rowKey) || seen.count(colKey) || seen.count(boxKey))
{
                    return false;
                }
                seen.insert(rowKey);
                seen.insert(colKey);
                seen.insert(boxKey);
            }
        }
    }
    return true;
}

int main() {
    vector<vector<char>> board(9, vector<char>(9));
    cout << "Enter the 9x9 Sudoku Board Row by Row Using '.' for Empty Cells
:- " << endl;
    for (int i = 0; i < 9; ++i) {
        cout << "Enter the Row " << i + 1 << " - ";
        for (int j = 0; j < 9; ++j) {
            cin >> board[i][j];
        }
    }
    bool result = isValidSudoku(board);
```

```
    cout << (result ? "\nTrue, the 9x9 Sudoku Board is Valid" : "\nFalse, the 9x9 Sudoku Board is not Valid") << endl;
    return 0;
}
```

Output

```
Enter the 9x9 Sudoku Board Row by Row Using '.' for Empty Cells :-
Enter the Row 1 - 5 3 . . 7 . . . .
Enter the Row 2 - 6 . . 1 9 5 . . .
Enter the Row 3 - . 9 8 . . . . 6 .
Enter the Row 4 - 8 . . . 6 . . . 3
Enter the Row 5 - 4 . . 8 . 3 . . 1
Enter the Row 6 - 7 . . . 2 . . . 6
Enter the Row 7 - . 6 . . . . 2 8 .
Enter the Row 8 - . . . 4 1 9 . . 5
Enter the Row 9 - . . . . 8 . . 7 9

True, the 9x9 Sudoku Board is Valid
```