National University of Computer & Emerging Sciences, Karachi
Final Examination (Fall-2023)
Dec 18, 2023, 12:30 PM – 3:30 PM

Course Code: CS2001	Course Name: Data Structures
Instructor Name: Dr. Jawwad A. Shamsi, Dr. Fahad Sherwani, Mr. Basit Ali, Ms. Sobia Iftikhar, Ms. Mubashra Fayyaz, Mr. Minhal Raza, Dr. Anam Qureshi, Ms. Sumaiyah Zahid, Ms. Sania Urooj, Mr. Farooq Zaidi, Mr. Shahbaz Siddiqui, Ms. Fizza Mansoor, Dr. Farrukh Hasan	
Student Roll No:	Section No:

Instructions:

- Return the question paper and make sure to keep it inside your answer sheet.
- Read each question completely before answering it. There are **6 questions in 4 pages**.
- In case of any ambiguity, you may make assumption. But your assumption should not contradict any statement in the question paper.
- You are **not allowed to write** anything on the question paper (except your ID and group).

Time: 180 minutes. **Max Marks**: 100 Points

Question: 1 [20 points] [CLO 2]

True/ False with reasoning. (Note: You need to give not more than 3 lines justification for each irrespective of whether the statement is true or false. There will be NO marks without justification) [20 points]

1. Is it possible to find the middle node of a singly linked list in constant time (O(1))?
2. Can a stack be efficiently implemented using two queues, and vice versa?
3. In a sorted array of distinct integers, a linear search is more efficient than a binary search for finding a specific element.
4. Among the bubble sort, insertion sort, and selection sort algorithms, selection sort is the most likely to have the best performance on a nearly sorted array because it is an adaptive algorithm.
5. The standard versions of both Merge sort and quick sort are both in-place sorting algorithms, allowing them to sort data without requiring additional memory space.
6. A balanced binary tree is also a complete binary tree.
7. In an AVL tree, after the insertion of a node, the tree may require rotations to maintain its balance factor and ensure that the height difference between the left and right subtrees is at most 2.
8. A binary heap is a suitable data structure for implementing a priority queue because it ensures that the element with the highest priority is always at the root.
9. In a weighted graph, where the edge weights are non-negative, applying Dijkstra's algorithm and Prim's algorithm for finding the minimum spanning tree will always yield the same result.
10. A hash table with a higher load factor is generally more efficient because it maximizes the utilization of available memory space.

Solution

1. False. The middle node of a singly linked list cannot be found in constant time, as it requires traversing the list at least once, resulting in a time complexity of O(n), where n is the number of nodes in the list.

2. True. A stack can be implemented using two queues, and a queue can be implemented using two stacks, with each operation (push, pop, enqueue, dequeue) taking O(1) amortized time.

3. False. In a sorted array of distinct integers, a binary search is more efficient than a linear search for finding a specific element.

4. False. Insertion sort will give the best performance because of it's adaptive nature.

5. False. Merge sort is not an in-place sorting algorithm as it requires additional memory proportional to the size of the input array. In contrast, quick sort is an in-place sorting algorithm, making it more memory-efficient.

6. False. A perfect binary tree is a specific type of complete binary tree where all levels are completely filled with nodes. However, a complete binary tree may not have all levels filled, but it fills levels from left to right.

7. False. In an AVL tree, after the insertion of a node, rotations may be necessary to maintain the balance factor and ensure that the height difference (balance factor) between the left and right subtrees is at most 1.

8. True. A binary heap is an efficient implementation for a priority queue, where the element with the highest priority is stored at the root. The binary heap structure allows for constant time (O(1)) retrieval of the highest priority element and logarithmic time (O(log n)) insertion and deletion.

9. False. While both Dijkstra's algorithm and Prim's algorithm use greedy approaches, they solve different problems. Dijkstra's algorithm finds the shortest paths from a single source to all other vertices, whereas Prim's algorithm finds the minimum spanning tree. The results may differ even in weighted graphs with non-negative edge weights.

10. False. A higher load factor in a hash table increases the likelihood of collisions, leading to performance degradation. An optimal load factor balances the table's occupancy and ensures efficient operations by minimizing collisions.

Question: 2 [10 points] [CLO 1]

You are given an array of size ***N*** containing distinct integers. An inversion in an array occurs when there are two indices i and j such that i < j but arr[i] > arr[j].
Your task is to implement an either algorithm or (pseudo-code) better than $O(n^2)$ to count the number of inversions in the array efficiently, where a positive integer N represents the size of the array of ***N*** distinct integers and is entered as input by the user.
Example:
Input:
N = 6

Array = [5, 3, 9, 1, 7, 6]
Output:
(5, 3) is an inversion because 5 > 3.
(5, 1) is an inversion because 5 > 1.
(9, 1) is an inversion because 9 > 1.
(7, 6) is an inversion because 7 > 6.
(9, 7) is an inversion because 9 > 7.
(9, 6) is an inversion because 9 > 6.
(3, 1) is an inversion because 3 > 1.
So, there are a total of 7 inversions in the given array. The problem solved above in $O(n^2)$ time. Find an another way to do it in more efficient manner.

Solution

```
#include <iostream>

using namespace std;

long long mergeAndCountInversions(int arr[], int temp[], int left, int mid, int right) {
    int i = left;
    int j = mid + 1;
    int k = left;
    long long inversions = 0;

    while (i <= mid && j <= right) {
        if (arr[i] <= arr[j]) {
            temp[k++] = arr[i++];
        } else {
            temp[k++] = arr[j++];
            inversions += (mid - i + 1);
        }
    }

    while (i <= mid) {
        temp[k++] = arr[i++];
    }

    while (j <= right) {
        temp[k++] = arr[j++];
    }

    for (int p = left; p <= right; p++) {
        arr[p] = temp[p];
    }

    return inversions;
}

long long mergeSortAndCountInversions(int arr[], int temp[], int left, int right) {
    long long inversions = 0;
    if (left < right) {
        int mid = left + (right - left) / 2;
```

```
        inversions += mergeSortAndCountInversions(arr, temp, left, mid);
        inversions += mergeSortAndCountInversions(arr, temp, mid + 1, right);
        inversions += mergeAndCountInversions(arr, temp, left, mid, right);
    }
    return inversions;
}

int main() {
    int N;
    cout << "Enter the size of the array: ";
    cin >> N;

    int arr[N];
    int temp[N];

    cout << "Enter the elements of the array: ";
    for (int i = 0; i < N; i++) {
        cin >> arr[i];
    }

    long long inversions = mergeSortAndCountInversions(arr, temp, 0, N - 1);

    cout << "Number of inversions: " << inversions << endl;


    return 0;
}
```

Question: 3 [10 points] [CLO 4]

Consider the following directed graph ***G :<V,E>*** as shown in figure-2. The 1st column shows the source vertex, 2nd column shows the destination vertex and 3rd column shows the cost between source and destination vertex.

```
5 4  0.35
4 7  0.37
5 7  0.28
5 1  0.32
4 0  0.38
0 2  0.26
3 7  0.39
1 3  0.29
7 2  0.34
6 2  0.40
3 6  0.52
6 0  0.58
6 4  0.93
```

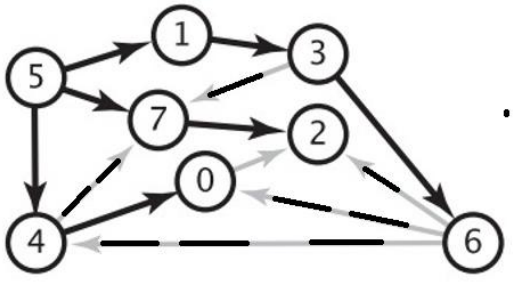
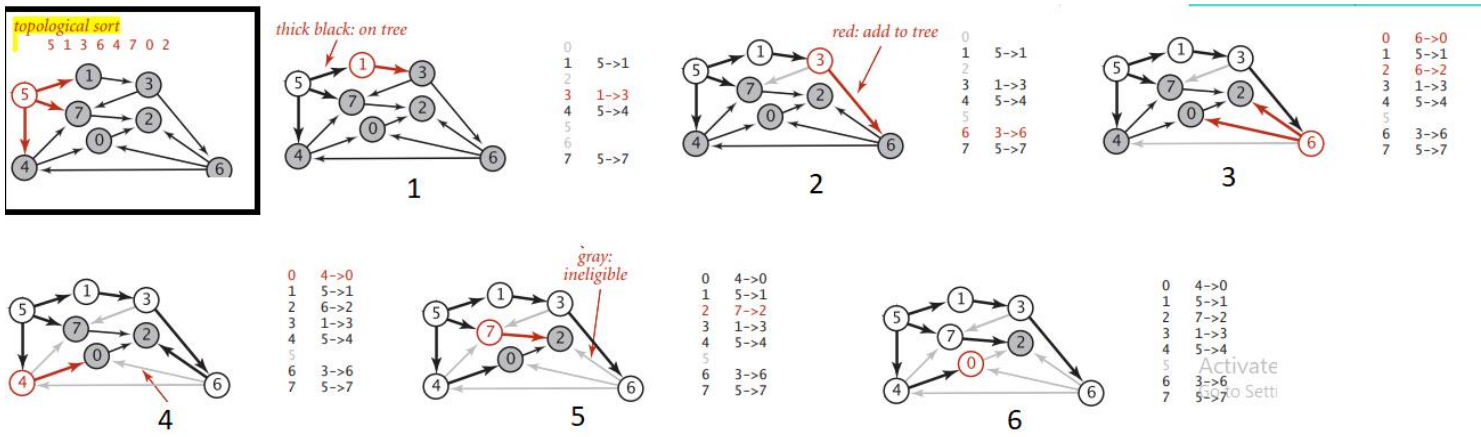


Figure 1

a. Perform topological sorting on the graph. Show each step clearly by showing the nodes in the underlying data structure at each stage. [5 points]



b. Construct the minimum spanning tree (MST) for the given graph using Kruskal's Algorithm. Show each step clearly. [5 points]

Through the adjacency list or from graph we can see that the tree has no cycle so it is already a minimum spanning tree.

Question: 4 [20 points] [CLO 3]

In Josephus problem, ***N*** people stand in a circle waiting to be executed. The counting starts at some point in the circle and proceeds in a specific direction around the circle. In each step, a certain number of people are skipped and the next person is executed (or eliminated). The elimination of people makes the circle smaller and smaller. At the last step, only one person remains who is declared the 'winner'.

Therefore, if there are ***N*** number of people and a number ***K*** which indicates that ***K–1*** people are skipped and ***k–th*** person in the circle is eliminated, then the problem is to choose a position in the initial circle so that the given person becomes the winner.

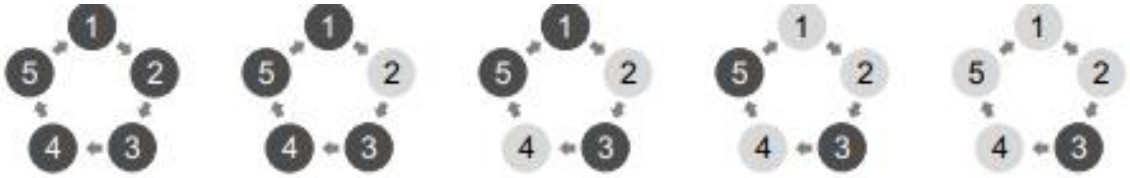


Figure 2

For example, if there are 5 (***N***) people and every second (***K***) person is eliminated, then first the person at position 2 is eliminated followed by the person at position 4 followed by person at position 1 and finally the person at position 5 is eliminated. Therefore, the person at position 3 becomes the winner.

Try the same process with N = 7 and K =3. You will find that person at position 4 is the winner. The elimination goes in the sequence of 3, 6, 2, 7, 5 and 1. Write a program which finds the solution of Josephus problem using a circular linked list.

Solution :

Problem and Algorithm Explanation : https://www.prepbytes.com/blog/linked-list/josephus-circle-using-circular-linked-list/

C++ Code

```
#include<iostream>

struct circularNode
{
	int data;
	circularNode* next;
};

circularNode *josephusPosition();
void showWinner(circularNode*);

int main()
{

	showWinner(josephusPosition());

   return 0;
}


circularNode *josephusPosition()
{
	circularNode *temp;
	circularNode *head;

	int players;
	int number;

	std::cout << "Enter the number of players in game: " << std::endl;
	std::cin >> players;
	std::cout << "Every xth person gets eliminated- enter x: " << std::endl;
	std::cin >> number;

	//Create a circular list containing the players in our game
	head = new circularNode;
	temp = head;
	temp->data = 1;
	for (int i = 2; i <= players; ++i)
	{
		temp->next = new circularNode;
		temp = temp->next;
		temp->data = i;
	}
	//close the circular list
	temp->next = head;
```

```
        //Eliminate every xth player as long until there is 1 remaining
        for (int count = players; count > 1; --count)
        {
                for (int i = 1; i < number; ++i)
                        temp = temp->next;
    //circularNode q will be used to delete dynamically allocated memory
                circularNode *q = temp->next;
    //Eliminate player from list
                temp->next = temp->next->next;
    //Free eliminated players node from memory
                delete q;
        }
        return temp;
}

void showWinner(circularNode* show)
{
        std::cout << "The winner is: " << show->data << std::endl;
}
```

Output:

```
Enter the number of players in game:
5
Every xth person gets eliminated- enter x:
2
The winner is: 3

Enter the number of players in game:
7
Every xth person gets eliminated- enter x:
3
The winner is: 4
```

C- Code

```
// Online C compiler to run C program online
#include <stdio.h>

    #include <stdlib.h>

    struct node

    {

        int number;

        struct node *next;

    };

    void create(struct node **);
```

```
void display(struct node *);
int survivor(struct node **, int);
int main()
{
    struct node *head = NULL;
    int survive, skip;
    create(&head);
    printf("The persons in circular list are:\n");
    display(head);
    printf("Enter the number of persons to be skipped: ");
    scanf("%d", &skip);
    survive = survivor(&head, skip);
    printf("The person to survive is : %d\n", survive);
    free(head);
    return 0;
}
int survivor(struct node **head, int k)
{
    struct node *p, *q;
    int i;
    q = p = *head;
    while (p->next != p)
    {
        for (i = 0; i < k - 1; i++)
        {
            q = p;
            p = p->next;
```

```
        }
        q->next = p->next;
        printf("%d has been killed.\n", p->number);
        free(p);
        p = q->next;
    }
    *head = p;
    return (p->number);
}
void create (struct node **head)
{
    struct node *temp, *rear;
    int a, ch;
    do
    {
        printf("Enter a number: ");
        scanf("%d", &a);
        temp = (struct node *)malloc(sizeof(struct node));
        temp->number = a;
        temp->next = NULL;
        if (*head == NULL)
        {
            *head = temp;
        }
        else
        {
```

```
            rear->next = temp;
        }
        rear = temp;
        printf("Do you want to add a number [1/0]? ");
        scanf("%d", &ch);
    } while (ch != 0);
    rear->next = *head;
}
void display(struct node *head)
{
    struct node *temp;
    temp = head;
    printf("%d   ", temp->number);
    temp = temp->next;
    while (head != temp)
    {
        printf("%d   ", temp->number);
        temp = temp->next;
    }
    printf("\n");
}
```

Output

```
Enter a number: 1
Do you want to add a number [1/0]? 1
Enter a number: 2
Do you want to add a number [1/0]? 1
Enter a number: 3
Do you want to add a number [1/0]? 1
Enter a number: 4
Do you want to add a number [1/0]? 1
Enter a number: 5
```

Do you want to add a number [1/0]? 1
Enter a number: 6
Do you want to add a number [1/0]? 1
Enter a number: 7
Do you want to add a number [1/0]? 0
The persons in circular list are:
1 2 3 4 5 6 7
Enter the number of persons to be skipped: 3
3 has been killed.
6 has been killed.
2 has been killed.
7 has been killed.
5 has been killed.
1 has been killed.
The person to survive is : 4

Enter a number: 1
Do you want to add a number [1/0]? 1
Enter a number: 2
Do you want to add a number [1/0]? 1
Enter a number: 3
Do you want to add a number [1/0]? 1
Enter a number: 4
Do you want to add a number [1/0]? 1
Enter a number: 5
Do you want to add a number [1/0]? 0
The persons in circular list are:
1 2 3 4 5
Enter the number of persons to be skipped: 2
2 has been killed.
4 has been killed.
1 has been killed.
5 has been killed.
The person to survive is : 3

Question: 5 [20 points] [CLO 4]

The following figure-3 describes a breadth first search (BFS)

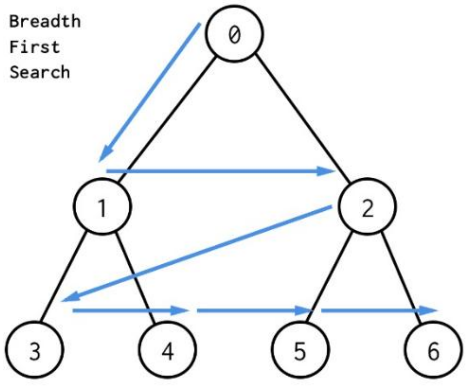


Figure 3

The nodes in an integer binary tree can be represented by the following data structure.

```
1. class BTNode { // Binary tree node
2. public:
3. int item; // data in this node
4. BTNode left; // left subtree or null
5. BTNode right; // right subtree or null if none
6. }
```

Give two implementations (code) of the definition of method BFS below to perform a breadth-first traversal of a binary tree and print the node data values in the order they are reached during the search.

```
// Perform a breadth-first traversal of the binary tree with
// root r and print the node values as they are encountered
public void BFS(BTNode r);
```

1. In your first solution, you may only use recursion and no other data structure in your implementation [10 points]
2. In your second solution you may create and use instances of other standard data structures (array's or vector's) and their operations if they are useful, without having to give details of their implementations. [5 points]
3. What will be the time complexity of both solutions?List pros and cons of both. [5 points]

Solution : Not Received Yet

Question: 6 [20 points] [CLO 3]

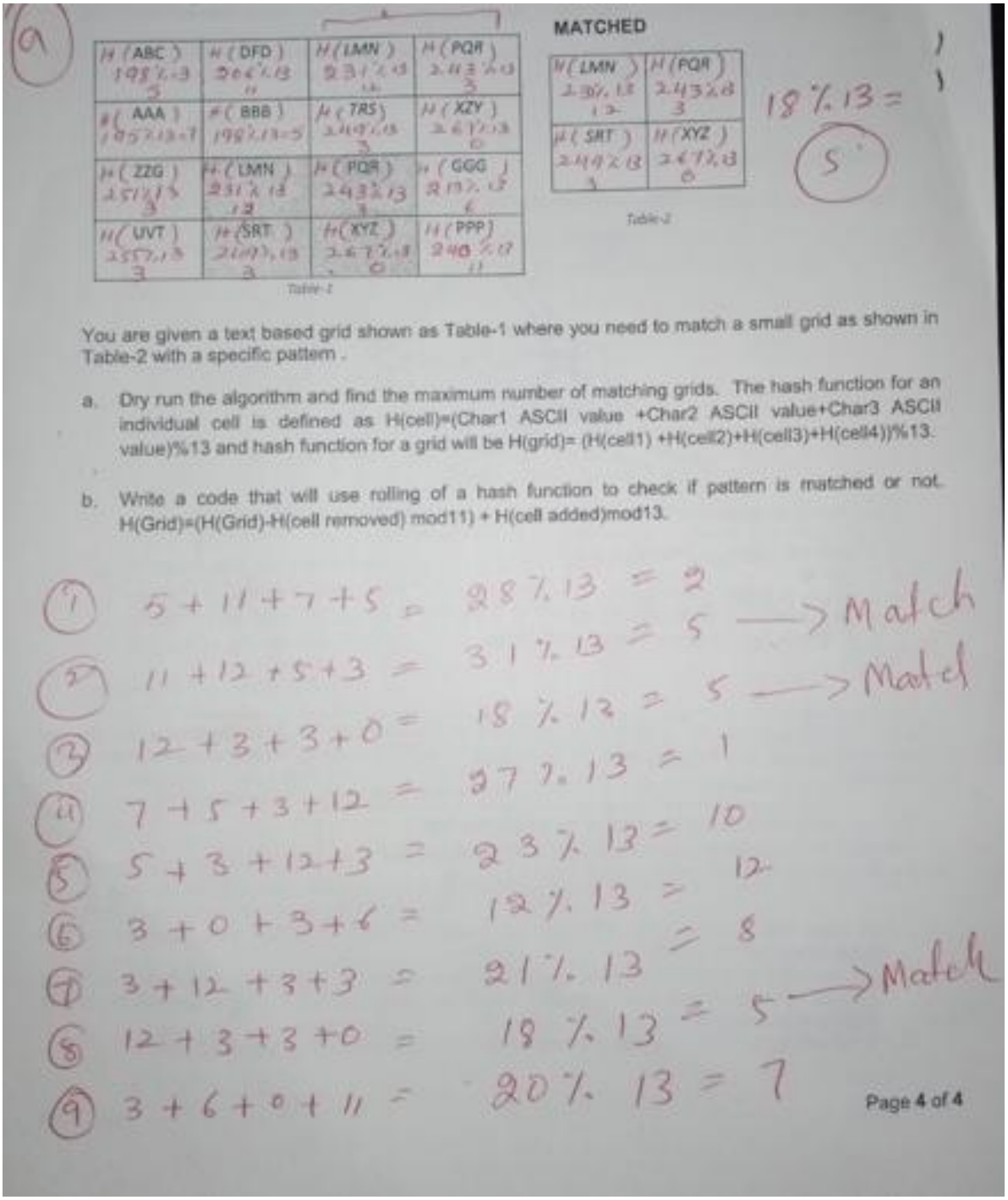
ABC	DFD	*LMN*	*PQR*
AAA	BBB	*TRS*	*XZY*
ZZG	LMN	PQR	GGG
UVT	SRT	XYZ	PPP

MATCHED

LMN	*PQR*
SRT	*XYZ*

You are given a text based grid shown as Table-1 where you need to match a small grid as shown in Table-2 with a specific pattern .

a. Dry run the algorithm and find the maximum number of matching grids. The hash function for an individual cell is defined as H(cell)=(Char1 ASCII value +Char2 ASCII value+Char3 ASCII value)%13 and hash function for a grid will be H(grid)= (H(cell1) +H(cell2)+H(cell3)+H(cell4))%13.



H(ABC)	H(DFD)	H(LMN)	H(PQR)
H(AAA)	H(BBB)	H(TRS)	H(XZY)
H(ZZG)	H(LMN)	H(PQR)	H(GGG)
H(UVT)	H(SRT)	H(XYZ)	H(PPP)

Table-1

MATCHED

H(LMN)	H(PQR)
H(SRT)	H(XYZ)

Table-2

18 % 13 = 5

You are given a text based grid shown as Table-1 where you need to match a small grid as shown in Table-2 with a specific pattern .

a. Dry run the algorithm and find the maximum number of matching grids. The hash function for an individual cell is defined as H(cell)=(Char1 ASCII value +Char2 ASCII value+Char3 ASCII value)%13 and hash function for a grid will be H(grid)= (H(cell1) +H(cell2)+H(cell3)+H(cell4))%13.

b. Write a code that will use rolling of a hash function to check if pattern is matched or not. H(Grid)=(H(Grid)-H(cell removed) mod11) + H(cell added)mod13.

(1) 5 + 11 + 7 + 5 = 28 % 13 = 2
(2) 11 + 12 + 5 + 3 = 31 % 13 = 5 → Match
(3) 12 + 3 + 3 + 0 = 18 % 12 = 5 → Match
(4) 7 + 5 + 3 + 12 = 27 % 13 = 1
(5) 5 + 3 + 12 + 3 = 23 % 13 = 10
(6) 3 + 0 + 3 + 6 = 12 % 13 = 12
(7) 3 + 12 + 3 + 3 = 21 % 13 = 8
(8) 12 + 3 + 3 + 0 = 18 % 13 = 5 → Match
(9) 3 + 6 + 0 + 11 = 20 % 13 = 7

Page 4 of 4

b. Write a code that will use rolling of a hash function to check if pattern is matched or not. H(Grid)=(H(Grid)-H(cell removed) mod11) + H(cell added)mod13.

Solution

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

```
int hashCell(const string& cell) {
    int hashValue = 0;
    for (char c : cell) {
        hashValue = (hashValue + c) % 13;
    }
    return hashValue;
}
int hashGrid(const vector<vector<string>>& grid, int mod) {
    int hashValue = 0;
    for (const auto& row : grid) {
        for (const string& cell : row) {
            hashValue = (hashValue + hashCell(cell)) % mod;
        }
    }
    return hashValue;
}

int updateHash(const vector<vector<string>>& grid, const string& cellRemoved, const string&
cellAdded, int mod) {
    int hashRemoved = hashCell(cellRemoved);
    int hashAdded = hashCell(cellAdded);
    int hashGridValue = hashGrid(grid, mod);
    hashGridValue = (hashGridValue - hashRemoved + hashAdded + mod) % mod;

    return hashGridValue;
}

bool rabinKarpSearch(const vector<vector<string>>& grid, const vector<vector<string>>& pattern) {
    int modGrid = 11;
    int modCell = 13;

    int hashPattern = hashGrid(pattern, modCell);

    for (int i = 0; i <= grid.size() - pattern.size(); ++i) {
        for (int j = 0; j <= grid[0].size() - pattern[0].size(); ++j) {

            vector<vector<string>> subgrid(pattern.size(), vector<string>(pattern[0].size()));
            for (int x = 0; x < pattern.size(); ++x) {
                for (int y = 0; y < pattern[0].size(); ++y) {
                    subgrid[x][y] = grid[i + x][j + y];
                }
            }

            int hashSubgrid = hashGrid(subgrid, modCell);

            if (hashSubgrid == hashPattern) {
```

```
            if (subgrid == pattern) {
                cout << "Pattern found at position (" << i << ", " << j << ")" << endl;
                return true;
            }
        }
    }
}

cout << "Pattern not found in the grid" << endl;
return false;
}

int main() {
    // Define the grid and pattern
    vector<vector<string>> grid = {
        {"ABC", "DFD", "LMN", "PQR"},
        {"AAA", "BBB", "SRT", "ZYX"},
        {"ZZG", "LMN", "PQR", "GGG"},
        {"UVT", "SRT", "XYZ", "PP"}
    };

    vector<vector<string>> pattern = {
        {"LMN", "PQR"},
        {"SRT", "XYZ"}
    };


    rabinKarpSearch(grid, pattern);

    return 0;
}
```