Questions for Question Bank

21CSC201J: Data Structures and Algorithms

Unit – II: Applications (Sparse Matrix, Polynomial Arithmetic, Joseph Problem)

Prepared by: Dr. B. Prakash

Q1 – Q5: Objective Type Questions

1. Which of the following is not the method to represent Sparse Matrix?
   a) Dictionary of Keys
   b) Linked List
   c) Array
   d) Heap

   Answer: d

2. The matrix contains m rows and n columns. The matrix is called Sparse Matrix if _____
   a) Total number of Zero elements > (m*n)/2
   b) Total number of Zero elements = m + n
   c) Total number of Zero elements = m/n
   d) Total number of Zero elements = m-n

   Answer: a

3. If p(x) is a polynomial of degree one and p(y) = 0, then y is said to be:

   a. Zero of p(x)
   b. Value of p(x)
   c. Constant of p(x)
   d. None of the above

   Answer: a

4. A polynomial's zeros can be represented graphically. The number of polynomial zeros equals the number of points on the graph of the polynomial:
   a. Intersects y-axis
   b. Intersects x-axis
   c. Intersects y-axis or x-axis
   d. None of the above

   Answer: b

5. Polynomial Addition is implemented using which data structure?
   a. Queue
   b. Linked List
   c. Tress
   d. Stack

Answer: b

Understanding question [Blooms level two]

6. Check whether a matrix is a sparse matrix or not.
   1 1 0
   0 0 2
   0 0 0

   Solution

- Let us assume ZERO in the matrix is greater than (row * column)/2.
- Then, the matrix is a sparse matrix otherwise not.

  Program

  Following is the program to check whether the given matrix is sparse matrix or not –

```
#include<stdio.h>
#include<stdlib.h>
int main(){
   int row,col,i,j,a[10][10],count = 0;
   printf("Enter row
");
   scanf("%d",&row);
   printf("Enter Column
");
   scanf("%d",&col);
   printf("Enter Element of Matrix1
");
   for(i = 0; i < row; i++){
      for(j = 0; j < col; j++){
         scanf("%d",&a[i][j]);
      }
   }
   printf("Elements are:
");
   for(i = 0; i < row; i++){
for(j = 0; j < col; j++){
         printf("%d\t",a[i][j]);
      }
      printf("
");
   }
   /*checking sparse of matrix*/
   for(i = 0; i < row; i++){
```

```
    for(j = 0; j < col; j++){
      if(a[i][j] == 0)
        count++;
    }
  }
  if(count > ((row * col)/2))
    printf("Matrix is a sparse matrix
");
  else
    printf("Matrix is not sparse matrix
");
}
```

7. Write an algorithm to demonstrate a polynomial using a linked list for i. Addition and Subtraction.
   Answer:

```
// C++ program for addition of two polynomials
// using Linked Lists
#include <bits/stdc++.h>
using namespace std;

// Node structure containing power and coefficient of
// variable
struct Node {
        int coeff;
        int pow;
        struct Node* next;
};

// Function to create new node
void create_node(int x, int y, struct Node** temp)
{
        struct Node *r, *z;
        z = *temp;
        if (z == NULL) {
                r = (struct Node*)malloc(sizeof(struct Node));
                r->coeff = x;
                r->pow = y;
                *temp = r;
                r->next = (struct Node*)malloc(sizeof(struct Node));
                r = r->next;
                r->next = NULL;
        }
        else {
                r->coeff = x;
                r->pow = y;
```

```
                r->next = (struct Node*)malloc(sizeof(struct Node));
                r = r->next;
                r->next = NULL;
        }
}

// Function Adding two polynomial numbers
void polyadd(struct Node* poly1, struct Node* poly2,
                        struct Node* poly)
{
        while (poly1->next && poly2->next) {
                // If power of 1st polynomial is greater then 2nd,
                // then store 1st as it is and move its pointer
                if (poly1->pow > poly2->pow) {
                        poly->pow = poly1->pow;
                        poly->coeff = poly1->coeff;
                        poly1 = poly1->next;
                }

                // If power of 2nd polynomial is greater then 1st,
                // then store 2nd as it is and move its pointer
                else if (poly1->pow < poly2->pow) {
                        poly->pow = poly2->pow;
                        poly->coeff = poly2->coeff;
                        poly2 = poly2->next;
                }

                // If power of both polynomial numbers is same then
                // add their coefficients
                else {
                        poly->pow = poly1->pow;
                        poly->coeff = poly1->coeff + poly2->coeff;
                        poly1 = poly1->next;
                        poly2 = poly2->next;
                }

                // Dynamically create new node
                poly->next
                        = (struct Node*)malloc(sizeof(struct Node));
                poly = poly->next;
                poly->next = NULL;
        }
        while (poly1->next || poly2->next) {
                if (poly1->next) {
                        poly->pow = poly1->pow;
                        poly->coeff = poly1->coeff;
                        poly1 = poly1->next;
                }
                if (poly2->next) {
                        poly->pow = poly2->pow;
```

```
                poly->coeff = poly2->coeff;
                poly2 = poly2->next;
            }
            poly->next
                = (struct Node*)malloc(sizeof(struct Node));
            poly = poly->next;
            poly->next = NULL;
        }
}

// Display Linked list
void show(struct Node* node)
{
        while (node->next != NULL) {
            printf("%dx^%d", node->coeff, node->pow);
            node = node->next;
            if (node->coeff >= 0) {
                if (node->next != NULL)
                    printf("+");
            }
        }
}

// Driver code
int main()
{
        struct Node *poly1 = NULL, *poly2 = NULL, *poly = NULL;

        // Create first list of 5x^2 + 4x^1 + 2x^0
        create_node(5, 2, &poly1);
        create_node(4, 1, &poly1);
        create_node(2, 0, &poly1);

        // Create second list of -5x^1 - 5x^0
        create_node(-5, 1, &poly2);
        create_node(-5, 0, &poly2);

        printf("1st Number: ");
        show(poly1);
        printf("\n2nd Number: ");
        show(poly2);

        poly = (struct Node*)malloc(sizeof(struct Node));
        // Function add two polynomial numbers
        polyadd(poly1, poly2, poly);
        // Display resultant List
        printf("\nAdded polynomial: ");
        show(poly);
        return 0;
}
```

8. State the Recursive approach in josephs problem.

   Answer:

A simple approach to solve this problem is to find the position of the step which would be called after each execution.

Therefore, given **N** persons, and skipping **K** persons during their deletion, **N - 1** persons will be left. Therefore, we need to call the recursive function for **N - 1** and **K** for the next iteration.

Now, for each remaining circle after execution, we need to find the last remaining person present i.e. if **Kth** person is executed, **K+1th** will be the next starting safe point the recursive call. Therefore, to keep a track of the position, perform **K%N + 1.**

The recursive function would be:

josephus(N, K) = (josephus(N - 1 , K) + K - 1) % N + 1.

You can also observe a pattern as follows:

n/k	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9
10	10	10	10	10	10	10	10	10	10	10

Algorithm

- If **N == 1,** return **1.** This is the termination condition for the function.
- Else, return **(josephus(N - 1 , K) + K - 1) % N + 1.**

Scenario Based

9. Given a 2D array, the task is to print the 2D in alternate manner (First row from left to right, then from right to left, and so on).

```
Input : arr[][2] = {{1, 2}
                   {2, 3}};
Output : 1  2  3  2

Input :arr[][3] = { { 7 , 2 , 3 },
                  { 2 , 3 , 4 },
                  { 5 , 6 ,  1 }};
Output : 7  2   3  4  3  2  5  6  1
```

Solution:

```
// C++ program to print matrix in alternate manner
#include<bits/stdc++.h>
using namespace std;
#define R 3
#define C 3

// Function for print matrix in alternate manner
void convert(int arr[R][C])
{
	bool leftToRight = true;
	for (int i=0; i<R; i++)
	{
		if (leftToRight)
		{
			for (int j=0; j<C; j++)
				printf("%d ", arr[i][j]);
		}
		else
		{
			for (int j=C-1; j>=0; j--)
				printf("%d ",arr[i][j]);
		}

		leftToRight = !leftToRight;
	}
}

// Driver code
int main()
{
	int arr[][C] =
	{
		{ 1 , 2 , 3 },
		{ 3 , 2 , 1 },
		{ 4 , 5 , 6 },
	};

	convert(arr);
	return 0;
}
```