perplexity

Array Concepts and C Implementations

1. Array

An **array** is a collection of elements of the same data type, stored in contiguous memory locations. Arrays allow random access to elements using indexes.

2. Row-Major and Column-Major Order

Row-Major Order

- **Definition:** In row-major order, multi-dimensional arrays are stored row by row. That is, all elements of the first row are stored in memory first, then the second row, and so on.

C Code Example (2D Array in Row-Major Order)

```c
#include <stdio.h>

int main() {
    // Declare and initialize a 2x3 array
    int arr[2][3] = { {1, 2, 3}, {4, 5, 6} };

    // Print elements in row-major order
    for (int i = 0; i < 2; i++) {           // Loop over rows
        for (int j = 0; j < 3; j++) {       // Loop over columns
            printf("%d ", arr[i][j]);       // Print each element
        }
    }
    return 0;
}
```

Explanation:
Each row is printed completely before moving to the next—that’s row-major order.

Column-Major Order

- **Definition:** In column-major order, multi-dimensional arrays are stored column by column. All elements from the first column are stored in memory first, followed by the second column, etc. C does **not natively** support column-major order, but it can be simulated.

C Code Example (Simulating 2D Array in Column-Major Order)

```c
#include <stdio.h>

int main() {
    // Declare a 2x3 array as a 1D array for column-major storage
    int rows = 2, cols = 3;
    int arr[6] = {1, 2, 3, 4, 5, 6}; // Simulated 2x3 matrix

    // Print elements in column-major order
    for (int j = 0; j < cols; j++) {             // Loop over columns
        for (int i = 0; i < rows; i++) {         // Loop over rows
            // Formula: index = row + col*rows
            printf("%d ", arr[i + j*rows]);      // Access column-major element
        }
    }
    return 0;
}
```

Explanation:
We access elements as if columns come first by using the formula: `arr[i + j*rows]`.

3. Sparse Matrix

Definition

A **sparse matrix** is a matrix in which most elements are zero. Efficient storage and computation can be achieved by storing only non-zero elements.

Storage Methods

- **Triplet (Coordinate) Representation:** Store three arrays: row indices, column indices, and values of non-zero elements.
- **Compressed Sparse Row (CSR):** Use three arrays for values, column indices, and pointers to the start of each row.

Implementation: Triplet Representation

C Code (Storing and Printing a Sparse Matrix)

```c
#include <stdio.h>

#define MAX 100

struct SparseMatrix {
    int row[MAX];
    int col[MAX];
    int val[MAX];
    int num_nonzero; // Count of nonzero elements
    int rows, cols;
};

void createSparse(int mat[][4], int rows, int cols, struct SparseMatrix *sm) {
    sm->num_nonzero = 0;
    sm->rows = rows;
    sm->cols = cols;
    for (int i = 0; i < rows; i++) {                      // Loop over rows
        for (int j = 0; j < cols; j++) {                  // Loop over cols
            if (mat[i][j] != 0) {                         // If element is nonzero
                sm->row[sm->num_nonzero] = i;             // Store row index
                sm->col[sm->num_nonzero] = j;             // Store col index
                sm->val[sm->num_nonzero] = mat[i][j];     // Store value
                sm->num_nonzero++;                        // Increment count
            }
        }
    }
}

void printSparse(struct SparseMatrix sm) {
    printf("Row Col Value\n");
    for (int i = 0; i < sm.num_nonzero; i++) {
        printf("%3d %3d %5d\n", sm.row[i], sm.col[i], sm.val[i]); // Print each nonzero e
    }
}

int main() {
    int mat[4][4] = {
        {0, 0, 3, 0},
        {4, 0, 0, 5},
        {0, 0, 0, 0},
        {0, 2, 0, 6}
    };
    struct SparseMatrix sm;
    createSparse(mat, 4, 4, &sm); // Convert to sparse matrix
    printSparse(sm);              // Print sparse matrix storage
    return 0;
}
```

Explanation:

- Nonzero elements are stored with their row/column indices.
- Only nonzero values are kept, saving memory for large sparse matrices.

Usage

- Efficient for large matrices with few nonzero elements (e.g., graphs, scientific computing).
- Saves memory and computation time compared to storing all elements.

4. Array Representation of Polynomials

- **Definition:** A polynomial $ P(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n $ can be stored in an array such that the coefficient of $ x^i $ is at index $ i $.

C Code (Polynomial Representation and Addition)

```c
#include <stdio.h>
#define MAX 10

void printPoly(int poly[], int n) {
    for (int i = 0; i < n; i++) {                 // Loop over coefficients
        printf("%dx^%d", poly[i], i);             // Print coefficient and degree
        if (i != n-1) printf(" + ");              // Print plus if not last term
    }
    printf("\n");
}

void addPoly(int poly1[], int poly2[], int n) {
    int sum[MAX];
    for (int i = 0; i < n; i++)
        sum[i] = poly1[i] + poly2[i];             // Add corresponding coefficients

    printPoly(sum, n);                            // Print the resulting polynomial
}

int main() {
    int poly1[MAX] = {2, 0, 5}; // 2 + 0x + 5x^2
    int poly2[MAX] = {1, 4, 2}; // 1 + 4x + 2x^2

    printf("Polynomial 1: ");
    printPoly(poly1, 3);                           // Print first polynomial

    printf("Polynomial 2: ");
    printPoly(poly2, 3);                           // Print second polynomial

    printf("Sum: ");
    addPoly(poly1, poly2, 3);                      // Print sum of polynomials

    return 0;
}
```

Explanation:

- Each index in the array contains the coefficient for that degree of the polynomial.
- Addition is performed by summing coefficients at the same index.

Summary Table

Concept	Definition/Description	C Storage Example
Row-Major Order	Array stored row by row	`arr[i][j]`
Column-Major Order	Array stored column by column (simulated in C)	`arr[i + j*rows]`
Sparse Matrix	Store only nonzeros as triplets (row, col, value)	`struct SparseMatrix`
Array Representation, Poly.	Each coefficient in array at index equal to term degree	`poly[i]`