explain xor in details in context of cp/dsa
everything in detail
every algortihm
cases
use case
from basic to advance give all codes in cpp
from basic to advnace all agorithm and imp code should be covered
all concepts

XOR (Exclusive OR) in Computer Programming & DSA

Table of Contents

1. Basic XOR Concept
2. Bitwise XOR Properties
3. Basic Algorithms
4. Advanced XOR Algorithms
5. XOR in Competitive Programming
6. XOR Tricks & Optimization
7. Real-world Applications

Basic XOR Concept

What is XOR?

XOR (Exclusive OR) is a binary operation that returns true only when inputs differ.

cpp

```cpp
// Truth Table
// A B | A XOR B
// 0 0 | 0
// 0 1 | 1
// 1 0 | 1
// 1 1 | 0
```

C++ Bitwise XOR Operator

cpp

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

int main() {
    int a = 5;    // 0101 in binary
    int b = 3;    // 0011 in binary
    int c = a ^ b; // 0110 in binary (6 in decimal)

    cout << "5 ^ 3 = " << c << endl; // Output: 6

    // Verification
    // 5:  0101
    // 3:  0011
    // XOR:0110 = 6

    return 0;
}
```

Bitwise XOR Properties

Important Properties

cpp

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

void demonstrateProperties() {
    int x = 5, y = 9, z = 13;

    // 1. Commutative: x ^ y = y ^ x
    cout << "Commutative:" << endl;
    cout << x << " ^ " << y << " = " << (x ^ y) << endl;
    cout << y << " ^ " << x << " = " << (y ^ x) << endl;

    // 2. Associative: (x ^ y) ^ z = x ^ (y ^ z)
    cout << "\nAssociative:" << endl;
    cout << "(" << x << " ^ " << y << ") ^ " << z << " = " << ((x ^ y) ^ z) << end
l;
    cout << x << " ^ (" << y << " ^ " << z << ") = " << (x ^ (y ^ z)) << endl;
```

```
    // 3. Identity: x ^ 0 = x
    cout << "\nIdentity:" << endl;
    cout << x << " ^ 0 = " << (x ^ 0) << endl;

    // 4. Self-inverse: x ^ x = 0
    cout << "\nSelf-inverse:" << endl;
    cout << x << " ^ " << x << " = " << (x ^ x) << endl;

    // 5. Inverse: x ^ y ^ y = x
    cout << "\nInverse property:" << endl;
    cout << x << " ^ " << y << " ^ " << y << " = " << (x ^ y ^ y) << endl;

    // 6. Even one's property: XOR of a number with itself even times = 0
    cout << "\nEven one's property:" << endl;
    cout << "X ^ X ^ X ^ X = " << (x ^ x ^ x ^ x) << endl;
}
```

XOR with Special Numbers

cpp

```
void xorWithSpecialNumbers() {
    // XOR with all 1's gives complement
    int num = 42; // 00101010

    cout << "Number: " << num << " (" << bitset<8>(num) << ")" << endl;
    cout << "XOR with 0: " << (num ^ 0) << endl;
    cout << "XOR with all 1's (" << 255 << "): " << (num ^ 255)
         << " (" << bitset<8>(num ^ 255) << ")" << endl;
    cout << "This is equivalent to ~num: " << (~num & 255)
         << " (" << bitset<8>(~num & 255) << ")" << endl;
}
```

Basic Algorithms

1. Swap Two Numbers without Temporary Variable

cpp

```
void swapWithXOR(int &a, int &b) {
    // Using XOR swap algorithm
    // Step-by-step:
    // a = a ^ b
    // b = a ^ b = (a ^ b) ^ b = a
```

```
    // a = a ^ b = (a ^ b) ^ a = b

    a = a ^ b;
    b = a ^ b; // Now b has original a
    a = a ^ b; // Now a has original b
}

void demonstrateSwap() {
    int x = 10, y = 20;
    cout << "Before swap: x = " << x << ", y = " << y << endl;
    swapWithXOR(x, y);
    cout << "After swap: x = " << x << ", y = " << y << endl;
}
```

2. Find Missing Number in Array

cpp

```
int findMissingNumber(vector<int>& nums, int n) {
    // Given array of size n-1 containing numbers from 1 to n
    // Find the missing number

    // Method 1: XOR approach
    int xor_all = 0;

    // XOR all numbers from 1 to n
    for(int i = 1; i <= n; i++) {
        xor_all ^= i;
    }

    // XOR with all array elements
    for(int num : nums) {
        xor_all ^= num;
    }

    return xor_all;
}

int findMissingNumberOptimized(vector<int>& nums, int n) {
    // Optimized version
    int result = n;

    for(int i = 0; i < nums.size(); i++) {
        result ^= i;
        result ^= nums[i];
    }
```

```
    return result;
}
```

3. Find Single Number in Array (Others appear twice)

cpp

```cpp
int singleNumber(vector<int>& nums) {
    // Given an array where every element appears twice except one
    // Find that single element

    int result = 0;
    for(int num : nums) {
        result ^= num;
    }
    return result;
}

// Variation: Find two numbers that appear once, others appear twice
pair<int, int> findTwoSingleNumbers(vector<int>& nums) {
    int xor_all = 0;
    for(int num : nums) {
        xor_all ^= num;
    }

    // Find rightmost set bit
    int rightmost_set_bit = xor_all & -xor_all;

    int num1 = 0, num2 = 0;

    for(int num : nums) {
        if(num & rightmost_set_bit) {
            num1 ^= num;
        } else {
            num2 ^= num;
        }
    }

    return {num1, num2};
}
```

4. Check if Two Numbers Have Opposite Signs

cpp

```cpp
bool haveOppositeSigns(int x, int y) {
    // XOR of sign bits
    // If signs are different, XOR of sign bits = 1
    // If signs are same, XOR of sign bits = 0

    return (x ^ y) < 0;
}
```

Advanced XOR Algorithms

1. XOR Subarray Queries

cpp

```cpp
class XORSubarray {
private:
    vector<int> prefixXOR;

public:
    XORSubarray(vector<int>& arr) {
        int n = arr.size();
        prefixXOR.resize(n + 1, 0);

        // Build prefix XOR array
        for(int i = 0; i < n; i++) {
            prefixXOR[i + 1] = prefixXOR[i] ^ arr[i];
        }
    }

    // Query XOR from index l to r (0-based)
    int query(int l, int r) {
        // XOR(l, r) = prefixXOR[r+1] ^ prefixXOR[l]
        return prefixXOR[r + 1] ^ prefixXOR[l];
    }

    // Find maximum XOR subarray
    int maxXORSubarray() {
        // Using Trie (implemented later)
        return findMaximumXOR(prefixXOR);
    }
};
```

2. XOR Trie for Maximum XOR

cpp

```cpp
class TrieNode {
public:
    TrieNode* children[2];

    TrieNode() {
        children[0] = children[1] = nullptr;
    }
};

class XORTrie {
private:
    TrieNode* root;
    int maxBits;

public:
    XORTrie(int maxNumber) {
        root = new TrieNode();
        maxBits = 32 - __builtin_clz(maxNumber); // Number of bits needed
    }

    void insert(int num) {
        TrieNode* node = root;

        for(int i = maxBits - 1; i >= 0; i--) {
            int bit = (num >> i) & 1;

            if(!node->children[bit]) {
                node->children[bit] = new TrieNode();
            }

            node = node->children[bit];
        }
    }

    int findMaxXOR(int num) {
        TrieNode* node = root;
        int maxXor = 0;

        for(int i = maxBits - 1; i >= 0; i--) {
            int bit = (num >> i) & 1;
            int oppositeBit = 1 - bit;

            // Try to go to opposite bit to maximize XOR
```

```
            if(node->children[oppositeBit]) {
                maxXor |= (1 << i);
                node = node->children[oppositeBit];
            } else {
                node = node->children[bit];
            }
        }

        return maxXor;
    }
};

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

    int maxNum = *max_element(nums.begin(), nums.end());
    XORTrie trie(maxNum);

    // Insert first number
    trie.insert(nums[0]);
    int maxXor = 0;

    for(int i = 1; i < nums.size(); i++) {
        maxXor = max(maxXor, trie.findMaxXOR(nums[i]));
        trie.insert(nums[i]);
    }

    return maxXor;
}
```

3. Gray Code Generation

cpp

```
vector<int> grayCode(int n) {
    // Gray code: consecutive numbers differ by exactly one bit
    // Formula: G(i) = i ^ (i >> 1)

    vector<int> result;
    for(int i = 0; i < (1 << n); i++) {
        result.push_back(i ^ (i >> 1));
    }
    return result;
}

bool isGrayCodeSequence(vector<int>& arr) {
```

```
    for(int i = 1; i < arr.size(); i++) {
        int diff = arr[i] ^ arr[i-1];

        // Check if diff has exactly one bit set
        if((diff & (diff - 1)) != 0) {
            return false;
        }
    }
    return true;
}
```

4. XOR Linked List (Memory Efficient)

cpp

```
class XORNode {
public:
    int data;
    XORNode* npx; // XOR of next and prev pointers

    XORNode(int val) {
        data = val;
        npx = nullptr;
    }
};

class XORLinkedList {
private:
    XORNode* head;

    XORNode* XOR(XORNode* a, XORNode* b) {
        return (XORNode*)((uintptr_t)a ^ (uintptr_t)b);
    }

public:
    XORLinkedList() {
        head = nullptr;
    }

    void insert(int data) {
        XORNode* newNode = new XORNode(data);

        // New node's npx = XOR(nullptr, head)
        newNode->npx = head;

        if(head) {
```

```
            // Update head's npx = XOR(newNode, next)
            head->npx = XOR(newNode, head->npx);
        }

        head = newNode;
    }

    void traverse() {
        XORNode* curr = head;
        XORNode* prev = nullptr;
        XORNode* next;

        while(curr) {
            cout << curr->data << " ";

            // Get next node
            next = XOR(prev, curr->npx);

            // Move forward
            prev = curr;
            curr = next;
        }
        cout << endl;
    }

    void traverseReverse() {
        XORNode* curr = head;
        XORNode* prev = nullptr;
        XORNode* next;

        // Go to end
        while(curr) {
            next = XOR(prev, curr->npx);
            prev = curr;
            curr = next;
        }

        // Now traverse backward from end
        curr = prev;
        prev = nullptr;

        while(curr) {
            cout << curr->data << " ";
            next = XOR(prev, curr->npx);
            prev = curr;
            curr = next;
        }
```

```
        cout << endl;
    }
};
```

5. Count Subarrays with XOR equal to K

cpp

```cpp
int countSubarraysWithXOR(vector<int>& arr, int k) {
    // Using prefix XOR and hash map
    unordered_map<int, int> prefixCount;
    int prefixXor = 0;
    int count = 0;

    prefixCount[0] = 1; // Empty subarray has XOR 0

    for(int num : arr) {
        prefixXor ^= num;

        // If prefixXor ^ k exists in map, we found subarrays
        if(prefixCount.find(prefixXor ^ k) != prefixCount.end()) {
            count += prefixCount[prefixXor ^ k];
        }

        prefixCount[prefixXor]++;
    }

    return count;
}

// Maximum length subarray with XOR k
int maxLengthSubarrayWithXOR(vector<int>& arr, int k) {
    unordered_map<int, int> prefixIndex;
    int prefixXor = 0;
    int maxLen = 0;

    prefixIndex[0] = -1; // XOR 0 at index -1

    for(int i = 0; i < arr.size(); i++) {
        prefixXor ^= arr[i];

        if(prefixIndex.find(prefixXor ^ k) != prefixIndex.end()) {
            maxLen = max(maxLen, i - prefixIndex[prefixXor ^ k]);
        }

        if(prefixIndex.find(prefixXor) == prefixIndex.end()) {
```

```
            prefixIndex[prefixXor] = i;
        }
    }

    return maxLen;
}
```

6. XOR of All Subarrays

cpp

```
int xorOfAllSubarrays(vector<int>& arr) {
    // XOR of all subarray XORs
    // Each element appears (i+1)*(n-i) times
    // If frequency is odd, include in result

    int n = arr.size();
    int result = 0;

    for(int i = 0; i < n; i++) {
        // Frequency of arr[i] in all subarrays
        long long freq = (long long)(i + 1) * (n - i);

        if(freq % 2 == 1) {
            result ^= arr[i];
        }
    }

    return result;
}
```

7. XOR of Numbers in a Range

cpp

```
int xorFrom1ToN(int n) {
    // Pattern observed:
    // XOR(1, n) =
    // n if n % 4 == 0
    // 1 if n % 4 == 1
    // n+1 if n % 4 == 2
    // 0 if n % 4 == 3

    switch(n % 4) {
```

```
        case 0: return n;
        case 1: return 1;
        case 2: return n + 1;
        case 3: return 0;
    }
    return 0;
}

int xorInRange(int L, int R) {
    // XOR(L, R) = XOR(1, L-1) ^ XOR(1, R)
    return xorFrom1ToN(L - 1) ^ xorFrom1ToN(R);
}
```

XOR in Competitive Programming

1. Game Theory - Nim Game

cpp

```
bool canWinNim(vector<int>& piles) {
    // Nim Game: Players take stones from piles
    // Winning condition: XOR of all piles != 0

    int nimSum = 0;
    for(int stones : piles) {
        nimSum ^= stones;
    }

    return nimSum != 0;
}

// Misère Nim (last move loses)
bool canWinMisereNim(vector<int>& piles) {
    int nimSum = 0;
    bool allOnes = true;

    for(int stones : piles) {
        nimSum ^= stones;
        if(stones > 1) allOnes = false;
    }

    if(allOnes) {
        // In Misère Nim with all piles of size 1:
        // Win if number of piles is even
        return piles.size() % 2 == 0;
```

```
    }

    // Otherwise, same as normal Nim
    return nimSum != 0;
}
```

2. XOR Basis (Linear Basis for XOR)

cpp

```cpp
class XORBasis {
private:
    vector<int> basis;
    int size;

public:
    XORBasis(int maxBits = 32) {
        basis.resize(maxBits, 0);
        size = 0;
    }

    void insert(int x) {
        for(int i = basis.size() - 1; i >= 0; i--) {
            if((x >> i) & 1) {
                if(basis[i] == 0) {
                    basis[i] = x;
                    size++;
                    return;
                }
                x ^= basis[i];
            }
        }
    }

    bool canRepresent(int x) {
        for(int i = basis.size() - 1; i >= 0; i--) {
            if((x >> i) & 1) {
                if(basis[i] == 0) return false;
                x ^= basis[i];
            }
        }
        return x == 0;
    }

    int getMaxXOR() {
        int result = 0;
```

```
        for(int i = basis.size() - 1; i >= 0; i--) {
            if((result ^ basis[i]) > result) {
                result ^= basis[i];
            }
        }
        return result;
    }

    int getMinXOR() {
        // Find smallest non-zero XOR
        for(int i = 0; i < basis.size(); i++) {
            if(basis[i] != 0) {
                return basis[i];
            }
        }
        return 0;
    }

    int getSize() {
        return size;
    }

    vector<int> getAllRepresentable() {
        vector<int> result;
        int total = 1 << size;

        for(int mask = 0; mask < total; mask++) {
            int current = 0;
            int temp = mask;

            for(int i = 0; i < basis.size(); i++) {
                if(basis[i] != 0) {
                    if(temp & 1) {
                        current ^= basis[i];
                    }
                    temp >>= 1;
                }
            }

            result.push_back(current);
        }

        sort(result.begin(), result.end());
        return result;
    }
};
```

3. XOR Convolution (Walsh-Hadamard Transform)

cpp

```cpp
void walshHadamardTransform(vector<int>& a, bool inverse = false) {
    // In-place Walsh-Hadamard Transform for XOR convolution
    // Used for polynomial multiplication under XOR

    int n = a.size();

    for(int len = 1; 2 * len <= n; len <<= 1) {
        for(int i = 0; i < n; i += 2 * len) {
            for(int j = 0; j < len; j++) {
                int u = a[i + j];
                int v = a[i + j + len];

                a[i + j] = u + v;
                a[i + j + len] = u - v;
            }
        }
    }

    if(inverse) {
        for(int i = 0; i < n; i++) {
            a[i] /= n;
        }
    }
}

vector<int> xorConvolution(vector<int>& a, vector<int>& b) {
    // Convolution where c[k] = sum(a[i] * b[j]) for all i^j = k

    int n = 1;
    while(n < max(a.size(), b.size())) n <<= 1;

    a.resize(n, 0);
    b.resize(n, 0);

    // Apply Walsh-Hadamard Transform
    walshHadamardTransform(a);
    walshHadamardTransform(b);

    // Multiply pointwise
    for(int i = 0; i < n; i++) {
        a[i] *= b[i];
    }
```

```
    // Inverse transform
    walshHadamardTransform(a, true);

    return a;
}
```

4. XOR Game - Grundy Numbers

cpp

```
int calculateMex(unordered_set<int>& s) {
    int mex = 0;
    while(s.find(mex) != s.end()) {
        mex++;
    }
    return mex;
}

int calculateGrundy(int n) {
    // Example: Game where you can split into XOR subgames
    // This is problem-specific

    vector<int> grundy(n + 1, 0);

    for(int i = 1; i <= n; i++) {
        unordered_set<int> s;

        // Example moves - problem specific
        for(int move = 1; move <= i; move++) {
            // Split into two piles
            s.insert(grundy[move - 1] ^ grundy[i - move]);
        }

        grundy[i] = calculateMex(s);
    }

    return grundy[n];
}
```

XOR Tricks & Optimization

1. Fast XOR Operations

cpp

```cpp
class XOROperations {
public:
    // Check if number is power of two
    bool isPowerOfTwo(int n) {
        return n > 0 && (n & (n - 1)) == 0;
    }

    // Get rightmost set bit
    int getRightmostSetBit(int n) {
        return n & -n;
    }

    // Clear rightmost set bit
    int clearRightmostSetBit(int n) {
        return n & (n - 1);
    }

    // Count set bits (popcount)
    int countSetBits(int n) {
        int count = 0;
        while(n) {
            n = n & (n - 1);
            count++;
        }
        return count;
    }

    // Check if i-th bit is set
    bool isBitSet(int n, int i) {
        return (n >> i) & 1;
    }

    // Set i-th bit
    int setBit(int n, int i) {
        return n | (1 << i);
    }

    // Clear i-th bit
    int clearBit(int n, int i) {
        return n & ~(1 << i);
    }

    // Toggle i-th bit
    int toggleBit(int n, int i) {
        return n ^ (1 << i);
```

```
    }

    // Get lowest unset bit
    int getLowestUnsetBit(int n) {
        return ~n & (n + 1);
    }

    // Swap adjacent bits
    int swapAdjacentBits(int n) {
        // Even bits shift right, odd bits shift left
        return ((n & 0xAAAAAAAA) >> 1) | ((n & 0x55555555) << 1);
    }

    // Reverse bits
    unsigned int reverseBits(unsigned int n) {
        n = ((n >> 1) & 0x55555555) | ((n & 0x55555555) << 1);
        n = ((n >> 2) & 0x33333333) | ((n & 0x33333333) << 2);
        n = ((n >> 4) & 0x0F0F0F0F) | ((n & 0x0F0F0F0F) << 4);
        n = ((n >> 8) & 0x00FF00FF) | ((n & 0x00FF00FF) << 8);
        n = ((n >> 16) & 0x0000FFFF) | ((n & 0x0000FFFF) << 16);
        return n;
    }
};
```

2. XOR for Parity Checking

cpp

```
class ParityChecker {
public:
    // Check parity (even or odd number of 1s)
    bool hasEvenParity(int n) {
        // XOR all bits
        n ^= n >> 16;
        n ^= n >> 8;
        n ^= n >> 4;
        n ^= n >> 2;
        n ^= n >> 1;

        return (n & 1) == 0;
    }

    bool hasOddParity(int n) {
        return !hasEvenParity(n);
    }
```

```
    // Fast parity using lookup table
    bool hasEvenParityFast(unsigned int n) {
        // 16-bit lookup table
        static const bool parityTable[256] = {
            // Precomputed parity for 0-255
            0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,
            1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,
            1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,
            0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,
            1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,
            0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,
            0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,
            1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0
        };

        n ^= n >> 16;
        n ^= n >> 8;
        return parityTable[n & 0xFF];
    }
};
```

3. XOR for Array Manipulation

cpp

```
class ArrayXORTricks {
public:
    // Find duplicate and missing numbers
    pair<int, int> findDuplicateAndMissing(vector<int>& nums, int n) {
        // Given array of size n with numbers 1 to n
        // One number is missing, one is duplicated

        int xor_all = 0;

        // XOR all numbers from 1 to n
        for(int i = 1; i <= n; i++) {
            xor_all ^= i;
        }

        // XOR with all array elements
        for(int num : nums) {
            xor_all ^= num;
        }

        // xor_all = missing ^ duplicate
        int rightmost_set_bit = xor_all & -xor_all;
```

```
        int missing_or_duplicate1 = 0;
        int missing_or_duplicate2 = 0;

        // Separate numbers based on rightmost set bit
        for(int i = 1; i <= n; i++) {
            if(i & rightmost_set_bit) {
                missing_or_duplicate1 ^= i;
            } else {
                missing_or_duplicate2 ^= i;
            }
        }

        for(int num : nums) {
            if(num & rightmost_set_bit) {
                missing_or_duplicate1 ^= num;
            } else {
                missing_or_duplicate2 ^= num;
            }
        }

        // Determine which is missing and which is duplicate
        for(int num : nums) {
            if(num == missing_or_duplicate1) {
                return {missing_or_duplicate1, missing_or_duplicate2}; // duplicat
e, missing
            }
        }

        return {missing_or_duplicate2, missing_or_duplicate1}; // duplicate, missin
g
    }

    // Check if array can be partitioned into two subsets with equal XOR
    bool canPartitionIntoEqualXOR(vector<int>& nums) {
        // If XOR of all elements is 0, then we can partition
        // Each subset will have XOR = total XOR = 0

        int total_xor = 0;
        for(int num : nums) {
            total_xor ^= num;
        }

        return total_xor == 0;
    }

    // Find element that appears odd times, others appear even times
```

```
    int findOddFrequencyElement(vector<int>& nums) {
        // All elements appear even times except one
        int result = 0;
        for(int num : nums) {
            result ^= num;
        }
        return result;
    }
};
```

4. XOR in Dynamic Programming

cpp

```
class XORDynamicProgramming {
public:
    // Count subsets with XOR equal to K
    int countSubsetsWithXOR(vector<int>& nums, int k) {
        int max_val = *max_element(nums.begin(), nums.end());
        int max_xor = (1 << (int)(log2(max_val) + 1)) - 1;

        vector<vector<int>> dp(nums.size() + 1, vector<int>(max_xor + 1, 0));
        dp[0][0] = 1;

        for(int i = 1; i <= nums.size(); i++) {
            for(int j = 0; j <= max_xor; j++) {
                dp[i][j] = dp[i-1][j] + dp[i-1][j ^ nums[i-1]];
            }
        }

        return dp[nums.size()][k];
    }

    // Space optimized version
    int countSubsetsWithXOROptimized(vector<int>& nums, int k) {
        int max_val = *max_element(nums.begin(), nums.end());
        int max_xor = (1 << (int)(log2(max_val) + 1)) - 1;

        vector<int> dp(max_xor + 1, 0);
        dp[0] = 1;

        for(int num : nums) {
            vector<int> temp = dp;
            for(int j = 0; j <= max_xor; j++) {
                dp[j] += temp[j ^ num];
            }
```

```
        }

        return dp[k];
    }
};
```

Real-world Applications

1. Simple Encryption/Decryption

cpp

```cpp
class XORCipher {
private:
    string key;

public:
    XORCipher(const string& k) : key(k) {}

    string encrypt(const string& plaintext) {
        string ciphertext = plaintext;

        for(size_t i = 0; i < plaintext.length(); i++) {
            ciphertext[i] = plaintext[i] ^ key[i % key.length()];
        }

        return ciphertext;
    }

    string decrypt(const string& ciphertext) {
        // XOR encryption is symmetric
        return encrypt(ciphertext);
    }

    void demonstrate() {
        string message = "Hello, XOR Encryption!";
        string key = "SecretKey123";

        XORCipher cipher(key);

        string encrypted = cipher.encrypt(message);
        string decrypted = cipher.decrypt(encrypted);

        cout << "Original: " << message << endl;
        cout << "Encrypted (hex): ";
```

```
        for(char c : encrypted) {
            printf("%02X ", (unsigned char)c);
        }
        cout << endl;
        cout << "Decrypted: " << decrypted << endl;
    }
};
```

2. RAID 5 Parity Calculation

cpp

```
class RAID5 {
public:
    // Calculate parity for RAID 5
    char calculateParity(const vector<char>& dataBlocks) {
        char parity = 0;
        for(char block : dataBlocks) {
            parity ^= block;
        }
        return parity;
    }

    // Recover lost block
    char recoverBlock(const vector<char>& availableBlocks, int missingIndex) {
        char recovered = 0;
        for(size_t i = 0; i < availableBlocks.size(); i++) {
            if(i != missingIndex) {
                recovered ^= availableBlocks[i];
            }
        }
        return recovered;
    }

    void demonstrate() {
        vector<char> data = {'A', 'B', 'C', 'D'};
        char parity = calculateParity(data);

        cout << "Data blocks: ";
        for(char d : data) cout << d << " ";
        cout << endl;
        cout << "Parity: " << parity << endl;

        // Simulate loss of block 2
        char lostBlock = data[2];
        data[2] = 0; // Mark as lost
```

```
        cout << "\nAfter losing block 2: ";
        for(char d : data) cout << (d ? d : '?') << " ";
        cout << endl;

        // Recover block 2
        char recovered = recoverBlock(data, 2);
        cout << "Recovered block 2: " << recovered << endl;

        // Verify
        data[2] = recovered;
        char newParity = calculateParity(data);
        cout << "New parity: " << newParity << " (matches original: "
             << (newParity == parity ? "Yes" : "No") << ")" << endl;
    }
};
```

3. Error Detection

cpp

```
class ErrorDetection {
public:
    // Simple parity check
    pair<string, char> addParity(const string& data) {
        char parity = 0;
        for(char c : data) {
            parity ^= c;
        }
        return {data, parity};
    }

    bool checkParity(const pair<string, char>& dataWithParity) {
        const string& data = dataWithParity.first;
        char receivedParity = dataWithParity.second;

        char calculatedParity = 0;
        for(char c : data) {
            calculatedParity ^= c;
        }

        return calculatedParity == receivedParity;
    }

    // Longitudinal Redundancy Check (LRC)
    string calculateLRC(const vector<string>& blocks) {
```

```
        if(blocks.empty()) return "";

        string lrc = blocks[0];

        for(size_t i = 1; i < blocks.size(); i++) {
            for(size_t j = 0; j < blocks[i].size(); j++) {
                lrc[j] ^= blocks[i][j];
            }
        }

        return lrc;
    }
};
```

4. Memory Optimization

cpp

```
class XORMemoryOptimization {
public:
    // Using XOR for doubly linked list with single pointer
    class MemoryEfficientNode {
    public:
        int data;
        MemoryEfficientNode* npx; // XOR of next and prev
```