**📜 Ultimate DSA Sheet: Tricks, Codes & Optimized Approaches**

💡 **Master Data Structures & Algorithms with optimized solutions, tricks, and code implementations in C++.**

**BY:Syntax\_Error**

**📌 Topics Covered:**

🔹 **Arrays**  
🔹 **Strings**  
🔹 **Sorting & Searching**  
🔹 **Hashing**  
🔹 **Linked List**  
🔹 **Stack & Queue**  
🔹 **Recursion & Backtracking**  
🔹 **Binary Tree & BST**  
🔹 **Heap & Priority Queue**  
🔹 **Graph Theory**  
🔹 **Dynamic Programming (DP)**  
🔹 **Greedy Algorithms**  
🔹 **Bit Manipulation**  
🔹 **Math & Number Theory**  
🔹 **Trie & Advanced Data Structures**

## ****📂 DSA Sheet Structure (Example for Arrays Section)****

### ****🟢 Arrays****

✅ **1. Sort an Array of 0s, 1s, and 2s (Dutch National Flag Algorithm)**  
📌 **Algorithm:** **Dutch National Flag Algorithm**  
📌 **Trick:** Use **three pointers (low, mid, high)** for **O(n) time**.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

void sortColors(vector<int>& nums) {

int low = 0, mid = 0, high = nums.size() - 1;

while (mid <= high) {

if (nums[mid] == 0) swap(nums[low++], nums[mid++]);

else if (nums[mid] == 1) mid++;

else swap(nums[mid], nums[high--]);

}

}

int main() {

vector<int> nums = {2, 0, 2, 1, 1, 0};

sortColors(nums);

for (int num : nums) cout << num << " ";

}

✅ **2. Find the Maximum Subarray Sum (Kadane’s Algorithm)**  
📌 **Algorithm:** **Kadane’s Algorithm**  
📌 **Trick:** Maintain **current sum (currSum)** and **max sum (maxSum)**. If currSum becomes negative, reset it.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int maxSubArray(vector<int>& nums) {

int maxSum = INT\_MIN, currSum = 0;

for (int num : nums) {

currSum = max(num, currSum + num);

maxSum = max(maxSum, currSum);

}

return maxSum;

}

int main() {

vector<int> nums = {-2, 1, -3, 4, -1, 2, 1, -5, 4};

cout << maxSubArray(nums);

}

✅ **3. Find the Next Permutation of an Array**  
📌 **Algorithm:** **Next Permutation Algorithm**  
📌 **Trick:** Find the first decreasing element from the end, swap it with the next larger element, and reverse the suffix.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

void nextPermutation(vector<int>& nums) {

int i = nums.size() - 2;

while (i >= 0 && nums[i] >= nums[i + 1]) i--; // Step 1: Find first decreasing element

if (i >= 0) {

int j = nums.size() - 1;

while (nums[j] <= nums[i]) j--; // Step 2: Find next greater element

swap(nums[i], nums[j]);

}

reverse(nums.begin() + i + 1, nums.end()); // Step 3: Reverse suffix

}

int main() {

vector<int> nums = {1, 2, 3};

nextPermutation(nums);

for (int num : nums) cout << num << " ";

}

✅ **4. Find the Majority Element (Boyer-Moore Voting Algorithm)**  
📌 **Algorithm:** **Boyer-Moore Majority Voting Algorithm**  
📌 **Trick:** Keep a **candidate** and a **count**; increase count for same element, decrease for different element.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

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() {

vector<int> nums = {2, 2, 1, 1, 1, 2, 2};

cout << majorityElement(nums);

}

✅ **5. Rearrange an Array Such That arr[i] = i**  
📌 **Algorithm:** **In-Place Swapping**  
📌 **Trick:** Place every element at its correct index using **swap technique** in O(n) time.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

void fixArray(vector<int>& arr) {

int n = arr.size();

for (int i = 0; i < n;) {

if (arr[i] >= 0 && arr[i] != i) swap(arr[i], arr[arr[i]]);

else i++;

}

}

int main() {

vector<int> arr = {-1, -1, 6, 1, 9, 3, 2, -1, 4, -1};

fixArray(arr);

for (int num : arr) cout << num << " ";

}

## ****📂 DSA Sheet Structure (Example for Strings Section)****

### ****🟢 Strings****

✅ **1. Longest Palindromic Substring (Expand Around Center Algorithm)**  
📌 **Algorithm:** **Expand Around Center**  
📌 **Trick:** For each character, expand outward while the substring remains a palindrome.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

string longestPalindrome(string s) {

int start = 0, maxLen = 0;

auto expand = [&](int l, int r) {

while (l >= 0 && r < s.size() && s[l] == s[r]) {

if (r - l + 1 > maxLen) {

start = l;

maxLen = r - l + 1;

}

l--; r++;

}

};

for (int i = 0; i < s.size(); i++) {

expand(i, i); // Odd length palindrome

expand(i, i+1); // Even length palindrome

}

return s.substr(start, maxLen);

}

int main() {

string s = "babad";

cout << longestPalindrome(s);

}

✅ **2. Check if Two Strings are Anagrams (Sorting & Hashing Approach)**  
📌 **Algorithm:** **Sorting / Hashing**  
📌 **Trick:** Sort both strings or use a frequency array for characters.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

bool isAnagram(string s, string t) {

if (s.size() != t.size()) return false;

vector<int> count(26, 0);

for (char c : s) count[c - 'a']++;

for (char c : t) {

if (--count[c - 'a'] < 0) return false;

}

return true;

}

int main() {

string s = "listen", t = "silent";

cout << (isAnagram(s, t) ? "Yes" : "No");

}

✅ **3. Implement strstr() (KMP Algorithm - String Matching)**  
📌 **Algorithm:** **KMP (Knuth-Morris-Pratt) Algorithm**  
📌 **Trick:** Precompute the **LPS (Longest Prefix Suffix)** array for efficient matching.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

vector<int> computeLPS(string pat) {

int n = pat.size(), j = 0;

vector<int> lps(n, 0);

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

while (j > 0 && pat[i] != pat[j]) j = lps[j - 1];

if (pat[i] == pat[j]) lps[i] = ++j;

}

return lps;

}

int strStr(string text, string pattern) {

if (pattern.empty()) return 0;

vector<int> lps = computeLPS(pattern);

int j = 0;

for (int i = 0; i < text.size(); i++) {

while (j > 0 && text[i] != pattern[j]) j = lps[j - 1];

if (text[i] == pattern[j]) j++;

if (j == pattern.size()) return i - j + 1;

}

return -1;

}

int main() {

string text = "hello", pattern = "ll";

cout << strStr(text, pattern);

}

✅ **4. Find the Longest Common Prefix in a List of Strings (Binary Search Approach)**  
📌 **Algorithm:** **Binary Search / Trie**  
📌 **Trick:** Find the minimum length string and compare all prefixes.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

string longestCommonPrefix(vector<string>& strs) {

if (strs.empty()) return "";

string prefix = strs[0];

for (int i = 1; i < strs.size(); i++) {

while (strs[i].find(prefix) != 0)

prefix = prefix.substr(0, prefix.size() - 1);

if (prefix.empty()) return "";

}

return prefix;

}

int main() {

vector<string> strs = {"flower", "flow", "flight"};

cout << longestCommonPrefix(strs);

}

✅ **5. Count and Say Sequence (Recursive Approach)**  
📌 **Algorithm:** **Recursive / Iterative Simulation**  
📌 **Trick:** Simulate the **count & say** pattern using recursion or iteration.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

string countAndSay(int n) {

if (n == 1) return "1";

string prev = countAndSay(n - 1), res = "";

int count = 1;

for (int i = 1; i < prev.size(); i++) {

if (prev[i] == prev[i - 1]) count++;

else {

res += to\_string(count) + prev[i - 1];

count = 1;

}

}

res += to\_string(count) + prev.back();

return res;

}

int main() {

int n = 5;

cout << countAndSay(n);

}

## ****📂 DSA Sheet Structure (Example for**** Sorting & Searching ****Section)****

### ****🟢**** Sorting & Searching

✅ **1. Longest Palindromic Substring (Expand Around Center Algorithm)**  
📌 **Algorithm:** **Expand Around Center**  
📌 **Trick:** For each character, expand outward while the substring remains a palindrome.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

string longestPalindrome(string s) {

int start = 0, maxLen = 0;

auto expand = [&](int l, int r) {

while (l >= 0 && r < s.size() && s[l] == s[r]) {

if (r - l + 1 > maxLen) {

start = l;

maxLen = r - l + 1;

}

l--; r++;

}

};

for (int i = 0; i < s.size(); i++) {

expand(i, i); // Odd length palindrome

expand(i, i+1); // Even length palindrome

}

return s.substr(start, maxLen);

}

int main() {

string s = "babad";

cout << longestPalindrome(s);

}

✅ **2. Check if Two Strings are Anagrams (Sorting & Hashing Approach)**  
📌 **Algorithm:** **Sorting / Hashing**  
📌 **Trick:** Sort both strings or use a frequency array for characters.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

bool isAnagram(string s, string t) {

if (s.size() != t.size()) return false;

vector<int> count(26, 0);

for (char c : s) count[c - 'a']++;

for (char c : t) {

if (--count[c - 'a'] < 0) return false;

}

return true;

}

int main() {

string s = "listen", t = "silent";

cout << (isAnagram(s, t) ? "Yes" : "No");

}

✅ **3. Implement strstr() (KMP Algorithm - String Matching)**  
📌 **Algorithm:** **KMP (Knuth-Morris-Pratt) Algorithm**  
📌 **Trick:** Precompute the **LPS (Longest Prefix Suffix)** array for efficient matching.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

vector<int> computeLPS(string pat) {

int n = pat.size(), j = 0;

vector<int> lps(n, 0);

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

while (j > 0 && pat[i] != pat[j]) j = lps[j - 1];

if (pat[i] == pat[j]) lps[i] = ++j;

}

return lps;

}

int strStr(string text, string pattern) {

if (pattern.empty()) return 0;

vector<int> lps = computeLPS(pattern);

int j = 0;

for (int i = 0; i < text.size(); i++) {

while (j > 0 && text[i] != pattern[j]) j = lps[j - 1];

if (text[i] == pattern[j]) j++;

if (j == pattern.size()) return i - j + 1;

}

return -1;

}

int main() {

string text = "hello", pattern = "ll";

cout << strStr(text, pattern);

}

✅ **4. Find the Longest Common Prefix in a List of Strings (Binary Search Approach)**  
📌 **Algorithm:** **Binary Search / Trie**  
📌 **Trick:** Find the minimum length string and compare all prefixes.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

string longestCommonPrefix(vector<string>& strs) {

if (strs.empty()) return "";

string prefix = strs[0];

for (int i = 1; i < strs.size(); i++) {

while (strs[i].find(prefix) != 0)

prefix = prefix.substr(0, prefix.size() - 1);

if (prefix.empty()) return "";

}

return prefix;

}

int main() {

vector<string> strs = {"flower", "flow", "flight"};

cout << longestCommonPrefix(strs);

}

✅ **5. Count and Say Sequence (Recursive Approach)**  
📌 **Algorithm:** **Recursive / Iterative Simulation**  
📌 **Trick:** Simulate the **count & say** pattern using recursion or iteration.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

string countAndSay(int n) {

if (n == 1) return "1";

string prev = countAndSay(n - 1), res = "";

int count = 1;

for (int i = 1; i < prev.size(); i++) {

if (prev[i] == prev[i - 1]) count++;

else {

res += to\_string(count) + prev[i - 1];

count = 1;

}

}

res += to\_string(count) + prev.back();

return res;

}

int main() {

int n = 5;

cout << countAndSay(n);

}

## ****📂 DSA Sheet Structure (Example for**** Hashing ****Section)****

### ****🟢**** Hashing

✅ **1. Two Sum (Using HashMap for O(1) Lookup)**  
📌 **Algorithm:** **Hash Map (Unordered Map in C++)**  
📌 **Trick:** Store each element’s index in a hashmap, then check for target - arr[i].  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

vector<int> twoSum(vector<int>& nums, int target) {

unordered\_map<int, int> mp;

for (int i = 0; i < nums.size(); i++) {

int complement = target - nums[i];

if (mp.find(complement) != mp.end())

return {mp[complement], i};

mp[nums[i]] = i;

}

return {};

}

int main() {

vector<int> nums = {2, 7, 11, 15};

int target = 9;

vector<int> res = twoSum(nums, target);

cout << res[0] << " " << res[1];

}

✅ **2. Longest Consecutive Sequence (Using HashSet for O(n) Complexity)**  
📌 **Algorithm:** **Hash Set (Unordered Set in C++)**  
📌 **Trick:** Insert all numbers into a set, then check only for the **starting numbers of sequences**.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int longestConsecutive(vector<int>& nums) {

unordered\_set<int> st(nums.begin(), nums.end());

int longest = 0;

for (int num : nums) {

if (st.find(num - 1) == st.end()) { // Start of a new sequence

int currNum = num, streak = 1;

while (st.find(currNum + 1) != st.end()) {

currNum++;

streak++;

}

longest = max(longest, streak);

}

}

return longest;

}

int main() {

vector<int> nums = {100, 4, 200, 1, 3, 2};

cout << longestConsecutive(nums);

}

✅ **3. Subarray Sum Equals K (Using Prefix Sum & HashMap)**  
📌 **Algorithm:** **Prefix Sum with HashMap**  
📌 **Trick:** Store **prefix sum frequencies** to check for sum - k in O(1).  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int subarraySum(vector<int>& nums, int k) {

unordered\_map<int, int> prefixSum;

prefixSum[0] = 1;

int sum = 0, count = 0;

for (int num : nums) {

sum += num;

if (prefixSum.find(sum - k) != prefixSum.end())

count += prefixSum[sum - k];

prefixSum[sum]++;

}

return count;

}

int main() {

vector<int> nums = {1, 1, 1};

int k = 2;

cout << subarraySum(nums, k);

}

✅ **4. Find Duplicates in an Array (Using HashMap for Frequency Counting)**  
📌 **Algorithm:** **Hash Map for Frequency Counting**  
📌 **Trick:** Store the **frequency of elements** and print those appearing more than once.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

vector<int> findDuplicates(vector<int>& nums) {

unordered\_map<int, int> freq;

vector<int> duplicates;

for (int num : nums) {

freq[num]++;

if (freq[num] == 2) duplicates.push\_back(num);

}

return duplicates;

}

int main() {

vector<int> nums = {4, 3, 2, 7, 8, 2, 3, 1};

vector<int> res = findDuplicates(nums);

for (int num : res) cout << num << " ";

}

✅ **5. Find First Non-Repeating Character (Using HashMap for Counting)**  
📌 **Algorithm:** **Hash Map + Order Maintenance**  
📌 **Trick:** First store frequency, then check the first element with count = 1.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

char firstUniqChar(string s) {

unordered\_map<char, int> freq;

for (char c : s) freq[c]++;

for (char c : s) {

if (freq[c] == 1) return c;

}

return '\_'; // Return '\_' if no unique character exists

}

int main() {

string s = "leetcode";

cout << firstUniqChar(s);

}

**🔥 Key Hashing Concepts Covered**

✔ **HashMap (unordered\_map) for O(1) Lookup**  
✔ **HashSet (unordered\_set) for Quick Searches**  
✔ **Prefix Sum & Frequency Maps for Subarrays**  
✔ **Efficient Handling of Duplicate / Unique Elements**

## ****📂 DSA Sheet Structure (Example for Linked List Section)****

### ****🟢 Linked List****

✅ **1. Reverse a Linked List (Iterative & Recursive Approach)**  
📌 **Algorithm:** **Two Pointer Method / Recursion**  
📌 **Trick:** Use **prev, curr, next** pointers and iterate through the list.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct ListNode {

int val;

ListNode\* next;

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

};

// Iterative Approach

ListNode\* reverseList(ListNode\* head) {

ListNode \*prev = NULL, \*curr = head;

while (curr) {

ListNode\* nextNode = curr->next;

curr->next = prev;

prev = curr;

curr = nextNode;

}

return prev;

}

// Recursive Approach

ListNode\* reverseListRecursive(ListNode\* head) {

if (!head || !head->next) return head;

ListNode\* newHead = reverseListRecursive(head->next);

head->next->next = head;

head->next = NULL;

return newHead;

}

// Utility function to print a linked list

void printList(ListNode\* head) {

while (head) {

cout << head->val << " ";

head = head->next;

}

}

int main() {

ListNode\* head = new ListNode(1);

head->next = new ListNode(2);

head->next->next = new ListNode(3);

head->next->next->next = new ListNode(4);

head = reverseList(head);

printList(head);

}

✅ **2. Detect a Cycle in a Linked List (Floyd’s Cycle Detection Algorithm)**  
📌 **Algorithm:** **Floyd’s Tortoise & Hare (Slow & Fast Pointers)**  
📌 **Trick:** Use two pointers, one moving twice as fast as the other. If they meet, there's a cycle.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct ListNode {

int val;

ListNode\* next;

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

};

bool hasCycle(ListNode\* head) {

ListNode \*slow = head, \*fast = head;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) return true;

}

return false;

}

int main() {

ListNode\* head = new ListNode(1);

head->next = new ListNode(2);

head->next->next = new ListNode(3);

head->next->next->next = head->next; // Creating a cycle

cout << (hasCycle(head) ? "Cycle Detected" : "No Cycle");

}

✅ **3. Find the Middle of a Linked List (Slow & Fast Pointer Approach)**  
📌 **Algorithm:** **Two Pointers (Tortoise & Hare)**  
📌 **Trick:** Move **slow by 1 step** and **fast by 2 steps**, when fast reaches the end, slow is at the middle.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct ListNode {

int val;

ListNode\* next;

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

};

ListNode\* findMiddle(ListNode\* head) {

ListNode \*slow = head, \*fast = head;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

}

return slow;

}

int main() {

ListNode\* head = new ListNode(1);

head->next = new ListNode(2);

head->next->next = new ListNode(3);

head->next->next->next = new ListNode(4);

head->next->next->next->next = new ListNode(5);

cout << "Middle element: " << findMiddle(head)->val;

}

✅ **4. Merge Two Sorted Linked Lists (Recursive Approach)**  
📌 **Algorithm:** **Merge Sort for Linked List**  
📌 **Trick:** Compare **head nodes** and recursively merge.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct ListNode {

int val;

ListNode\* next;

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

};

ListNode\* mergeTwoLists(ListNode\* l1, ListNode\* l2) {

if (!l1) return l2;

if (!l2) return l1;

if (l1->val < l2->val) {

l1->next = mergeTwoLists(l1->next, l2);

return l1;

} else {

l2->next = mergeTwoLists(l1, l2->next);

return l2;

}

}

// Utility function to print a linked list

void printList(ListNode\* head) {

while (head) {

cout << head->val << " ";

head = head->next;

}

}

int main() {

ListNode\* l1 = new ListNode(1);

l1->next = new ListNode(3);

l1->next->next = new ListNode(5);

ListNode\* l2 = new ListNode(2);

l2->next = new ListNode(4);

l2->next->next = new ListNode(6);

ListNode\* mergedHead = mergeTwoLists(l1, l2);

printList(mergedHead);

}

✅ **5. Remove N-th Node from End of List (Two Pointer Trick)**  
📌 **Algorithm:** **Fast & Slow Pointer**  
📌 **Trick:** Move fast pointer n steps ahead, then move both until fast reaches the end.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct ListNode {

int val;

ListNode\* next;

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

};

ListNode\* removeNthFromEnd(ListNode\* head, int n) {

ListNode\* dummy = new ListNode(0);

dummy->next = head;

ListNode \*fast = dummy, \*slow = dummy;

for (int i = 0; i <= n; i++) fast = fast->next;

while (fast) {

fast = fast->next;

slow = slow->next;

}

slow->next = slow->next->next;

return dummy->next;

}

// Utility function to print a linked list

void printList(ListNode\* head) {

while (head) {

cout << head->val << " ";

head = head->next;

}

}

int main() {

ListNode\* head = new ListNode(1);

head->next = new ListNode(2);

head->next->next = new ListNode(3);

head->next->next->next = new ListNode(4);

head->next->next->next->next = new ListNode(5);

head = removeNthFromEnd(head, 2);

printList(head);

}

**🔥 Key Linked List Concepts Covered**

✔ **Reverse a Linked List (Iterative & Recursive)**  
✔ **Detect a Cycle (Floyd's Cycle Detection Algorithm)**  
✔ **Find the Middle Element (Slow & Fast Pointers)**  
✔ **Merge Two Sorted Lists (Recursion)**  
✔ **Remove N-th Node from End (Two Pointer Trick)**

## ****📂 DSA Sheet Structure (Example for**** Stack & Queue ****Section)****

### ****🟢**** Stack & Queue

✅ **1. Next Greater Element (Using Monotonic Stack)**  
📌 **Algorithm:** **Monotonic Stack**  
📌 **Trick:** Traverse **from right to left**, maintain a decreasing stack.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

vector<int> nextGreaterElement(vector<int>& nums) {

stack<int> st;

vector<int> res(nums.size(), -1);

for (int i = nums.size() - 1; i >= 0; i--) {

while (!st.empty() && st.top() <= nums[i])

st.pop();

if (!st.empty()) res[i] = st.top();

st.push(nums[i]);

}

return res;

}

int main() {

vector<int> nums = {2, 1, 3, 5, 4};

vector<int> res = nextGreaterElement(nums);

for (int num : res) cout << num << " ";

}

✅ **2. Valid Parentheses (Using Stack for Matching Brackets)**  
📌 **Algorithm:** **Stack for Balanced Brackets**  
📌 **Trick:** Use a stack to **push opening brackets** and **pop on matching closing brackets**.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

bool isValid(string s) {

stack<char> st;

unordered\_map<char, char> pairs = {{')', '('}, {']', '['}, {'}', '{'}};

for (char c : s) {

if (pairs.count(c)) {

if (st.empty() || st.top() != pairs[c]) return false;

st.pop();

} else {

st.push(c);

}

}

return st.empty();

}

int main() {

string s = "{[()]}";

cout << (isValid(s) ? "Valid" : "Invalid");

}

✅ **3. Largest Rectangle in Histogram (Using Stack to Maintain Heights)**  
📌 **Algorithm:** **Monotonic Stack for Max Area Calculation**  
📌 **Trick:** **Store indices of increasing heights**, pop when a lower height is found.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int largestRectangleArea(vector<int>& heights) {

stack<int> st;

int maxArea = 0;

heights.push\_back(0); // Sentinel for easy calculation

for (int i = 0; i < heights.size(); i++) {

while (!st.empty() && heights[st.top()] > heights[i]) {

int h = heights[st.top()];

st.pop();

int w = st.empty() ? i : i - st.top() - 1;

maxArea = max(maxArea, h \* w);

}

st.push(i);

}

return maxArea;

}

int main() {

vector<int> heights = {2, 1, 5, 6, 2, 3};

cout << largestRectangleArea(heights);

}

**🔥 Queue Problems**

✅ **4. Implement a Queue Using Two Stacks**  
📌 **Algorithm:** **Two Stacks (Push and Pop Efficient Implementations)**  
📌 **Trick:** Use two stacks - one for push (input stack) and one for pop (output stack).  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

class MyQueue {

stack<int> input, output;

public:

void push(int x) { input.push(x); }

int pop() {

if (output.empty()) {

while (!input.empty()) {

output.push(input.top());

input.pop();

}

}

int val = output.top();

output.pop();

return val;

}

int peek() {

if (output.empty()) {

while (!input.empty()) {

output.push(input.top());

input.pop();

}

}

return output.top();

}

bool empty() { return input.empty() && output.empty(); }

};

int main() {

MyQueue q;

q.push(1);

q.push(2);

cout << q.peek() << endl; // 1

cout << q.pop() << endl; // 1

cout << q.empty() << endl; // false

}

✅ **5. Sliding Window Maximum (Using Monotonic Deque for O(n))**  
📌 **Algorithm:** **Deque (Double-Ended Queue) for Optimized Window Handling**  
📌 **Trick:** Store **indices of elements**, remove smaller elements from back.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

vector<int> maxSlidingWindow(vector<int>& nums, int k) {

deque<int> dq;

vector<int> res;

for (int i = 0; i < nums.size(); i++) {

// Remove elements out of the current window

if (!dq.empty() && dq.front() == i - k) dq.pop\_front();

// Remove smaller elements as they won't be needed

while (!dq.empty() && nums[dq.back()] < nums[i]) dq.pop\_back();

dq.push\_back(i);

if (i >= k - 1) res.push\_back(nums[dq.front()]); // Front contains max

}

return res;

}

int main() {

vector<int> nums = {1, 3, -1, -3, 5, 3, 6, 7};

int k = 3;

vector<int> res = maxSlidingWindow(nums, k);

for (int num : res) cout << num << " ";

}

**🔥 Key Stack & Queue Concepts Covered**

✔ **Monotonic Stack (Next Greater Element, Histogram Area)**  
✔ **Balanced Parentheses Matching (Stack for Brackets)**  
✔ **Queue using Two Stacks (Stack-Based Queue Implementation)**  
✔ **Sliding Window Maximum (Optimized Deque for O(n) Complexity)**

## ****📂 DSA Sheet Structure (Example for**** Recursion & Backtracking ****Section)****

### ****🟢**** Recursion & Backtracking

✅ **1. Print All Subsequences of an Array**  
📌 **Algorithm:** **Recursive Subset Generation**  
📌 **Trick:** Either **pick** or **don’t pick** an element to generate all subsequences.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

void printSubsequences(vector<int>& arr, vector<int>& sub, int index) {

if (index == arr.size()) {

for (int num : sub) cout << num << " ";

cout << endl;

return;

}

sub.push\_back(arr[index]);

printSubsequences(arr, sub, index + 1);

sub.pop\_back();

printSubsequences(arr, sub, index + 1);

}

int main() {

vector<int> arr = {1, 2, 3};

vector<int> sub;

printSubsequences(arr, sub, 0);

}

✅ **2. Generate All Possible Parentheses (Balanced Parentheses)**  
📌 **Algorithm:** **Backtracking (Recursive State Change)**  
📌 **Trick:** Maintain **count of open and close brackets**, ensuring valid placement.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

void generateParentheses(string s, int open, int close, int n) {

if (s.size() == 2 \* n) {

cout << s << endl;

return;

}

if (open < n) generateParentheses(s + "(", open + 1, close, n);

if (close < open) generateParentheses(s + ")", open, close + 1, n);

}

int main() {

int n = 3;

generateParentheses("", 0, 0, n);

}

✅ **3. Tower of Hanoi (Classic Recursion)**  
📌 **Algorithm:** **Recursive Disk Movement**  
📌 **Trick:** Move n-1 disks to auxiliary peg, move the nth disk to the target, then move n-1 disks from auxiliary to target.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

void towerOfHanoi(int n, char from, char to, char aux) {

if (n == 0) return;

towerOfHanoi(n - 1, from, aux, to);

cout << "Move disk " << n << " from " << from << " to " << to << endl;

towerOfHanoi(n - 1, aux, to, from);

}

int main() {

int n = 3;

towerOfHanoi(n, 'A', 'C', 'B');

}

**🔥 Backtracking Problems**

✅ **4. N-Queens Problem (Place Queens Safely on Chessboard)**  
📌 **Algorithm:** **Backtracking with Safe Placement Check**  
📌 **Trick:** Use an **array to track safe columns & diagonals** to optimize backtracking.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

bool isSafe(vector<string>& board, int row, int col, int n) {

for (int i = 0; i < row; i++)

if (board[i][col] == 'Q') return false;

for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)

if (board[i][j] == 'Q') return false;

for (int i = row, j = col; i >= 0 && j < n; i--, j++)

if (board[i][j] == 'Q') return false;

return true;

}

void solveNQueens(vector<string>& board, int row, int n) {

if (row == n) {

for (string s : board) cout << s << endl;

cout << endl;

return;

}

for (int col = 0; col < n; col++) {

if (isSafe(board, row, col, n)) {

board[row][col] = 'Q';

solveNQueens(board, row + 1, n);

board[row][col] = '.';

}

}

}

int main() {

int n = 4;

vector<string> board(n, string(n, '.'));

solveNQueens(board, 0, n);

}

✅ **5. Rat in a Maze (Find All Paths in a Grid)**  
📌 **Algorithm:** **Backtracking with Direction Movement**  
📌 **Trick:** Try **all four possible moves** (down, left, right, up) and backtrack when stuck.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

void solveMaze(vector<vector<int>>& maze, int x, int y, vector<string>& paths, string path, vector<vector<int>>& visited) {

int n = maze.size();

if (x == n - 1 && y == n - 1) {

paths.push\_back(path);

return;

}

string dir = "DLRU";

int dx[] = {1, 0, 0, -1};

int dy[] = {0, -1, 1, 0};

for (int i = 0; i < 4; i++) {

int newX = x + dx[i];

int newY = y + dy[i];

if (newX >= 0 && newY >= 0 && newX < n && newY < n && !visited[newX][newY] && maze[newX][newY] == 1) {

visited[newX][newY] = 1;

solveMaze(maze, newX, newY, paths, path + dir[i], visited);

visited[newX][newY] = 0;

}

}

}

int main() {

vector<vector<int>> maze = {{1, 0, 0, 0},

{1, 1, 0, 1},

{0, 1, 0, 0},

{1, 1, 1, 1}};

int n = maze.size();

vector<vector<int>> visited(n, vector<int>(n, 0));

vector<string> paths;

visited[0][0] = 1;

solveMaze(maze, 0, 0, paths, "", visited);

for (string path : paths) cout << path << endl;

}

**🔥 Key Recursion & Backtracking Concepts Covered**

✔ **Recursion: Print Subsequences, Generate Parentheses, Tower of Hanoi**  
✔ **Backtracking: N-Queens, Rat in a Maze**  
✔ **Optimized Safe Placement (N-Queens), Direction Movement (Maze)**

## ****📂 DSA Sheet Structure (Example for**** Binary Tree & BST ****Section)****

### ****🟢**** Binary Tree & BST

✅ **1. Lowest Common Ancestor (LCA) of Two Nodes in a Binary Tree**  
📌 **Algorithm:** **Recursive DFS Approach**  
📌 **Trick:** If a node is found in both left and right subtrees, it is the LCA.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(NULL), right(NULL) {}

};

TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

if (!root || root == p || root == q) return root;

TreeNode\* left = lowestCommonAncestor(root->left, p, q);

TreeNode\* right = lowestCommonAncestor(root->right, p, q);

return left ? (right ? root : left) : right;

}

int main() {

TreeNode\* root = new TreeNode(3);

root->left = new TreeNode(5);

root->right = new TreeNode(1);

root->left->left = new TreeNode(6);

root->left->right = new TreeNode(2);

root->right->left = new TreeNode(0);

root->right->right = new TreeNode(8);

TreeNode\* lca = lowestCommonAncestor(root, root->left, root->right);

cout << "LCA: " << lca->val;

}

✅ **2. Zig-Zag Level Order Traversal**  
📌 **Algorithm:** **BFS (Level Order Traversal with Deque)**  
📌 **Trick:** Use a **queue** for BFS and maintain **left-to-right or right-to-left order** using a flag.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(NULL), right(NULL) {}

};

vector<vector<int>> zigzagLevelOrder(TreeNode\* root) {

vector<vector<int>> res;

if (!root) return res;

queue<TreeNode\*> q;

q.push(root);

bool leftToRight = true;

while (!q.empty()) {

int size = q.size();

vector<int> level(size);

for (int i = 0; i < size; i++) {

TreeNode\* node = q.front();

q.pop();

int index = leftToRight ? i : (size - 1 - i);

level[index] = node->val;

if (node->left) q.push(node->left);

if (node->right) q.push(node->right);

}

res.push\_back(level);

leftToRight = !leftToRight;

}

return res;

}

int main() {

TreeNode\* root = new TreeNode(1);

root->left = new TreeNode(2);

root->right = new TreeNode(3);

root->left->left = new TreeNode(4);

root->left->right = new TreeNode(5);

root->right->left = new TreeNode(6);

root->right->right = new TreeNode(7);

vector<vector<int>> res = zigzagLevelOrder(root);

for (auto level : res) {

for (int val : level) cout << val << " ";

cout << endl;

}

}

✅ **3. Diameter of a Binary Tree**  
📌 **Algorithm:** **DFS with Height Calculation**  
📌 **Trick:** Maintain a **global variable for max diameter** while computing height.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(NULL), right(NULL) {}

};

int maxDiameter = 0;

int findHeight(TreeNode\* root) {

if (!root) return 0;

int leftHeight = findHeight(root->left);

int rightHeight = findHeight(root->right);

maxDiameter = max(maxDiameter, leftHeight + rightHeight);

return 1 + max(leftHeight, rightHeight);

}

int diameterOfBinaryTree(TreeNode\* root) {

findHeight(root);

return maxDiameter;

}

int main() {

TreeNode\* root = new TreeNode(1);

root->left = new TreeNode(2);

root->right = new TreeNode(3);

root->left->left = new TreeNode(4);

root->left->right = new TreeNode(5);

cout << "Diameter: " << diameterOfBinaryTree(root);

}

**🔥 Binary Search Tree (BST) Problems**

✅ **4. Validate a Binary Search Tree (BST)**  
📌 **Algorithm:** **Inorder Traversal with Range Check**  
📌 **Trick:** Ensure **all left nodes are smaller** and **all right nodes are larger** using min/max range.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(NULL), right(NULL) {}

};

bool validateBST(TreeNode\* root, long minVal, long maxVal) {

if (!root) return true;

if (root->val <= minVal || root->val >= maxVal) return false;

return validateBST(root->left, minVal, root->val) &&

validateBST(root->right, root->val, maxVal);

}

bool isValidBST(TreeNode\* root) {

return validateBST(root, LONG\_MIN, LONG\_MAX);

}

int main() {

TreeNode\* root = new TreeNode(2);

root->left = new TreeNode(1);

root->right = new TreeNode(3);

cout << (isValidBST(root) ? "Valid BST" : "Invalid BST");

}

✅ **5. Kth Smallest Element in a BST**  
📌 **Algorithm:** **Inorder Traversal (Sorted Order in BST)**  
📌 **Trick:** Use **inorder traversal** and count nodes to find the Kth smallest element.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(NULL), right(NULL) {}

};

void inorder(TreeNode\* root, int& k, int& ans) {

if (!root) return;

inorder(root->left, k, ans);

if (--k == 0) ans = root->val;

inorder(root->right, k, ans);

}

int kthSmallest(TreeNode\* root, int k) {

int ans = -1;

inorder(root, k, ans);

return ans;

}

int main() {

TreeNode\* root = new TreeNode(5);

root->left = new TreeNode(3);

root->right = new TreeNode(6);

root->left->left = new TreeNode(2);

root->left->right = new TreeNode(4);

root->left->left->left = new TreeNode(1);

int k = 3;

cout << "Kth Smallest: " << kthSmallest(root, k);

}

**🔥 Key Binary Tree & BST Concepts Covered**

✔ **Binary Tree: Lowest Common Ancestor, Zig-Zag Traversal, Diameter Calculation**  
✔ **BST: Validation, Kth Smallest Element**  
✔ **Efficient Traversals (BFS, DFS, Inorder, Preorder, Postorder)**

## ****📂 DSA Sheet Structure (Example for**** Heap & Priority Queue ****Section)****

### ****🟢**** Heap & Priority Queue

✅ **1. Find Kth Largest Element in an Array**  
📌 **Algorithm:** **Min-Heap (Priority Queue)**  
📌 **Trick:** Maintain a **min-heap of size K**. If the heap size exceeds K, remove the smallest element.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int findKthLargest(vector<int>& nums, int k) {

priority\_queue<int, vector<int>, greater<int>> minHeap;

for (int num : nums) {

minHeap.push(num);

if (minHeap.size() > k) minHeap.pop();

}

return minHeap.top();

}

int main() {

vector<int> nums = {3, 2, 3, 1, 2, 4, 5, 5, 6};

int k = 4;

cout << findKthLargest(nums, k) << endl; // Output: 4

}

✅ **2. Merge K Sorted Linked Lists**  
📌 **Algorithm:** **Min-Heap (Priority Queue of Node Pointers)**  
📌 **Trick:** Push the first node of each list into a **min-heap**. Pop the smallest, push its next node.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct ListNode {

int val;

ListNode\* next;

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

};

struct Compare {

bool operator()(ListNode\* a, ListNode\* b) {

return a->val > b->val;

}

};

ListNode\* mergeKLists(vector<ListNode\*>& lists) {

priority\_queue<ListNode\*, vector<ListNode\*>, Compare> minHeap;

for (auto list : lists)

if (list) minHeap.push(list);

ListNode\* dummy = new ListNode(0);

ListNode\* tail = dummy;

while (!minHeap.empty()) {

tail->next = minHeap.top();

minHeap.pop();

tail = tail->next;

if (tail->next) minHeap.push(tail->next);

}

return dummy->next;

}

✅ **3. Top K Frequent Elements in an Array**  
📌 **Algorithm:** **Min-Heap (Frequency Count & Heap of Size K)**  
📌 **Trick:** Use **unordered\_map** for frequency count, then maintain a **min-heap** of top K elements.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

vector<int> topKFrequent(vector<int>& nums, int k) {

unordered\_map<int, int> freq;

for (int num : nums) freq[num]++;

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> minHeap;

for (auto& it : freq) {

minHeap.push({it.second, it.first});

if (minHeap.size() > k) minHeap.pop();

}

vector<int> res;

while (!minHeap.empty()) {

res.push\_back(minHeap.top().second);

minHeap.pop();

}

return res;

}

✅ **4. Median of a Stream of Integers**  
📌 **Algorithm:** **Two Heaps (Max-Heap for Left, Min-Heap for Right)**  
📌 **Trick:** Keep the **left half** of numbers in a **max-heap** and the **right half** in a **min-heap**. Balance sizes.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

class MedianFinder {

priority\_queue<int> left; // Max-Heap (Lower half)

priority\_queue<int, vector<int>, greater<int>> right; // Min-Heap (Upper half)

public:

void addNum(int num) {

if (left.empty() || num <= left.top()) left.push(num);

else right.push(num);

if (left.size() > right.size() + 1) {

right.push(left.top());

left.pop();

} else if (right.size() > left.size()) {

left.push(right.top());

right.pop();

}

}

double findMedian() {

if (left.size() > right.size()) return left.top();

return (left.top() + right.top()) / 2.0;

}

};

int main() {

MedianFinder mf;

mf.addNum(1);

mf.addNum(2);

cout << mf.findMedian() << endl; // Output: 1.5

mf.addNum(3);

cout << mf.findMedian() << endl; // Output: 2

}

✅ **5. Task Scheduler (CPU Scheduling with Cooldown Time)**  
📌 **Algorithm:** **Max-Heap (Task Frequency Count & Greedy Scheduling)**  
📌 **Trick:** Use a **max-heap to store the most frequent tasks** and execute them with **cooldown periods**.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int leastInterval(vector<char>& tasks, int n) {

unordered\_map<char, int> freq;

for (char task : tasks) freq[task]++;

priority\_queue<int> maxHeap;

for (auto f : freq) maxHeap.push(f.second);

int cycles = 0;

while (!maxHeap.empty()) {

vector<int> temp;

for (int i = 0; i <= n; i++) {

if (!maxHeap.empty()) {

temp.push\_back(maxHeap.top() - 1);

maxHeap.pop();

}

}

for (int t : temp)

if (t > 0) maxHeap.push(t);

cycles += maxHeap.empty() ? temp.size() : n + 1;

}

return cycles;

}

int main() {

vector<char> tasks = {'A', 'A', 'A', 'B', 'B', 'B'};

int n = 2;

cout << leastInterval(tasks, n) << endl; // Output: 8

}

**🔥 Key Heap & Priority Queue Concepts Covered**

✔ **Min-Heap for Kth Largest Element, Top K Frequent Elements**  
✔ **Max-Heap for Task Scheduling, Median Calculation**  
✔ **Priority Queue for Efficient Merging of K Sorted Lists**

## ****📂 DSA Sheet Structure (Example for**** Graph Theory ****Section)****

### ****🟢**** Graph Theory

✅ **1. Find the Number of Connected Components in an Undirected Graph**  
📌 **Algorithm:** **DFS / BFS (Graph Traversal)**  
📌 **Trick:** Iterate over all nodes, run **DFS/BFS** for each unvisited node, and count the connected components.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

void dfs(int node, vector<vector<int>>& adj, vector<bool>& visited) {

visited[node] = true;

for (int neighbor : adj[node]) {

if (!visited[neighbor]) dfs(neighbor, adj, visited);

}

}

int countConnectedComponents(int n, vector<pair<int, int>>& edges) {

vector<vector<int>> adj(n);

vector<bool> visited(n, false);

for (auto& edge : edges) {

adj[edge.first].push\_back(edge.second);

adj[edge.second].push\_back(edge.first);

}

int count = 0;

for (int i = 0; i < n; i++) {

if (!visited[i]) {

dfs(i, adj, visited);

count++;

}

}

return count;

}

int main() {

int n = 5;

vector<pair<int, int>> edges = {{0, 1}, {1, 2}, {3, 4}};

cout << countConnectedComponents(n, edges) << endl; // Output: 2

}

✅ **2. Detect a Cycle in a Graph (Directed & Undirected)**  
📌 **Algorithm:** **DFS for Undirected Graph, DFS with Recursion Stack for Directed Graph**  
📌 **Trick:** For **undirected graphs**, track parent nodes. For **directed graphs**, use a **recursion stack**.  
💻 **Code (Directed Graph - DFS)**:

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

bool dfs(int node, vector<vector<int>>& adj, vector<int>& visited, vector<int>& recStack) {

visited[node] = 1;

recStack[node] = 1;

for (int neighbor : adj[node]) {

if (!visited[neighbor] && dfs(neighbor, adj, visited, recStack)) return true;

else if (recStack[neighbor]) return true;

}

recStack[node] = 0;

return false;

}

bool detectCycle(int n, vector<pair<int, int>>& edges) {

vector<vector<int>> adj(n);

for (auto& edge : edges) adj[edge.first].push\_back(edge.second);

vector<int> visited(n, 0), recStack(n, 0);

for (int i = 0; i < n; i++)

if (!visited[i] && dfs(i, adj, visited, recStack)) return true;

return false;

}

int main() {

int n = 4;

vector<pair<int, int>> edges = {{0, 1}, {1, 2}, {2, 0}, {2, 3}};

cout << detectCycle(n, edges) << endl; // Output: 1 (true)

}

✅ **3. Find the Shortest Path in an Unweighted Graph (BFS Approach)**  
📌 **Algorithm:** **Breadth-First Search (BFS)**  
📌 **Trick:** BFS guarantees the shortest path in **unweighted graphs** by exploring level-wise.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

vector<int> shortestPath(int n, vector<pair<int, int>>& edges, int src) {

vector<vector<int>> adj(n);

for (auto& edge : edges) {

adj[edge.first].push\_back(edge.second);

adj[edge.second].push\_back(edge.first);

}

vector<int> dist(n, INT\_MAX);

queue<int> q;

dist[src] = 0;

q.push(src);

while (!q.empty()) {

int node = q.front();

q.pop();

for (int neighbor : adj[node]) {

if (dist[node] + 1 < dist[neighbor]) {

dist[neighbor] = dist[node] + 1;

q.push(neighbor);

}

}

}

return dist;

}

int main() {

int n = 6;

vector<pair<int, int>> edges = {{0, 1}, {0, 2}, {1, 3}, {2, 3}, {3, 4}, {4, 5}};

vector<int> dist = shortestPath(n, edges, 0);

for (int d : dist) cout << d << " "; // Output: 0 1 1 2 3 4

}

✅ **4. Dijkstra’s Algorithm (Shortest Path in Weighted Graph)**  
📌 **Algorithm:** **Priority Queue (Min-Heap) + Dijkstra's Algorithm**  
📌 **Trick:** Use a **min-heap (priority queue)** to always expand the node with the smallest known distance.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

vector<int> dijkstra(int n, vector<pair<int, int>> adj[], int src) {

vector<int> dist(n, INT\_MAX);

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

dist[src] = 0;

pq.push({0, src});

while (!pq.empty()) {

int d = pq.top().first, node = pq.top().second;

pq.pop();

if (d > dist[node]) continue;

for (auto& edge : adj[node]) {

int nextNode = edge.first, weight = edge.second;

if (dist[node] + weight < dist[nextNode]) {

dist[nextNode] = dist[node] + weight;

pq.push({dist[nextNode], nextNode});

}

}

}

return dist;

}

int main() {

int n = 5;

vector<pair<int, int>> adj[5] = {

{{1, 10}, {2, 3}},

{{3, 2}},

{{1, 4}, {3, 8}},

{{4, 7}},

{}

};

vector<int> dist = dijkstra(n, adj, 0);

for (int d : dist) cout << d << " "; // Output: 0 7 3 9 16

}

✅ **5. Topological Sorting (Kahn’s Algorithm - BFS)**  
📌 **Algorithm:** **Kahn's Algorithm (BFS) or DFS**  
📌 **Trick:** Use **in-degree array & queue** to find nodes with zero dependencies.  
💻 **Code (Kahn's Algorithm - BFS)**:

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

vector<int> topologicalSort(int n, vector<pair<int, int>>& edges) {

vector<vector<int>> adj(n);

vector<int> inDegree(n, 0), topoSort;

for (auto& edge : edges) {

adj[edge.first].push\_back(edge.second);

inDegree[edge.second]++;

}

queue<int> q;

for (int i = 0; i < n; i++)

if (inDegree[i] == 0) q.push(i);

while (!q.empty()) {

int node = q.front();

q.pop();

topoSort.push\_back(node);

for (int neighbor : adj[node]) {

if (--inDegree[neighbor] == 0) q.push(neighbor);

}

}

return topoSort;

}

int main() {

int n = 6;

vector<pair<int, int>> edges = {{5, 2}, {5, 0}, {4, 0}, {4, 1}, {2, 3}, {3, 1}};

vector<int> topoOrder = topologicalSort(n, edges);

for (int node : topoOrder) cout << node << " "; // Output: 5 4 2 3 1 0

}

**🔥 Key Graph Theory Concepts Covered**

✔ **Graph Traversal (BFS, DFS), Cycle Detection**  
✔ **Shortest Paths (Dijkstra, BFS), Connected Components**  
✔ **Topological Sorting for DAGs**

## ****📂 DSA Sheet Structure (Example for**** Dynamic Programming (DP) ****Section)****

### ****🟢**** Dynamic Programming (DP)

✅ **1. Longest Common Subsequence (LCS)**  
📌 **Algorithm:** **DP with Memoization / Tabulation**  
📌 **Trick:** Use a **2D DP table**, where dp[i][j] represents the LCS of substrings s1[0...i] and s2[0...j].  
💻 **Code (Bottom-Up Approach):**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int longestCommonSubsequence(string s1, string s2) {

int n = s1.size(), m = s2.size();

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

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

for (int j = 1; j <= m; j++) {

if (s1[i - 1] == s2[j - 1]) dp[i][j] = 1 + dp[i - 1][j - 1];

else dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);

}

}

return dp[n][m];

}

int main() {

string s1 = "abcde", s2 = "ace";

cout << longestCommonSubsequence(s1, s2) << endl; // Output: 3

}

✅ **2. 0/1 Knapsack Problem**  
📌 **Algorithm:** **DP with Recursion / Tabulation**  
📌 **Trick:** Use **a DP table where dp[i][j] stores the max value possible with i items and capacity j**.  
💻 **Code (Bottom-Up Approach):**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int knapsack(vector<int>& weights, vector<int>& values, int W) {

int n = weights.size();

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

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

for (int j = 1; j <= W; j++) {

if (weights[i - 1] <= j)

dp[i][j] = max(dp[i - 1][j], values[i - 1] + dp[i - 1][j - weights[i - 1]]);

else

dp[i][j] = dp[i - 1][j];

}

}

return dp[n][W];

}

int main() {

vector<int> weights = {2, 3, 4, 5};

vector<int> values = {3, 4, 5, 6};

int W = 5;

cout << knapsack(weights, values, W) << endl; // Output: 7

}

✅ **3. Coin Change (Fewest Coins to Reach Sum)**  
📌 **Algorithm:** **DP with Unbounded Knapsack Pattern**  
📌 **Trick:** **Use dp[i] to store the minimum number of coins required to reach sum i.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int coinChange(vector<int>& coins, int amount) {

vector<int> dp(amount + 1, INT\_MAX);

dp[0] = 0;

for (int coin : coins) {

for (int j = coin; j <= amount; j++) {

if (dp[j - coin] != INT\_MAX)

dp[j] = min(dp[j], 1 + dp[j - coin]);

}

}

return dp[amount] == INT\_MAX ? -1 : dp[amount];

}

int main() {

vector<int> coins = {1, 2, 5};

int amount = 11;

cout << coinChange(coins, amount) << endl; // Output: 3

}

✅ **4. Longest Increasing Subsequence (LIS)**  
📌 **Algorithm:** **DP + Binary Search (O(n log n))**  
📌 **Trick:** Use a **DP array with Binary Search (Patience Sorting Technique)** for optimal performance.  
💻 **Code (O(n log n) using Binary Search):**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int lengthOfLIS(vector<int>& nums) {

vector<int> lis;

for (int num : nums) {

auto it = lower\_bound(lis.begin(), lis.end(), num);

if (it == lis.end()) lis.push\_back(num);

else \*it = num;

}

return lis.size();

}

int main() {

vector<int> nums = {10, 9, 2, 5, 3, 7, 101, 18};

cout << lengthOfLIS(nums) << endl; // Output: 4

}

✅ **5. Matrix Chain Multiplication (MCM)**  
📌 **Algorithm:** **DP with Recursion & Memoization**  
📌 **Trick:** **Use recursion with dp[i][j] storing min cost of multiplying matrices from i to j**.  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

vector<vector<int>> dp;

int mcm(vector<int>& arr, int i, int j) {

if (i == j) return 0;

if (dp[i][j] != -1) return dp[i][j];

int minCost = INT\_MAX;

for (int k = i; k < j; k++) {

int cost = mcm(arr, i, k) + mcm(arr, k + 1, j) + arr[i - 1] \* arr[k] \* arr[j];

minCost = min(minCost, cost);

}

return dp[i][j] = minCost;

}

int main() {

vector<int> arr = {40, 20, 30, 10, 30};

int n = arr.size();

dp.assign(n, vector<int>(n, -1));

cout << mcm(arr, 1, n - 1) << endl; // Output: 26000

}

**🔥 Key Dynamic Programming Concepts Covered**

✔ **LCS & Subsequence-based DP**  
✔ **Knapsack Variations (0/1, Unbounded, Coin Change)**  
✔ **Optimization Problems (MCM, LIS)**  
✔ **State Transition Formulation & Recursion with Memoization**

## ****📂 DSA Sheet Structure (Example for**** Greedy Algorithms ****Section)****

### ****🟢**** Greedy Algorithms

✅ **1. Activity Selection Problem**  
📌 **Algorithm:** **Greedy Approach (Earliest Finish Time First)**  
📌 **Trick:** **Sort activities by finish time and always select the next non-overlapping activity.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct Activity {

int start, end;

};

bool compare(Activity a, Activity b) {

return a.end < b.end;

}

int maxActivities(vector<Activity>& activities) {

sort(activities.begin(), activities.end(), compare);

int count = 1, lastEnd = activities[0].end;

for (int i = 1; i < activities.size(); i++) {

if (activities[i].start >= lastEnd) {

count++;

lastEnd = activities[i].end;

}

}

return count;

}

int main() {

vector<Activity> activities = {{1, 3}, {2, 5}, {3, 9}, {6, 8}};

cout << maxActivities(activities) << endl; // Output: 2

}

✅ **2. Huffman Coding (Optimal Prefix Codes)**  
📌 **Algorithm:** **Greedy + Priority Queue (Min-Heap)**  
📌 **Trick:** **Use a min-heap to build the optimal prefix tree for variable-length encoding.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct Node {

char ch;

int freq;

Node \*left, \*right;

Node(char c, int f) : ch(c), freq(f), left(NULL), right(NULL) {}

};

struct Compare {

bool operator()(Node\* a, Node\* b) {

return a->freq > b->freq;

}

};

void printCodes(Node\* root, string str) {

if (!root) return;

if (root->ch != '#') cout << root->ch << ": " << str << endl;

printCodes(root->left, str + "0");

printCodes(root->right, str + "1");

}

void huffmanCoding(vector<char>& chars, vector<int>& freqs) {

priority\_queue<Node\*, vector<Node\*>, Compare> minHeap;

for (int i = 0; i < chars.size(); i++)

minHeap.push(new Node(chars[i], freqs[i]));

while (minHeap.size() > 1) {

Node\* left = minHeap.top(); minHeap.pop();

Node\* right = minHeap.top(); minHeap.pop();

Node\* newNode = new Node('#', left->freq + right->freq);

newNode->left = left;

newNode->right = right;

minHeap.push(newNode);

}

printCodes(minHeap.top(), "");

}

int main() {

vector<char> chars = {'a', 'b', 'c', 'd', 'e', 'f'};

vector<int> freqs = {5, 9, 12, 13, 16, 45};

huffmanCoding(chars, freqs);

}

✅ **3. Minimum Coins for Change**  
📌 **Algorithm:** **Greedy (Largest Coin First)**  
📌 **Trick:** **Use the largest possible denomination first to minimize the total number of coins.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int minCoins(vector<int>& coins, int amount) {

sort(coins.rbegin(), coins.rend());

int count = 0;

for (int coin : coins) {

if (amount == 0) break;

count += amount / coin;

amount %= coin;

}

return count;

}

int main() {

vector<int> coins = {1, 5, 10, 25, 100};

int amount = 93;

cout << minCoins(coins, amount) << endl; // Output: 5 (25+25+25+10+5+1+1+1)

}

✅ **4. Fractional Knapsack**  
📌 **Algorithm:** **Greedy (Sort by Value-to-Weight Ratio)**  
📌 **Trick:** **Sort items by value/weight and take as much of the most valuable item as possible.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct Item {

int weight, value;

};

bool compare(Item a, Item b) {

return (double)a.value / a.weight > (double)b.value / b.weight;

}

double fractionalKnapsack(vector<Item>& items, int W) {

sort(items.begin(), items.end(), compare);

double totalValue = 0.0;

for (auto& item : items) {

if (W == 0) break;

int take = min(W, item.weight);

totalValue += take \* ((double)item.value / item.weight);

W -= take;

}

return totalValue;

}

int main() {

vector<Item> items = {{10, 60}, {20, 100}, {30, 120}};

int W = 50;

cout << fractionalKnapsack(items, W) << endl; // Output: 240

}

✅ **5. Job Sequencing Problem**  
📌 **Algorithm:** **Greedy + Sorting (Sort by Profit & Use a Deadline Array)**  
📌 **Trick:** **Sort jobs by profit and try to place them in the latest available slot before the deadline.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct Job {

char id;

int deadline, profit;

};

bool compare(Job a, Job b) {

return a.profit > b.profit;

}

int jobSequencing(vector<Job>& jobs) {

sort(jobs.begin(), jobs.end(), compare);

int maxDeadline = 0;

for (auto& job : jobs) maxDeadline = max(maxDeadline, job.deadline);

vector<int> slots(maxDeadline + 1, -1);

int totalProfit = 0;

for (auto& job : jobs) {

for (int i = job.deadline; i > 0; i--) {

if (slots[i] == -1) {

slots[i] = job.id;

totalProfit += job.profit;

break;

}

}

}

return totalProfit;

}

int main() {

vector<Job> jobs = {{'a', 2, 100}, {'b', 1, 19}, {'c', 2, 27}, {'d', 1, 25}, {'e', 3, 15}};

cout << jobSequencing(jobs) << endl; // Output: 142

}

**🔥 Key Greedy Concepts Covered**

✔ **Activity Selection (Earliest Finish Time First)**  
✔ **Huffman Coding (Prefix Encoding with Min-Heap)**  
✔ **Coin Change (Largest Coin First Strategy)**  
✔ **Knapsack (Fractional Approach with Sorting by Value/Weight)**  
✔ **Job Scheduling (Sort by Profit & Use Deadlines Effectively)**

### 

## ****📂 DSA Sheet Structure (Example for**** Bit Manipulation ****Section)****

### ****🟢**** Bit Manipulation

✅ **1. Find the Only Non-Repeating Element (XOR Trick)**  
📌 **Algorithm:** **Bitwise XOR**  
📌 **Trick:** **XOR cancels out duplicate elements, leaving only the unique one.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int findUnique(vector<int>& nums) {

int unique = 0;

for (int num : nums) {

unique ^= num;

}

return unique;

}

int main() {

vector<int> nums = {4, 3, 2, 4, 2, 3, 7};

cout << findUnique(nums) << endl; // Output: 7

}

✅ **2. Check if a Number is Power of 2**  
📌 **Algorithm:** **Bitwise AND Trick**  
📌 **Trick:** **A power of 2 has exactly one 1 bit, so n & (n - 1) == 0.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

bool isPowerOfTwo(int n) {

return (n > 0) && ((n & (n - 1)) == 0);

}

int main() {

cout << isPowerOfTwo(16) << endl; // Output: 1 (true)

cout << isPowerOfTwo(18) << endl; // Output: 0 (false)

}

✅ **3. Count the Number of 1s in Binary Representation (Brian Kernighan’s Algorithm)**  
📌 **Algorithm:** **Repeated AND with n-1**  
📌 **Trick:** **Each operation removes the last set bit (1) in O(log n).**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int countSetBits(int n) {

int count = 0;

while (n) {

n &= (n - 1);

count++;

}

return count;

}

int main() {

cout << countSetBits(15) << endl; // Output: 4 (Binary: 1111)

}

✅ **4. Find the Two Non-Repeating Elements in an Array**  
📌 **Algorithm:** **Bitwise XOR + Rightmost Set Bit Separation**  
📌 **Trick:** **XOR finds the combined difference, then we split based on the rightmost set bit.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

void findTwoUnique(vector<int>& nums) {

int xor\_all = 0;

for (int num : nums) xor\_all ^= num;

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;

}

cout << num1 << " " << num2 << endl;

}

int main() {

vector<int> nums = {3, 4, 3, 7, 8, 4};

findTwoUnique(nums); // Output: 7 8

}

✅ **5. Find the XOR of All Numbers from 1 to N (XOR Pattern Trick)**  
📌 **Algorithm:** **Mathematical Pattern for XOR Sequence**  
📌 **Trick:** **Use a precomputed pattern to compute in O(1).**  
📌 **Pattern:**

* n % 4 == 0 → result = n
* n % 4 == 1 → result = 1
* n % 4 == 2 → result = n + 1
* n % 4 == 3 → result = 0  
  💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int xorTillN(int n) {

if (n % 4 == 0) return n;

if (n % 4 == 1) return 1;

if (n % 4 == 2) return n + 1;

return 0;

}

int main() {

cout << xorTillN(10) << endl; // Output: 11

}

### ****🔥 Key Bit Manipulation Concepts Covered****

✔ **XOR Properties for Finding Unique Elements**  
✔ **Checking Power of Two Using AND Trick**  
✔ **Brian Kernighan’s Algorithm for Counting Set Bits**  
✔ **Finding Two Unique Numbers Using Rightmost Set Bit**  
✔ **Fast Computation of XOR from 1 to N Using a Pattern**

## ****📂 DSA Sheet Structure (Example for**** Math & Number Theory ****Section)****

### ****🟢**** Math & Number Theory

✅ **1. Check if a Number is Prime**  
📌 **Algorithm:** **Trial Division with Optimization**  
📌 **Trick:** **Check divisibility only up to √n and skip even numbers after 2.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

bool isPrime(int n) {

if (n < 2) return false;

if (n == 2 || n == 3) return true;

if (n % 2 == 0 || n % 3 == 0) return false;

for (int i = 5; i \* i <= n; i += 6) {

if (n % i == 0 || n % (i + 2) == 0) return false;

}

return true;

}

int main() {

cout << isPrime(29) << endl; // Output: 1 (true)

cout << isPrime(30) << endl; // Output: 0 (false)

}

✅ **2. Find All Prime Numbers Up to N (Sieve of Eratosthenes)**  
📌 **Algorithm:** **Sieve of Eratosthenes**  
📌 **Trick:** **Mark all multiples of a prime starting from p^2, reducing time to O(n log log n).**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

vector<bool> sieve(int n) {

vector<bool> isPrime(n + 1, true);

isPrime[0] = isPrime[1] = false;

for (int i = 2; i \* i <= n; i++) {

if (isPrime[i]) {

for (int j = i \* i; j <= n; j += i) {

isPrime[j] = false;

}

}

}

return isPrime;

}

int main() {

int n = 50;

vector<bool> primes = sieve(n);

for (int i = 2; i <= n; i++) {

if (primes[i]) cout << i << " ";

}

cout << endl;

}

✅ **3. Find GCD (Greatest Common Divisor) Using Euclidean Algorithm**  
📌 **Algorithm:** **Euclidean Algorithm**  
📌 **Trick:** **Use gcd(a, b) = gcd(b, a % b) until b == 0 for O(log(min(a, b))) complexity.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int gcd(int a, int b) {

while (b) {

a %= b;

swap(a, b);

}

return a;

}

int main() {

cout << gcd(56, 98) << endl; // Output: 14

}

✅ **4. Compute Modular Exponentiation (Fast Powering)**  
📌 **Algorithm:** **Binary Exponentiation**  
📌 **Trick:** **Use x^y mod p efficiently with O(log y) complexity.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int modExp(int x, int y, int p) {

int res = 1;

x %= p;

while (y > 0) {

if (y & 1) res = (1LL \* res \* x) % p;

x = (1LL \* x \* x) % p;

y >>= 1;

}

return res;

}

int main() {

cout << modExp(2, 10, 1000000007) << endl; // Output: 1024

}

✅ **5. Find Modular Multiplicative Inverse (Using Extended Euclidean Algorithm)**  
📌 **Algorithm:** **Extended Euclidean Algorithm**  
📌 **Trick:** **If a and m are coprime, then x in ax ≡ 1 (mod m) can be found using Extended GCD.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

int gcdExtended(int a, int b, int &x, int &y) {

if (b == 0) {

x = 1, y = 0;

return a;

}

int x1, y1;

int g = gcdExtended(b, a % b, x1, y1);

x = y1;

y = x1 - (a / b) \* y1;

return g;

}

int modInverse(int a, int m) {

int x, y;

int g = gcdExtended(a, m, x, y);

if (g != 1) return -1; // No inverse if gcd(a, m) ≠ 1

return (x % m + m) % m;

}

int main() {

cout << modInverse(3, 7) << endl; // Output: 5 (because 3 \* 5 ≡ 1 (mod 7))

}

### ****🔥 Key Math & Number Theory Concepts Covered****

✔ **Prime Checking (Optimized Trial Division)**  
✔ **Finding All Primes (Sieve of Eratosthenes)**  
✔ **Greatest Common Divisor (GCD) Using Euclidean Algorithm**  
✔ **Modular Exponentiation (Fast Powering in O(log y))**  
✔ **Modular Multiplicative Inverse Using Extended Euclidean Algorithm**

## ****📂 DSA Sheet Structure (Example for**** Trie & Advanced Data Structures****)****

### ****🟢**** Trie & Advanced Data Structures

✅ **1. Implement Trie (Prefix Tree) for Word Insertion & Search**  
📌 **Algorithm:** **Trie Data Structure**  
📌 **Trick:** **Store words character by character in nodes, marking end of words.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct TrieNode {

TrieNode\* children[26];

bool isEndOfWord;

TrieNode() {

isEndOfWord = false;

for (int i = 0; i < 26; i++) children[i] = nullptr;

}

};

class Trie {

public:

TrieNode\* root;

Trie() { root = new TrieNode(); }

void insert(string word) {

TrieNode\* node = root;

for (char c : word) {

if (!node->children[c - 'a'])

node->children[c - 'a'] = new TrieNode();

node = node->children[c - 'a'];

}

node->isEndOfWord = true;

}

bool search(string word) {

TrieNode\* node = root;

for (char c : word) {

if (!node->children[c - 'a']) return false;

node = node->children[c - 'a'];

}

return node->isEndOfWord;

}

};

int main() {

Trie trie;

trie.insert("apple");

cout << trie.search("apple") << endl; // Output: 1 (true)

cout << trie.search("app") << endl; // Output: 0 (false)

}

✅ **2. Find Longest Common Prefix Using Trie**  
📌 **Algorithm:** **Trie with Common Prefix Traversal**  
📌 **Trick:** **Move down the Trie while only one child exists.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

struct TrieNode {

TrieNode\* children[26];

int count;

TrieNode() {

count = 0;

for (int i = 0; i < 26; i++) children[i] = nullptr;

}

};

class Trie {

public:

TrieNode\* root;

Trie() { root = new TrieNode(); }

void insert(string word) {

TrieNode\* node = root;

for (char c : word) {

if (!node->children[c - 'a'])

node->children[c - 'a'] = new TrieNode();

node = node->children[c - 'a'];

node->count++;

}

}

string longestCommonPrefix() {

TrieNode\* node = root;

string prefix = "";

while (node) {

int nextIndex = -1;

int childrenCount = 0;

for (int i = 0; i < 26; i++) {

if (node->children[i]) {

nextIndex = i;

childrenCount++;

}

}

if (childrenCount != 1) break;

prefix += char('a' + nextIndex);

node = node->children[nextIndex];

}

return prefix;

}

};

int main() {

Trie trie;

vector<string> words = {"flower", "flow", "flight"};

for (string word : words) trie.insert(word);

cout << trie.longestCommonPrefix() << endl; // Output: "fl"

}

✅ **3. Implement LRU (Least Recently Used) Cache**  
📌 **Algorithm:** **Doubly Linked List + Hash Map**  
📌 **Trick:** **Use a hash map for O(1) access and a linked list to track usage.**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

class LRUCache {

public:

int capacity;

list<pair<int, int>> lruList;

unordered\_map<int, list<pair<int, int>>::iterator> cache;

LRUCache(int cap) { capacity = cap; }

int get(int key) {

if (cache.find(key) == cache.end()) return -1;

lruList.splice(lruList.begin(), lruList, cache[key]);

return cache[key]->second;

}

void put(int key, int value) {

if (cache.find(key) != cache.end()) {

lruList.splice(lruList.begin(), lruList, cache[key]);

cache[key]->second = value;

return;

}

if (lruList.size() == capacity) {

cache.erase(lruList.back().first);

lruList.pop\_back();

}

lruList.push\_front({key, value});

cache[key] = lruList.begin();

}

};

int main() {

LRUCache lru(2);

lru.put(1, 10);

lru.put(2, 20);

cout << lru.get(1) << endl; // Output: 10

lru.put(3, 30);

cout << lru.get(2) << endl; // Output: -1 (Evicted)

}

✅ **4. Implement Segment Tree for Range Sum Query**  
📌 **Algorithm:** **Segment Tree with Lazy Propagation**  
📌 **Trick:** **Use a tree array for efficient range queries in O(log n).**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

class SegmentTree {

vector<int> tree;

int n;

void build(vector<int>& arr, int node, int start, int end) {

if (start == end) {

tree[node] = arr[start];

} else {

int mid = (start + end) / 2;

build(arr, 2 \* node, start, mid);

build(arr, 2 \* node + 1, mid + 1, end);

tree[node] = tree[2 \* node] + tree[2 \* node + 1];

}

}

int query(int node, int start, int end, int l, int r) {

if (r < start || end < l) return 0;

if (l <= start && end <= r) return tree[node];

int mid = (start + end) / 2;

return query(2 \* node, start, mid, l, r) + query(2 \* node + 1, mid + 1, end, l, r);

}

public:

SegmentTree(vector<int>& arr) {

n = arr.size();

tree.resize(4 \* n);

build(arr, 1, 0, n - 1);

}

int rangeSum(int l, int r) {

return query(1, 0, n - 1, l, r);

}

};

int main() {

vector<int> arr = {1, 2, 3, 4, 5};

SegmentTree segTree(arr);

cout << segTree.rangeSum(1, 3) << endl; // Output: 9

}

✅ **5. Implement Fenwick Tree (Binary Indexed Tree) for Prefix Sum**  
📌 **Algorithm:** **Fenwick Tree**  
📌 **Trick:** **Efficient prefix sum updates in O(log n).**  
💻 **Code:**

cpp

CopyEdit

#include <bits/stdc++.h>

using namespace std;

class FenwickTree {

vector<int> BIT;

int n;

public:

FenwickTree(int size) {

n = size;

BIT.assign(n + 1, 0);

}

void update(int index, int val) {

while (index <= n) {

BIT[index] += val;

index += index & -index;

}

}

int query(int index) {

int sum = 0;

while (index > 0) {

sum += BIT[index];

index -= index & -index;

}

return sum;

}

};

int main() {

FenwickTree fen(5);

fen.update(1, 5);

fen.update(3, 7);

cout << fen.query(3) << endl; // Output: 12

}

### ****🔥 Key Trie & Advanced Data Structure Concepts Covered****

✔ **Trie for String Operations**  
✔ **LRU Cache (Hash Map + Doubly Linked List)**  
✔ **Segment Tree for Range Queries**  
✔ **Fenwick Tree (Binary Indexed Tree)**

**🎉 Congratulations on Completing This DSA Guide! 🎉**

If you've reached this point, you’ve taken a significant step toward mastering **Data Structures and Algorithms**. Completing this entire PDF from start to finish is a testament to your dedication and hard work. Keep practicing, keep learning, and keep pushing your limits—your efforts will pay off in coding competitions, interviews, and real-world problem-solving! 🚀

If this guide helped you, **I’d love to hear your feedback!** Feel free to **connect with me on Instagram (@Syntax\_Error)** for more **DSA problems, tricks, and full-stack development content.** Let's keep growing together! 💡🔥

### ****⚠ Copyright Notice ⚠****

This PDF is **created and owned by Abhishek Rathor (@Syntax\_Error)**. Unauthorized copying, reproduction, or distribution of this content **without proper credit** is strictly prohibited. If anyone attempts to misuse this content or claim it as their own, a **copyright claim will be filed** against them.

If you wish to share or reference this material, please ensure **proper attribution to the original creator (@Syntax\_Error)**.

Thank you for respecting the effort behind this work! 🙌