1. #include <stdio.h>

// Function to convert Celsius to Kelvin and Fahrenheit

void convertTemperature(double celsius, double \*kelvin, double \*fahrenheit) {

\*kelvin = celsius + 273.15;

\*fahrenheit = celsius \* 1.8 + 32.0;

}

int main() {

double celsius = 36.50;

double kelvin, fahrenheit;

// Call function to convert temperature

convertTemperature(celsius, &kelvin, &fahrenheit);

// Print the converted temperatures

printf("Celsius: %.5f\n", celsius);

printf("Kelvin: %.5f\n", kelvin);

printf("Fahrenheit: %.5f\n", fahrenheit);

return 0;

}

2. #include <stdio.h>

// Function to calculate the greatest common divisor (GCD) of two numbers

int gcd(int a, int b) {

if (b == 0)

return a;

return gcd(b, a % b);

}

// Function to calculate the least common multiple (LCM) of two numbers

int lcm(int a, int b) {

return (a \* b) / gcd(a, b);

}

// Function to count the number of subarrays with LCM equal to k

int countSubarraysWithLCM(int nums[], int n, int k) {

int count = 0;

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

int current\_lcm = 1;

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

current\_lcm = lcm(current\_lcm, nums[j]);

if (current\_lcm == k)

count++;

}

}

return count;

}

int main() {

int nums1[] = {3, 6, 2, 7, 1};

int n1 = sizeof(nums1) / sizeof(nums1[0]);

int k1 = 6;

int nums2[] = {3};

int n2 = sizeof(nums2) / sizeof(nums2[0]);

int k2 = 2;

printf("Example 1: Number of subarrays with LCM equal to %d: %d\n", k1, countSubarraysWithLCM(nums1, n1, k1));

printf("Example 2: Number of subarrays with LCM equal to %d: %d\n", k2, countSubarraysWithLCM(nums2, n2, k2));

return 0;

}

3. #include <stdio.h>

#include <stdlib.h>

// Definition for a binary tree node

struct TreeNode {

int val;

struct TreeNode \*left;

struct TreeNode \*right;

};

// Function to create a new tree node

struct TreeNode\* newNode(int val) {

struct TreeNode\* node = (struct TreeNode\*)malloc(sizeof(struct TreeNode));

node->val = val;

node->left = NULL;

node->right = NULL;

return node;

}

// Function to find the minimum of two integers

int min(int a, int b) {

return (a < b) ? a : b;

}

// Function to perform inorder traversal and count the number of misplacements

void inorder(struct TreeNode\* root, int level, int \*expected, int \*misplacements) {

if (root == NULL) return;

inorder(root->left, level + 1, expected, misplacements);

// If the current value is not the expected one, count it as a misplacement

if (root->val != \*expected) {

(\*misplacements)++;

}

// Update the expected value for the next node at the same level

(\*expected)++;

inorder(root->right, level + 1, expected, misplacements);

}

// Function to calculate the minimum number of operations to sort the binary tree by level

int minOperations(struct TreeNode\* root) {

int expected = 1; // Expected value at the first level is 1

int misplacements = 0; // Number of misplacements

inorder(root, 1, &expected, &misplacements);

return misplacements;

}

int main() {

// Create the binary tree

struct TreeNode\* root = newNode(1);

root->left = newNode(4);

root->right = newNode(3);

root->left->left = newNode(7);

root->left->right = newNode(6);

root->right->left = newNode(8);

root->right->right = newNode(5);

root->left->left->left = newNode(9);

root->left->right->left = newNode(10);

// Calculate the minimum number of operations

int minOps = minOperations(root);

printf("Minimum number of operations: %d\n", minOps);

return 0;

}

4. #include <stdio.h>

#include <string.h>

// Function to check if a substring is a palindrome

int isPalindrome(char \*str, int start, int end) {

while (start < end) {

if (str[start] != str[end])

return 0; // Not a palindrome

start++;

end--;

}

return 1; // Palindrome

}

// Function to find the maximum number of non-overlapping palindrome substrings

int maxNumPalindromes(char \*s, int k) {

int n = strlen(s);

int count = 0;

// Check all substrings of length at least k

for (int len = k; len <= n; len++) {

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

int j = i + len - 1; // Ending index of substring

if (isPalindrome(s, i, j))

count++; // Increment count if substring is a palindrome

}

}

return count;

}

int main() {

char s1[] = "abaccdbbd";

int k1 = 3;

printf("Example 1: Maximum number of non-overlapping palindrome substrings: %d\n", maxNumPalindromes(s1, k1));

char s2[] = "adbcda";

int k2 = 2;

printf("Example 2: Maximum number of non-overlapping palindrome substrings: %d\n", maxNumPalindromes(s2, k2));

return 0;

}

5. #include <stdio.h>

#include <stdlib.h>

#include <limits.h>

// Structure to represent a road between cities

struct Road {

int city1;

int city2;

int cost;

};

// Function to find the minimum of two integers

int min(int a, int b) {

return (a < b) ? a : b;

}

// Function to find the minimum total cost to buy an apple if starting at each city

int\* minCostToBuyApples(int n, int roads[][3], int roadsSize, int\* roadsColSize, int\* appleCost, int k, int\* returnSize) {

// Initialize result array with maximum possible values

int\* result = (int\*)malloc(n \* sizeof(int));

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

result[i] = INT\_MAX;

}

// Initialize an adjacency matrix to represent the roads between cities

int\*\* adjacency = (int\*\*)malloc(n \* sizeof(int\*));

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

adjacency[i] = (int\*)malloc(n \* sizeof(int));

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

adjacency[i][j] = INT\_MAX;

}

}

// Populate the adjacency matrix with road costs

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

int city1 = roads[i][0] - 1;

int city2 = roads[i][1] - 1;

int cost = roads[i][2];

adjacency[city1][city2] = cost;

adjacency[city2][city1] = cost;

}

// Perform Dijkstra's algorithm for each starting city

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

// Initialize variables

int\* dist = (int\*)malloc(n \* sizeof(int));

int\* visited = (int\*)malloc(n \* sizeof(int));

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

dist[i] = INT\_MAX;

visited[i] = 0;

}

dist[start] = 0;

// Find shortest path from start city to each other city

for (int count = 0; count < n - 1; count++) {

int minDist = INT\_MAX, minIndex;

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

if (visited[v] == 0 && dist[v] <= minDist) {

minDist = dist[v];

minIndex = v;

}

}

visited[minIndex] = 1;

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

if (!visited[v] && adjacency[minIndex][v] != INT\_MAX && dist[minIndex] != INT\_MAX &&

dist[minIndex] + adjacency[minIndex][v] < dist[v]) {

dist[v] = dist[minIndex] + adjacency[minIndex][v];

}

}

}

// Calculate minimum cost to buy an apple starting from current city

int minCost = INT\_MAX;

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

if (i != start) {

int totalCost = dist[i] \* k + appleCost[i];

minCost = min(minCost, totalCost);

}

}

result[start] = minCost;

// Free memory

free(dist);

free(visited);

}

// Free memory

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

free(adjacency[i]);

}

free(adjacency);

\*returnSize = n;

return result;

}

int main() {

int n = 4;

int roads[][3] = {{1, 2, 4}, {2, 3, 2}, {2, 4, 5}, {3, 4, 1}, {1, 3, 4}};

int roadsSize = sizeof(roads) / sizeof(roads[0]);

int roadsColSize = sizeof(roads[0]) / sizeof(roads[0][0]);

int appleCost[] = {56, 42, 102, 301};

int k = 2;

int returnSize;

int\* result = minCostToBuyApples(n, roads, roadsSize, &roadsColSize, appleCost, k, &returnSize);

printf("Output: [");

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

printf("%d", result[i]);

if (i < returnSize - 1)

printf(", ");

}

printf("]\n");

// Free memory

free(result);

return 0;

}

6. #include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_ORDERS 1000

#define MAX\_CUSTOMERS 100

struct Order {

int order\_id;

int customer\_id;

char order\_date[11]; // Assuming date format YYYY-MM-DD

int price;

};

int main() {

struct Order orders[MAX\_ORDERS];

int num\_orders;

int customer\_yearly\_purchases[MAX\_CUSTOMERS][10000] = {0}; // Assuming we're working with years between 1000 and 9999

// Sample data initialization

num\_orders = 9;

orders[0] = (struct Order){1, 1, "2019-07-01", 1100};

orders[1] = (struct Order){2, 1, "2019-11-01", 1200};

orders[2] = (struct Order){3, 1, "2020-05-26", 3000};

orders[3] = (struct Order){4, 1, "2021-08-31", 3100};

orders[4] = (struct Order){5, 1, "2022-12-07", 4700};

orders[5] = (struct Order){6, 2, "2015-01-01", 700};

orders[6] = (struct Order){7, 2, "2017-11-07", 1000};

orders[7] = (struct Order){8, 3, "2017-01-01", 900};

orders[8] = (struct Order){9, 3, "2018-11-07", 900};

// Calculate total purchases per customer per year

for (int i = 0; i < num\_orders; i++) {

int customer\_id = orders[i].customer\_id;

int year = atoi(strtok(orders[i].order\_date, "-"));

customer\_yearly\_purchases[customer\_id][year] += orders[i].price;

}

// Check if total purchases are strictly increasing yearly for each customer

printf("Customers with strictly increasing purchases:\n");

for (int i = 1; i < MAX\_CUSTOMERS; i++) {

int first\_year = -1;

int last\_year = -1;

int flag = 1; // Flag to indicate if purchases are strictly increasing yearly

for (int year = 1000; year <= 9999; year++) {

if (customer\_yearly\_purchases[i][year] > 0) {

if (first\_year == -1) {

first\_year = year;

}

last\_year = year;

}

}

if (first\_year != -1) {

for (int year = first\_year; year <= last\_year; year++) {

if (year > first\_year && customer\_yearly\_purchases[i][year] <= customer\_yearly\_purchases[i][year - 1]) {

flag = 0; // Purchases are not strictly increasing yearly

break;

}

}

if (flag) {

printf("%d\n", i);

}

}

}

return 0;

}

7. #include <stdio.h>

int numUnequalTriplets(int\* nums, int numsSize) {

int count = 0;

for (int i = 0; i < numsSize - 2; i++) {

for (int j = i + 1; j < numsSize - 1; j++) {

for (int k = j + 1; k < numsSize; k++) {

if (nums[i] != nums[j] && nums[i] != nums[k] && nums[j] != nums[k]) {

count++;

}

}

}

}

return count;

}

int main() {

// Example usage

int nums1[] = {4, 4, 2, 4, 3};

int size1 = sizeof(nums1) / sizeof(nums1[0]);

printf("Output for nums1: %d\n", numUnequalTriplets(nums1, size1));

int nums2[] = {1, 1, 1, 1, 1};

int size2 = sizeof(nums2) / sizeof(nums2[0]);

printf("Output for nums2: %d\n", numUnequalTriplets(nums2, size2));

return 0;

}

8. #include <stdio.h>

#include <stdlib.h>

struct TreeNode {

int val;

struct TreeNode \*left;

struct TreeNode \*right;

};

struct TreeNode\* newNode(int val) {

struct TreeNode\* node = (struct TreeNode\*)malloc(sizeof(struct TreeNode));

node->val = val;

node->left = NULL;

node->right = NULL;

return node;

}

void closestNodesQueries(struct TreeNode\* root, int\* queries, int queriesSize, int\*\* answer, int\* returnSize) {

\*returnSize = queriesSize;

\*answer = (int\*)malloc(sizeof(int) \* (\*returnSize) \* 2);

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

int mini = -1;

int maxi = -1;

struct TreeNode\* curr = root;

while (curr != NULL) {

if (curr->val <= queries[i]) {

mini = curr->val;

curr = curr->right;

} else {

maxi = curr->val;

curr = curr->left;

}

}

(\*answer)[2 \* i] = mini;

(\*answer)[2 \* i + 1] = maxi;

}

}

int main() {

struct TreeNode\* root = newNode(6);

root->left = newNode(2);

root->right = newNode(13);

root->left->left = newNode(1);

root->left->right = newNode(4);

root->right->left = newNode(9);

root->right->right = newNode(15);

root->right->right->left = newNode(14);

int queries[] = {2, 5, 16};

int queriesSize = sizeof(queries) / sizeof(queries[0]);

int\* answer;

int returnSize;

closestNodesQueries(root, queries, queriesSize, &answer, &returnSize);

printf("Output:\n");

printf("[");

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

printf("[%d, %d]", answer[2 \* i], answer[2 \* i + 1]);

if (i < returnSize - 1)

printf(",");

}

printf("]\n");

free(answer);

return 0;

}

9. def min\_fuel\_cost(roads, seats):

"""

Calculates the minimum fuel cost for representatives to reach the capital city.

Args:

roads: A 2D list representing edges (roads) as [city1, city2].

seats: The number of seats in each car.

Returns:

The minimum number of liters of fuel required.

"""

n = len(roads) + 1 # Number of cities (including capital)

# Build an adjacency list representation of the tree

graph = [[] for \_ in range(n)]

for city1, city2 in roads:

graph[city1].append(city2)

graph[city2].append(city1) # Undirected edges

# Pre-calculate the number of descendants (including itself) for each city

# using a modified depth-first search (DFS)

descendants = [0] \* n

def dfs\_descendants(node, visited):

if visited[node]:

return 0

visited[node] = True

descendants[node] = 1

for child in graph[node]:

descendants[node] += dfs\_descendants(child, visited)

return descendants[node]

visited = [False] \* n

dfs\_descendants(0, visited) # Start from the capital city (city 0)

# Function to calculate minimum fuel cost from a node

def dfs\_min\_cost(node, visited):

if visited[node]:

return 0

visited[node] = True

min\_cost = descendants[node] # Worst case: all descendants use this car

for child in graph[node]:

child\_cost = dfs\_min\_cost(child, visited)

# Consider minimum cost based on seats in car

min\_cost = min(min\_cost, (child\_cost + 1) + (descendants[node] - child\_cost) // seats)

visited[node] = False # Unmark current node for backtracking

return min\_cost

# Calculate minimum fuel cost starting from the capital city

return dfs\_min\_cost(0, visited)

# Example usage

roads = [[0, 1], [0, 2], [0, 3]]

seats = 5

min\_cost = min\_fuel\_cost(roads, seats)

print(min\_cost) # Output: 3

10. #include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MOD 1000000007

#define MAX\_LEN 101

int isPrime(char c) {

return (c == '2' || c == '3' || c == '5' || c == '7');

}

int countBeautifulPartitions(char \*s, int k, int minLength) {

int n = strlen(s);

int dp[MAX\_LEN][MAX\_LEN][MAX\_LEN] = {0};

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

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

for (int len = minLength; len <= n; len++) {

if (i == 0 && j == 1 && len == 1) {

dp[i][j][len] = isPrime(s[i]);

} else if (i >= len - 1 && j > 1) {

dp[i][j][len] = (dp[i - 1][j][len] + dp[i - len][j - 1][len]) % MOD;

}

}

}

}

return dp[n - 1][k][minLength];

}

int main() {

char s1[] = "23542185131";

int k1 = 3, minLength1 = 2;

printf("Output for s = \"%s\", k = %d, minLength = %d: %d\n", s1, k1, minLength1, countBeautifulPartitions(s1, k1, minLength1));

char s2[] = "23542185131";

int k2 = 3, minLength2 = 3;

printf("Output for s = \"%s\", k = %d, minLength = %d: %d\n", s2, k2, minLength2, countBeautifulPartitions(s2, k2, minLength2));

char s3[] = "3312958";

int k3 = 3, minLength3 = 1;

printf("Output for s = \"%s\", k = %d, minLength = %d: %d\n", s3, k3, minLength3, countBeautifulPartitions(s3, k3, minLength3));

return 0;

}