Design & Analysis of Algorithm LAB

(CIC-359)

Faculty Name : Dr. Moolchand Sharma Name : Amit Singhal

Enrollment No. : 11614802722

Semester : V

Group : C6



Maharaja Agrasen Institute of Technology, PSP Area,

Sector – 22, Rohini, New Delhi – 110085



**MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY**

**VISION**

“To attain global excellence through ****education, innovation, research,**** and ****work ethics****with the commitment to ****serve humanity.”****

**MISSION**

****M1.****  To promote diversification by adopting advancement in science, technology, management,

and allied discipline through continuous learning

****M2.****  To foster ****moral values**** in students and equip them for developing sustainable solutions to

serve both national and global needs in society and industry.

****M3.****  To ****digitize educational resources and process****for enhanced teaching and effective

learning.

****M4.****  To cultivate an ****environment**** supporting ****incubation, product development, technology****

****transfer, capacity building and entrepreneurship****.

****M5.****  To encourage ****faculty-student networking with alumni, industry, institutions,****and

other ****stakeholders**** for collective engagement.



**Department of Computer Science and Engineering**

**MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY**

**VISION**

**"To attain global excellence through education, innovation, research, and work ethics in the field of Computer Science and engineering with the commitment to serve humanity."**

**MISSION**

****M1.**** To lead in the advancement of computer science and engineering through internationally recognized research and education.  
****M2.****   To prepare students for full and ethical participation in a diverse society and encourage lifelong learning.  
****M3.**** To foster development of problem solving and communication skills as an integral component of the profession.  
****M4.**** To impart knowledge, skills and cultivate an environment supporting incubation, product development, technology transfer, capacity building and entrepreneurship in the field of computer science and engineering.  
****M5.**** To encourage faculty, student’s networking with alumni, industry, institutions, and other stakeholders for collective engagement.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S.No.** | **Experiment** | **Date of** | **M** | **A** | **R** | **K** | **S** | **Total** | **Signature** |
|  | **Name** | **Perf.** | **R1** | **R2** | **R3** | **R4** | **R5** | **Marks** |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

**Index**

Paper Code: **CIC - 359** Name : **Amit Singhal**

Paper: **Design & Analysis of Algorithm Lab** Enrollment No. : **11614802722**

Semester/Group : **5C6**

Branch : **CSE-I**

LIST OF EXPERIMENTS (As prescribed by G.G.S.I.P.U)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S.No.** | **Experiment** | **Date of** | **M** | **A** | **R** | **K** | **S** | **Total** | **Signature** |
|  | **Name** | **Perf.** | **R1** | **R2** | **R3** | **R4** | **R5** | **Marks** |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

LIST OF EXPERIMENTS (Beyond the syllabus prescribed by G.G.S.I.P.U)

LIST OF EXPERIMENTS (Beyond the syllabus prescribed by G.G.S.I.P.U)

**Lab Exercise - 1**

* AIM ::

WAP in C++ to implement Bubble, Merge, Quick & Insertion Sort and also evaluate time in each.

**1. Bubble Sort**

Source\_Code ::

### **#include <ctime>**

### **#include <iostream>**

### **#include <vector>**

### **using namespace std;**

### **void bubbleSort(vector<int>& arr)**

### **{**

### **int n = arr.size();**

### **for (int i = 0; i < n - 1; i++) {**

### **for (int j = 0; j < n - i - 1; j++) {**

### **if (arr[j] > arr[j + 1]) {**

### **swap(arr[j], arr[j + 1]);**

### **}**

### **}**

### **}**

### **}**

### **int main()**

### **{**

### **cout << "5C6 - Amit Singhal (11614802722)" << endl;**

### **vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };**

### **cout << "Unsorted Array: ";**

### **for (int num : arr) {**

### **cout << num << " ";**

### **}**

### **cout << endl;**

### **clock\_t start = clock();**

### **bubbleSort(arr);**

### **clock\_t end = clock();**

### **cout << "Bubble Sort:" << endl;**

### **cout << "Sorted Array: ";**

### **for (int num : arr) {**

### **cout << num << " ";**

### **}**

### **cout << endl;**

### **double time\_taken\_ms = double(end - start) \* 1000.0**

### **/ CLOCKS\_PER\_SEC; // Convert to milliseconds**

### **cout << "Time taken: " << time\_taken\_ms << " milliseconds" << endl;**

### **return 0;**

### **}**

### 

Output ::

**2. Merge Sort**

Source\_Code ::

### **#include <ctime>**

### **#include <iostream>**

### **#include <vector>**

### **using namespace std;**

### **void merge(vector<int>& arr, int l, int m, int r)**

### **{**

### **int n1 = m - l + 1;**

### **int n2 = r - m;**

### **vector<int> L(n1), R(n2);**

### **for (int i = 0; i < n1; i++)**

### **L[i] = arr[l + i];**

### **for (int i = 0; i < n2; i++)**

### **R[i] = arr[m + 1 + i];**

### **int i = 0, j = 0, k = l;**

### **while (i < n1 && j < n2) {**

### **if (L[i] <= R[j]) {**

### **arr[k] = L[i];**

### **i++;**

### **} else {**

### **arr[k] = R[j];**

### **j++;**

### **}**

### **k++;**

### **}**

### **while (i < n1) {**

### **arr[k] = L[i];**

### **i++;**

### **k++;**

### **}**

### **while (j < n2) {**

### **arr[k] = R[j];**

### **j++;**

### **k++;**

### **}**

### **}**

### **void mergeSort(vector<int>& arr, int l, int r)**

### **{**

### **if (l < r) {**

### **int m = l + (r - l) / 2;**

### **mergeSort(arr, l, m);**

### **mergeSort(arr, m + 1, r);**

### **merge(arr, l, m, r);**

### **}**

### **}**

### **int main()**

### **{**

### **cout << "5C6 - Amit Singhal (11614802722)" << endl;**

### **vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };**

### **cout << "Unsorted Array: ";**

### **for (int num : arr) {**

### **cout << num << " ";**

### **}**

### **cout << endl;**

### **clock\_t start = clock();**

### **mergeSort(arr, 0, arr.size() - 1);**

### **clock\_t end = clock();**

### **cout << "Merge Sort:" << endl;**

### **cout << "Sorted Array: ";**

### **for (int num : arr) {**

### **cout << num << " ";**

### **}**

### **cout << endl;**

### **double time\_taken\_ms = double(end - start) \* 1000.0**

### **/ CLOCKS\_PER\_SEC; // Convert to milliseconds**

### **cout << "Time taken: " << time\_taken\_ms << " milliseconds" << endl;**

### **return 0;**

### **}**

Output ::

### 

**3. Quick Sort**

Source\_Code ::

### **#include <ctime>**

### **#include <iostream>**

### **#include <vector>**

### **using namespace std;**

### **int partition(vector<int>& arr, int low, int high)**

### **{**

### **int pivot = arr[high];**

### **int i = (low - 1);**

### **for (int j = low; j <= high - 1; j++) {**

### **if (arr[j] < pivot) {**

### **i++;**

### **swap(arr[i], arr[j]);**

### **}**

### **}**

### **swap(arr[i + 1], arr[high]);**

### **return (i + 1);**

### **}**

### **void quickSort(vector<int>& arr, int low, int high)**

### **{**

### **if (low < high) {**

### **int pi = partition(arr, low, high);**

### **quickSort(arr, low, pi - 1);**

### **quickSort(arr, pi + 1, high);**

### **}**

### **}**

### **int main()**

### **{**

### **cout << "5C6 - Amit Singhal (11614802722)" << endl;**

### **vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };**

### **cout << "Unsorted Array: ";**

### **for (int num : arr) {**

### **cout << num << " ";**

### **}**

### **cout << endl;**

### **clock\_t start = clock();**

### **quickSort(arr, 0, arr.size() - 1);**

### **clock\_t end = clock();**

### **cout << "Quick Sort:" << endl;**

### **cout << "Sorted Array: ";**

### **for (int num : arr) {**

### **cout << num << " ";**

### **}**

### **cout << endl;**

### **double time\_taken\_ms = double(end - start) \* 1000.0**

### **/ CLOCKS\_PER\_SEC; // Convert to milliseconds**

### **cout << "Time taken: " << time\_taken\_ms << " milliseconds" << endl;**

### **return 0;**

### **}**

### 

Output ::

**4. Insertion Sort**

Source\_Code ::

### **#include <ctime>**

### **#include <iostream>**

### **#include <vector>**

### **using namespace std;**

### **void insertionSort(vector<int>& arr)**

### **{**

### **int n = arr.size();**

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

### **int key = arr[i];**

### **int j = i - 1;**

### **while (j >= 0 && arr[j] > key) {**

### **arr[j + 1] = arr[j];**

### **j = j - 1;**

### **}**

### **arr[j + 1] = key;**

### **}**

### **}**

### **int main()**

### **{**

### **cout << "5C6 - Amit Singhal (11614802722)" << endl;**

### **vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };**

### **cout << "Unsorted Array: ";**

### **for (int num : arr) {**

### **cout << num << " ";**

### **}**

### **cout << endl;**

### **clock\_t start = clock();**

### **insertionSort(arr);**

### **clock\_t end = clock();**

### **cout << "Insertion Sort:" << endl;**

### **cout << "Sorted Array: ";**

### **for (int num : arr) {**

### **cout << num << " ";**

### **}**

### **cout << endl;**

### **double time\_taken\_ms = double(end - start) \* 1000.0**

### **/ CLOCKS\_PER\_SEC; // Convert to milliseconds**

### **cout << "Time taken: " << time\_taken\_ms << " milliseconds" << endl;**

### **return 0;**

### **}**

### 

Output ::

Output ::

**Lab Exercise - 2**

* AIM ::

WAP in C++ to implement Linear & Binary Search and also evaluate time in each.

**1. Linear Search**

### **#include <ctime>**

Source\_Code ::

### **#include <iostream>**

### **#include <vector>**

### **using namespace std;**

### **int linearSearch(const vector<int>& arr, int x)**

### **{**

### **for (int i = 0; i < arr.size(); i++) {**

### **if (arr[i] == x) {**

### **return i;**

### **}**

### **}**

### **return -1; // Element not found**

### **}**

### **int main()**

### **{**

### **cout << "5C6 - Amit Singhal (11614802722)" << endl;**

### **vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };**

### **cout << "Array: ";**

### **for (int num : arr) {**

### **cout << num << " ";**

### **}**

### **cout << endl;**

### **int x;**

### **cout << "Enter the element to search: ";**

### **cin >> x;**

### **clock\_t start = clock();**

### **int index = linearSearch(arr, x);**

### **clock\_t end = clock();**

### **if (index != -1) {**

### **cout << "Element found at index: " << index << endl;**

### **} else {**

### **cout << "Element not found" << endl;**

### **}**

### **double time\_taken\_ms = double(end - start) \* 1000.0**

### **/ CLOCKS\_PER\_SEC; // Convert to milliseconds**

### **cout << "Time taken: " << time\_taken\_ms << " milliseconds" << endl;**

### **return 0;**

### **}**

Output ::

### 

**2. Binary Search**

### **#include <algorithm>**

Source\_Code ::

### **#include <ctime>**

### **#include <iostream>**

### **#include <vector>**

### **using namespace std;**

### **int binarySearch(const vector<int>& arr, int x)**

### **{**

### **int left = 0, right = arr.size() - 1;**

### **while (left <= right) {**

### **int mid = left + (right - left) / 2;**

### **if (arr[mid] == x) {**

### **return mid;**

### **}**

### **if (arr[mid] < x) {**

### **left = mid + 1;**

### **} else {**

### **right = mid - 1;**

### **}**

### **}**

### **return -1; // Element not found**

### **}**

### **int main()**

### **{**

### **cout << "5C6 - Amit Singhal (11614802722)" << endl;**

### **vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };**

### **sort(arr.begin(), arr.end()); // Binary search requires a sorted array**

### **cout << "Array: ";**

### **for (int num : arr) {**

### **cout << num << " ";**

### **}**

### **cout << endl;**

### **int x;**

### **cout << "Enter the element to search: ";**

### **cin >> x;**

### **clock\_t start = clock();**

### **int index = binarySearch(arr, x);**

### **clock\_t end = clock();**

### **if (index != -1) {**

### **cout << "Element found at index: " << index << endl;**

### **} else {**

### **cout << "Element not found" << endl;**

### **}**

### **double time\_taken\_ms = double(end - start) \* 1000.0**

### **/ CLOCKS\_PER\_SEC; // Convert to milliseconds**

### **cout << "Time taken: " << time\_taken\_ms << " milliseconds" << endl;**

### **return 0;**

### **}**

Output ::

### 

**Lab Exercise - 3**

* AIM ::

WAP in C++ to implement Huffman Coding & also evaluate its time complexity.

#include <ctime>

Source\_Code ::

#include <iomanip>

#include <iostream>

#include <queue>

#include <unordered\_map>

#include <vector>

using namespace std;

// Node of Huffman Tree

struct Node {

char ch;

int freq;

Node \*left, \*right;

Node(char ch, int freq, Node\* left = nullptr, Node\* right = nullptr)

{

this->ch = ch;

this->freq = freq;

this->left = left;

this->right = right;

}

};

// Comparison function for priority queue

struct compare {

bool operator()(Node\* left, Node\* right)

{

return left->freq > right->freq;

}

};

// Function to build the Huffman Tree

Node\* buildHuffmanTree(const unordered\_map<char, int>& freq)

{

priority\_queue<Node\*, vector<Node\*>, compare> pq;

for (auto pair : freq) {

pq.push(new Node(pair.first, pair.second));

}

while (pq.size() != 1) {

Node\* left = pq.top();

pq.pop();

Node\* right = pq.top();

pq.pop();

int sum = left->freq + right->freq;

pq.push(new Node('\0', sum, left, right));

}

return pq.top();

}

// Function to encode the input string

void encode(

Node\* root, const string& str, unordered\_map<char, string>& huffmanCode)

{

if (root == nullptr)

return;

if (!root->left && !root->right) {

huffmanCode[root->ch] = str;

}

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

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

}

// Function to decode the encoded string

string decode(Node\* root, const string& str)

{

string result = "";

Node\* curr = root;

for (char bit : str) {

if (bit == '0') {

curr = curr->left;

} else {

curr = curr->right;

}

if (!curr->left && !curr->right) {

result += curr->ch;

curr = root;

}

}

return result;

}

int main()

{

cout << "\n5C6 - Amit Singhal (11614802722)" << endl;

string text;

cout << "\nEnter the text to encode: ";

getline(cin, text);

unordered\_map<char, int> freq;

for (char ch : text) {

freq[ch]++;

}

clock\_t start = clock();

Node\* root = buildHuffmanTree(freq);

clock\_t end = clock();

double time\_taken\_build\_tree

= double(end - start) \* 1000.0 / CLOCKS\_PER\_SEC;

unordered\_map<char, string> huffmanCode;

start = clock();

encode(root, "", huffmanCode);

end = clock();

double time\_taken\_encoding = double(end - start) \* 1000.0 / CLOCKS\_PER\_SEC;

cout << "\nCharacter Encoding Table:" << endl;

cout << "------------------------------" << endl;

cout << setw(10) << "Character" << setw(20) << "Huffman Code" << endl;

cout << "------------------------------" << endl;

for (auto pair : huffmanCode) {

cout << setw(10) << pair.first << setw(20) << pair.second << endl;

}

cout << "------------------------------" << endl;

cout << "Time taken to build Huffman Tree: " << time\_taken\_build\_tree

<< " milliseconds" << endl;

string encodedString = "";

for (char ch : text) {

encodedString += huffmanCode[ch];

}

cout << "\nEncoded String: " << encodedString << endl;

cout << "Time taken for encoding: " << time\_taken\_encoding

<< " milliseconds" << endl;

start = clock();

string decodedString = decode(root, encodedString);

end = clock();

double time\_taken\_decoding = double(end - start) \* 1000.0 / CLOCKS\_PER\_SEC;

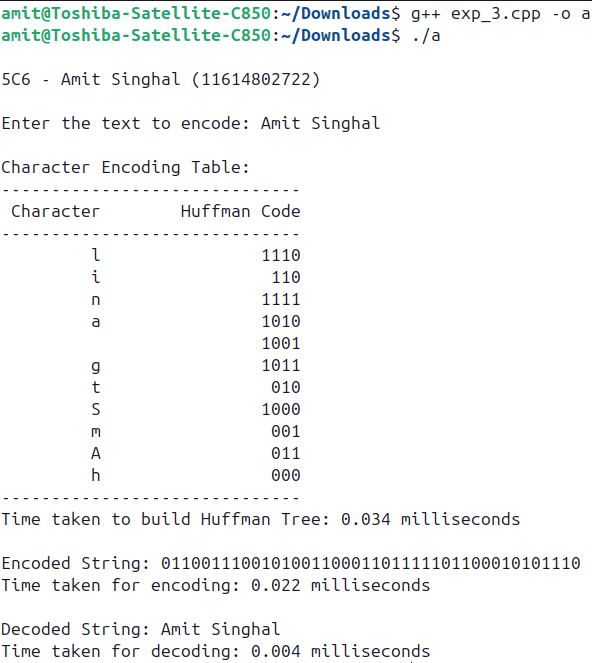
cout << "\nDecoded String: " << decodedString << endl;

cout << "Time taken for decoding: " << time\_taken\_decoding

<< " milliseconds" << endl;

return 0;

}



Output ::

**Lab Exercise - 4**

* AIM ::

WAP in C++ to find Minimum Spanning Tree for a Graph & also evaluate its time complexity.

Source\_Code ::

#include <chrono>

#include <climits>

#include <iomanip>

#include <iostream>

#include <vector>

using namespace std;

using namespace std::chrono;

struct Edge {

int src, dest, weight;

};

// Function to display the graph in a table format

void displayGraph(int V, const vector<Edge>& edges)

{

cout << "Original Graph:\n";

cout << setw(10) << left << "Edges" << setw(10) << left << "Weights"

<< endl;

cout << "-----------------" << endl;

for (const auto& edge : edges) {

cout << setw(1) << edge.src << " - " << setw(8) << edge.dest << setw(10)

<< edge.weight << endl;

}

}

// Function to convert edge list to adjacency matrix

vector<vector<int>> toAdjacencyMatrix(int V, const vector<Edge>& edges)

{

vector<vector<int>> adjMatrix(V, vector<int>(V, 0));

for (const auto& edge : edges) {

adjMatrix[edge.src][edge.dest] = edge.weight;

adjMatrix[edge.dest][edge.src]

= edge.weight; // Since the graph is undirected

}

return adjMatrix;

}

// Function to find the vertex with the minimum key value

int minKey(const vector<int>& key, const vector<bool>& inMST)

{

int min = INT\_MAX, min\_index;

for (int v = 0; v < key.size(); ++v) {

if (!inMST[v] && key[v] < min) {

min = key[v];

min\_index = v;

}

}

return min\_index;

}

// Function to implement Prim's algorithm to find the MST

void primMST(int V, const vector<vector<int>>& graph)

{

vector<int> parent(V, -1); // Array to store constructed MST

vector<int> key(V, INT\_MAX); // Key values to pick minimum weight edge

vector<bool> inMST(

V, false); // To represent vertices not yet included in MST

key[0] = 0; // Start from the first vertex

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

int u = minKey(key, inMST);

inMST[u] = true;

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

if (graph[u][v] && !inMST[v] && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

}

// Print the constructed MST

cout << "\nMinimum Spanning Tree (MST):\n";

cout << setw(10) << left << "Edges" << setw(10) << left << "Weights"

<< endl;

cout << "------------------" << endl;

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

cout << setw(1) << parent[i] << " - " << setw(8) << i << setw(10)

<< graph[i][parent[i]] << endl;

}

}

int main() {

cout << "\n5C6 - Amit Singhal (11614802722)\n" << endl;

int V = 4; // Number of vertices in the graph

vector<Edge> edges = { { 0, 1, 7 }, { 0, 2, 9 }, { 0, 3, 14 },

{ 1, 2, 10 }, { 1, 3, 15 }, { 2, 3, 11 } };

displayGraph(V, edges);

// Convert edge list to adjacency matrix

vector<vector<int>> adjMatrix = toAdjacencyMatrix(V, edges);

// Measure the time taken to find the MST

auto start = high\_resolution\_clock::now();

primMST(V, adjMatrix);

auto stop = high\_resolution\_clock::now();

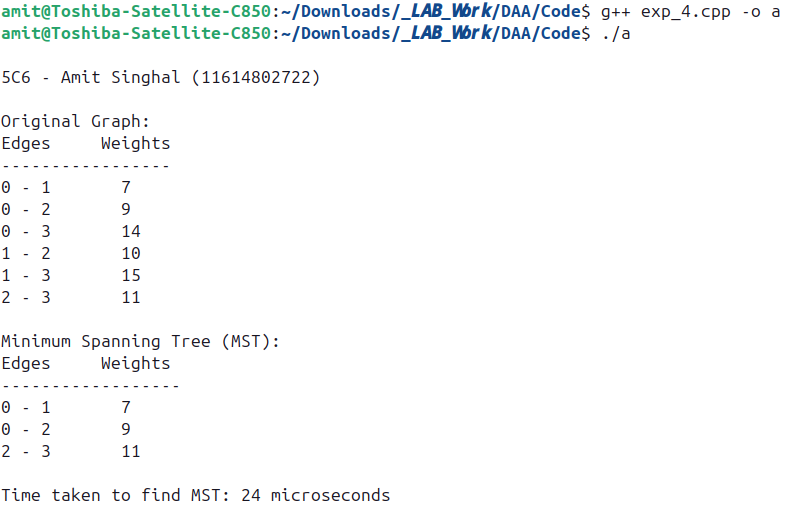
auto duration = duration\_cast<microseconds>(stop - start);

cout << "\nTime taken to find MST: " << duration.count()

<< " microseconds\n";

return 0;

}



Output ::

**Lab Exercise - 5**

* AIM ::

WAP in C++ to implement Dijsktra’s Algorithm & also calculate time complexity to find the shortest path

Source\_Code ::

#include <chrono>

#include <climits>

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

using namespace std::chrono;

// Structure to represent an edge in the graph

struct Edge {

int to;

int weight;

};

// Function to add an edge to the adjacency list

void addEdge(vector<vector<Edge> >& graph, int u, int v, int weight) {

graph[u].push\_back({v, weight});

graph[v].push\_back({u, weight}); // For undirected graph

}

// Function to display the graph

void displayGraph(const vector<vector<Edge> >& graph) {

cout << "Graph adjacency list representation:\n";

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

cout << "Node " << i << ": ";

for (const auto& edge : graph[i]) {

cout << "(to: " << edge.to << ", weight: " << edge.weight << ") ";

}

cout << endl;

}

}

// Dijkstra's algorithm implementation

vector<int> dijkstra(const vector<vector<Edge> >& graph,

int source,

int64\_t& timeTaken) {

int n = graph.size();

vector<int> dist(n, INT\_MAX); // Distance array, initialized to infinity

dist[source] = 0; // Distance to source is 0

// Priority queue to store {distance, node}

priority\_queue<pair<int, int>, vector<pair<int, int> >,

greater<pair<int, int> > >

pq;

pq.push({0, source});

// Measure time start

auto start = high\_resolution\_clock::now();

while (!pq.empty()) {

int u = pq.top().second; // Get the node with the smallest distance

int d = pq.top().first; // Get the distance of that node

pq.pop();

// If the distance in the queue is greater than the already found

// shortest distance, skip

if (d > dist[u])

continue;

// Explore the neighbors of node u

for (const auto& edge : graph[u]) {

int v = edge.to;

int weight = edge.weight;

// Relaxation step

if (dist[u] + weight < dist[v]) {

dist[v] = dist[u] + weight;

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

}

}

}

// Measure time end

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<nanoseconds>(stop - start);

timeTaken = duration.count(); // Time in nanoseconds

return dist;

}

int main() {

cout << "\n5C6 - Amit Singhal (11614802722)\n" << endl;

int n, e, source;

// Input: Number of nodes and edges

cout << "Enter the number of nodes and edges: ";

cin >> n >> e;

vector<vector<Edge> > graph(n);

// Input: Edges

cout << "\nEnter the edges (u, v, weight):\n";

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

int u, v, weight;

cin >> u >> v >> weight;

addEdge(graph, u, v, weight);

}

cout << endl;

// Display the graph

displayGraph(graph);

// Input: Source node

cout << "\nEnter the source node: ";

cin >> source;

// Time taken for calculating the shortest paths

int64\_t totalTime;

// Find shortest paths from the source node to all other nodes

vector<int> dist = dijkstra(graph, source, totalTime);

// Display shortest distances from the source to all other nodes

cout << "\nShortest distances from node " << source << " to all other nodes:\n";

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

if (dist[i] == INT\_MAX) {

cout << "To node " << i << " : Unreachable\n";

} else {

cout << "To node " << i << " : " << dist[i] << endl;

}

}

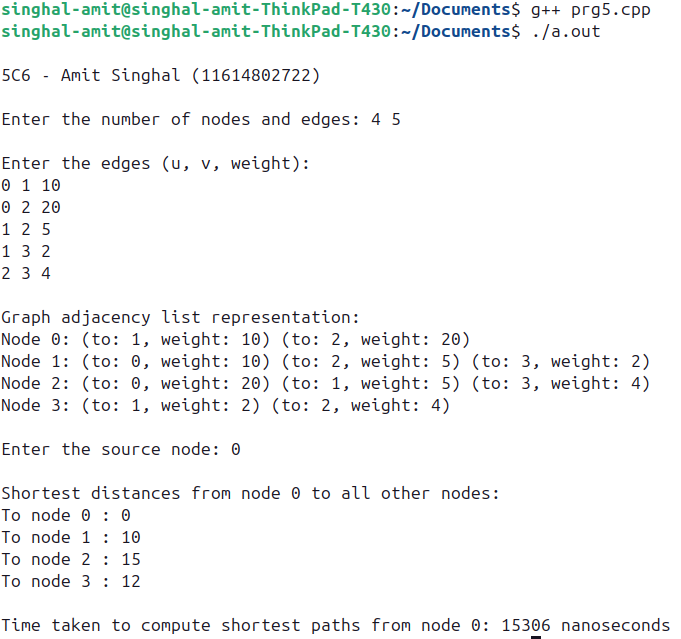
// Display time complexity

cout << "\nTime taken to compute shortest paths from node " << source

<< ": " << totalTime << " nanoseconds" << endl;

return 0;

}



Output ::

**Lab Exercise – 6**

AIM ::

WAP in C++ to implement Bellman Ford Algorithm & analyse its time complexity.

Source\_Code ::

#include <chrono>

#include <climits>

#include <iostream>

#include <vector>

using namespace std;

using namespace std::chrono;

// Structure to represent an edge in the graph

struct Edge

{

int from;

int to;

int weight;

};

// Function to add an edge to the graph

void addEdge(vector<Edge> &edges, int u, int v, int weight)

{

edges.push\_back({u, v, weight});

}

// Function to display the graph

void displayGraph(const vector<Edge> &edges, int n)

{

cout << "Graph edge list representation:\n";

for (const auto &edge : edges)

{

cout << "Edge from " << edge.from << " to " << edge.to << " with weight " << edge.weight << endl;

}

}

// Bellman-Ford algorithm implementation

vector<int> bellmanFord(const vector<Edge> &edges, int n, int source, bool &hasNegativeCycle, int64\_t &timeTaken)

{

vector<int> dist(n, INT\_MAX); // Distance array, initialized to infinity

dist[source] = 0; // Distance to source is 0

// Measure time start

auto start = high\_resolution\_clock::now();

// Relax all edges |V| - 1 times

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

{

for (const auto &edge : edges)

{

int u = edge.from;

int v = edge.to;

int weight = edge.weight;

if (dist[u] != INT\_MAX && dist[u] + weight < dist[v])

{

dist[v] = dist[u] + weight;

}

}

}

// Check for negative-weight cycles

hasNegativeCycle = false;

for (const auto &edge : edges)

{

int u = edge.from;

int v = edge.to;

int weight = edge.weight;

if (dist[u] != INT\_MAX && dist[u] + weight < dist[v])

{

hasNegativeCycle = true;

break;

}

}

// Measure time end

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<nanoseconds>(stop - start);

timeTaken = duration.count(); // Time in nanoseconds

return dist;

}

int main()

{

cout << "\n5C6 - Amit Singhal (11614802722)\n"

<< endl;

int n, e, source;

// Input: Number of nodes and edges

cout << "Enter the number of nodes and edges: ";

cin >> n >> e;

vector<Edge> edges;

// Input: Edges

cout << "\nEnter the edges (u, v, weight):\n";

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

{

int u, v, weight;

cin >> u >> v >> weight;

addEdge(edges, u, v, weight);

}

cout << endl;

// Display the graph

displayGraph(edges, n);

// Input: Source node

cout << "\nEnter the source node: ";

cin >> source;

// Time taken for calculating the shortest paths

int64\_t totalTime;

bool hasNegativeCycle;

// Find shortest paths from the source node to all other nodes using Bellman-Ford

vector<int> dist = bellmanFord(edges, n, source, hasNegativeCycle, totalTime);

// Check for negative-weight cycle

if (hasNegativeCycle)

{

cout << "\nGraph contains a negative weight cycle. Shortest paths cannot be computed.\n";

}

else

{

// Display shortest distances from the source to all other nodes

cout << "\nShortest distances from node " << source

<< " to all other nodes:\n";

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

{

if (dist[i] == INT\_MAX)

{

cout << "To node " << i << " : Unreachable\n";

}

else

{

cout << "To node " << i << " : " << dist[i] << endl;

}

}

// Display time complexity

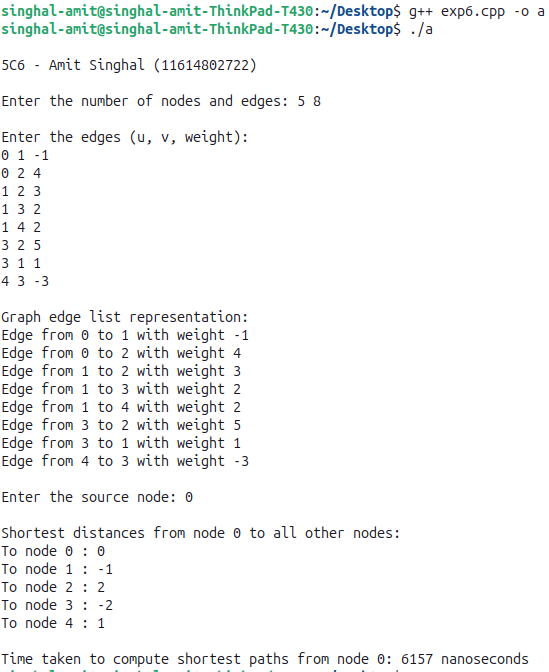
cout << "\nTime taken to compute shortest paths from node " << source

<< ": " << totalTime << " nanoseconds" << endl;

}

return 0;

}



Output ::

**Lab Exercise – 7**

AIM ::

WAP in C++ to implement N Queen’s problem using Back Tracking & analyse its time complexity.

#include <iostream>

Source\_Code ::

#include <vector>

#include <chrono>

using namespace std;

using namespace std::chrono;

bool isSafe(const vector<vector<int>> &board, int row, int col, int N)

{

// Check this row on left side

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

if (board[row][i])

return false;

// Check upper diagonal on left side

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

if (board[i][j])

return false;

// Check lower diagonal on left side

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

if (board[i][j])

return false;

return true;

}

bool solveNQueensUtil(vector<vector<int>> &board, int col, int N)

{

// If all queens are placed, return true

if (col >= N)

return true;

// Try placing queens in all rows of this column

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

{

if (isSafe(board, i, col, N))

{

// Place the queen

board[i][col] = 1;

// Recur to place the rest of the queens

if (solveNQueensUtil(board, col + 1, N))

return true;

// If placing queen in board[i][col] doesn't lead to a solution, remove it

board[i][col] = 0;

}

}

// If queen can't be placed in any row in this column, return false

return false;

}

bool solveNQueens(int N, int64\_t &timeTaken)

{

vector<vector<int>> board(N, vector<int>(N, 0));

// Measure time start

auto start = high\_resolution\_clock::now();

if (!solveNQueensUtil(board, 0, N))

{

cout << "Solution does not exist" << endl;

return false;

}

// Measure time end

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<nanoseconds>(stop - start);

timeTaken = duration.count(); // Time in nanoseconds

// Print the solution

cout << "Solution for " << N << "-Queens problem:" << endl;

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

{

for (int j = 0; j < N; j++)

cout << (board[i][j] ? "Q " : ". ");

cout << endl;

}

return true;

}

int main()

{

cout << "\n5C6 - Amit Singhal (11614802722)\n"

<< endl;

int N;

// Input: Size of the chessboard

cout << "Enter the number of queens (N): ";

cin >> N;

int64\_t totalTime;

// Solve the N-Queens problem

if (solveNQueens(N, totalTime))

{

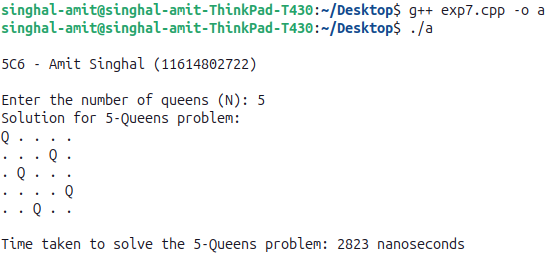
// Display time complexity

cout << "\nTime taken to solve the " << N << "-Queens problem: " << totalTime << " nanoseconds" << endl;

}

return 0;

}



Output ::

**Lab Exercise – 8**

AIM ::

WAP in C++ to implement Matrix Multiplication & analyse its time complexity.

Normal Matrix Multiplication Algorithm

#include <iostream>

Source\_Code ::

#include <vector>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Function to multiply two matrices using normal algorithm

vector<vector<int>> multiplyNormal(const vector<vector<int>> &A, const vector<vector<int>> &B, int N, int64\_t &timeTaken)

{

vector<vector<int>> C(N, vector<int>(N, 0));

// Measure time start

auto start = high\_resolution\_clock::now();

// Perform matrix multiplication

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

{

for (int j = 0; j < N; j++)

{

for (int k = 0; k < N; k++)

{

C[i][j] += A[i][k] \* B[k][j];

}

}

}

// Measure time end

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<nanoseconds>(stop - start);

timeTaken = duration.count(); // Time in nanoseconds

return C;

}

int main()

{

cout << "\n5C6 - Amit Singhal (11614802722)\n"

<< endl;

int N;

// Input: Matrix size (NxN)

cout << "Enter the size of the matrix (N x N): ";

cin >> N;

vector<vector<int>> A(N, vector<int>(N));

vector<vector<int>> B(N, vector<int>(N));

// Input: Matrix A

cout << "\nEnter matrix A values:\n";

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

for (int j = 0; j < N; j++)

cin >> A[i][j];

// Input: Matrix B

cout << "\nEnter matrix B values:\n";

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

for (int j = 0; j < N; j++)

cin >> B[i][j];

int64\_t totalTime;

// Multiply matrices using normal algorithm

vector<vector<int>> C = multiplyNormal(A, B, N, totalTime);

// Display result matrix

cout << "\nResultant matrix after normal multiplication:\n";

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

{

for (int j = 0; j < N; j++)

cout << C[i][j] << " ";

cout << endl;

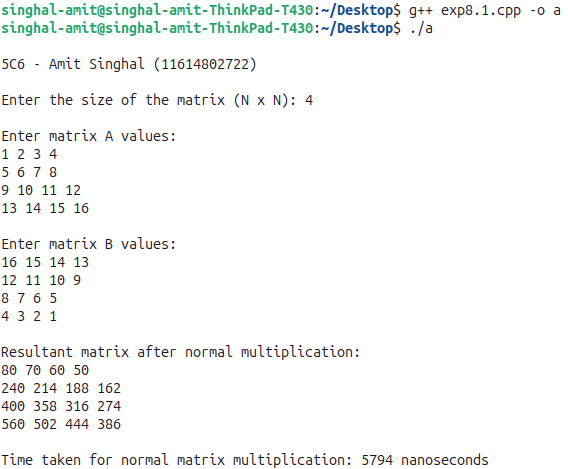
}

// Display time complexity

cout << "\nTime taken for normal matrix multiplication: " << totalTime << " nanoseconds\n";

return 0;

}



Output ::

Strassen’s Matrix Multiplication Algorithm

#include <iostream>

Source\_Code ::

#include <vector>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Function to add two matrices

vector<vector<int>> addMatrix(const vector<vector<int>> &A, const vector<vector<int>> &B, int N)

{

vector<vector<int>> C(N, vector<int>(N));

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

for (int j = 0; j < N; j++)

C[i][j] = A[i][j] + B[i][j];

return C;

}

// Function to subtract two matrices

vector<vector<int>> subtractMatrix(const vector<vector<int>> &A, const vector<vector<int>> &B, int N)

{

vector<vector<int>> C(N, vector<int>(N));

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

for (int j = 0; j < N; j++)

C[i][j] = A[i][j] - B[i][j];

return C;

}

// Function to perform Strassen's matrix multiplication

vector<vector<int>> strassenMultiply(const vector<vector<int>> &A, const vector<vector<int>> &B, int N, int64\_t &timeTaken)

{

vector<vector<int>> C(N, vector<int>(N, 0));

// Base case for recursion

if (N == 1)

{

C[0][0] = A[0][0] \* B[0][0];

return C;

}

// Measure time start

auto start = high\_resolution\_clock::now();

int newSize = N / 2;

vector<int> z(newSize, 0);

vector<vector<int>> a11(newSize, z), a12(newSize, z), a21(newSize, z), a22(newSize, z);

vector<vector<int>> b11(newSize, z), b12(newSize, z), b21(newSize, z), b22(newSize, z);

vector<vector<int>> c11(newSize, z), c12(newSize, z), c21(newSize, z), c22(newSize, z);

vector<vector<int>> p1(newSize, z), p2(newSize, z), p3(newSize, z), p4(newSize, z), p5(newSize, z), p6(newSize, z), p7(newSize, z);

// Dividing the matrices into quadrants

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

{

for (int j = 0; j < newSize; j++)

{

a11[i][j] = A[i][j];

a12[i][j] = A[i][j + newSize];

a21[i][j] = A[i + newSize][j];

a22[i][j] = A[i + newSize][j + newSize];

b11[i][j] = B[i][j];

b12[i][j] = B[i][j + newSize];

b21[i][j] = B[i + newSize][j];

b22[i][j] = B[i + newSize][j + newSize];

}

}

// Calculating p1 to p7

p1 = strassenMultiply(addMatrix(a11, a22, newSize), addMatrix(b11, b22, newSize), newSize, timeTaken);

p2 = strassenMultiply(addMatrix(a21, a22, newSize), b11, newSize, timeTaken);

p3 = strassenMultiply(a11, subtractMatrix(b12, b22, newSize), newSize, timeTaken);

p4 = strassenMultiply(a22, subtractMatrix(b21, b11, newSize), newSize, timeTaken);

p5 = strassenMultiply(addMatrix(a11, a12, newSize), b22, newSize, timeTaken);

p6 = strassenMultiply(subtractMatrix(a21, a11, newSize), addMatrix(b11, b12, newSize), newSize, timeTaken);

p7 = strassenMultiply(subtractMatrix(a12, a22, newSize), addMatrix(b21, b22, newSize), newSize, timeTaken);

// Calculating c11, c12, c21, c22

c11 = addMatrix(subtractMatrix(addMatrix(p1, p4, newSize), p5, newSize), p7, newSize);

c12 = addMatrix(p3, p5, newSize);

c21 = addMatrix(p2, p4, newSize);

c22 = addMatrix(subtractMatrix(addMatrix(p1, p3, newSize), p2, newSize), p6, newSize);

// Combining the results into matrix C

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

{

for (int j = 0; j < newSize; j++)

{

C[i][j] = c11[i][j];

C[i][j + newSize] = c12[i][j];

C[i + newSize][j] = c21[i][j];

C[i + newSize][j + newSize] = c22[i][j];

}

}

// Measure time end

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<nanoseconds>(stop - start);

timeTaken = duration.count(); // Time in nanoseconds

return C;

}

int main()

{

cout << "\n5C6 - Amit Singhal (11614802722)\n"

<< endl;

int N;

// Input: Matrix size (NxN)

cout << "Enter the size of the matrix (N x N, must be a power of 2): ";

cin >> N;

vector<vector<int>> A(N, vector<int>(N));

vector<vector<int>> B(N, vector<int>(N));

// Input: Matrix A

cout << "\nEnter matrix A values:\n";

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

for (int j = 0; j < N; j++)

cin >> A[i][j];

// Input: Matrix B

cout << "\nEnter matrix B values:\n";

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

for (int j = 0; j < N; j++)

cin >> B[i][j];

int64\_t totalTime;

// Multiply matrices using Strassen's algorithm

vector<vector<int>> C = strassenMultiply(A, B, N, totalTime);

// Display result matrix

cout << "\nResultant matrix after Strassen's multiplication:\n";

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

{

for (int j = 0; j < N; j++)

cout << C[i][j] << " ";

cout << endl;

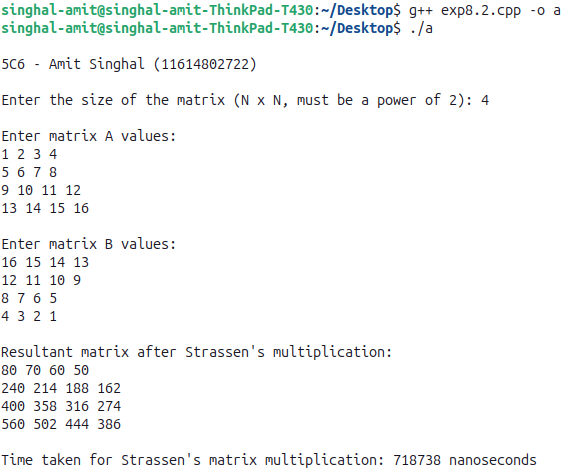
}

// Display time complexity

cout << "\nTime taken for Strassen's matrix multiplication: " << totalTime << " nanoseconds\n";

return 0;

}



Output ::

**Lab Exercise – 9**

AIM ::

WAP in C++ to implement Longest Common Subsequence problem & analyse its time complexity.

Iterative Approach

#include <iostream>

Source\_Code ::

#include <vector>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Function to find LCS using iterative DP approach

int lcsIterative(string X, string Y, int64\_t &timeTaken)

{

int m = X.length();

int n = Y.length();

// Create a 2D table to store lengths of LCS

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

// Measure time start

auto start = high\_resolution\_clock::now();

// Build the dp table in a bottom-up manner

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

{

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

{

if (i == 0 || j == 0)

dp[i][j] = 0;

else if (X[i - 1] == Y[j - 1])

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

else

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

}

}

// Measure time end

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<nanoseconds>(stop - start);

timeTaken = duration.count(); // Time in nanoseconds

// The length of LCS is in dp[m][n]

return dp[m][n];

}

int main()

{

cout << "\n5C6 - Amit Singhal (11614802722)\n"

<< endl;

string X, Y;

// Input: Two strings

cout << "Enter first string: ";

cin >> X;

cout << "Enter second string: ";

cin >> Y;

int64\_t totalTime;

// Find the length of LCS using iterative approach

int lcsLength = lcsIterative(X, Y, totalTime);

// Display the length of LCS

cout << "\nLength of Longest Common Subsequence (Iterative): " << lcsLength << endl;

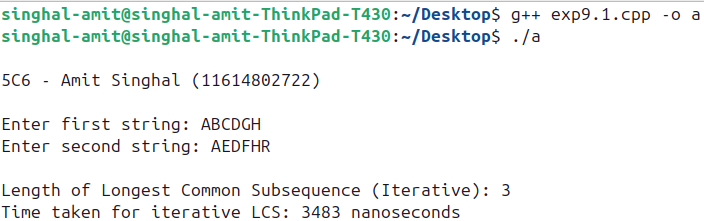
// Display time complexity

cout << "Time taken for iterative LCS: " << totalTime << " nanoseconds\n";

return 0;

}

Output ::



Recursive Approach

#include <iostream>

Source\_Code ::

#include <vector>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Function to find LCS using recursive approach with memoization

int lcsRecursive(string X, string Y, int m, int n, vector<vector<int>> &memo)

{

// Base case: if either string is empty

if (m == 0 || n == 0)

return 0;

// Check if the result is already computed

if (memo[m][n] != -1)

return memo[m][n];

// If characters match, store the result in memo

if (X[m - 1] == Y[n - 1])

memo[m][n] = 1 + lcsRecursive(X, Y, m - 1, n - 1, memo);

else

// If characters don't match, take the max of two options

memo[m][n] = max(lcsRecursive(X, Y, m - 1, n, memo), lcsRecursive(X, Y, m, n - 1, memo));

return memo[m][n];

}

int main()

{

cout << "\n5C6 - Amit Singhal (11614802722)\n" << endl;

string X, Y;

// Input: Two strings

cout << "Enter first string: ";

cin >> X;

cout << "Enter second string: ";

cin >> Y;

int m = X.length();

int n = Y.length();

// Initialize memoization table with -1

vector<vector<int>> memo(m + 1, vector<int>(n + 1, -1));

// Measure time start

auto start = high\_resolution\_clock::now();

// Find the length of LCS using recursive approach

int lcsLength = lcsRecursive(X, Y, m, n, memo);

// Measure time end

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<nanoseconds>(stop - start);

int64\_t totalTime = duration.count(); // Time in nanoseconds

// Display the length of LCS

cout << "\nLength of Longest Common Subsequence (Recursive): " << lcsLength << endl;

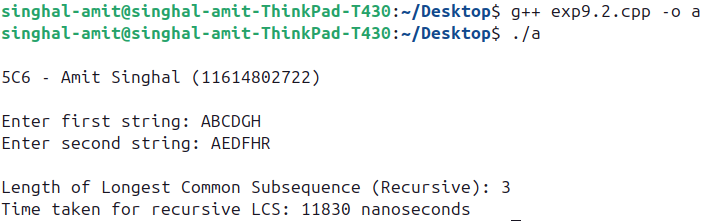
// Display time complexity

cout << "Time taken for recursive LCS: " << totalTime << " nanoseconds\n";

return 0;

}

Output ::



**Lab Exercise – 10**

AIM ::

WAP in C++ to implement naive String Matching algorithm, Rabin Karp algorithm and Knuth Morris Pratt algorithm & analyse its time complexity.

1) Naive String Matching Algorithm

#include <iostream>

Source\_Code ::

#include <chrono>

using namespace std;

using namespace std::chrono;

// Function for Naive String Matching Algorithm

void naiveStringMatching(string text, string pattern, int64\_t &timeTaken)

{

int n = text.length();

int m = pattern.length();

bool found = false;

// Measure time start

auto start = high\_resolution\_clock::now();

// Slide pattern over text one by one

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

{

int j;

// Check for match at this position

for (j = 0; j < m; j++)

{

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

break;

}

// If pattern[0...m-1] matches text[i...i+m-1], a match is found

if (j == m)

{

cout << "Pattern found at index " << i << endl;

found = true;

}

}

// Measure time end

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<nanoseconds>(stop - start);

timeTaken = duration.count(); // Time in nanoseconds

if (!found)

{

cout << "Pattern not found in the text." << endl;

}

}

int main()

{

cout << "\n5C6 - Amit Singhal (11614802722)\n"

<< endl;

string text, pattern;

// Input: Text and Pattern

cout << "Enter the text: ";

cin >> text;

cout << "Enter the pattern: ";

cin >> pattern;

int64\_t totalTime;

// Call the naive string matching function

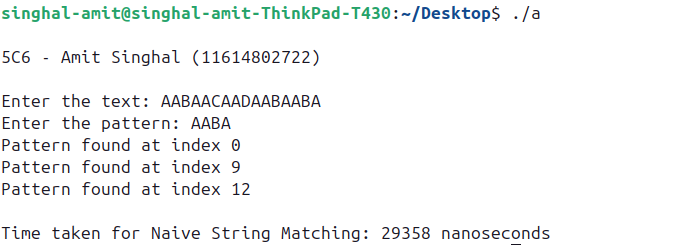
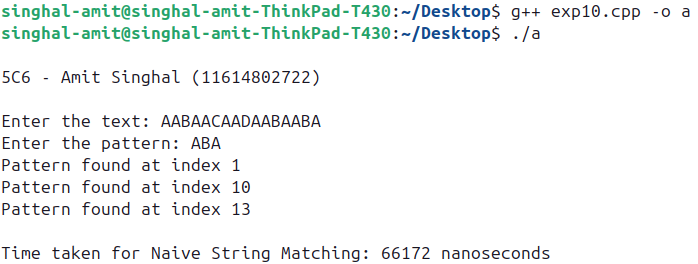
naiveStringMatching(text, pattern, totalTime);

// Display the time complexity

cout << "\nTime taken for Naive String Matching: " << totalTime << " nanoseconds" << endl;

return 0;

}



Output ::

2) Rabin Karp Algorithm

#include <iostream>

Source\_Code ::

#include <chrono>

using namespace std;

using namespace std::chrono;

#define d 256 // Number of characters in the input alphabet

// Rabin-Karp function

void rabinKarp(string text, string pattern, int q, int64\_t &timeTaken)

{

int n = text.length();

int m = pattern.length();

int i, j;

int p = 0; // Hash value for pattern

int t = 0; // Hash value for text

int h = 1;

bool found = false;

// Measure time start

auto start = high\_resolution\_clock::now();

// The value of h would be "pow(d, m-1)%q"

for (i = 0; i < m - 1; i++)

h = (h \* d) % q;

// Calculate the hash value of pattern and first window of text

for (i = 0; i < m; i++)

{

p = (d \* p + pattern[i]) % q;

t = (d \* t + text[i]) % q;

}

// Slide the pattern over text one by one

for (i = 0; i <= n - m; i++)

{

// Check the hash values of the current window of text and pattern

if (p == t)

{

// If hash values match, check characters one by one

for (j = 0; j < m; j++)

{

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

break;

}

// If the pattern is found

if (j == m)

{

cout << "Pattern found at index " << i << endl;

found = true;

}

}

// Calculate hash value for next window of text

if (i < n - m)

{

t = (d \* (t - text[i] \* h) + text[i + m]) % q;

// We might get a negative value of t, convert it to positive

if (t < 0)

t = (t + q);

}

}

// Measure time end

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<nanoseconds>(stop - start);

timeTaken = duration.count(); // Time in nanoseconds

if (!found)

{

cout << "Pattern not found in the text." << endl;

}

}

int main()

{

cout << "\n5C6 - Amit Singhal (11614802722)\n"

<< endl;

string text, pattern;

int q = 101; // A prime number

// Input: Text and Pattern

cout << "Enter the text: ";

cin >> text;

cout << "Enter the pattern: ";

cin >> pattern;

int64\_t totalTime;

// Call the Rabin-Karp algorithm function

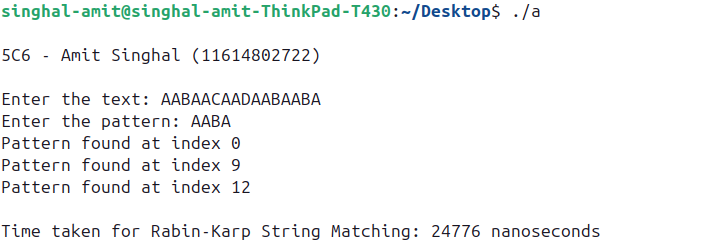
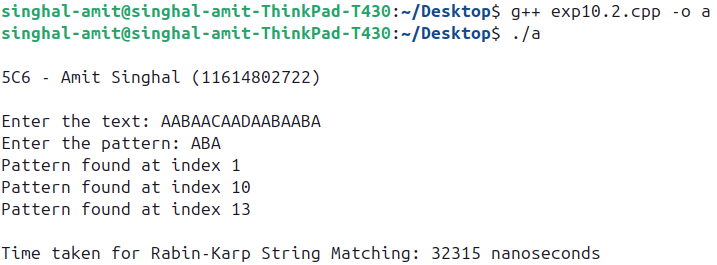
rabinKarp(text, pattern, q, totalTime);

// Display the time complexity

cout << "\nTime taken for Rabin-Karp String Matching: " << totalTime << " nanoseconds" << endl;

return 0;

}



Output ::

3) Knuth Morris Pratt Algorithm

#include <iostream>

Source\_Code ::

#include <vector>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Function to compute the Longest Prefix Suffix (LPS) array

void computeLPSArray(string pattern, vector<int> &lps)

{

int len = 0; // Length of the previous longest prefix suffix

lps[0] = 0; // LPS[0] is always 0

// The loop calculates lps[i] for i = 1 to m-1

for (int i = 1; i < pattern.length();)

{

if (pattern[i] == pattern[len])

{

len++;

lps[i] = len;

i++;

}

else

{

// This is tricky. Consider the example "AAACAAAA" and i = 7.

if (len != 0)

{

len = lps[len - 1];

}

else

{

lps[i] = 0;

i++;

}

}

}

}

// KMP algorithm implementation

void KMPSearch(string text, string pattern, int64\_t &timeTaken)

{

int m = pattern.length();

int n = text.length();

vector<int> lps(m); // LPS array for the pattern

computeLPSArray(pattern, lps); // Preprocess the pattern

// Measure time start

auto start = high\_resolution\_clock::now();

int i = 0; // Index for text

int j = 0; // Index for pattern

bool found = false;

while (i < n)

{

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

{

i++;

j++;

}

if (j == m)

{

cout << "Pattern found at index " << (i - j) << endl;

found = true;

j = lps[j - 1]; // Continue to search for next occurrence

}

else if (i < n && pattern[j] != text[i]) // mismatch after j matches

{

if (j != 0)

j = lps[j - 1];

else

i++;

}

}

// Measure time end

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<nanoseconds>(stop - start);

timeTaken = duration.count(); // Time in nanoseconds

if (!found)

{

cout << "Pattern not found in the text." << endl;

}

}

int main()

{

cout << "\n5C6 - Amit Singhal (11614802722)\n"

<< endl;

string text, pattern;

// Input: Text and Pattern

cout << "Enter the text: ";

cin >> text;

cout << "Enter the pattern: ";

cin >> pattern;

int64\_t totalTime;

// Call the KMP algorithm function

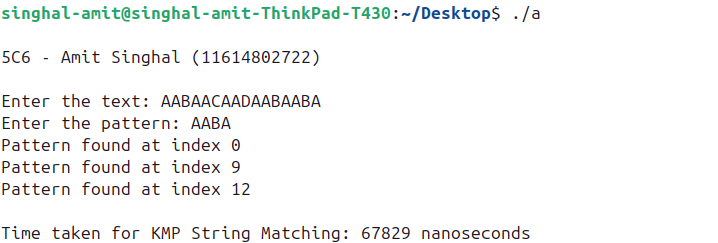
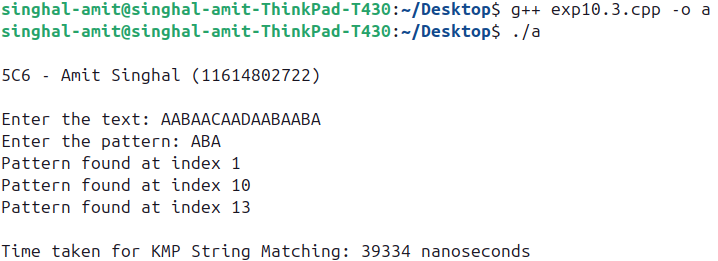
KMPSearch(text, pattern, totalTime);

// Display the time complexity

cout << "\nTime taken for KMP String Matching: " << totalTime << " nanoseconds" << endl;

return 0;

}



Output ::