**A blue and white logo

Description automatically generated**

# Name: Apurba Koirala

# Reg no: 22BCE3799

# Subject Code: BCSE204P

# Course Title: Design and Analysis of Algorithms Lab

# Lab Slot: L39 + L40

# Guided by: Dr. IYAPPAN P

# Lab Assessment 3

A white background with black text

AI-generated content may be incorrect.

1. KMP Algorithm

Algorithm:

KMP\_String\_Matcher(T, P):

n = length(T)

m = length(P)

π = Compute\_LPS(P)

i = 0

j = 0

while i < n:

if T[i] == P[j]:

i = i + 1

j = j + 1

if j == m:

print("Pattern occurs at index", i - j)

j = π[j - 1]

else:

if j != 0:

j = π[j - 1]

else:

i = i + 1

Compute\_LPS(P):

m = length(P)

LPS = array of size m

LPS[0] = 0

len = 0

i = 1

while i < m:

if P[i] == P[len]:

len = len + 1

LPS[i] = len

i = i + 1

else:

if len != 0:

len = LPS[len - 1]

else:

LPS[i] = 0

i = i + 1

return LPS

Source Code:

#include <iostream>

#include <string>

#include <vector>

using namespace std;

void constructLps(string &pat, vector<int> &lps) {

int len = 0;

lps[0] = 0;

int i = 1;

while (i < pat.length()) {

if (pat[i] == pat[len]) {

len++;

lps[i] = len;

i++;

}

else {

if (len != 0) {

len = lps[len - 1];

}

else {

lps[i] = 0;

i++;

}

}

}

}

vector<int> search(string &pat, string &txt) {

int n = txt.length();

int m = pat.length();

vector<int> lps(m);

vector<int> res;

constructLps(pat, lps);

int i = 0;

int j = 0;

while (i < n) {

if (txt[i] == pat[j]) {

i++;

j++;

if (j == m) {

res.push\_back(i - j);

j = lps[j - 1];

}

}

else {

if (j != 0)

j = lps[j - 1];

else

i++;

}

}

return res;

}

int main() {

string txt = "SAQSPAPGPGGAS";

string pat = "PGGA";

vector<int> res = search(pat, txt);

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

cout << res[i] << " ";

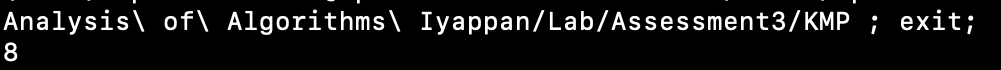
return 0;

}

Input:



Output:



= 8 is the index

Time Complexity Analysis:

A yellow lined notebook with writing on it

AI-generated content may be incorrect.

1. Rabin-Karp Algorithm

Algorithm:

Rabin\_Karp\_Matcher(T, P, d, q):

n = length(T)

m = length(P)

h = d^(m-1) mod q

p = 0

t = 0

for i = 0 to m-1:

p = (d \* p + P[i]) mod q

t = (d \* t + T[i]) mod q

for s = 0 to n-m:

if p == t:

if P[0..m-1] == T[s..s+m-1]:

print "Pattern occurs at index", s

if s < n-m:

t = (d \* (t - T[s] \* h) + T[s + m]) mod q

if t < 0:

t = t + q

Source Code:

#include <iostream>

#include <string>

using namespace std;

void search(string pat, string txt, int q)

{

int M = pat.size();

int N = txt.size();

int i, j;

int p = 0;

int t = 0;

int h = 1;

int d = 256;

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

h = (h \* d) % q;

for (i = 0; i < M; i++) {

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

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

}

for (i = 0; i <= N - M; i++) {

if (p == t) {

for (j = 0; j < M; j++) {

if (txt[i + j] != pat[j]) {

break;

}

}

if (j == M)

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

<< endl;

}

if (i < N - M) {

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

if (t < 0)

t = (t + q);

}

}

}

int main()

{

string txt = "SAQSPAPGPGGAS";

string pat = "PGGA";

int q = INT\_MAX;

search(pat, txt, q);

return 0;

}

Input:

A black background with white text

AI-generated content may be incorrect.

Output:

A black background with white text

AI-generated content may be incorrect.

At index 8 again

Time Complexity Analysis:

A yellow lined paper with writing on it

AI-generated content may be incorrect.

A diagram of a graph

AI-generated content may be incorrect.

Algorithm:

createGraph(V, E):

Initialize Graph with V vertices and E edges

Return Graph

displayDistances(distances):

Print "Vertex Distance from Source"

For each (vertex, distance) in distances:

Print vertex, distance

bellmanFord(graph, source):

Initialize distances for all vertices as ∞, except source = 0

Repeat (V-1) times:

For each edge (u, v, w) in graph:

If distances[u] + w < distances[v]:

Update distances[v] = distances[u] + w

For each edge (u, v, w) in graph:

If distances[u] + w < distances[v]:

Print "Negative cycle detected"

Return

displayDistances(distances)

Source Code:

#include <iostream>

#include <cstdlib>

#include <climits>

#include <map>

using namespace std;

struct Edge {

char startVertex, endVertex;

int edgeWeight;

};

struct Graph {

int numVertices, numEdges;

struct Edge\* edges;

};

struct Graph\* initializeGraph(int vertices, int edges) {

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->numEdges = edges;

graph->edges = (struct Edge\*)malloc(graph->numEdges \* sizeof(struct Edge));

return graph;

}

void displayFinalDistances(map<char, int>& distanceMap) {

cout << "\nVertex\tMinimum Distance from Source" << endl;

for (auto& entry : distanceMap) {

cout << entry.first << "\t\t" << entry.second << endl;

}

}

void bellmanFord(struct Graph\* graph, char sourceVertex) {

int vertices = graph->numVertices;

int edges = graph->numEdges;

map<char, int> shortestDistances;

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

shortestDistances[graph->edges[i].startVertex] = INT\_MAX;

shortestDistances[graph->edges[i].endVertex] = INT\_MAX;

}

shortestDistances[sourceVertex] = 0;

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

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

char u = graph->edges[j].startVertex;

char v = graph->edges[j].endVertex;

int weight = graph->edges[j].edgeWeight;

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

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

}

}

}

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

char u = graph->edges[i].startVertex;

char v = graph->edges[i].endVertex;

int weight = graph->edges[i].edgeWeight;

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

cout << "\nWarning: The graph contains a negative weight cycle!" << endl;

return;

}

}

displayFinalDistances(shortestDistances);

}

int main() {

int vertices, edges;

char source;

cout << "Enter the number of vertices: ";

cin >> vertices;

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

cin >> edges;

cout << "Enter the source vertex: ";

cin >> source;

struct Graph\* graph = initializeGraph(vertices, edges);

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

cout << "\nEnter properties for edge " << i + 1 << " (Start Vertex, End Vertex, Weight): ";

cin >> graph->edges[i].startVertex >> graph->edges[i].endVertex >> graph->edges[i].edgeWeight;

}

bellmanFord(graph, source);

return 0;

}

Input:

A screenshot of a computer screen

AI-generated content may be incorrect.

Output:

A black screen with white text

AI-generated content may be incorrect.

Time Complexity Analysis:

A yellow paper with writing on it

AI-generated content may be incorrect.

A diagram of a diagram

AI-generated content may be incorrect.

Algorithm

FloydWarshall(graph, vertices):

numVertices = size of vertices

for k from 0 to numVertices - 1:

for i from 0 to numVertices - 1:

for j from 0 to numVertices - 1:

start = vertices[i]

end = vertices[j]

intermediate = vertices[k]

if graph[start][intermediate] is not INF and graph[intermediate][end] is not INF:

if graph[start][end] is INF or graph[start][end] > graph[start][intermediate] + graph[intermediate][end]:

graph[start][end] = graph[start][intermediate] + graph[intermediate][end]

Source Code

#include <iostream>

#include <vector>

#include <map>

#include <climits>

using namespace std;

void floydWarshall(map<char, map<char, int> > &graph, vector<char> &vertices) {

int numVertices = vertices.size();

for (int k = 0; k < numVertices; k++) {

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

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

char start = vertices[i], end = vertices[j], intermediate = vertices[k];

if (graph[start][intermediate] != INT\_MAX && graph[intermediate][end] != INT\_MAX &&

(graph[start][end] == INT\_MAX || graph[start][end] > graph[start][intermediate] + graph[intermediate][end])) {

graph[start][end] = graph[start][intermediate] + graph[intermediate][end];

}

}

}

}

}

int main() {

int numVertices, numEdges;

cout << "Enter the number of vertices: ";

cin >> numVertices;

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

cin >> numEdges;

vector<char> vertices(numVertices);

map<char, map<char, int> > graph;

cout << "Enter the vertex labels: ";

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

cin >> vertices[i];

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

graph[vertices[i]][vertices[j]] = (i == j) ? 0 : INT\_MAX;

}

}

cout << "Enter edges in format (source destination weight):" << endl;

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

char source, destination;

int weight;

cin >> source >> destination >> weight;

graph[source][destination] = weight;

}

floydWarshall(graph, vertices);

cout << "\nAll-Pairs Shortest Paths:\n ";

for (char vertex : vertices) {

cout << vertex << "\t";

}

cout << "\n-------------------------------------------------\n";

for (char start : vertices) {

cout << start << " | ";

for (char end : vertices) {

if (graph[start][end] == INT\_MAX)

cout << "INF\t";

else

cout << graph[start][end] << "\t";

}

cout << endl;

}

return 0;

}

Input

A computer screen shot of a black screen

AI-generated content may be incorrect.

Output

A black and white screen with numbers and a black background

AI-generated content may be incorrect.

Time Complexity Analysis

Yellow lined notebooks with mathematical equations

AI-generated content may be incorrect.