**THEORY OF COMPUTING**

**Msc. Artificial Intelligence**

**Submitted by,**

**ASHIMA SHIBU**

**Msc. CS.AI (1 sem)**

**LAB CYCLE 1**

***INDEX***

|  |  |  |
| --- | --- | --- |
| **NO:** | **TOPIC** | **PAGE** |
| **1** | **Fibinocci series** |  |
| **2** | **Palindrome** |  |
| **3** | **Armstrong** |  |
| **4** | **Reverse form of a 1-D array** |  |
| **5** | **Largest number of an array** |  |
| **6** | **Numbers return arranged number** |  |
| **7** | **Two stacks in a single array** |  |
| **8** | **Individual numbers return the arranged numbers** |  |
| **9** | **Circular queue using linked list** |  |
| **10** | **Find a number in a singly linked list** |  |
| **11** |  |  |
| **12** | **Magic triangle** |  |
| **13** | **Convert the given binary to decimal** |  |
| **14** | **Binary search in a given set of numbers** |  |
| **15** | **All pair shortest path algorithm** |  |

**1.Write a function to print a fibinocci series upto the n terms?**

**CODE:**

#include <stdio.h>

void printFibonacci(int n) {

int t1 = 0, t2 = 1, nextTerm;

printf("%d, %d, ", t1, t2);

for (int i = 3; i <= n; i++)

{

nextTerm = t1 + t2;

printf("%d ", nextTerm);

t1 = t2;

t2 = nextTerm;

}

}

int main()

{

int n;

printf("Enter the number of terms: ");

scanf("%d", &n);

printf("Fibonacci Series up to %d terms: ", n);

printFibonacci(n);

return 0;

}

**OUTPUT:**

Enter the number of terms: 4

Fibonacci Series up to 4 terms: 0, 1, 1 2

**2.Write a function that will accept a number and return its palindrome?**

**CODE:**

#include <stdio.h>

int reverseNumber(int num) {

int reverse = 0;

while (num != 0) {

int remainder = num % 10;

reverse = reverse \* 10 + remainder;

num /= 10;

}

return reverse;

}

void checkPalindrome(int num) {

if (num < 0) {

printf("%d is not a palindrome\n", num);

} else {

int reversedNum = reverseNumber(num);

if (num == reversedNum) {

printf("%d is a palindrome\n", num);

} else {

printf("%d is not a palindrome\n", num);

}

}

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

printf("Reversed number: %d\n", reverseNumber(num));

checkPalindrome(num);

return 0;

}

**OUTPUT:**

Enter a number: 543

Reversed number: 345

543 is not a palindrome

**3.Write a function that will accept a number and return its armstrong ?**

**CODE:**

#include <stdio.h>

#include <math.h>

int countDigits(int num) {

int count = 0;

while (num != 0) {

num /= 10;

count++;

}

return count;

}

int armstrongNumber(int num) {

int digits = countDigits(num);

int armstrong = 0;

int temp = num;

while (temp != 0)

{

int digit = temp % 10;

armstrong += pow(digit, digits);

temp /= 10;

}

return armstrong;

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

int armstrong = armstrongNumber(num);

if (num == armstrong) {

printf("%d is an Armstrong number. Its Armstrong value is %d.\n", num, armstrong);

} else {

printf("%d is not an Armstrong number.\n", num);

}

return 0;

}

**OUTPUT:**

Enter a number: 153

153 is an Armstrong number. Its Armstrong value is 153.

Enter a number: 45

45 is not an Armstrong number.

**4.Write a function that will return the reverse form of a single dimension array?**

**CODE:**

#include <stdio.h>

void reverseArray(int arr[], int size) {

int temp;

int start = 0;

int end = size - 1;

while (start < end) {

temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

start++;

end--;

}

}

void printArray(int arr[], int size) {

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

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

}

printf("\n");

}

int main() {

int size;

printf("Enter the size of the array: ");

scanf("%d", &size);

int arr[size];

printf("Enter %d elements:\n", size);

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

scanf("%d", &arr[i]);

}

printf("Original array: ");

printArray(arr, size);

reverseArray(arr, size);

printf("Reversed array: ");

printArray(arr, size);

return 0;

}

**OUTPUT:**

Enter the size of the array: 5

Enter 5 elements:

1

2

3

4

5

Original array: 1 2 3 4 5

Reversed array: 5 4 3 2 1

**5**.**Write a function to find the largest number in a given array of ‘m’ rows and ‘n’ columns?**

**CODE:**

#include <stdio.h>

int findLargest(int arr[][10], int m, int n) {

int max = arr[0][0];

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

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

if (arr[i][j] > max) {

max = arr[i][j];

}

}

}

return max;

}

void printArray(int arr[][10], int m, int n) {

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

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

printf("%d ", arr[i][j]);

}

printf("\n");

}

}

int main() {

int m, n;

printf("Enter the number of rows (m): ");

scanf("%d", &m);

printf("Enter the number of columns (n): ");

scanf("%d", &n);

int arr[m][10];

printf("Enter the elements of the 2D array:\n");

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

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

scanf("%d", &arr[i][j]);

}

}

printf("Original 2D array:\n");

printArray(arr, m, n);

int largest = findLargest(arr, m, n);

printf("Largest number in the 2D array: %d\n", largest);

return 0;

}

**OUTPUT:**

Enter the number of rows (m): 2

Enter the number of columns (n): 2

Enter the elements of the 2D array:

1

2

3

4

Original 2D array:

1 2

3 4

Largest number in the 2D array: 4

**6.Write a function that will read a series of individual numbers which when ended with blank enter it will return the arranged number?**

**CODE:**

#include <stdio.h>

#define MAX\_NUMBERS 100

int main() {

int arr[MAX\_NUMBERS];

int i = 0;

printf("Enter numbers (press enter after each number, and a blank line to finish input):\n");

while (i < MAX\_NUMBERS) {

char line[1024];

if (fgets(line, sizeof(line), stdin) == NULL) {

break;

}

if (line[0] == '\n') {

break;

}

int num;

if (sscanf(line, "%d", &num) != 1) {

printf("Invalid input. Please enter a number.\n");

continue;

}

arr[i++] = num;

}

printf("Arranged numbers: ");

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

printf("%d ", arr[j]);

}

printf("\n");

return 0;

}

**OUTPUT:**

Enter numbers (press enter after each number, and a blank line to finish input):

1

2

3

4

5

Arranged numbers: 1 2 3 4 5

**7.Implement two stacks in a single array?**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

typedef struct {

int\* data;

int top1;

int top2;

int size;

} TwoStacks;

TwoStacks\* createTwoStacks(int size) {

TwoStacks\* twoStacks = (TwoStacks\*) malloc(sizeof(TwoStacks));

twoStacks->data = (int\*) malloc(size \* sizeof(int));

twoStacks->top1 = -1;

twoStacks->top2 = size;

twoStacks->size = size;

return twoStacks;

}

int isEmptyStack1(TwoStacks\* twoStacks) {

return twoStacks->top1 == -1;

}

int isEmptyStack2(TwoStacks\* twoStacks) {

return twoStacks->top2 == twoStacks->size;

}

int isFull(TwoStacks\* twoStacks) {

return twoStacks->top1 + 1 == twoStacks->top2;

}

void pushStack1(TwoStacks\* twoStacks, int value) {

if (isFull(twoStacks)) {

printf("Stack overflow\n");

return;

}

twoStacks->data[++twoStacks->top1] = value;

}

void pushStack2(TwoStacks\* twoStacks, int value) {

if (isFull(twoStacks)) {

printf("Stack overflow\n");

return;

}

twoStacks->data[--twoStacks->top2] = value;

}

int popStack1(TwoStacks\* twoStacks) {

if (isEmptyStack1(twoStacks)) {

printf("Stack underflow\n");

return -1;

}

return twoStacks->data[twoStacks->top1--];

}

int popStack2(TwoStacks\* twoStacks) {

if (isEmptyStack2(twoStacks)) {

printf("Stack underflow\n");

return -1;

}

return twoStacks->data[twoStacks->top2++];

}

void displayStacks(TwoStacks\* twoStacks) {

int i;

printf("Stack 1: ");

for (i = 0; i <= twoStacks->top1; i++) {

printf("%d ", twoStacks->data[i]);

}

printf("\n");

printf("Stack 2: ");

for (i = twoStacks->top2; i < twoStacks->size; i++) {

printf("%d ", twoStacks->data[i]);

}

printf("\n");

}

int main() {

int size;

printf("Enter the size of the array: ");

scanf("%d", &size);

TwoStacks\* twoStacks = createTwoStacks(size);

int choice, value;

while (1) {

printf("1. Push to Stack 1\n");

printf("2. Push to Stack 2\n");

printf("3. Pop from Stack 1\n");

printf("4. Pop from Stack 2\n");

printf("5. Display Stacks\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter element to push to Stack 1: ");

scanf("%d", &value);

pushStack1(twoStacks, value);

break;

case 2:

printf("Enter element to push to Stack 2: ");

scanf("%d", &value);

pushStack2(twoStacks, value);

break;

case 3:

value = popStack1(twoStacks);

if (value != -1) {

printf("Popped element from Stack 1: %d\n", value);

}

break;

case 4:

value = popStack2(twoStacks);

if (value != -1) {

printf("Popped element from Stack 2: %d\n", value);

}

break;

case 5:

displayStacks(twoStacks);

break;

case 6:

return 0;

default:

printf("Invalid choice\n");

}

}

return 0;

}

**OUTPUT:**

Enter the size of the array: 5

1. Push to Stack 1

2. Push to Stack 2

3. Pop from Stack 1

4. Pop from Stack 2

5. Display Stacks

6. Exit

Enter your choice: 1

Enter element to push to Stack 1: 4

1. Push to Stack 1

2. Push to Stack 2

3. Pop from Stack 1

4. Pop from Stack 2

5. Display Stacks

6. Exit

Enter your choice: 1

Enter element to push to Stack 1: 5

1. Push to Stack 1

2. Push to Stack 2

3. Pop from Stack 1

4. Pop from Stack 2

5. Display Stacks

6. Exit

Enter your choice: 2

Enter element to push to Stack 2: 4

1. Push to Stack 1

2. Push to Stack 2

3. Pop from Stack 1

4. Pop from Stack 2

5. Display Stacks

6. Exit

Enter your choice: 2

Enter element to push to Stack 2: 6

1. Push to Stack 1

2. Push to Stack 2

3. Pop from Stack 1

4. Pop from Stack 2

5. Display Stacks

6. Exit

Enter your choice: 2

Enter element to push to Stack 2: 7

1. Push to Stack 1

2. Push to Stack 2

3. Pop from Stack 1

4. Pop from Stack 2

5. Display Stacks

6. Exit

Enter your choice: 2

Enter element to push to Stack 2: 8

Stack overflow

1. Push to Stack 1

2. Push to Stack 2

3. Pop from Stack 1

4. Pop from Stack 2

5. Display Stacks

6. Exit

Enter your choice: 5

Stack 1: 4 5

Stack 2: 7 6 4

1. Push to Stack 1

2. Push to Stack 2

3. Pop from Stack 1

4. Pop from Stack 2

5. Display Stacks

6. Exit

Enter your choice: 4

Popped element from Stack 2: 7

1. Push to Stack 1

2. Push to Stack 2

3. Pop from Stack 1

4. Pop from Stack 2

5. Display Stacks

6. Exit

Enter your choice: 5

Stack 1: 4 5

Stack 2: 6 4

1. Push to Stack 1

2. Push to Stack 2

3. Pop from Stack 1

4. Pop from Stack 2

5. Display Stacks

6. Exit

Enter your choice: 6

**8.Implement circular queue using linked list?**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

typedef struct {

Node\* front;

Node\* rear;

int size;

} CircularQueue;

void initQueue(CircularQueue\* q) {

q->front = NULL;

q->rear = NULL;

q->size = 0;

}

void enqueue(CircularQueue\* q, int x) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = x;

newNode->next = NULL;

if (q->front == NULL) {

q->front = newNode;

q->rear = newNode;

} else {

q->rear->next = newNode;

q->rear = newNode;

}

q->size++;

}

int dequeue(CircularQueue\* q) {

if (q->front == NULL) {

printf("Queue is empty\n");

return -1;

}

int x = q->front->data;

Node\* temp = q->front;

q->front = q->front->next;

if (q->front == NULL) {

q->rear = NULL;

}

free(temp);

q->size--;

return x;

}

void displayQueue(CircularQueue\* q) {

Node\* temp = q->front;

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->next;

if (temp == q->front) {

break;

}

}

printf("\n");

}

int main() {

CircularQueue q;

initQueue(&q);

int choice, x;

while (1) {

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display Queue\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter element to enqueue: ");

scanf("%d", &x);

enqueue(&q, x);

break;

case 2:

printf("Dequeued element: %d\n", dequeue(&q));

break;

case 3:

displayQueue(&q);

break;

case 4:

return 0;

default:

printf("Invalid choice\n");

}

}

return 0;

}

**OUTPUT:**

1. Enqueue

2. Dequeue

3. Display Queue

4. Exit

Enter your choice: 1

Enter element to enqueue: 6

1. Enqueue

2. Dequeue

3. Display Queue

4. Exit

Enter your choice: 1

Enter element to enqueue: 8

1. Enqueue

2. Dequeue

3. Display Queue

4. Exit

Enter your choice: 3

6 8

1. Enqueue

2. Dequeue

3. Display Queue

4. Exit

Enter your choice: 2

Dequeued element: 6

1. Enqueue

2. Dequeue

3. Display Queue

4. Exit

Enter your choice: 3

8

1. Enqueue

2. Dequeue

3. Display Queue

4. Exit

Enter your choice: 4

**9.Write a function to reverse a singly linked list?**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

typedef struct {

Node\* head;

} LinkedList;

void initList(LinkedList\* list) {

list->head = NULL;

}

void append(LinkedList\* list, int x) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = x;

newNode->next = NULL;

if (list->head == NULL) {

list->head = newNode;

} else {

Node\* temp = list->head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

}

void reverseList(LinkedList\* list) {

Node\* prev = NULL;

Node\* current = list->head;

Node\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

list->head = prev;

}

void displayList(LinkedList\* list) {

Node\* temp = list->head;

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->next;

}

printf("\n");

}

int main() {

LinkedList list;

initList(&list);

int choice, x;

while (1) {

printf("1. Append\n");

printf("2. Reverse List\n");

printf("3. Display List\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter element to append: ");

scanf("%d", &x);

append(&list, x);

break;

case 2:

reverseList(&list);

break;

case 3:

displayList(&list);

break;

case 4:

return 0;

default:

printf("Invalid choice\n");

}

}

return 0;

}

**OUTPUT:**

1. Append

2. Reverse List

3. Display List

4. Exit

Enter your choice: 1

Enter element to append: 6

1. Append

2. Reverse List

3. Display List

4. Exit

Enter your choice: 1

Enter element to append: 7

1. Append

2. Reverse List

3. Display List

4. Exit

Enter your choice: 1

Enter element to append: 5

1. Append

2. Reverse List

3. Display List

4. Exit

Enter your choice: 3

6 7 5

1. Append

2. Reverse List

3. Display List

4. Exit

Enter your choice: 2

1. Append

2. Reverse List

3. Display List

4. Exit

Enter your choice: 3

5 7 6

1. Append

2. Reverse List

3. Display List

4. Exit

Enter your choice: 4

**10.Write a function to find a number in a singly linked list?**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

Node\* createNode(int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

if (!newNode) {

printf("Memory error\n");

return NULL;

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

int findNode(Node\* head, int key) {

Node\* current = head;

int position = 1;

while (current != NULL) {

if (current->data == key) {

return position;

}

current = current->next;

position++;

}

return -1;

}

void printList(Node\* head) {

while (head != NULL) {

printf("%d ", head->data);

head = head->next;

}

printf("\n");

}

int main() {

int n;

printf("Enter the number of elements in the linked list: ");

scanf("%d", &n);

Node\* head = NULL;

Node\* current = NULL;

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

int data;

printf("Enter element %d: ", i + 1);

scanf("%d", &data);

Node\* newNode = createNode(data);

if (head == NULL) {

head = newNode;

current = head;

} else {

current->next = newNode;

current = current->next;

}

}

printf("Linked list: ");

printList(head);

int key;

printf("Enter a value to search for: ");

scanf("%d", &key);

int position = findNode(head, key);

if (position != -1) {

printf("Found %d at position %d\n", key, position);

} else {

printf("%d not found in the list\n", key);

}

return 0;

}

**OUTPUT:**

Enter the number of elements in the linked list: 4

Enter element 1: 1

Enter element 2: 6

Enter element 3: 4

Enter element 4: 8

Linked list: 1 6 4 8

Enter a value to search for: 4

Found 4 at position 3

**11.Write a function to create a linked list where it will insert a given number in the list?**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

Node\* createNode(int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

if (!newNode) {

printf("Memory error\n");

return NULL;

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void insertAtBeginning(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

newNode->next = \*head;

\*head = newNode;

}

}

void insertAtEnd(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

Node\* current = \*head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

}

}

void insertAtPosition(Node\*\* head, int data, int position) {

Node\* newNode = createNode(data);

if (position == 1) {

insertAtBeginning(head, data);

} else {

Node\* current = \*head;

int count = 1;

while (current != NULL && count < position - 1) {

current = current->next;

count++;

}

if (current == NULL) {

printf("Position out of range\n");

return;

}

newNode->next = current->next;

current->next = newNode;

}

}

void printList(Node\* head) {

while (head != NULL) {

printf("%d ", head->data);

head = head->next;

}

printf("\n");

}

int main() {

Node\* head = NULL;

int n;

printf("Enter the number of nodes: ");

scanf("%d", &n);

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

int data;

printf("Enter data for node %d: ", i + 1);

scanf("%d", &data);

insertAtEnd(&head, data);

}

printf("Linked list: ");

printList(head);

int position, data;

printf("Enter a position to insert a new node: ");

scanf("%d", &position);

printf("Enter data for the new node: ");

scanf("%d", &data);

insertAtPosition(&head, data, position);

printf("Linked list after inserting at position %d: ", position);

printList(head);

return 0;

}

**OUTPUT:**

Enter the number of nodes: 5

Enter data for node 1: 6

Enter data for node 2: 4

Enter data for node 3: 8

Enter data for node 4: 9

Enter data for node 5: 3

Linked list: 6 4 8 9 3

Enter a position to insert a new node: 2

Enter data for the new node: 7

Linked list after inserting at position 2: 6 7 4 8 9 3

**12.Print a magic triangle?**

**CODE:**

#include <stdio.h>

int main() {

int rows, i, j, num;

printf("Enter the number of rows: ");

scanf("%d", &rows);

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

num = 1;

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

printf("%4d", num);

num = num \* (i - j) / (j + 1);

}

printf("\n");

}

return 0;

}

**OUTPUT:**

Enter the number of rows: 5

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

**13.Write a program to convert the given binary to decimal?**

**CODE:**

#include <stdio.h>

#include <string.h>

int binaryToDecimal(char \*binary) {

int decimal = 0;

int base = 1;

int len = strlen(binary);

for (int i = len - 1; i >= 0; i--) {

if (binary[i] == '1') {

decimal += base;

}

base \*= 2;

}

return decimal;

}

int main() {

char binary[100];

printf("Enter a binary number: ");

scanf("%s", binary);

for (int i = 0; i < strlen(binary); i++) {

if (binary[i] != '0' && binary[i] != '1') {

printf("Invalid binary number. Please enter only 0s and 1s.\n");

return 1;

}

}

int decimal = binaryToDecimal(binary);

printf("Decimal equivalent: %d\n", decimal);

return 0;

}

**OUTPUT:**

Enter a binary number: 1001

Decimal equivalent: 9

**14.Write a program to implement binary search in a given set of numbers?**

**CODE:**

#include <stdio.h>

int binarySearch(int arr[], int low, int high, int target) {

while (low <= high) {

int mid = (low + high) / 2;

if (arr[mid] == target) {

return mid;

}

if (arr[mid] > target) {

high = mid - 1;

}

else {

low = mid + 1;

}

}

return -1;

}

int main() {

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter the elements in ascending order: \n");

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

scanf("%d", &arr[i]);

}

int target;

printf("Enter the target element to search: ");

scanf("%d", &target);

int result = binarySearch(arr, 0, n - 1, target);

if (result != -1) {

printf("Element found at index %d\n", result);

} else {

printf("Element not found\n");

}

return 0;

}

**OUTPUT:**

Enter the number of elements: 5

Enter the elements in ascending order:

1

5

6

7

8

Enter the target element to search: 5

Element found at index 1

**15.Write a program to find the shortest path for a given graph using all pair shortest path algorithm?**

**CODE:**

#include <stdio.h>

#define MAX\_VERTICES 100

#define INT\_MAX 200

void floydWarshall(int graph[][MAX\_VERTICES], int numVertices) {

int i, j, k;

int distance[MAX\_VERTICES][MAX\_VERTICES];

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

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

distance[i][j] = graph[i][j];

}

}

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

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

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

if (distance[i][k] + distance[k][j] < distance[i][j]) {

distance[i][j] = distance[i][k] + distance[k][j];

}

}

}

}

printf("Shortest distances:\n");

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

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

if (distance[i][j] == INT\_MAX) {

printf("INF ");

} else {

printf("%d ", distance[i][j]);

}

}

printf("\n");

}

printf("Shortest paths:\n");

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

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

if (i == j) {

printf("0 ");

} else if (distance[i][j] == INT\_MAX) {

printf("No path ");

} else {

printf("%d ", distance[i][j]);

}

}

printf("\n");

}

}

int main() {

int numVertices;

printf("Enter the number of vertices: ");

scanf("%d", &numVertices);

int graph[MAX\_VERTICES][MAX\_VERTICES];

printf("Enter the adjacency matrix (%d x %d):\n", numVertices, numVertices);

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

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

printf("Enter weight of edge from %d to %d: ", i, j);

scanf("%d", &graph[i][j]);

if (i != j && graph[i][j] == 0) {

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

}

}

}

floydWarshall(graph, numVertices);

return 0;

}

**OUTPUT:**

Enter the number of vertices: 4

Enter the adjacency matrix (4 x 4):

Enter weight of edge from 0 to 0: 0

Enter weight of edge from 0 to 1: 3

Enter weight of edge from 0 to 2: 8

Enter weight of edge from 0 to 3: 0

Enter weight of edge from 1 to 0: 0

Enter weight of edge from 1 to 1: 0

Enter weight of edge from 1 to 2: 2

Enter weight of edge from 1 to 3: 5

Enter weight of edge from 2 to 0: 0

Enter weight of edge from 2 to 1: 0

Enter weight of edge from 2 to 2: 0

Enter weight of edge from 2 to 3: 1

Enter weight of edge from 3 to 0: 0

Enter weight of edge from 3 to 1: 0

Enter weight of edge from 3 to 2: 0

Enter weight of edge from 3 to 3: 0

Shortest distances:

0 3 5 6

INF 0 2 3

INF INF 0 1

INF INF INF 0

Shortest paths:

0 3 5 6

No path 0 2 3

No path No path 0 1

No path No path No path 0