Lab

GCD

```
#include <stdio.h>

int gcd(int a, int b) {
    if (a == 0)
        return b;
    return gcd(b % a, a);
}

int main() {
    int a, b;
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);
    printf("GCD(%d, %d) = %d\\n", a, b, gcd(a, b));
    return 0;
}
```

```
Enter two integers: 2 4
GCD(2, 4) = 2
```

Fibonacci

```
#include <stdio.h>

void fib(int a, int b, int sum, int N) {
    if (N != 0) {
        printf(" %d", a);
        sum = a + b;
        a = b;
        b = sum;
        N--;
        fib(a, b, sum, N);
```

```
    }
}

int main() {
    int N;
    printf("Enter the number of terms: ");
    scanf("%d", &N);
    printf("Fibonacci series of %d terms:", N);
    fib(0, 1, 0, N);
    return 0;
}
```

```
Enter the number of terms: 4
Fibonacci series of 4 terms: 0 1 1 2
```

Sum of elements in Array

```
#include <stdio.h>
#include <stdlib.h>

int sum(int arr[], int n) {
    if (n == 0) {
        return 0;
    } else {
        return arr[0] + sum(arr + 1, n - 1);
    }
}

int main() {
    int n;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);
    int *arr = (int *)malloc(n * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return 1;
```

```
    }
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Sum of the elements of the array: %d \n", sum(a
rr, n));
    free(arr);
    return 0;
}
```

```
Enter the number of elements in the array: 4
Enter the elements of the array:
2 4 6 8
Sum of the elements of the array: 20
```

Warshal's Algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 4 // Size of the graph

void printMatrix(int graph[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", graph[i][j]);
        }
        printf("\n");
    }
}

void warshall(int graph[N][N]) {
    for (int k = 0; k < N; k++) {
        for (int i = 0; i < N; i++) {
```

```
            for (int j = 0; j < N; j++) {
                graph[i][j] = graph[i][j] || (graph[i][k] &
& graph[k][j]);
            }
        }
    }
}

int main() {
    int graph[N][N] = {
        {1, 0, 1, 0},
        {1, 1, 0, 0},
        {0, 1, 0, 1},
        {0, 0, 1, 1}
    };

    printf("Original Graph: \n");
    printMatrix(graph);

    warshall(graph);

    printf("\nTransitive Closure:\n");
    printMatrix(graph);

    return 0;
}
```

```
Original Graph:
1 0 1 0
1 1 0 0
0 1 0 1
0 0 1 1

Transitive Closure:
1 1 1 1
1 1 1 1
```

```
1 1 1 1
1 1 1 1
```

String Matching

```
#include <stdio.h>
#include <string.h>

void search(char* pat, char* txt) {
    int M = strlen(pat);
    int N = strlen(txt);

    // Slide the pattern over the text one by one
    for (int i = 0; i <= N - M; i++) {
        int j;

        // Check if the pattern matches the substring of te
xt
        for (j = 0; j < M; j++)
            if (txt[i + j] != pat[j])
                break;

        // If the pattern is found, print the index
        if (j == M)
            printf("Pattern found at index %d \n", i);
    }
}

int main() {
    char txt[] = "AABAACAADAABAAABAA";
    char pat[] = "AABA";

    // Call the search function
    search(pat, txt);
```

```
    return 0;
}
```

```
Pattern found at index 0
Pattern found at index 9
Pattern found at index 13
```

Dijkstra's

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>

#define V 9 // Number of vertices in the graph

int minDistance(int dist[], bool sptSet[]) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (!sptSet[v] && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

void printSolution(int dist[]) {
    printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t\t %d\n", i, dist[i]);
}

void dijkstra(int graph[V][V], int src) {
    int dist[V];    // Holds the shortest distance from src
to i
    bool sptSet[V]; // True if vertex i is included in the
shortest path tree
```

```
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet);
        sptSet[u] = true;

        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v] && dist[u] != INT
_MAX && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    printSolution(dist);
}

int main() {
    int graph[V][V] = {
        {0, 4, 0, 0, 0, 0, 0, 8, 0},
        {4, 0, 8, 0, 0, 0, 0, 11, 0},
        {0, 8, 0, 7, 0, 4, 0, 0, 2},
        {0, 0, 7, 0, 9, 14, 0, 0, 0},
        {0, 0, 0, 9, 0, 10, 0, 0, 0},
        {0, 0, 4, 14, 10, 0, 2, 0, 0},
        {0, 0, 0, 0, 0, 2, 0, 1, 6},
        {8, 11, 0, 0, 0, 0, 1, 0, 7},
        {0, 0, 2, 0, 0, 0, 6, 7, 0}
    };

    dijkstra(graph, 0); // Starting from vertex 0

    return 0;
}
```

```
Vertex  Distance from Source
0            0
1            4
2            12
3            19
4            21
5            11
6            9
7            8
8            14
```

Prim’s

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>

#define V 5 // Number of vertices in the graph

int minKey(int key[], bool mstSet[]) {
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;
    return min_index;
}

void printMST(int parent[], int graph[V][V]) {
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d \t%d \n", parent[i], i, graph[i][pa
rent[i]]);
}

void primMST(int graph[V][V]) {
    int parent[V]; // Array to store constructed MST
```

```
    int key[V];    // Key values used to pick minimum weigh
t edge in cut
    bool mstSet[V]; // To represent the set of vertices inc
luded in MST

    // Initialize all keys as INFINITE and mstSet[] as fals
e
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    // Always include the first vertex in MST
    key[0] = 0; // Make key 0 so that this vertex is picked
as the first vertex
    parent[0] = -1; // First node is always the root of MST

    // The MST will have V vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum key vertex from the set of vert
ices not yet processed
        int u = minKey(key, mstSet);
        mstSet[u] = true; // Add the picked vertex to the M
ST

        // Update key value and parent index of the adjacen
t vertices of the picked vertex
        for (int v = 0; v < V; v++)
            // Update the key only if graph[u][v] is smalle
r than key[v]
            if (graph[u][v] && mstSet[v] == false && graph
[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
    }

    // Print the constructed MST
    printMST(parent, graph);
}
```

```
int main() {
    // Example graph represented as an adjacency matrix
    int graph[V][V] = {
        {0, 2, 0, 6, 0},
        {2, 0, 3, 8, 5},
        {0, 3, 0, 0, 7},
        {6, 8, 0, 0, 9},
        {0, 5, 7, 9, 0}
    };

    primMST(graph); // Call the function to find the MST
    return 0;
}
```

```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```