# Code for Rabin Miller Primality Test

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

long long modular_pow(long long base, long long exp, long long mod) {
    long long result = 1;
    base = base % mod;
    while (exp > 0) {
        if (exp % 2 == 1)
            result = (result * base) % mod;
        exp = exp >> 1;
        base = (base * base) % mod;
    }
    return result;
}

// Rabin-Miller primality test
int is_prime(int n, int k) {
    if (n <= 1 || n == 4) return 0;
    if (n <= 3) return 1;
    int d = n - 1;
    while (d % 2 == 0) d /= 2;

    for (int i = 0; i < k; i++) {
        int a = 2 + rand() % (n - 4);
        long long x = modular_pow(a, d, n);

        if (x == 1 || x == n - 1)
            continue;

        int is_composite = 1;
        while (d != n - 1) {
            x = (x * x) % n;
            d *= 2;
            if (x == 1) return 0;
            if (x == n - 1) {
                is_composite = 0;
                break;
            }
        }
        if (is_composite) return 0;
    }
    return 1;
}

int main() {
    int n, k = 5;
    printf("Enter the number to test: ");
    scanf("%d", &n);

    if (is_prime(n, k))
        printf("%d is a prime number.\n", n);
    else
        printf("%d is not a prime number.\n", n);

    return 0;
}
```

**Output:**

```
Enter the number to test: 11
11 is a prime number.
```

```
Enter the number to test: 15
15 is not a prime number.
```

# Code for Traveling Salesperson Problem(np complete problem)

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

#define V 4
#define INF INT_MAX

int tsp(int graph[V][V], int visited[], int currPos, int n, int count, int cost, int *ans) {
    if (count == n && graph[currPos][0]) {
        *ans = cost+graph[currPos][0]<*ans ? cost+graph[currPos][0] : *ans;
        return *ans;
    }

    for (int i = 0; i < n; i++) {
        if (!visited[i] && graph[currPos][i]) {
            visited[i] = 1;
            tsp(graph, visited, i, n, count+1, cost+graph[currPos][i], ans);
            visited[i] = 0;
        }
    }
    return *ans;
}

int main() {
    int graph[V][V] = {
        {0, 10, 15, 20},
        {10, 0, 35, 25},
        {15, 35, 0, 30},
        {20, 25, 30, 0}
    };

    int visited[V];
    for (int i = 0; i < V; i++)
        visited[i] = 0;

    visited[0] = 1;
    int ans = INF;

    ans = tsp(graph, visited, 0, V, 1, 0, &ans);

    printf("The minimum cost is %d\n", ans);
    return 0;
}
```

# Output

```
The minimum cost is 80
```

# Code for Ford Fulkerson Algorithm

```c
#include <stdio.h>
#include <string.h>
#include <limits.h>
#define MAX 100

int dfs(int rGraph[MAX][MAX], int source, int sink, int parent[], int V) {
    int visited[MAX] = {0};
    int stack[MAX], top = -1;

    stack[++top] = source;
    visited[source] = 1;
    parent[source] = -1;

    while (top >= 0) {
        int u = stack[top--];
        for (int v = 0; v < V; v++) {
            if (!visited[v] && rGraph[u][v] > 0) {
                stack[++top] = v;
                parent[v] = u;
                visited[v] = 1;

                if (v == sink)
                    return 1;
            }
        }
    }
    return 0;
}

int fordFulkerson(int graph[MAX][MAX], int source, int sink, int V) {
    int rGraph[MAX][MAX], u, v;

    for (u = 0; u < V; u++)
        for (v = 0; v < V; v++)
            rGraph[u][v] = graph[u][v];

    int parent[MAX];
    int max_flow = 0;

    while (dfs(rGraph, source, sink, parent, V)) {
        int path_flow = INT_MAX;


        for (v = sink; v != source; v = parent[v]) {
            u = parent[v];
            path_flow = (path_flow < rGraph[u][v]) ? path_flow : rGraph[u][v];
        }

        for (v = sink; v != source; v = parent[v]) {
            u = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }

        max_flow += path_flow;
    }

    return max_flow;
```

```
}

int main() {
    int V, graph[MAX][MAX];
    printf("Enter the number of vertices: ");
    scanf("%d", &V);

    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            scanf("%d", &graph[i][j]);

    int source, sink;
    printf("Enter the source and sink vertices: ");
    scanf("%d %d", &source, &sink);

    printf("The maximum possible flow is: %d\n", fordFulkerson(graph, source, sink, V));

    return 0;
}
```

# Output:

```
Enter the number of vertices: 4
Enter the adjacency matrix:
0 10  5  0
0  0 15 10
0  0  0 10
0  0  0  0
Enter the source and sink vertices: 0 3
The maximum possible flow is: 15
```