1. Floyd's Algorithm

```
#include <iostream>
using namespace std;

#define INF 99999
#define V 4

int main() {
    int dist[V][V] = {
        {0,   5,   INF, 10},
        {INF, 0,   3,   INF},
        {INF, INF, 0,   1},
        {INF, INF, INF, 0}
    };

    // Floyd's Algorithm
    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }

    cout << "Shortest Path Matrix:\n";
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INF)
                cout << "INF ";
            else
                cout << dist[i][j] << " ";
        }
        cout << endl;
```

```
    }

    return 0;
}
```

2. Warshall's Algorithm

```
#include <iostream>
using namespace std;

#define V 4

int main() {
    int reach[V][V] = {
        {0, 1, 0, 0},
        {0, 0, 1, 0},
        {0, 0, 0, 1},
        {0, 0, 0, 0}
    };

    // Warshall's Algorithm
    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                reach[i][j] = reach[i][j] || (reach[i][k] && reach[k][j]);
            }
        }
    }

    cout << "Transitive Closure Matrix:\n";
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            cout << reach[i][j] << " ";
        }
        cout << endl;
    }
```

```
    return 0;
}
```

3. Dijkstra's Algorithm

```
#include <iostream>
using namespace std;

#define V 5
#define INF 99999

int main() {
    int graph[V][V] = {
        {0,   10,  INF, 30, 100},
        {INF, 0,   50,  INF, INF},
        {INF, INF, 0,   INF, 10},
        {INF, INF, 20,  0,   60},
        {INF, INF, INF, INF, 0}
    };

    int dist[V], visited[V] = {0};

    // Initialize distances
    for (int i = 0; i < V; i++)
        dist[i] = INF;
    dist[0] = 0;   // Source

    // Dijkstra's Algorithm
    for (int count = 0; count < V - 1; count++) {
        int u = -1, min = INF;

        // Find unvisited vertex with minimum distance
        for (int i = 0; i < V; i++) {
            if (!visited[i] && dist[i] < min) {
                min = dist[i];
```

```
            u = i;
        }
    }

    visited[u] = 1;

    // Relax edges
    for (int v = 0; v < V; v++) {
        if (!visited[v] && graph[u][v] != INF &&
            dist[u] + graph[u][v] < dist[v]) {
            dist[v] = dist[u] + graph[u][v];
        }
    }
}

cout << "Shortest distances from source 0:\n";
for (int i = 0; i < V; i++)
    cout << "0 -> " << i << " = " << dist[i] << endl;

return 0;
}
```

4. Brute Force String Search

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string text, pattern;

    cout << "Enter text: ";
    cin >> text;

    cout << "Enter pattern: ";
    cin >> pattern;
```

```
    int n = text.length();
    int m = pattern.length();

    cout << "Pattern found at positions: ";

    for (int i = 0; i <= n - m; i++) {
        int j;
        for (j = 0; j < m; j++) {
            if (text[i + j] != pattern[j])
                break;
        }
        if (j == m) {
            cout << i << " ";
        }
    }

    return 0;
}
```