Computer And Communication Network
ASSIGNMENT - 1

Name : Alkesh shukla
Roll no : S20210020252

---

1. Write code to implement the IPv4 fragmentation by taking inputs of payload and MTU details from the user. Each fragment must take the shortest path following Dijkstra's Algorithm in the considered topology between any source and destination pairs of userchoice. Below points must be considered for realizing the code.
a) Fragmentation and Dijkstra’s algorithm must work for any topology and you may
be asked to run code on a custom topology during evaluation.
b) Source and destination nodes must be taken as user inputs.
c) Shortest path must be calculated before sending every fragmentation to the
destination assuming that there may be topology changes in the network. Also,
each fragment should take the current shortest path from source to destination.

C++ code :

```
#include <iostream>
#include <vector>
#include <limits>
#include <cstring>

using namespace std;

const int INFINITY = INT_MAX;
const int IP_header_size = 20;
vector<int> dijkstra_algorithm(vector<vector<int>> accepting_graph, int
accepting_start_vertex)
```

```
{
    int numberOfVertex = accepting_graph.size();
    vector<int> shortestDistance(numberOfVertex, INFINITY);
    vector<bool> node_visited(numberOfVertex, false);
    shortestDistance[accepting_start_vertex] = 0;
    for (int i = 0; i < numberOfVertex - 1; i++)
    {
        int u = -1;
        for (int j = 0; j < numberOfVertex; j++)
        {
            if (!node_visited[j] && (u == -1 || shortestDistance[j] <
shortestDistance[u]))
            {
                u = j;
            }
        }
        node_visited[u] = true;
        for (int v = 0; v < numberOfVertex; v++)
        {
            if (accepting_graph[u][v] != 0 && !node_visited[v])
            {
                shortestDistance[v] = min(shortestDistance[v],
shortestDistance[u] + accepting_graph[u][v]);
            }
        }
    }
    return shortestDistance;
}

int calculate_num_fragment(int accepting_payload_size, int accepting_mtu)
{
    int payload_size_max = accepting_mtu - IP_header_size;
    int noOfFragments = accepting_payload_size / payload_size_max;
    if (accepting_payload_size % payload_size_max != 0)
    {
        noOfFragments++;
    }
    return noOfFragments;
}
int noOfFragments;
```

```
vector<int> payload_fragments(int accepting_payload, int accepting_mtu)
{
    vector<int> fragments;
    int payload_size_max = accepting_mtu - IP_header_size;
    int actual_payload_size = accepting_payload - IP_header_size;
    noOfFragments = calculate_num_fragment(accepting_payload,
accepting_mtu);
    int actual_value = actual_payload_size;
    while (actual_value > payload_size_max)
    {
        actual_value = actual_value - payload_size_max;

        fragments.push_back(accepting_mtu);
    }
    fragments.push_back(actual_value + IP_header_size);
    return fragments;
}

int main()
{
    int accepting_payload;
    cout << "Enter the accepting_payload : ";
    cin >> accepting_payload;
    int accepting_mtu;
    cout << "Enter the Maximum transmission unit (MTU) : ";
    cin >> accepting_mtu;
    vector<int> fragments = payload_fragments(accepting_payload,
accepting_mtu);
    // fragmentation detail that how many fragment hace and each fragment
    // how many byte data can be transfer from sourcse to destination
    for (int i = 0; i < fragments.size(); i++)
    {
        cout<<endl;
        int frg = i+1;
        cout << "Fragment " << i + 1 << ": " << fragments[i] << " bytes"
<< endl;
        int numberOfVertex, numOfEdges,
accepting_start_vertex,destination;
        cout<<"Enter the Network topology and Source and Destination of
"<< i+1<< " fragment "<<endl;
```

```
        cout << "Enter the Node,Edges,starting point and destination point
:" << endl;
        cin >> numberOfVertex >> numOfEdges >> accepting_start_vertex >>
destination;
        vector<vector<int>> accepting_graph(numberOfVertex,
vector<int>(numberOfVertex, 0));
        for (int i = 0; i < numOfEdges; i++)
        {
            int u, v, w;
            cout << "Enter the starting_point,end_point and cost " << i +
1 << " edge : ";
            cin >> u >> v >> w;
            accepting_graph[u][v] = w;
            accepting_graph[v][u] = w;
        }
        vector<int> shortestDistance = dijkstra_algorithm(accepting_graph,
accepting_start_vertex);
        // dijkstra_algorithm algorithm gives the
        int flag = 0;
        for (int i = 0; i < numberOfVertex; i++)
        {
            if (shortestDistance[i] == INFINITY || shortestDistance[i] <
0)
            {
                flag =1;
                break;
            }
            else if ( i == destination)
            {
                cout << "Shortest path from " << accepting_start_vertex <<
" to " << i << " of " <<frg<<" fragment is " << shortestDistance[i] <<
endl;
            }
        }
        if(flag ==1){
                cout << "No path from source to destination exists to
travel through them."<< endl;
        }
    }
    return 0;
```

```
}
```

Result :

```
PS C:\Users\abc\OneDrive\Documents\sem-4\CCN> cd "c:\Users\abc\OneDrive\Documents\sem-4\CCN\CCN_ASSIGNMENT_1\" ; if ($?) { g++ S2021002
0252__Alkesh_Shukla_CCN_Assignment_1.cpp -o S20210020252__Alkesh_Shukla_CCN_Assignment_1 } ; if ($?) { .\S20210020252__Alkesh_Shukla_CC
N_Assignment_1 }
Enter the accepting_payload : 4000
Enter the Maximum transmission unit (MTU) : 1500

Fragment 1: 1500 bytes
Enter the Network topology and Source and Destination of 1 fragment
Enter the Node,Edges,starting point and destination point :
6 8 0 5
Enter the starting_point,end_point and cost 1 edge : 0 1 7
Enter the starting_point,end_point and cost 2 edge : 0 2 12
Enter the starting_point,end_point and cost 3 edge : 1 2 2
Enter the starting_point,end_point and cost 4 edge : 1 3 9
Enter the starting_point,end_point and cost 5 edge : 2 4 10
Enter the starting_point,end_point and cost 6 edge : 3 4 4
Enter the starting_point,end_point and cost 7 edge : 3 5 1
Enter the starting_point,end_point and cost 8 edge : 4 5 5
Shortest path from 0 to 5 of 1 fragment is 17

Fragment 2: 1500 bytes
Enter the Network topology and Source and Destination of 2 fragment
Enter the Node,Edges,starting point and destination point :
6 6 0 5
Enter the starting_point,end_point and cost 1 edge : 1 2 2
Enter the starting_point,end_point and cost 2 edge : 1 3 9
Enter the starting_point,end_point and cost 3 edge : 2 4 10
Enter the starting_point,end_point and cost 4 edge : 3 4 4
Enter the starting_point,end_point and cost 5 edge : 3 5 1
Enter the starting_point,end_point and cost 6 edge : 4 5 5
No path from source to destination exists to travel through them.
```

```
Fragment 3: 1040 bytes
Enter the Network topology and Source and Destination of 3 fragment
Enter the Node,Edges,starting point and destination point :
6 8 2 5
Enter the starting_point,end_point and cost 1 edge : 0 1 7
Enter the starting_point,end_point and cost 2 edge : 0 2 12
Enter the starting_point,end_point and cost 3 edge : 1 2 2
Enter the starting_point,end_point and cost 4 edge : 1 3 9
Enter the starting_point,end_point and cost 5 edge : 2 4 10
Enter the starting_point,end_point and cost 6 edge : 3 4 4
Enter the starting_point,end_point and cost 7 edge : 3 5 1
Enter the starting_point,end_point and cost 8 edge : 4 5 5
Shortest path from 2 to 5 of 3 fragment is 12
PS C:\Users\abc\OneDrive\Documents\sem-4\CCN\CCN_ASSIGNMENT_1>
```