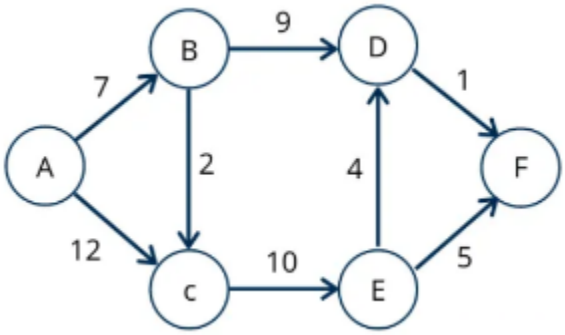
CCN Assignment

Name: Tushar Roll No.: S20210010231

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 user choice. 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.



CPP Code:

```cpp
#include <iostream>
#include <vector>
#include <set>
#include <limits>
#include <algorithm>
#include <string>
#include <iomanip>
using namespace std;

const int infinity = numeric_limits<int>::max();

class Fragment
```

```
{
    public:
        string fragmentNumber;
        int fragLength;
        int offset;
        int fragFlag;
        string fragId;

        Fragment(string fragnum, int length, int frag_offset, int
flag, string id)
        {
            fragmentNumber = fragnum;
            fragLength = length;
            offset = frag_offset;
            fragFlag = flag;
            fragId = id;
        }
};

void printFragmentDetails(vector<Fragment> fragmentDetails){

cout<<"<-------------------------------------------------FRAGMENT
DETAILS------------------------------------------------>"<<endl;
    cout
    << left
    << setw(30)
    << "Fragment Number"
    << left
    << setw(30)
    << "Fragment Length"
    << left
    << setw(20)
    << "Id"
    << left
    << setw(20)
    << "Offset"
    << left
    << setw(20)
```

```
    << "Flag"
    << endl;
    for(int i=0;i<fragmentDetails.size();i++){
        cout
        << left
        << setw(30)
        << fragmentDetails[i].fragmentNumber
        << left
        << setw(30)
        << fragmentDetails[i].fragLength
        << left
        << setw(20)
        << fragmentDetails[i].fragId
        << left
        << setw(20)
        << fragmentDetails[i].offset
        << left
        << setw(20)
        << fragmentDetails[i].fragFlag
        << endl;
    }

cout<<"<----------------------------------------------------------------
-------------------------------------------------->"<<endl;
}

vector<Fragment> creatingFragmentDetails(vector<int> fragment){
    vector<Fragment> fragmentDetails;
    string id = "X";
    int fragSum = 0;
    for(int i = 0; i<fragment.size();i++){
        string fragNum = "Fragment " + to_string(i+1);
        int offset = fragSum / 8;
        int flag = 1;
        if(i==fragment.size()-1){
            flag = 0;
        }
        int length = fragment[i];
```

```
fragmentDetails.push_back(Fragment(fragNum,length,offset,flag,id));
        fragSum+=fragment[i];
    }
    return fragmentDetails;
}

vector<int> algorithm_dijkstra_application(vector<vector<pair<int,
int>>> input_graph, int index_starting) {
    int n = input_graph.size();
    vector<int> shortest_distance_calculated(n, infinity);
    set<pair<int, int>> node_unvisited;
    shortest_distance_calculated[index_starting] = 0;
    node_unvisited.insert({0, index_starting});
    while (!node_unvisited.empty()) {
        int index_U = node_unvisited.begin()->second;
        node_unvisited.erase(node_unvisited.begin());
        for (auto graph_edge : input_graph[index_U]) {
            int edge_first_vertex = graph_edge.first;
            int edge_weight = graph_edge.second;
            if (shortest_distance_calculated[index_U] + edge_weight <
shortest_distance_calculated[edge_first_vertex]) {

node_unvisited.erase({shortest_distance_calculated[edge_first_vertex]
, edge_first_vertex});
                shortest_distance_calculated[edge_first_vertex] =
shortest_distance_calculated[index_U] + edge_weight;

node_unvisited.insert({shortest_distance_calculated[edge_first_vertex
], edge_first_vertex});
            }
        }
    }
    return shortest_distance_calculated;
}
int findingNumFragments;
vector<int> creatingSetOfFragments(int input_payload, int input_mtu)
{
```

```
     int payloadAfterRemovingHeaderSize = input_payload-20;
    int mtuAfterRemovingHeaderSize = input_mtu-20;
    findingNumFragments = input_payload/mtuAfterRemovingHeaderSize;
    if(input_payload % mtuAfterRemovingHeaderSize != 0){
        findingNumFragments++;
    }
    cout<<"Total number of fragments : "<< findingNumFragments<<endl;
    int j=0;
    vector<int> vectorOfFragments(findingNumFragments);
    for (int i = 0; i < findingNumFragments; i++){
        while(payloadAfterRemovingHeaderSize >
mtuAfterRemovingHeaderSize){
            payloadAfterRemovingHeaderSize =
payloadAfterRemovingHeaderSize-mtuAfterRemovingHeaderSize;
            vectorOfFragments[j]=input_mtu;
            j++;
        }
        vectorOfFragments[j] = payloadAfterRemovingHeaderSize+20;
    }
    return vectorOfFragments;
}

int main() {
   int payload_Size;
    cout << "Note: The IP_HEADER_SIZE of 20 has been included in the
Payload and MTU. \n";
    cout << "Enter the initial payload size : ";
    cin>>payload_Size;
    int input_mtu;
    cout << "Enter maximum transmitting unit(MTU) : ";
    cin >> input_mtu;
    vector<int> fragment = creatingSetOfFragments(payload_Size,
input_mtu);
    vector<Fragment> fragmentDetails =
creatingFragmentDetails(fragment);
    printFragmentDetails(fragmentDetails);
    vector<int> ints;
    for (int i = 0; i < fragment.size(); i++) {
```

```
        cout << "Fragment " << i+1 << ": " << fragment[i] << endl;
        cout<<"Enter the number of vertices and no of edges in the
graph. For example (6 8):";
        int vertices, graph_edge;
        cin >> vertices >> graph_edge;
        int index_starting,dest;
        cout<<"Enter the starting and end points. For example (0 5):
";
        cin>> index_starting >> dest;
        vector<vector<pair<int, int>>> input_graph(vertices);
        cout<<"Enter source,destination,weight. For example (0 1
7)"<<endl;
        for (int i = 0; i < graph_edge; i++) {
            int x, y, z;
            cin >> x >> y >> z;
            input_graph[x].push_back({y, z});
        }
        vector<int> dist =
algorithm_dijkstra_application(input_graph, index_starting);
        int flag = 0;
        int temp;
        for (temp = 0; temp < vertices; temp++) {
            if (dist[temp] == infinity && temp == dest) {
                flag=1;
                break;
            }
            else if(temp == dest) {
                cout << "Shortest path from " << index_starting << "
to " << temp << " is " << dist[temp] << endl;
            }
        }
        if(flag == 1){
            cout << "No path from source to destination exists" <<
endl;
        }

    }
```

```
    return 0;
}
// 0 1 7
// 0 2 12
// 1 2 2
// 1 3 9
// 2 4 10
// 4 3 4
// 3 5 1
// 4 5 5
```

Output:

```
D:\Tushar\desktop\code>cd "d:\Tushar\desktop\code\ccn\" && g++ S20210010231_Tushar_CCN_Assignment.cpp -o S20210010231_Tushar_CCN_Assignment && "d:\Tushar\desktop\code\ccn\"S2021
0010231_Tushar_CCN_Assignment
Note: The IP_HEADER_SIZE of 20 has been included in the Payload and MTU.
Enter the initial payload size : 4000
Enter maximum transmitting unit(MTU) : 1500
Total number of fragments : 3
<----------------------------------------------------FRAGMENT DETAILS---------------------------------------------->
Fragment Number             Fragment Length             Id                Offset            Flag
Fragment 1                  1500                        X                 0                 1
Fragment 2                  1500                        X                 187               1
Fragment 3                  1040                        X                 375               0
<------------------------------------------------------------------------------------------------------------------>
Fragment 1: 1500
Enter the number of vertices and no of edges in the graph. For example (6 8):6 8
Enter the starting and end points. For example (0 5): 0 5
Enter source,destination,weight. For example (0 1 7)
0 1 7
0 2 12
1 2 2
1 3 9
2 4 10
4 3 4
4 5 5
3 5 1
Shortest path from 0 to 5 is 17
Fragment 2: 1500
Enter the number of vertices and no of edges in the graph. For example (6 8):6 6
Enter the starting and end points. For example (0 5): 0 5
Enter source,destination,weight. For example (0 1 7)
1 2 2
1 3 9
2 4 10
4 3 4
4 5 5
3 5 1
No path from source to destination exists
```

```
Fragment 3: 1040
Enter the number of vertices and no of edges in the graph. For example (6 8):6 8
Enter the starting and end points. For example (0 5): 2 5
Enter source,destination,weight. For example (0 1 7)
0 1 7
0 2 12
1 2 2
1 3 9
2 4 10
4 3 4
4 5 5
3 5 1
Shortest path from 2 to 5 is 15
```

Code Explanation:

In this code, I have created a class called Fragment which has got all the required fields for a fragment.
First of all with the help of payload and mtu, I am creating a set of fragments which is then passed into createFragmentDetails to list the details of all the fragments. After that printFragmentDetails will print all the fragments. Then asking for the network topology and passing that topology to the algorithm_dijkstra_application to find the shortest path from source to destination.
The above graph has been used as an example and its corresponding output has been shown above.