Chapter 10 - exercise 2

•Assume we assign a group of vertices to a processor and send the edge/weight information for those vertices.
find a vertex u such that
d[u] = min{d[v]|v in V-VT}
•Assume each processor knows if its vertex is in V-VT
•Using the code from Chp 10 Exercise 1, write code so each process now computes the min.
•You will need to add a weight to each edge.
•You will need a way to flag if an edge is in V-VT
•You will need to assign the initial d[v] values.
•You will need a way to combine the results to determine the final u.

Graded Rubric
_____ (1 Points) Able to flag V-VT
_____ (1 Points) Able to set initial d[v] values
_____ (2 Points) Able to send all data to each processor
_____ (1 Points) Added a weight to each edge
_____ (1 Points) Able to compute local d[u]
_____ (2 Points) Able to combine local d[u] to determine actual value for u
_____ (1 Points) Test function that ensures code is working

Answer:

```
/*
so from the exercise 1
I need to add weight to each edge
a way to flag vertex if edge is in V-VT
able to assign initial value to d[u]
a way to combine the the results something like send and
recieve as well not bcasting

just an idea implement serial version for 24 to know if its
correct or not initially
then you can pump it up to larger scale

hint hint should be node 4

 */
#include
<mpi.h>
// Include MPI library
```

```
#include
<iostream>
// Include iostream for input/output
#include
<vector>
// Include vector for dynamic arrays
#include
<limits>
// Include limits for numeric limits

using namespace
std;
// Use the standard namespace

class Vertex
{
// Define Vertex class
public:
    int
value;
// Value of the vertex
    vector<pair<int, int> >
edges;
// Edges now store a pair of (neighbor, weight)
    bool
in_VT;
// Flag indicating if vertex is in V-VT

    Vertex(int val) : value(val), in_VT(false)
{}                                                  //
Constructor initializing value and in_VT
};

class Graph
{
// Define Graph class
public:
    vector<Vertex>
vertices;
// Vector of vertices

    Graph(int n)
{
// Constructor initializing n vertices
```

```
        for (int i = 0; i < n; ++i)
{
// Loop to create vertices

vertices.emplace_back(i);
// Add vertex to the vector
        }
    }

    void add_edge(int u, int v, int weight)
{                                                          //
Function to add an edge
        vertices[u].edges.emplace_back(v,
weight);                                                     //
Add edge to vertex u
    }

    const Vertex& get_vertex(int i) const
{                                                            //
Function to get a vertex
        return
vertices[i];
// Return vertex at index i
    }

    void print_graph() const
{
// Function to print the graph
        cout << "Adjacency List with Weights:
\n";                                                    // Print
header
        for (const auto& vertex : vertices)
{                                                           // Loop
through vertices
            cout << "Vertex " << vertex.value << " (in V-VT: "
<< vertex.in_VT << "): ";            // Print vertex info
            for (const auto& edge : vertex.edges)
{                                                     // Loop
through edges
                cout << "(" << edge.first << ", " <<
edge.second << ") ";                             // Print edge
info
            }
            cout <<
```

```
endl;
// New line
        }
    }
};

int main()
{
// Main function
    MPI_Init(NULL,
NULL);
// Initialize MPI

    int world_rank,
world_size;
// Variables for rank and size
    MPI_Comm_rank(MPI_COMM_WORLD,
&world_rank);
// Get rank of the process
    MPI_Comm_size(MPI_COMM_WORLD,
&world_size);
// Get number of processes

    const int n =
8;
// Number of vertices
    Graph
g(n);
// Create graph with n vertices

    if (world_rank == 0)
{
// If master process

// Initialize graph with edges and weights
        g.add_edge(0, 1,
3);
// Add edge from 0 to 1 with weight 3
        g.add_edge(0, 2,
1);
// Add edge from 0 to 2 with weight 1
        g.add_edge(1, 4,
7);
// Add edge from 1 to 4 with weight 7
```

```
        g.add_edge(1, 5,
2);
// Add edge from 1 to 5 with weight 2
        g.add_edge(2, 6,
4);
// Add edge from 2 to 6 with weight 4
        g.add_edge(2, 7,
6);
// Add edge from 2 to 7 with weight 6


// Initialize in_VT and d[v] values
        for (int i = 0; i < n; ++i)
{
// Loop through vertices
            g.vertices[i].in_VT = (i % 2 ==
0);                                                  //
Example: Even vertices are in V-VT
        }


// Send edge data to other processes
        for (int i = 1; i < world_size; ++i)
{                                                   // Loop
through processes
            int num_edges =
g.get_vertex(i).edges.size();
// Get number of edges
            MPI_Send(&num_edges, 1, MPI_INT, i, 0,
MPI_COMM_WORLD);                                    // Send number
of edges
            for (const auto& edge : g.get_vertex(i).edges)
{                                       // Loop through edges
                int data[2] = {edge.first,
edge.second};                                        //
Create data array
                MPI_Send(data, 2, MPI_INT, i, 0,
MPI_COMM_WORLD);                                    // Send edge
data
            }
            MPI_Send(&g.vertices[i].in_VT, 1, MPI_C_BOOL, i, 0,
MPI_COMM_WORLD);                    // Send in_VT flag
        }
    } else
```

```
{
// If worker process
        int
num_edges;
// Variable for number of edges
        MPI_Recv(&num_edges, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);                  // Receive number of edges

g.vertices[world_rank].edges.resize(num_edges);
// Resize edges vector
        for (int i = 0; i < num_edges; ++i)
{                                                          // Loop
through edges
            int
data[2];
// Data array
            MPI_Recv(data, 2, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);                    // Receive edge data
            g.vertices[world_rank].edges[i] =
make_pair(data[0], data[1]);                           // Assign
edge data
        }
        MPI_Recv(&g.vertices[world_rank].in_VT, 1, MPI_C_BOOL,
0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE); // Receive in_VT flag
    }


// Compute local minimum d[u]
    int local_min =
numeric_limits<int>::max();
// Initialize local minimum
    int local_u =
-1;
// Initialize local vertex

    if (g.vertices[world_rank].in_VT)
{                                                             /
/ If vertex is in V-VT
        for (const auto& edge : g.get_vertex(world_rank).edges)
{                                    // Loop through edges
            if (edge.second < local_min)
{                                                          //
If edge weight is less than local minimum
                local_min =
```

```
edge.second;
// Update local minimum
                local_u =
g.vertices[world_rank].value;
// Update local vertex
            }
        }
    }


// Reduce to find the global minimum d[u]
    int
global_min;
// Variable for global minimum
    MPI_Allreduce(&local_min, &global_min, 1, MPI_INT, MPI_MIN,
MPI_COMM_WORLD);                    // Reduce to find global
minimum

    if (world_rank == 0)
{
// If master process
        cout << "Global minimum d[u]: " << global_min <<
endl;                                   // Print global
minimum
    }


MPI_Finalize();
// Finalize MPI
    return
0;
// Return success
}
```

Goal	Implementation
1. Add weights to edges	Weights are added using add_edge(u, v, weight). Each edge has a specific cost or distance.
2. Flag edges in V-VT	in_VT boolean flag determines if a vertex is in the V-VT subset.

3. Assign initial d[v] **values**	Each vertex's d[v] is initialized to numeric_limits<int>::max().
4. Combine results for final u	Use MPI_Allreduce to find the global minimum d[u].