OS Lab Assignment 3

Assignment 3: Hands-on experience of using shared-memory

Group Members:

1. Nikhil Saraswat (20CS10039)
2. Abhijeet Singh (20CS30001)
3. Amit Kumar (20CS30003)
4. Gopal (20CS30021)

Graph Data Structure

```
struct node
{
   int degree;
   int head;
   node()
   {
       degree = 0;
       head = -1;
   }
};

struct edgeData
{
   int to, cost, flow;
   int nxt; /* nxt: is the index of the next edge */
};

struct graph
{
   int node_count, edge_count;
   node nodes[MAX_NODES];
   edgeData edges[MAX_EDGES];
   graph(int n)
   {
       node_count = n;
       edge_count = 0;
   }
   void addEdge(int from, int to, int cost)
   {
       edges[edge_count].to = to;
       edges[edge_count].cost = cost;
       edges[edge_count].nxt = nodes[from].head;
       nodes[from].head = edge_count;
       nodes[from].degree++;
       edge_count++;
   }
};
```

We will represent this graph using only two arrays:

node nodes[MAX_NODES] : struct node contains all information about a vertex in graph. head will represent the index of the latest added outgoing edge in the edges array. Degree is number of edges it is connected to.

edgeData edges[MAX_EDGES] : it contains all the needed data from the edge. and the edgeData contains nxt which will point to the index of the next edge.

struct graph : node_count and edge_count represent the actual number of nodes and edges in the graph. addEdge() function adds an edge in the graph.

Design Strategy for optimization

- When new nodes are added, in each consumer process a vector of source nodes and non source nodes is created consisting of the extra nodes that have been added.
- Then multi-sourced-dijkstra is run in each consumer process with the extra source nodes that have been added.
- By just doing the above step, distance for some of the non source extra nodes distance might not get updated.

- So we are calling one more function to update the distance of extra non source nodes, and updating distance of other non source nodes as a result of previous updation.
- In this function we are first pushing all the extra non source nodes into a priority-queue, and then for all these nodes we are trying to update the distance of these nodes using their neighbors.
- Then when the distance of these extra non source nodes have been updated, we are trying to update the distance of neighbors of these nodes. And if the distance of their neighbors have reduced then we are pushing these neighbors into the priority-queue and running dijkstra's algorithm further on these nodes.
- Thus in our optimized version we are not trying to compute the distance of all the nodes but rather proceeding in only those directions in which distance is getting updated.
- Why does this work?

  Key observation is that the maximum shortest distance is 5 for all nodes. Thus for our optimized algorithm updation will take place only for extra nodes and in the neighboring region of these extra

nodes. While in an unoptimized version we would have to update distance for all the nodes. Thus our optimized version is computationally much more efficient than the normal unoptimized version.

Table for running average execution time of each iteration

Number of Iterations	Unoptimized Version (in ms)	Optimized Version (in ms)
1	7.073	6.398
2	6.628	3.199
3	6.593	2.221
4	6.566	1.666
5	6.530	1.372
6	6.520	1.184
7	6.545	1.015
8	6.576	0.908
9	6.605	0.807
10	6.595	0.751
15	6.292	0.553
20	5.838	0.432
25	5.667	0.358
30	5.383	0.322