Smit Shah - 201701009

Q3. Incidence to Adjacency matrix

```
void inc2adj(int **inc, int **adj, int n, int m)
{
    // Converts a given incidence matrix to adjacency matrix.
    // Uses the fact that the sum of columns in an incidence matrix
is 2.

    int i, j;

    // Here both the inner and outer loop can be parallelizable but
again due to the temp_vertex variable, there needs to be a
    // critical section. Hence, we parallelize the outer loop to
avoid these issues. We use static scheduling here because all
    // the loops will have similar load so to increase efficiency, we
use static scheduling.

    //#pragma omp parallel for private(j, dist) schedule(static)
    for (i = 1; i <= m; i++)
    {
        int temp_vertex = -1;
        for (j = 1; j <= n; j++)
        {
            if (inc[j][i] != 0)
            {
                if (temp_vertex == -1)
                {
                    temp_vertex = j;
                }
                else
                {
                    adj[temp_vertex][j] = 1;
                    adj[j][temp_vertex] = 1;
                    break;
                }
            }
        }
    }
```

```
}
```

Inner loop parallelization

```
void inc2adj(int **inc, int **adj, int n, int m)
{
    // Converts a given incidence matrix to adjacency matrix.
    // Uses the fact that the sum of columns in an incidence matrix
is 2.

    int i, j;

    for (i = 1; i <= m; i++)
    {
        int temp_vertex_1 = -1, temp_vertex_2 = -1;
        // The above two variables are shared between the threads.
        // IPC is required when brea
        for (j = 1; j <= n; j++)
        {
            if (inc[j][i] != 0)
            {
                if (temp_vertex_1 == -1)
                {
                    temp_vertex_1 = j;
                }
                else if (temp_vertex_2 == -1)
                {
                    temp_vertex_2 = j;
                    break;
                }
            }
        }
        //Critical section. Use locks or omp critical.
        mutex_lock();
        adj[temp_vertex_1][temp_vertex_2]++;
        adj[temp_vertex_2][temp_vertex_1]++;
        mutex_unlock();
    }
}
```