Smit Shah - 201701009

Q3. Adjacency to Incidence Matrix

```
void adj2inc(int **adj, int **inc, int n, int m)
{
    // Converts a given adjacency matrix to incidence matrix.
    // Uses lower-half matrix (symmetric) to calculate the adjacency
matrix

    int i, j, edge_id = 1, no_edges = 0;

    // outer loop parallelizable. We can also parallelize inner loop
but that will involve usage of critical sections
    // We use dynamic scheduling which means that the compiler will
schedule the given process during runtime.
    // We do this because each iteration does not have equal amount
of load (iterations closer to n will be more loaded)
    //#pragma omp parallel for private(j, dist) schedule(dynamic)
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= i; j++)
        {
            if (adj[i][j] != 0)
            {
                i == j ? no_edges = adj[i][j] / 2 : no_edges =
adj[i][j];
                for (int k = 0; k < n_edges; k++)
                {
                    inc[i][edge_id] = 1;
                    inc[j][edge_id] = 1;
                    edge_id++;
                }
            }
        }
    }
}
```

Inner Loop Parallelization

```
void adj2inc(int **adj, int **inc, int n, int m)
```

```
{
    // Converts a given adjacency matrix to incidence matrix.
    // Uses lower-half matrix (symmetric) to calculate the adjacency
matrix

    int i, j, edge_id = 1, no_edges = 0;

    for (i = 1; i <= n; i++)
    {
        //#pragma omp parallel for private(j, dist)
        for (j = 1; j <= i; j++)
        {
            if (adj[i][j] != 0)
            {
                i == j ? no_edges = adj[i][j] / 2 : no_edges =
adj[i][j];
                for (int k = 0; k < n_edges; k++)
                {
                    //Critical section. Use locks or omp critical.
                    mutex_lock();
                    inc[i][edge_id] = 1;
                    inc[j][edge_id] = 1;
                    edge_id++;
                    mutex_unlock();
                }
            }
        }
    }
}
```