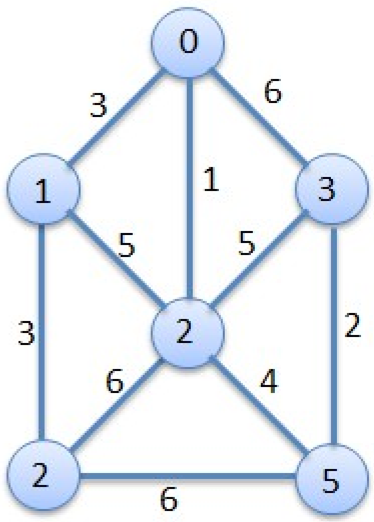
Assignment-B6
Implement concurrent prims algorithm using OPENMP

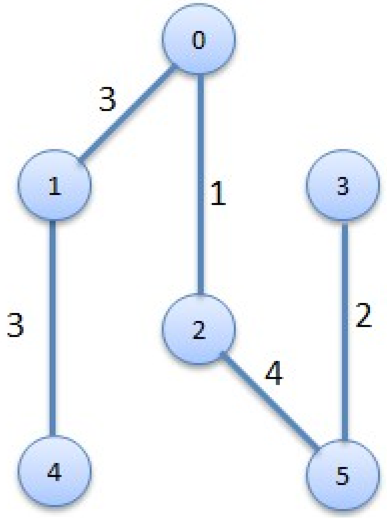
Prim's Algorithm

- greedy algorithm that finds a minimum spanning tree for a connected weighted undirected graph
- finds a subset of the edges
  - forms a tree that includes every vertex
  - the total weight of all the edges in the tree is minimized
- directly based on the MST(minimum spanning tree) property

Example



A Simple Weighted Graph



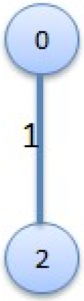
Minimum-Cost Spanning Tree

Example

- **Procedure for finding Minimum Spanning Tree**

Step1

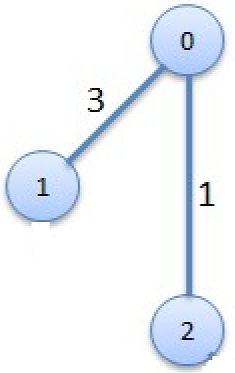
No. of Nodes	0	1	2	3	4	5
Distance	0	3	**1**	6	∞	∞
Distance From		0	0	0		



Example

Step2

No. of Nodes	0	1	2	3	4	5
Distance	0	**3**	0	5	6	4
Distance From		0		2	2	2



Similarly…..

Step3

No. of Nodes	0	1	2	3	4	5
Distance	0	0	0	5	**3**	4
Distance From				2	1	2

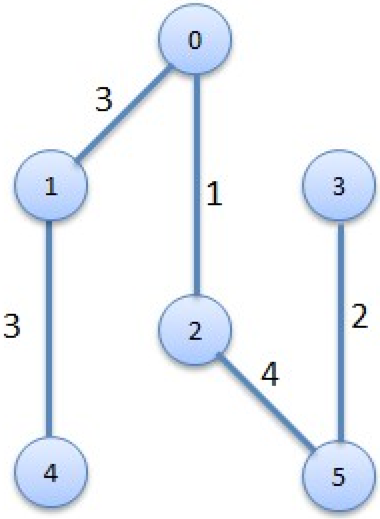
Example

Step4

No. of Nodes	0	1	2	3	4	5
Distance	0	0	0	5	0	**4**
Distance From				2		2

Step5

No. of Nodes	0	1	2	3	4	5
Distance	0	0	0	**3**	0	0
Distance From				2		2



Minimum Cost = 1+2+3+3+4 = 13

Program description

```
struct prim_data {
            int edges_weight[DIM][DIM];
            int dimension;
            int U[DIM];
            int total_min_distance;
            int count_nodes_in_mst;
            };

struct prim_data prim;
```

- Declaring a structure that will be further shared by entire program.
- Structure contains all declared variables that will also be shared among the threads

Program Description

```
#pragma omp parallel default(none),private(i,j),shared(prim)
  {
  int tid = omp_get_thread_num();
  printf("thread %d starting\n",tid);
   #pragma omp for
     for(i=0; i<prim.dimension; i++){
           for(j=0; j<prim.dimension; j++) {
                 printf("%d\t\n",prim.edges_weight[i][j]);
           }
      }
```

- All variables in OpenMP are shared by default. If you want a set
- of private variables you will need to specify these variables in a
- parallel pragma directive in a private clause.

If you use

#pragma omp parallel default none

You need to specify the private variables and shared variables.

- For instance:

#pragma omp parallel default(none) private(i,j) shared(a,b)

Program Description

```
#pragma omp for
    for(i=0; i<prim.dimension; i++){
        for(j=0; j<prim.dimension; j++) {
            printf("%d\t\n",prim.edges_weight[i][j]);
        }
    }
```

- #pragma omp parallel spawns a group of threads, while
- #pragma omp for
- divides loop iterations between the spawned threads.

- You can do both things at once with the
fused #pragma omp parallel for directive.

Program Description

```
void initialization(void) {

	int i,j;

	prim.total_min_distance = 0;
	prim.count_nodes_in_mst = 0;

	//initializing the U set
	for(i = 0; i < prim.dimension; i++) prim.U[i] = -1;

	//storing the first node into the U set
	prim.U[0] = 0;
	//deleting the first node
	delete_elements( prim.U[0] );
	//incrementing by one the number of node that are inside the U set
	prim.count_nodes_in_mst++;
}
```

Program Description

```
//initializing the data
       initialization();

       //calculating for all the nodes
       for(k = 0; k < prim.dimension -1; k++)
       {
           min_distance = 1000;
              //for every node in minimum spanning tree
              for(i = 0; i < prim.count_nodes_in_mst; i++)
              {
                     //declaring OpenMP's derective with the appropriate scheduling...
                     #pragma omp parallel for
                for(j = 0; j < prim.dimension; j++)
                {
                            //find the minimum weight
                    if(prim.edges_weight[prim.U[i]][j] > min_distance ||
prim.edges_weight[prim.U[i]][j]==0)
                    {
                        continue;
                    }
```

Program Description

```
 else
           {
            #pragma omp critical
             {
            min_distance = prim.edges_weight[prim.U[i]][j];
            new_next_element = j;
      }
 }



//Adding the local min_distance to the total_min_distance
       prim.total_min_distance += min_distance;
       //Adding the next node in the U set
       prim.U[i] = new_next_element;
//Substructing the elements of the column in which  the new node is
assosiated with
       delete_elements( new_next_element );
       //Increasing the nodes that they are in the MST
       prim.count_nodes_in_mst++;
    }
```

Program Description

```
//Print all the nodes in MST in the way that they stored in the U set
     for(i = 0 ; i < prim.dimension; i++) {
         printf("%d ",prim.U[i] + 1);
         if( i < prim.dimension - 1 ) printf("-> ");
     }

     printf("\n\n");
     printf("Total minimun distance: %d\n\n",
prim.total_min_distance);
     printf("\nProgram terminates now..\n");
       return 0;
```

Thank You