MPI_CART_COORDS(comm, rank, maxdims, coords)

IN	comm	communicator with Cartesian structure (handle)
IN	rank	rank of a process within group of comm (integer)
IN	maxdims	length of vector coords in the calling program (integer)
OUT	coords	integer array (of size ndims) containing the Cartesian coordinates of specified process (array of integers)

```
int MPI_Cart_coords(MPI_Comm comm, int rank, int maxdims, int *coords)
```

```
MPI_CART_COORDS(COMM, RANK, MAXDIMS, COORDS, IERROR)
    INTEGER COMM, RANK, MAXDIMS, COORDS(*), IERROR
```

```
void MPI::Cartcomm::Get_coords(int rank, int maxdims, int coords[]) const
```

The inverse mapping, rank-to-coordinates translation is provided by MPI_CART_COORDS.

If comm is associated with a zero-dimensional Cartesian topology, coords will be unchanged.

MPI_GRAPH_NEIGHBORS_COUNT(comm, rank, nneighbors)

IN	comm	communicator with graph topology (handle)
IN	rank	rank of process in group of comm (integer)
OUT	nneighbors	number of neighbors of specified process (integer)

```
int MPI_Graph_neighbors_count(MPI_Comm comm, int rank, int *nneighbors)
```

```
MPI_GRAPH_NEIGHBORS_COUNT(COMM, RANK, NNEIGHBORS, IERROR)
    INTEGER COMM, RANK, NNEIGHBORS, IERROR
```

```
int MPI::Graphcomm::Get_neighbors_count(int rank) const
```

MPI_GRAPH_NEIGHBORS(comm, rank, maxneighbors, neighbors)

IN	comm	communicator with graph topology (handle)
IN	rank	rank of process in group of comm (integer)
IN	maxneighbors	size of array neighbors (integer)
OUT	neighbors	ranks of processes that are neighbors to specified process (array of integer)

```
int MPI_Graph_neighbors(MPI_Comm comm, int rank, int maxneighbors,
              int *neighbors)
```

```
MPI_GRAPH_NEIGHBORS(COMM, RANK, MAXNEIGHBORS, NEIGHBORS, IERROR)
    INTEGER COMM, RANK, MAXNEIGHBORS, NEIGHBORS(*), IERROR
```

```
void MPI::Graphcomm::Get_neighbors(int rank, int maxneighbors, int
            neighbors[]) const
```

MPI_GRAPH_NEIGHBORS_COUNT and MPI_GRAPH_NEIGHBORS provide adjacency information for a general graph topology. The returned count and array of neighbors for the queried rank will both include *all* neighbors and reflect the same edge ordering as was specified by the original call to MPI_GRAPH_CREATE. Specifically, MPI_GRAPH_NEIGHBORS_COUNT and MPI_GRAPH_NEIGHBORS will return values based on the original `index` and `edges` array passed to MPI_GRAPH_CREATE (assuming that `index[-1]` effectively equals zero):

- The `count` returned from MPI_GRAPH_NEIGHBORS_COUNT will be (`index[rank] - index[rank-1]`).
- The `neighbors` array returned from MPI_GRAPH_NEIGHBORS will be `edges[index[rank-1]]` through `edges[index[rank]]`.

Example 7.3 Assume there are four processes 0, 1, 2, 3 with the following adjacency matrix (note that some neighbors are listed multiple times):

process	neighbors
0	1, 1, 3
1	0, 0
2	3
3	0, 2, 2

Thus, the input arguments to MPI_GRAPH_CREATE are:

nnodes =	4
index =	3, 5, 6, 9
edges =	1, 1, 3, 0, 0, 3, 0, 2, 2

Therefore, calling MPI_GRAPH_NEIGHBORS_COUNT and MPI_GRAPH_NEIGHBORS for each of the 4 processes will return:

Input rank	**Count**	**Neighbors**
0	3	1, 1, 3
1	2	0, 0
2	1	3
3	3	0, 2, 2

Example 7.4 Suppose that `comm` is a communicator with a shuffle-exchange topology. The group has 2^n members. Each process is labeled by $a_1, \ldots, a_n$ with $a_i \in \{0, 1\}$, and has three neighbors: exchange$(a_1, \ldots, a_n) = a_1, \ldots, a_{n-1}, \bar{a}_n$ ($\bar{a} = 1 - a$), shuffle$(a_1, \ldots, a_n) = a_2, \ldots, a_n, a_1$, and unshuffle$(a_1, \ldots, a_n) = a_n, a_1, \ldots, a_{n-1}$. The graph adjacency list is illustrated below for $n = 3$.