```python
import heapq
from graph import Graph

# read the graph
def read_graph_from_file(file_descriptor:str) -> Graph:
    with open(file_descriptor, "r") as file:
        tokens = file.readline().split(" ")
        n, m = int(tokens[0].strip()), int(tokens[1].strip().removesuffix("\n"))

        graph = Graph(n)

        while m > 0:
            tokens = file.readline().split(" ")
            v_from, v_to, cost = int(tokens[0].strip()), int(tokens[1].strip()), int(tokens[2].strip().removesuffix("\n"))
            graph.add_edge(v_from, v_to, cost)

            m -= 1

    return graph

def prim(graph: Graph, starting_vertex: int) -> Graph:

    if not graph.existing_vertex(starting_vertex): # if the vertex is invalid or does not exist raise exception
        raise Exception("No such vertex in the graph.\n")

    mst = Graph()
    pq = []
    n = graph.get_nr_of_vertices

    mst.add_vertex(starting_vertex) # adding the starting vertex to the mst

    # Add all edges from the starting vertex to the priority queue
    for neighbor_node, cost in graph.get_neighbors(starting_vertex):
        heapq.heappush(pq, (cost, starting_vertex, neighbor_node))

    while mst.get_nr_of_vertices < n and pq: # the mst should contain all nodes of the initial connected graph
        cost, u, v = heapq.heappop(pq) # get the edge with the current minimum cost from the priority queue
        if mst.existing_vertex(u) != mst.existing_vertex(v): # one of the endpoints of the edge should be in the mst already, and one shouldn't
            new_node = v if not mst.existing_vertex(v) else u # getting the vertex that is not in the mst yet
            mst.add_vertex(new_node) # add the vertex to the mst
            mst.add_edge(u, v, cost) # add the edge on the mst

            # Add all edges from the last vertex added to the priority queue
            for neighbor_node, edge_cost in graph.get_neighbors(new_node):
                if not mst.existing_vertex(neighbor_node):
                    heapq.heappush(pq, (edge_cost, new_node, neighbor_node))

    return mst # return the result


def solve():
    try:
        graph = read_graph_from_file("graph1.txt")

        starting_vertex = int(input("Insert the starting vertex here: "))
        mst = prim(graph, starting_vertex)
        print(f"\nThe MST starting from vertex {starting_vertex} is:\n{mst.get_nr_of_vertices} {mst.get_nr_of_vertices - 1}")

        mst_cost = 0
        for v_from, v_to, cost in mst.edges_iterator():
            print(f"{v_from} {v_to} {cost}")
            mst_cost += cost

        print(f"\nThe cost of the MST is {mst_cost}.")
    except Exception as e:
        print(str(e) + '\n')

solve()
```