Title

Reasoning

We can use the depth first or breadth first search in order to find the number of unconnected components, then add one edge to connect each of these components.

Psudocode

```
def dfs(v, pointer to seen):
    seen[v] = true
    for every neighbor u of v:
        if not seen[u]: DFS(u)
    time ++
    fin[v] = time

def min_edges_to_connect(n, edges):
    adj = empty array for every element of n
    for u, v in edges:
        adj[u].append(v)
        adj[v].append(u)

    visited = []
    for int i = 0 to n: visited[i] = False
    components = 0

    for i in range(n):
        if not visited[i]:
            dfs(i, visited)
            components += 1
    return components - 1
```

Proof

If we seperate the graph into components, then all of these seperated segments can be connected by roads using one less road then there are segments. We can tell what segments are connected by using the dfs algorithm.

Running Time

Estimate

The running time is $O(n+m)$

Running Time Reasoning

```
def dfs(v, seen): # DFS takes O(n+m)
    seen[v] = true
    for every neighbor u of v:
        if not seen[u]: DFS(u)
    time ++
    fin[v] = time

def min_edges_to_connect(n, edges):
    adj = empty array for every element of n # O(n)
    for u, v in edges: # O(m)
        adj[u].append(v)
        adj[v].append(u)

    visited = []
    for int i = 0 to n: visited[i] = False # O(n)
    components = 0

    for i in range(n):
        if not visited[i]: # Only runs dfs for each new component, so dfs
is only called for smaller chunks of the graphs, meaning it still runs in
O(n+m) time
            dfs(i, visited)
            components += 1
    return components - 1
```

Total time is $O(n+m)+O(n)+O(m)$ or $O(n+m)$