Algorithm:

```
findMST(G', T, modified_edge):
    V = G'.vertices
    E = G'.edges

    if modified_edge not in T.edges:
        T.add_edge(modified_edge)

        cycle = findCycle(T, modified_edge)
        max_edge = max(cycle.edges, key=lambda edge: edge.weight)
        T.remove_edge(max_edge)

    return T

findCycle(T, modified_edge):
    visited = set()
    parent = dict()
    start, end = modified_edge

     DFS(u, p):
        visited.add(u)
        parent[u] = p
        for v in T.neighbors(u):
            if v not in visited:
                DFS(v, u)
            else if v != p:
                return (u, v)

    cycle_start, cycle_end = DFS(start, None)

    cycle = [cycle_end]
```

```
current = cycle_start
while current != cycle_end:
    cycle.append(current)
    current = parent[current]

return cycle
```

توضیح الگوریتم:

اگر modified_edge قبلاً در T نباشد، به T اضافه می‌شود. سپس تابع findCycle برای شناسایی cycle که با افزودن modified_edge به T تشکیل می‌شود، فراخوانی می‌شود. این cycle به این دلیل تشکیل می‌شود که افزودن modified_edge به T یک cycle ایجاد می‌کند و MST تشکیل می‌شود ولی cycle , MST ندارد. هنگامی که cycle شناسایی شد، تابع بالاترین وزن را از cycle حذف می کند و T را با حذف این یال به روز می کند. این تضمین می کند که درخت حاصل یک MST معتبر باقی می ماند. سپس T به روز شده توسط تابع findMST برگردانده می شود.

تابع findCycle از Depth-First Search (DFS) برای شناسایی cycle استفاده می‌کند که با افزودن modified_edge به T شکل می‌گیرد.

Time complexity: O(V+E)