**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 MST , cycleندارد. 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)