8월 14일

```
edge_index=torch. tensor([[ 3,  2,  2,  9,  9,  9, 10,  3,  1,  1,  2,  2,  2,  2,  2,  2,  3,  5,
          6,  7,  8,  8,  8,  8,  9,  9,  0,  0,  0,  8,  5,  7,  5,  7,  1,  3,
          4,  6,  7,  4,  5,  6,  7,  9, 10,  2,  1,  1, 10,  1,  7,  9, 10,  1,
          6,  1,  2,  3],
                          [ 8,  5,  7,  5,  7,  1,  3,  4,  6,  7,  4,  5,  6,  7,  9, 10,  2,  1,
          1, 10,  1,  7,  9, 10,  1,  6,  1,  2,  3,  3,  2,  2,  9,  9,  9, 10,
          3,  1,  1,  2,  2,  2,  2,  2,  2,  3,  5,  6,  7,  8,  8,  8,  8,  9,
          9,  0,  0,  0]])
  edge_type=torch. tensor([ 4,  7,  7,  7,  7,  9, 10, 12, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
        17, 17, 17, 17, 17, 17, 17, 17,  0,  0,  0, 23, 26, 26, 26, 26, 28, 29,
        31, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36,
        36, 19, 19, 19])
```

```
def preprocess_adjacency_list(edge_index, edge_type):
    adj_list = {}

    for i in range(edge_index.shape[1]):
        source_node = edge_index[0, i].item()
        target_node = edge_index[1, i].item()
        edge_type_idx = edge_type[i].item()

        if source_node not in adj_list:
            adj_list[source_node] = []
        adj_list[source_node].append((target_node, edge_type_idx))

    return adj_list
```

```
def find_cycles(edge_index, edge_type):
    num_nodes = max(edge_index.max().item(), edge_index.max().item()) + 1

    adj_list = {i: [] for i in range(num_nodes)}
    for i, j in edge_index.T.tolist():
        adj_list[i].append(j)

    def find_cycles_dfs(node, start_node, depth, visited, path, cycles):
        visited[node] = True
        path.append(node)

        if depth == 2:
            if start_node in adj_list[node]:
                cycle = list(path)
                min_idx = cycle.index(min(cycle))
                cycle = cycle[min_idx:] + cycle[:min_idx]
                cycles.add(tuple(cycle))
        elif depth < 2:
            for neighbor in adj_list[node]:
                if not visited[neighbor]:
                    find_cycles_dfs(neighbor, start_node, depth + 1, visited, path, cycles)

        path.pop()
        visited[node] = False

    cycles = set()
    for start_node in range(num_nodes):
        visited = [False] * num_nodes
        find_cycles_dfs(start_node, start_node, 0, visited, [], cycles)
```

```
    node_list = [list(cycle) for cycle in cycles]
    return node_list
```

```
def get_cycle_triples_from_adj_list(adj_list, node_list):
  # 인접 리스트와 노드 리스트는 미리 만들어 놓는다.

    cycle_triples_list = []

    for cycle_nodes in node_list:
        cycle_triples = []
        cycle_length = len(cycle_nodes)

        for i in range(cycle_length):
            source_node = cycle_nodes[i]
            target_node = cycle_nodes[(i + 1) % cycle_length]  # Wrap around to the first node for the last edge

            if source_node in adj_list:
                for target, edge_type in adj_list[source_node]:
                    if target == target_node:
                        cycle_triples.append((source_node, edge_type, target_node))

        cycle_triples_list.append(cycle_triples)

    return cycle_triples_list
```

인접 리스트를 pickle파일로 저장, node_list를 pickle 파일로 저장

adj_list =

마지막으로 get_cycle_triples from adj_list에 대입

이렇게 해도 triple은 워낙 많아서 시간이 걸린다.

그런 상황에서 만약에 cycle triple을 구성하는 에지의 타입이 모두 17, 36이면 제외하면 어떨까?

내일 할 일

get_cycle_triples_from_adj_list를 전체 adj_list, 전체 node_list에 돌릴 수 있는 코드 만들기