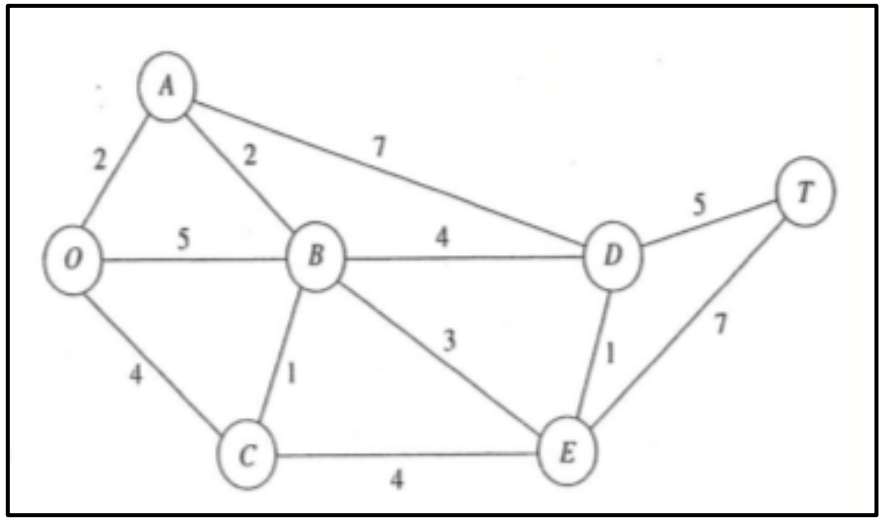
Dijkstra 알고리즘 파이썬 구현

산업공학과 201913257 정용희

목차

1. 네트워킹 경로 정보 저장



- dictionary형태로 저장

★ 딕셔너리 형태로 저장하는 이유

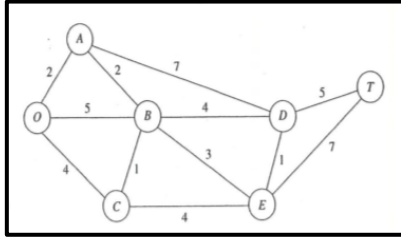
: dictionary는 {key : value} 형태로 저장되기 때문에 key를 노드로 하고, value를 거리로 저장하면 쉽게 알아볼 수 있음

```
original_dict = {
    'O': {'A':2, 'B':5, 'C':4},
    'A': {'B':2, 'D':7},
    'B': {'C':1, 'D':4, 'E':3},
    'C': {'E':4},
    'D': {'E':1, 'T':5},
    'E': {'T':7},
    'T': {}
}
```

노드 'O' 에서 갈 수 있는 인접노드
= { 노드 'A' (거리=2), 노드 'B' (거리=5), 노드 'C' (거리=4) }

2. Dijkstra 함수

2.1. 반복 전, 필요한 list와 dict 생성



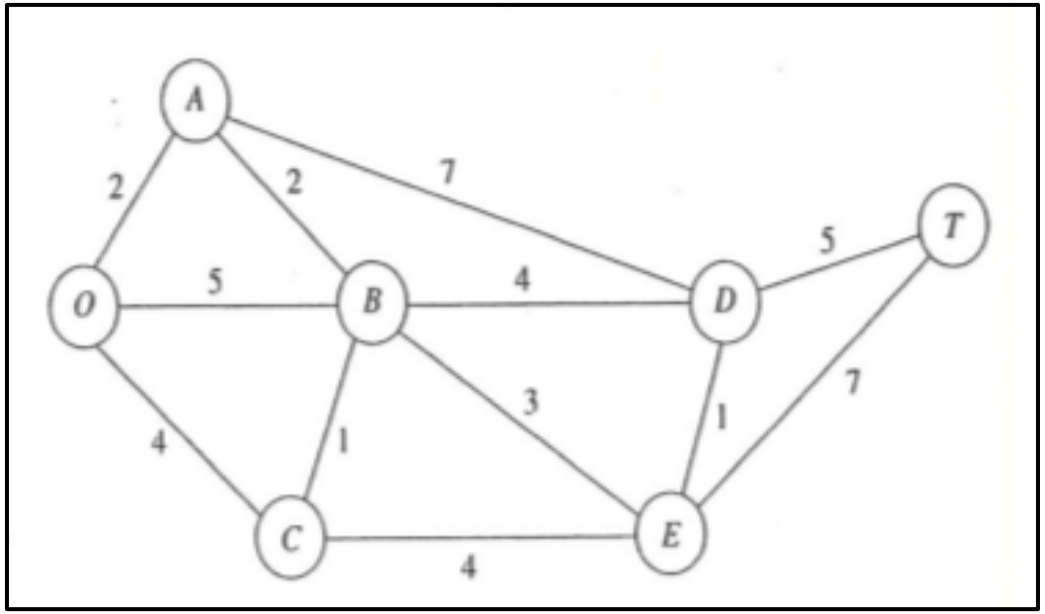

```python
def dijkstra(start, original_dict):

    # 시작 노드에서 다른 노드까지의 거리를 무한대로 초기화한 딕셔너리를 생성
    distances = {node: float('inf') for node in original_dict}
    print(f'처음 거리(distances): {distances}', '₩n')

    # 시작 목록을 생성
    orig_list = [(start, 0)]   # -> 처음에 [('O', 0)]이 들어가게 됨
    print(f'시작 목록(orig_list): {orig_list}')
    print()

    # 시작 노드의 거리를 0으로 설정
    distances[start] = 0
    print(f'시작 노드의 거리를 0으로 설정한 거리(distances): {distances}', '₩n')

    # 직전 노드를 저장할 딕셔너리를 생성
    prev_node = {node: None for node in original_dict}
    print(f'직전 노드의 딕셔너리 생성(prev_node): {prev_node}', '₩n')
    print('==========================================반복 시작==========================================', '₩n')
```

```
처음 거리(distances): {'O': inf, 'A': inf, 'B': inf, 'C': inf, 'D': inf, 'E': inf, 'T': inf}

시작 목록(orig_list): [('O', 0)]

시작 노드의 거리를 0으로 설정한 거리(distances): {'O': 0, 'A': inf, 'B': inf, 'C': inf, 'D': inf, 'E': inf, 'T': inf}

직전 노드의 딕셔너리 생성(prev_node): {'O': None, 'A': None, 'B': None, 'C': None, 'D': None, 'E': None, 'T': None}

==========================================반복 시작==========================================
```

전	노드	O(시작)	A	B	C	D	E	T
	거리	inf	inf	inf	inf	inf	inf	inf

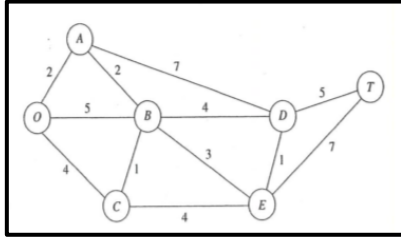
후	노드	O(시작)	A	B	C	D	E	T
	거리	0	inf	inf	inf	inf	inf	inf

* inf: 무한대



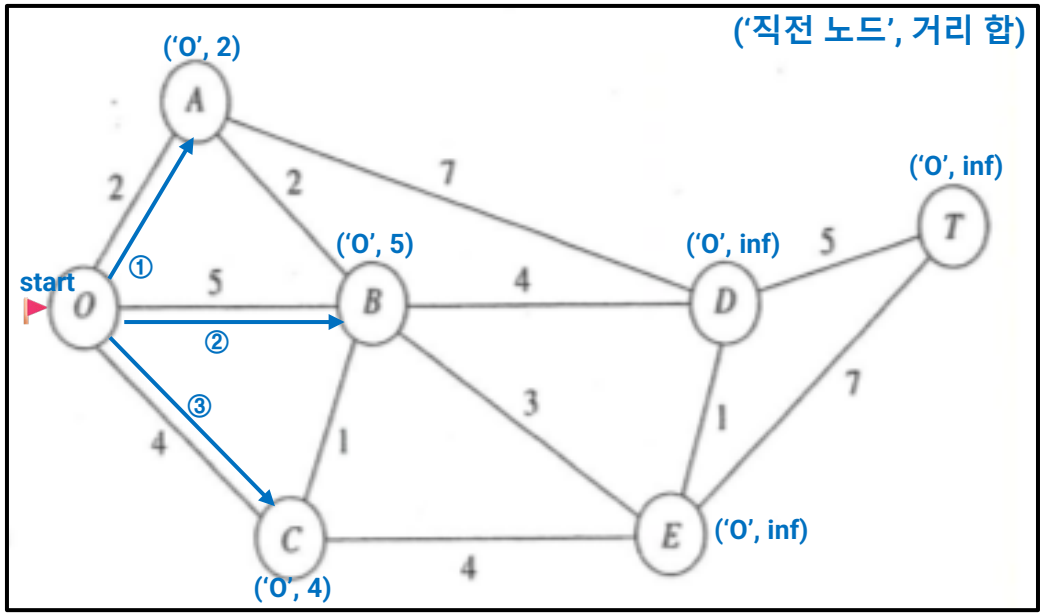
2. Dijkstra 함수

2.2. 반복 : 마지막 반복에 orig_list에 아무것도 없을 때 종료




```python
# 목록이 빌 때까지 반복
while orig_list:
    # 시작 노드로부터 최소 거리를 가진 노드를 찾습니다
    node, current_distance = min(orig_list)
    print(f'최소 거리를 가진 노드(min(orig_list)): {min(orig_list)}')

    orig_list.remove((node, current_distance))
    print(f'최소 거리를 가진 노드를 제거한 목록(orig_list): {orig_list}', '₩n')

    # 현재 거리가 저장된 거리보다 큰 경우 건너뜁니다
    if current_distance > distances[node]:
        continue

    # 인접한 노드와 그 거리를 반복하면서 확인
    for next_node, distance in original_dict[node].items():

        # 시작 노드에서 인접 노드까지의 새로운 거리를 계산합니다
        new_distance = distances[node] + distance
        print(f"출발 노드(node='{node}')에서 인접 노드(node='{next_node}') 까지의 거리: {new_distance}")

        # 새로운 거리가 저장된 거리보다 작으면 업데이트합니다
        if new_distance < distances[next_node]:
            distances[next_node] = new_distance

            # 직전 노드를 업데이트합니다
            prev_node[next_node] = node
            print(f'업데이트 된 직전 노드 딕셔너리: {prev_node}')

            # 새로운 거리와 인접한 노드를 목록에 추가합니다
            orig_list.append((next_node, new_distance))   # new_distance -> 노드에 도달하기 위해 더해진 거리가 기록됨

            print(f'업데이트 된 거리: {distances}', '₩n')
    print()
    print(f'업데이트 된 목록(orig_list): {orig_list}')
    print('==========================================한 사이클 종료==========================================', '₩n')
print('==========================================반복 종료==========================================', '₩n')
```

```
최소 거리를 가진 노드(min(orig_list)): ('O', 0)
최소 거리를 가진 노드를 제거한 목록(orig_list): []

① 출발 노드(node='O')에서 인접 노드(node='A') 까지의 거리: 2
업데이트 된 직전 노드 딕셔너리: {'O': None, 'A': 'O', 'B': None, 'C': None, 'D': None, 'E': None, 'T': None}
업데이트 된 거리: {'O': 0, 'A': 2, 'B': inf, 'C': inf, 'D': inf, 'E': inf, 'T': inf}

② 출발 노드(node='O')에서 인접 노드(node='B') 까지의 거리: 5
업데이트 된 직전 노드 딕셔너리: {'O': None, 'A': 'O', 'B': 'O', 'C': None, 'D': None, 'E': None, 'T': None}
업데이트 된 거리: {'O': 0, 'A': 2, 'B': 5, 'C': inf, 'D': inf, 'E': inf, 'T': inf}

③ 출발 노드(node='O')에서 인접 노드(node='C') 까지의 거리: 4
업데이트 된 직전 노드 딕셔너리: {'O': None, 'A': 'O', 'B': 'O', 'C': 'O', 'D': None, 'E': None, 'T': None}
업데이트 된 거리: {'O': 0, 'A': 2, 'B': 5, 'C': 4, 'D': inf, 'E': inf, 'T': inf}

업데이트 된 목록(orig_list): [('A', 2), ('B', 5), ('C', 4)]
==========================================한 사이클 종료==========================================
```

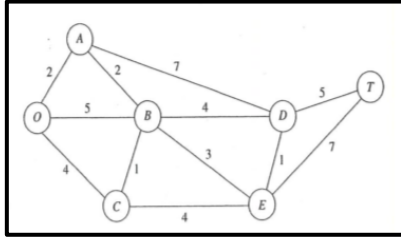


전	노드	O(시작)	A	B	C	D	E	T
	거리	0	inf	inf	inf	inf	inf	inf

후	노드	O(시작)	A	B	C	D	E	T
	거리	0	2	5	4	inf	inf	inf

2. Dijkstra 함수

2.2. 반복 : 마지막 반복에 orig_list에 아무것도 없을 때 종료

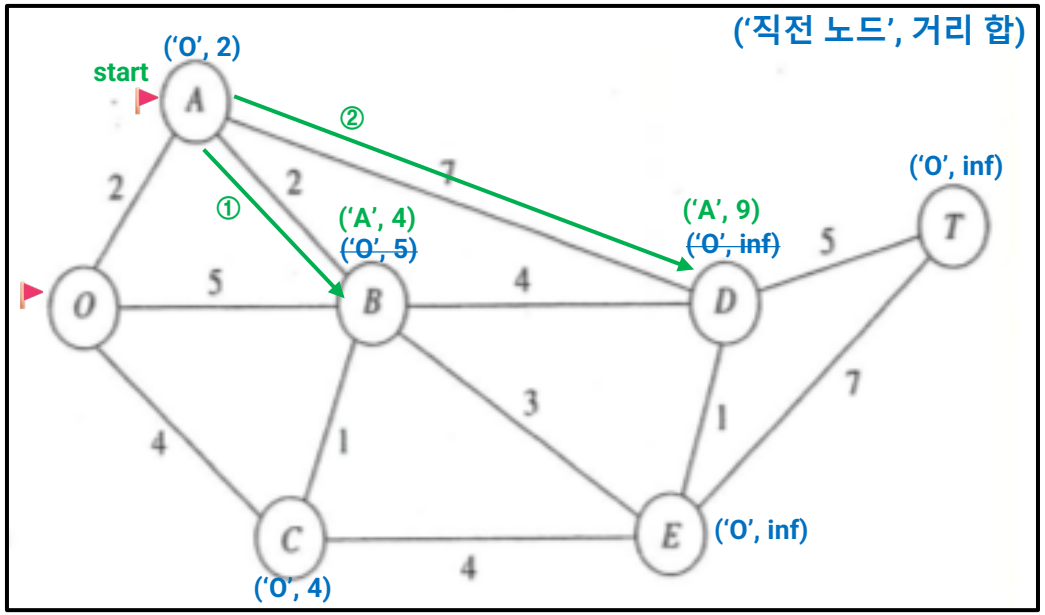



```python
# 목록이 빌 때까지 반복
while orig_list:
    # 시작 노드로부터 최소 거리를 가진 노드를 찾습니다
    node, current_distance = min(orig_list)
    print(f'최소 거리를 가진 노드(min(orig_list)): {min(orig_list)}')

    orig_list.remove((node, current_distance))
    print(f'최소 거리를 가진 노드를 제거한 목록(orig_list): {orig_list}', '₩n')

    # 현재 거리가 저장된 거리보다 큰 경우 건너뜁니다
    if current_distance > distances[node]:
        continue

    # 인접한 노드와 그 거리를 반복하면서 확인
    for next_node, distance in original_dict[node].items():

        # 시작 노드에서 인접 노드까지의 새로운 거리를 계산합니다
        new_distance = distances[node] + distance
        print(f"출발 노드(node='{node}')에서 인접 노드(node='{next_node}') 까지의 거리: {new_distance}")

        # 새로운 거리가 저장된 거리보다 작으면 업데이트합니다
        if new_distance < distances[next_node]:
            distances[next_node] = new_distance

            # 직전 노드를 업데이트합니다
            prev_node[next_node] = node
            print(f'업데이트 된 직전 노드 딕셔너리: {prev_node}')

            # 새로운 거리와 인접한 노드를 목록에 추가합니다
            orig_list.append((next_node, new_distance))   # new_distance -> 노드에 도달하기 위해 더해진 거리가 기록됨

            print(f'업데이트 된 거리: {distances}', '₩n')
    print()
    print(f'업데이트 된 목록(orig_list): {orig_list}')
    print('==========================================한 사이클 종료==========================================', '₩n')
print('==========================================반복 종료==========================================', '₩n')
```

```
최소 거리를 가진 노드(min(orig_list)): ('A', 2)
최소 거리를 가진 노드를 제거한 목록(orig_list): [('B', 5), ('C', 4)]

① 출발 노드(node='A')에서 인접 노드(node='B') 까지의 거리: 4
업데이트 된 직전 노드 딕셔너리: {'O': None, 'A': 'O', 'B': 'A', 'C': 'O', 'D': None, 'E': None, 'T': None}
업데이트 된 거리: {'O': 0, 'A': 2, 'B': 4, 'C': 4, 'D': inf, 'E': inf, 'T': inf}

② 출발 노드(node='A')에서 인접 노드(node='D') 까지의 거리: 9
업데이트 된 직전 노드 딕셔너리: {'O': None, 'A': 'O', 'B': 'A', 'C': 'O', 'D': 'A', 'E': None, 'T': None}
업데이트 된 거리: {'O': 0, 'A': 2, 'B': 4, 'C': 4, 'D': 9, 'E': inf, 'T': inf}

업데이트 된 목록(orig_list): [('B', 5), ('C', 4), ('B', 4), ('D', 9)]
==========================================한 사이클 종료==========================================
```

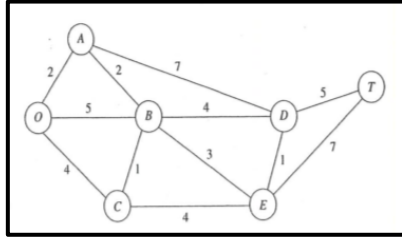
전	노드	O(시작)	A	B	C	D	E	T
	거리	0	2	5	4	inf	inf	inf

후	노드	O(시작)	A	B	C	D	E	T
	거리	0	2	4	4	9	inf	inf



2. Dijkstra 함수

2.2. 반복 : 마지막 반복에 orig_list에 아무것도 없을 때 종료

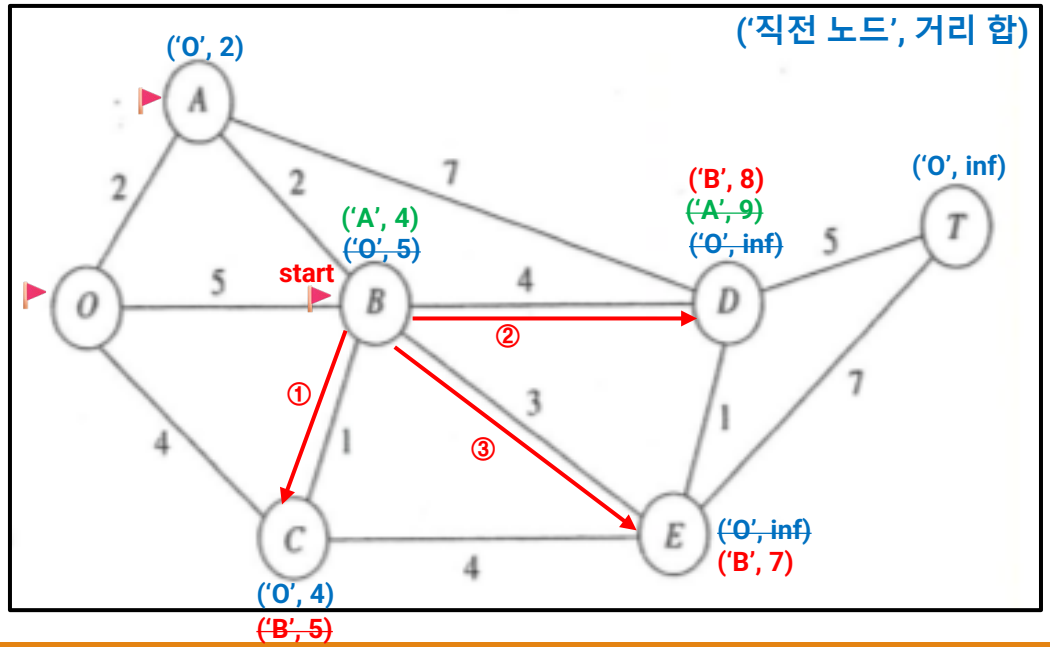



```python
# 목록이 빌 때까지 반복
while orig_list:
    # 시작 노드로부터 최소 거리를 가진 노드를 찾습니다
    node, current_distance = min(orig_list)
    print(f'최소 거리를 가진 노드(min(orig_list)): {min(orig_list)}')

    orig_list.remove((node, current_distance))
    print(f'최소 거리를 가진 노드를 제거한 목록(orig_list): {orig_list}', '₩n')

    # 현재 거리가 저장된 거리보다 큰 경우 건너뜁니다
    if current_distance > distances[node]:
        continue

    # 인접한 노드와 그 거리를 반복하면서 확인
    for next_node, distance in original_dict[node].items():

        # 시작 노드에서 인접 노드까지의 새로운 거리를 계산합니다
        new_distance = distances[node] + distance
        print(f"출발 노드(node='{node}')에서 인접 노드(node='{next_node}') 까지의 거리: {new_distance}")

        # 새로운 거리가 저장된 거리보다 작으면 업데이트합니다
        if new_distance < distances[next_node]:
            distances[next_node] = new_distance

            # 직전 노드를 업데이트합니다
            prev_node[next_node] = node
            print(f'업데이트 된 직전 노드 딕셔너리: {prev_node}')

            # 새로운 거리와 인접한 노드를 목록에 추가합니다
            orig_list.append((next_node, new_distance))   # new_distance -> 노드에 도달하기 위해 더해진 거리가 기록됨

            print(f'업데이트 된 거리: {distances}', '₩n')
    print()
    print(f'업데이트 된 목록(orig_list): {orig_list}')
    print('==========================================한 사이클 종료==========================================', '₩n')
print('==========================================반복 종료==========================================', '₩n')
```

```
최소 거리를 가진 노드(min(orig_list)): ('B', 4)
최소 거리를 가진 노드를 제거한 목록(orig_list): [('B', 5), ('C', 4), ('D', 9)]

① 출발 노드(node='B')에서 인접 노드(node='C') 까지의 거리: 5
② 출발 노드(node='B')에서 인접 노드(node='D') 까지의 거리: 8
업데이트 된 직전 노드 딕셔너리: {'O': None, 'A': 'O', 'B': 'A', 'C': 'O', 'D': 'B', 'E': None, 'T': None}
업데이트 된 거리: {'O': 0, 'A': 2, 'B': 4, 'C': 4, 'D': 8, 'E': inf, 'T': inf}

③ 출발 노드(node='B')에서 인접 노드(node='E') 까지의 거리: 7
업데이트 된 직전 노드 딕셔너리: {'O': None, 'A': 'O', 'B': 'A', 'C': 'O', 'D': 'B', 'E': 'B', 'T': None}
업데이트 된 거리: {'O': 0, 'A': 2, 'B': 4, 'C': 4, 'D': 8, 'E': 7, 'T': inf}

업데이트 된 목록(orig_list): [('B', 5), ('C', 4), ('D', 9), ('D', 8), ('E', 7)]
==========================================한 사이클 종료==========================================
```

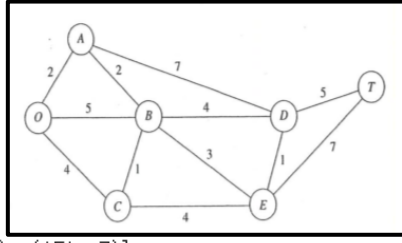
전	노드	O(시작)	A	B	C	D	E	T
	거리	0	2	4	4	9	inf	inf

후	노드	O(시작)	A	B	C	D	E	T
	거리	0	2	4	4	8	7	inf



2. Dijkstra 함수

2.2. 반복 : 마지막 반복에 orig_list에 아무것도 없을 때 종료

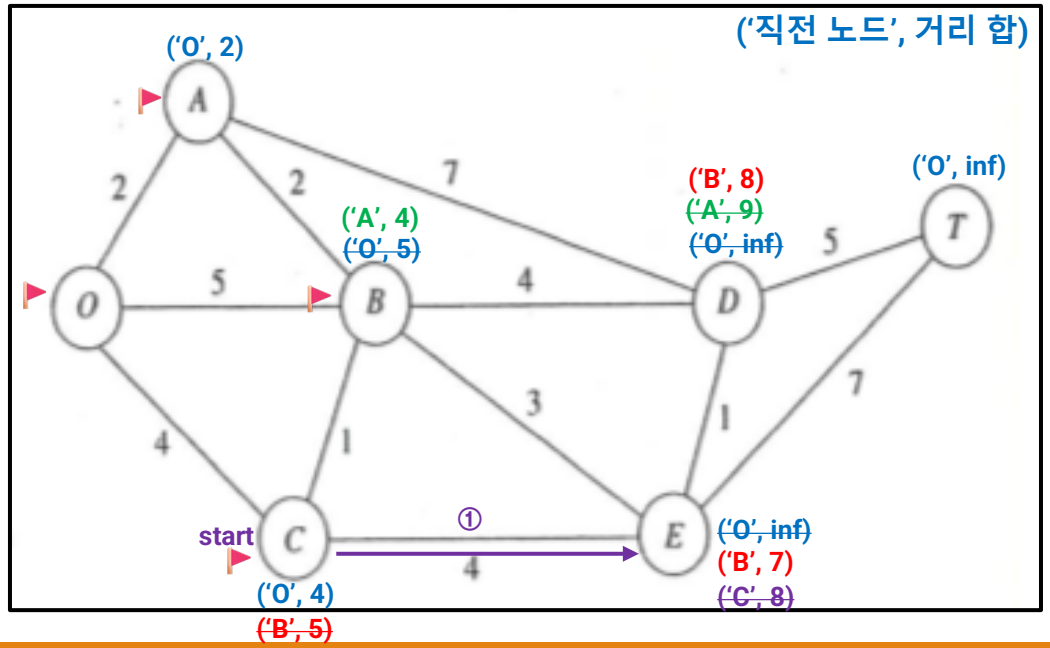



```python
# 목록이 빌 때까지 반복
while orig_list:
    # 시작 노드로부터 최소 거리를 가진 노드를 찾습니다
    node, current_distance = min(orig_list)
    print(f'최소 거리를 가진 노드(min(orig_list)): {min(orig_list)}')

    orig_list.remove((node, current_distance))
    print(f'최소 거리를 가진 노드를 제거한 목록(orig_list): {orig_list}', '₩n')

    # 현재 거리가 저장된 거리보다 큰 경우 건너뜁니다
    if current_distance > distances[node]:
        continue

    # 인접한 노드와 그 거리를 반복하면서 확인
    for next_node, distance in original_dict[node].items():

        # 시작 노드에서 인접 노드까지의 새로운 거리를 계산합니다
        new_distance = distances[node] + distance
        print(f"출발 노드(node='{node}')에서 인접 노드(node='{next_node}') 까지의 거리: {new_distance}")

        # 새로운 거리가 저장된 거리보다 작으면 업데이트합니다
        if new_distance < distances[next_node]:
            distances[next_node] = new_distance

            # 직전 노드를 업데이트합니다
            prev_node[next_node] = node
            print(f'업데이트 된 직전 노드 딕셔너리: {prev_node}')

            # 새로운 거리와 인접한 노드를 목록에 추가합니다
            orig_list.append((next_node, new_distance))   # new_distance -> 노드에 도달하기 위해 더해진 거리가 기록됨

            print(f'업데이트 된 거리: {distances}', '₩n')
    print()
    print(f'업데이트 된 목록(orig_list): {orig_list}')
    print('==========================================한 사이클 종료==========================================', '₩n')
print('==========================================반복 종료==========================================', '₩n')
```

```
최소 거리를 가진 노드(min(orig_list)): ('B', 5)
최소 거리를 가진 노드를 제거한 목록(orig_list): [('C', 4), ('D', 9), ('D', 8), ('E', 7)]

최소 거리를 가진 노드(min(orig_list)): ('C', 4)
최소 거리를 가진 노드를 제거한 목록(orig_list): [('D', 9), ('D', 8), ('E', 7)]

① 출발 노드(node='C')에서 인접 노드(node='E') 까지의 거리: 8

업데이트 된 목록(orig_list): [('D', 9), ('D', 8), ('E', 7)]
==========================================한 사이클 종료==========================================
```

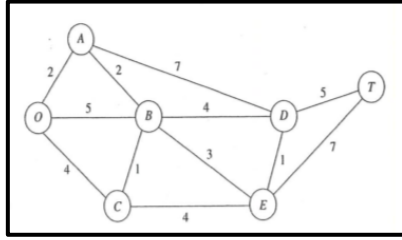
전	노드	O(시작)	A	B	C	D	E	T
	거리	0	2	4	4	8	7	inf

후	노드	O(시작)	A	B	C	D	E	T
	거리	0	2	4	4	8	7	inf



2. Dijkstra 함수

2.2. 반복 : 마지막 반복에 orig_list에 아무것도 없을 때 종료

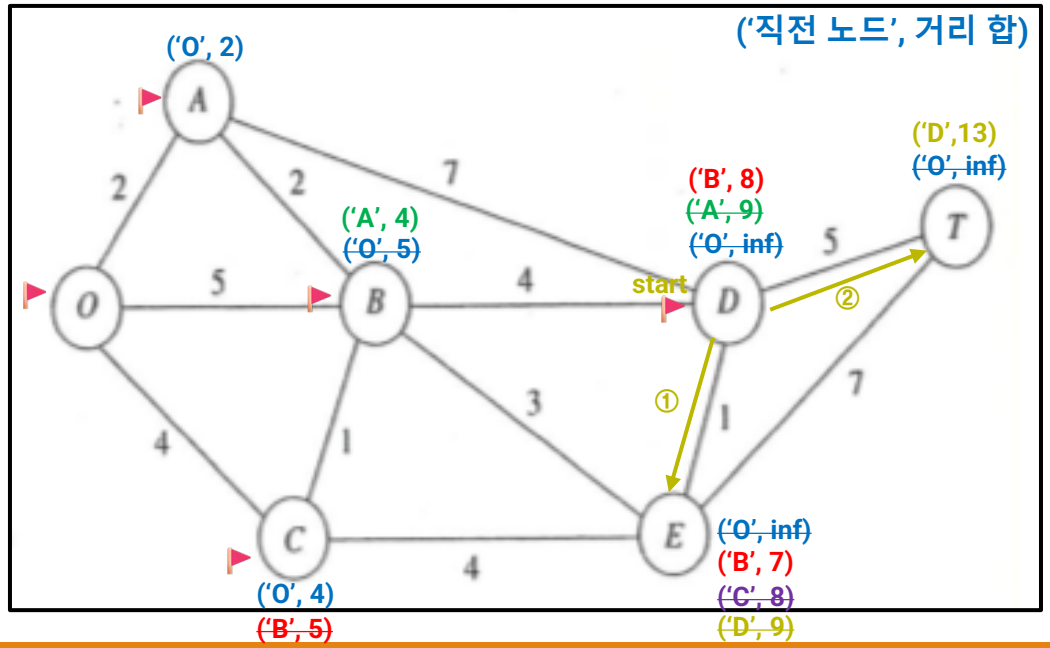



```python
# 목록이 빌 때까지 반복
while orig_list:
    # 시작 노드로부터 최소 거리를 가진 노드를 찾습니다
    node, current_distance = min(orig_list)
    print(f'최소 거리를 가진 노드(min(orig_list)): {min(orig_list)}')

    orig_list.remove((node, current_distance))
    print(f'최소 거리를 가진 노드를 제거한 목록(orig_list): {orig_list}', '₩n')

    # 현재 거리가 저장된 거리보다 큰 경우 건너뜁니다
    if current_distance > distances[node]:
        continue

    # 인접한 노드와 그 거리를 반복하면서 확인
    for next_node, distance in original_dict[node].items():

        # 시작 노드에서 인접 노드까지의 새로운 거리를 계산합니다
        new_distance = distances[node] + distance
        print(f"출발 노드(node='{node}')에서 인접 노드(node='{next_node}') 까지의 거리: {new_distance}")

        # 새로운 거리가 저장된 거리보다 작으면 업데이트합니다
        if new_distance < distances[next_node]:
            distances[next_node] = new_distance

            # 직전 노드를 업데이트합니다
            prev_node[next_node] = node
            print(f'업데이트 된 직전 노드 딕셔너리: {prev_node}')

            # 새로운 거리와 인접한 노드를 목록에 추가합니다
            orig_list.append((next_node, new_distance))   # new_distance -> 노드에 도달하기 위해 더해진 거리가 기록됨

            print(f'업데이트 된 거리: {distances}', '₩n')
    print()
    print(f'업데이트 된 목록(orig_list): {orig_list}')
    print('==========================================한 사이클 종료==========================================', '₩n')
print('==========================================반복 종료==========================================', '₩n')
```

```
최소 거리를 가진 노드(min(orig_list)): ('D', 8)
최소 거리를 가진 노드를 제거한 목록(orig_list): [('D', 9), ('E', 7)]

① 출발 노드(node='D')에서 인접 노드(node='E') 까지의 거리: 9
② 출발 노드(node='D')에서 인접 노드(node='T') 까지의 거리: 13
업데이트 된 직전 노드 딕셔너리: {'O': None, 'A': 'O', 'B': 'A', 'C': 'O', 'D': 'B', 'E': 'B', 'T': 'D'}
업데이트 된 거리: {'O': 0, 'A': 2, 'B': 4, 'C': 4, 'D': 8, 'E': 7, 'T': 13}

업데이트 된 목록(orig_list): [('D', 9), ('E', 7), ('T', 13)]
==========================================한 사이클 종료==========================================
```

전	노드	O(시작)	A	B	C	D	E	T
	거리	0	2	4	4	8	7	inf

후	노드	O(시작)	A	B	C	D	E	T
	거리	0	2	4	4	8	7	13

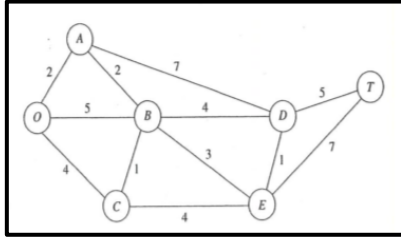


2. Dijkstra 함수

2.2. 반복 : 마지막 반복에 orig_list에 아무것도 없을 때 종료

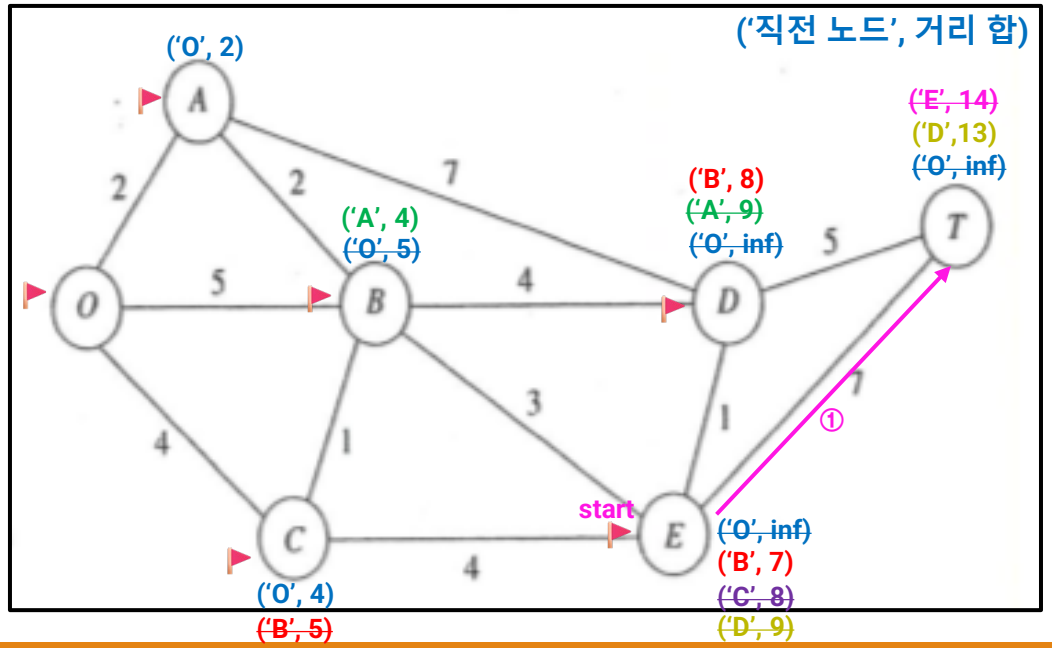
```python
# 목록이 빌 때까지 반복
while orig_list:
    # 시작 노드로부터 최소 거리를 가진 노드를 찾습니다
    node, current_distance = min(orig_list)
    print(f'최소 거리를 가진 노드(min(orig_list)): {min(orig_list)}')

    orig_list.remove((node, current_distance))
    print(f'최소 거리를 가진 노드를 제거한 목록(orig_list): {orig_list}', '₩n')

    # 현재 거리가 저장된 거리보다 큰 경우 건너뜁니다
    if current_distance > distances[node]:
        continue

    # 인접한 노드와 그 거리를 반복하면서 확인
    for next_node, distance in original_dict[node].items():

        # 시작 노드에서 인접 노드까지의 새로운 거리를 계산합니다
        new_distance = distances[node] + distance
        print(f"출발 노드(node='{node}')에서 인접 노드(node='{next_node}') 까지의 거리: {new_distance}")

        # 새로운 거리가 저장된 거리보다 작으면 업데이트합니다
        if new_distance < distances[next_node]:
            distances[next_node] = new_distance

            # 직전 노드를 업데이트합니다
            prev_node[next_node] = node
            print(f'업데이트 된 직전 노드 딕셔너리: {prev_node}')

            # 새로운 거리와 인접한 노드를 목록에 추가합니다
            orig_list.append((next_node, new_distance))   # new_distance -> 노드에 도달하기 위해 더해진 거리가 기록됨

            print(f'업데이트 된 거리: {distances}', '₩n')
    print()
    print(f'업데이트 된 목록(orig_list): {orig_list}')
    print('==========================================한 사이클 종료==========================================', '₩n')
print('==========================================반복 종료==========================================', '₩n')
```

```
최소 거리를 가진 노드(min(orig_list)): ('D', 9)
최소 거리를 가진 노드를 제거한 목록(orig_list): [('E', 7), ('T', 13)]

최소 거리를 가진 노드(min(orig_list)): ('E', 7)
최소 거리를 가진 노드를 제거한 목록(orig_list): [('T', 13)]

① 출발 노드(node='E')에서 인접 노드(node='T') 까지의 거리: 14

업데이트 된 목록(orig_list): [('T', 13)]
==========================================한 사이클 종료==========================================
```



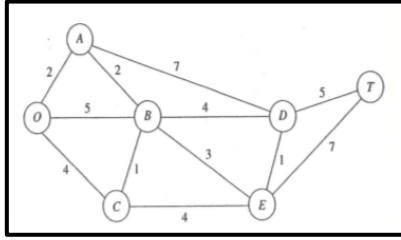
전	노드	O(시작)	A	B	C	D	E	T
	거리	0	2	4	4	8	7	13

후	노드	O(시작)	A	B	C	D	E	T
	거리	0	2	4	4	8	7	13



2. Dijkstra 함수

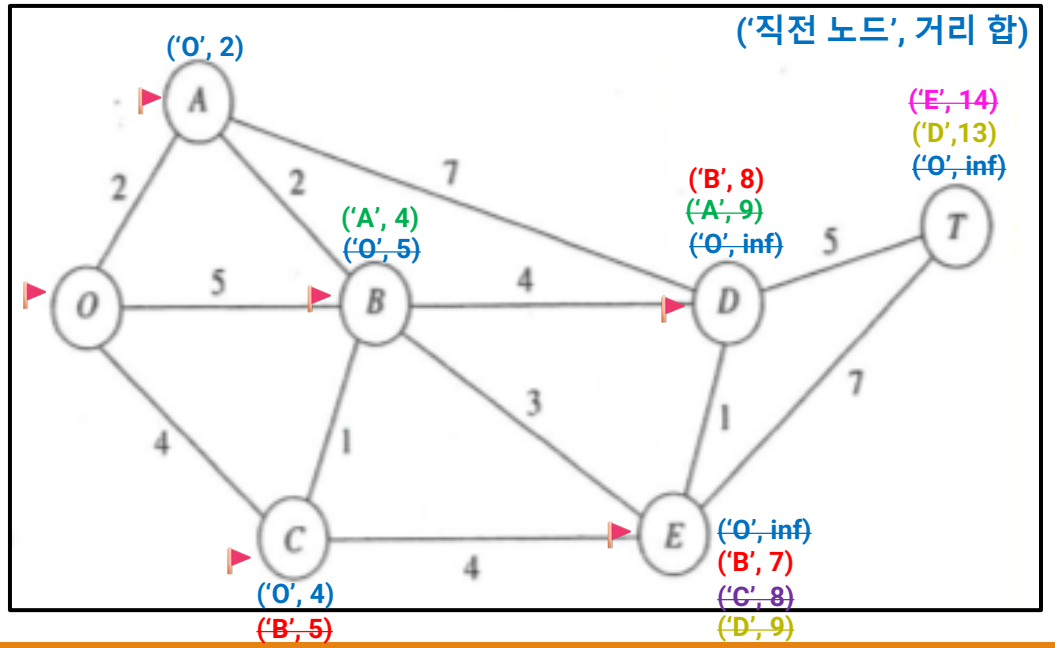
2.2. 반복 : 마지막 반복에 orig_list에 아무것도 없을 때 종료




```python
# 목록이 빌 때까지 반복
while orig_list:
    # 시작 노드로부터 최소 거리를 가진 노드를 찾습니다
    node, current_distance = min(orig_list)
    print(f'최소 거리를 가진 노드(min(orig_list)): {min(orig_list)}')

    orig_list.remove((node, current_distance))
    print(f'최소 거리를 가진 노드를 제거한 목록(orig_list): {orig_list}', '₩n')

    # 현재 거리가 저장된 거리보다 큰 경우 건너뜁니다
    if current_distance > distances[node]:
        continue

    # 인접한 노드와 그 거리를 반복하면서 확인
    for next_node, distance in original_dict[node].items():

        # 시작 노드에서 인접 노드까지의 새로운 거리를 계산합니다
        new_distance = distances[node] + distance
        print(f"출발 노드(node='{node}')에서 인접 노드(node='{next_node}') 까지의 거리: {new_distance}")

        # 새로운 거리가 저장된 거리보다 작으면 업데이트합니다
        if new_distance < distances[next_node]:
            distances[next_node] = new_distance

            # 직전 노드를 업데이트합니다
            prev_node[next_node] = node
            print(f'업데이트 된 직전 노드 딕셔너리: {prev_node}')

            # 새로운 거리와 인접한 노드를 목록에 추가합니다
            orig_list.append((next_node, new_distance))   # new_distance -> 노드에 도달하기 위해 더해진 거리가 기록됨

            print(f'업데이트 된 거리: {distances}', '₩n')
    print()
    print(f'업데이트 된 목록(orig_list): {orig_list}')
    print('==========================================한 사이클 종료==========================================', '₩n')
print('==========================================반복 종료==========================================', '₩n')
```

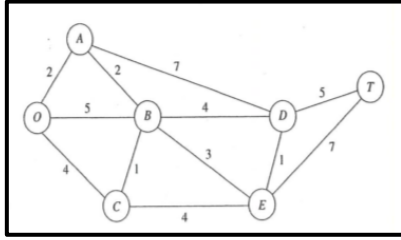
```
최소 거리를 가진 노드(min(orig_list)): ('T', 13)
최소 거리를 가진 노드를 제거한 목록(orig_list): []

업데이트 된 목록(orig_list): []
==========================================한 사이클 종료==========================================

==========================================반복 종료==========================================
```

최종	노드	O(시작)	A	B	C	D	E	T
	거리	0	2	4	4	8	7	13



2. Dijkstra 함수

2.3. 최종 결과 출력 : 각 노드에 대한 최단 경로와 최단 거리 출력

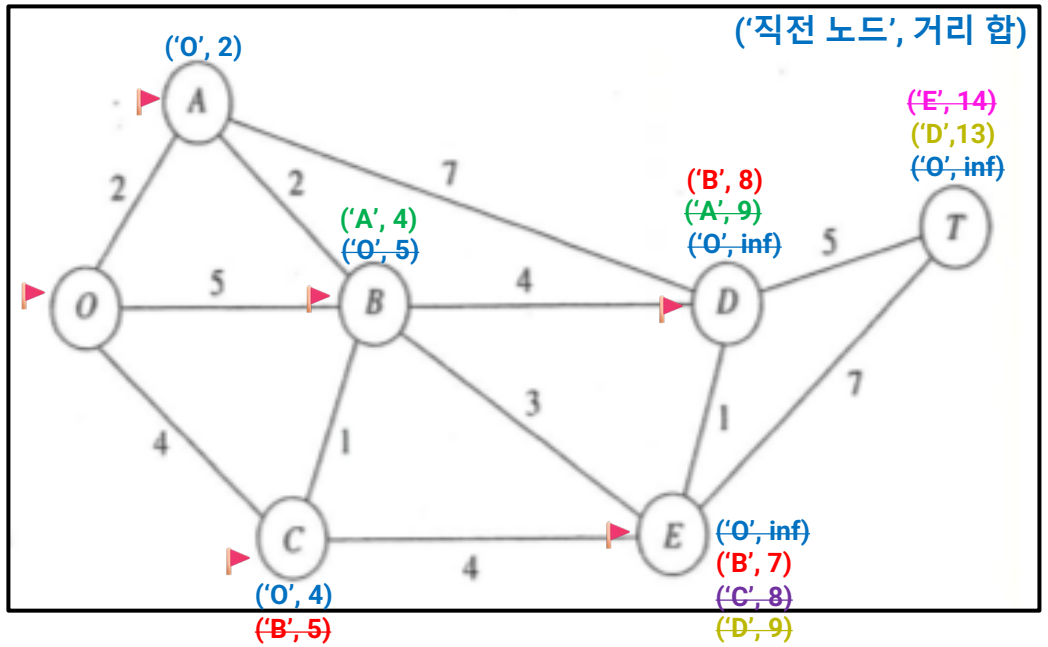



```
    # 각 노드까지의 최단 거리와 직전 노드를 출력합니다
    for node, distance in distances.items():
        if prev_node[node] is not None:
            path = get_shortest_path(prev_node, node)
            shortest_path = '->'.join(path)
            print(f"노드 {node}: 최단 거리 = {distance}, 최단 경로 = {shortest_path}")

def get_shortest_path(prev_node, current_node):
    path = []
    while current_node is not None:
        path.insert(0, current_node)
        current_node = prev_node[current_node]
    return path
```

```
[ ]  result = dijkstra('O', original_dict)
     print(result)
```

```
노드 A: 최단 거리 = 2, 최단 경로 = O->A
노드 B: 최단 거리 = 4, 최단 경로 = O->A->B
노드 C: 최단 거리 = 4, 최단 경로 = O->C
노드 D: 최단 거리 = 8, 최단 경로 = O->A->B->D
노드 E: 최단 거리 = 7, 최단 경로 = O->A->B->E
노드 T: 최단 거리 = 13, 최단 경로 = O->A->B->D->T
None
```

최종	노드	O(시작)	A	B	C	D	E	T
	거리	0	2	4	4	8	7	13



감사합니다

https://colab.research.google.com/drive/16pd72MBlugUGjDyZOfADHbgCuwhOC5SD?usp=sharing