R E P O R T

실습 및 과제 4

과목명	알고리즘및실습
분반	2 분 반
교수	한 연 희
학번	2020136129
이름	최 수 연
제출일	2022년 6월 3일 금요일

목차

서론

본 과제에서는 Prim 알고리즘과 Kruskal 알고리즘을 통한 최소 비용 신장 트리 알고리즘을 구현함으로써 Prim 알고리즘과 Kruskal 알고리즘에 대해 이해할 수 있다.

또한, Dijkstra 알고리즘과 Bellman-ford 알고리즘, floyd-warshall 알고리즘을 통해 최단 경로 알고리즘을 구현하고, 이때 주어진 그래프에 음의 사이클이 있는 경우를 고려하여 Bellman-ford 알고리즘을 구현하여 음의 사이클 발견 시 예외 처리가 되도록 함으로써 Dijkstra 알고리즘과 Bellman-ford 알고리즘, floyd-warshall 알고리즘에 대해 이해할 수 있고, 음의 사이클 예외 처리에 대한 이해 및 구현 능력을 향상시킬 수 있다.

그리고 위 알고리즘에 대한 트리를 구하는 과정을 직접 그려 시각화해봄으로써 각 알고리즘에 대해 더 쉽게 이해할 수 있다.

연쇄 행렬들의 곱을 계산하는 최적 순서 및 비용을 계산함으로써 재귀적 속성을 올바르게 이해하고, 그에 따른 Dynamic Programming 풀이 절차를 이해할 수 있다.

허프만 알고리즘을 구하는 과정을 제시하고, 직접 구한 최적 이진 트리를 기반으로 허프만 코드를 제시함으로써 허프만 코드에 대해 이해하고 설명할 수 있다.

본론

[문제 1] 최소 비용 신장 트리 알고리즘 구현

```
import sys

class Graph:
    def __init__(self, adjacency_list, directed=False):
        self.adjacency_list = adjacency_list # 인접리스트
        self.nodes = set() # 노드
        self.edges = set() # 간선
        self.num_nodes = 0 # 노드 개수
        self.num_edges = 0 # 간선 개수

        if directed: # directed가 True이면, 인접 리스트의 노드를 하나씩 순차적으로 반복
            for node in adjacency_list:
                for adjacency_node in adjacency_list[node]:
                    weight = adjacency_list[node][adjacency_node]
                    self._add_node_and_edge(node, adjacency_node, weight)
        else: # directed가 False이면,
            for node in adjacency_list:
                for adjacency_node in adjacency_list[node]:
                    edge_exist_conditions = [
                        (node, adjacency_node, adjacency_list[node][adjacency_node]) in self.edges,
                        (adjacency_node, node, adjacency_list[adjacency_node][node]) in self.edges,
                    ]
                    if any(edge_exist_conditions):
                        assert adjacency_list[node][adjacency_node] == adjacency_list[adjacency_node][node]
                    else:
                        weight = adjacency_list[node][adjacency_node]
                        self._add_node_and_edge(node, adjacency_node, weight)

    def _add_node_and_edge(self, s, d, weight):
        if s not in self.nodes:
            self.nodes.add(s)
            self.num_nodes += 1

        if d not in self.nodes:
            self.nodes.add(d)
```

```
            self.num_nodes += 1

        self.edges.add((s, d, weight))
        self.num_edges += 1


class DisjointSet:
    def __init__(self, vertices):
        self.vertices = vertices
        self.parent = {}
        self.rank = {}

    def make_set(self):
        for v in self.vertices:
            self.parent[v] = v
            self.rank[v] = 0

    def find_set(self, item):
        if self.parent[item] != item:
            self.parent[item] = self.find_set(self.parent[item])

        return self.parent[item]

    def union(self, x, y):
        xroot = self.find_set(x)
        yroot = self.find_set(y)

        # 이곳에 코딩을 추가하세요. (약 5~7라인)
        if self.rank[xroot] > self.rank[yroot]:
            self.parent[yroot] = xroot # 랭크가 낮은 노드의 부모를 수정
        else:
            self.parent[xroot] = yroot # 랭크가 낮은 노드의 부모를 수정
            if self.rank[xroot] == self.rank[yroot]: # 랭크가 동일할 땐
                self.rank[yroot] += 1 # 붙임을 받은 노드의 랭크를 1 증가

class SpanningTree:
    def __init__(self, graph):
        self.graph = graph
        self.prim_tree = {}
        self.kruskal_tree = []

    def print_solution(self, algorithm=None):
        if algorithm == "prim":
            print("*** Prim Solution ***")
```

```
            for v, u in self.prim_tree.items():
                print("{0} - {1}".format(u, v))
        elif algorithm == "kruskal":
            print("*** Kruskal Solution ***")
            for u, v in self.kruskal_tree:
                print("{0} - {1}".format(u, v))
        else:
            raise ValueError()

    def prim(self, start_node='1'):
        S = set()
        d = {}
        for node in self.graph.nodes:
            d[node] = sys.maxsize
        d[start_node] = 0

        while len(S) != len(self.graph.nodes):
            V_minus_S = self.graph.nodes - S

                # 이곳에 코딩을 추가하세요. (약 8~10라인)
            node = extract_min(V_minus_S, d) # 가장 작은 node 구함
            S.add(node) # S 집합에 node 추가
            for i in graph.adjacency_list[node]: # node의 인접한 정점들을 한 번씩
방문
                if i in V_minus_S and graph.adjacency_list[node][i] < d[i]: # 만약
외부 정점이고 인접 정점이 d[i]보다 작으면
                    d[i] = graph.adjacency_list[node][i] # d[i]에 가중치 값 대입
                    self.prim_tree[i] = node # node를 tree에 저장

    def extract_min(self, V_minus_S, d):
        min = sys.maxsize
        selected_node = None

        # 이곳에 코딩을 추가하세요. (약 5~7라인)
        for node in V_minus_S: # 외부 정점 중 가중치가 가장 작은 node 고르기
            if d[node] < min: # 현재 d[node]가 min보다 작으면
                min = d[node] # min에 작은 값 갱신
                selected_node = node # 해당 node를 selected_node에 대입
        return selected_node # 가장 작은 node 반환

    def kruskal(self):
        ds = DisjointSet(self.graph.nodes)
        ds.make_set()
```

```
        e = 0
        sorted_edges = sorted(self.graph.edges, key=lambda edge: edge[2]) # 간선의 가중치 오름차순으로 정렬

        while e < self.graph.num_nodes - 1:
                # 이곳에 코딩을 추가하세요. (약 9~11라인)
            min_edge = sorted_edges.pop(0) # 가장 작은 edge를 pop함
            if ds.find_set(min_edge[0]) != ds.find_set(min_edge[1]): # 두 정점이 서로 다른 집합에 존재할 경우
                ds.union(min_edge[0], min_edge[1]) # 하나의 집합으로 묶음
                self.kruskal_tree.append((min_edge[0], min_edge[1])) # tree에 넣음
                e += 1 # 다음 계산을 위해 e에 1을 추가(while문이므로)

if __name__ == "__main__":
    adjacency_list = {
        '1': {'2': 8, '3': 9, '4': 11},
        '2': {'1': 8, '5': 14},
        '3': {'1': 9, '4': 13, '5': 5, '6': 12},
        '4': {'1': 11, '3': 13, '6': 9, '7': 8},
        '5': {'2': 14, '3': 5},
        '6': {'3': 12, '4': 9, '7': 7},
        '7': {'4': 8, '6': 7}
    }

    graph = Graph(adjacency_list=adjacency_list)
    print("[Graph] number of nodes: {0}, number of edges: {1}".format(graph.num_nodes, graph.num_edges))
    print(graph.edges)
    st = SpanningTree(graph=graph)
    st.prim(start_node='1')
    st.print_solution(algorithm="prim")

    st.kruskal()
    st.print_solution(algorithm="kruskal")
```

```
[Graph] number of nodes: 7, number of edges: 10
{('3', '5', 5), ('1', '4', 11), ('2', '5', 14), ('3', '4', 13), ('1', '2', 8), ('6', '7', 7), ('4', '7', 8), ('1', '3', 9), ('3', '6', 12), ('4', '6', 9)}
*** Prim Solution ***
1 - 2
1 - 3
1 - 4
3 - 5
7 - 6
4 - 7
*** Kruskal Solution ***
3 - 5
6 - 7
1 - 2
4 - 7
1 - 3
1 - 4
```

[문제 2] 최단 경로 알고리즘 구현

```
import sys

class Graph:
    def __init__(self, adjacency_list, directed=False):
        self.adjacency_list = adjacency_list
        self.nodes = set()
        self.edges = set()
        self.num_nodes = 0
        self.num_edges = 0

        if directed:
            for node in adjacency_list:
                for adjacency_node in adjacency_list[node]:
                    weight = adjacency_list[node][adjacency_node]
                    self._add_node_and_edge(node, adjacency_node, weight)
        else:
            for node in adjacency_list:
                for adjacency_node in adjacency_list[node]:
                    edge_exist_conditions = [
                        (node, adjacency_node, adjacency_list[node][adjacency_no
de]) in self.edges,
                        (adjacency_node, node, adjacency_list[adjacency_node][no
de]) in self.edges,
                    ]
                    if any(edge_exist_conditions):
                        assert adjacency_list[node][adjacency_node] == adjacency
_list[adjacency_node][node]
                    else:
                        weight = adjacency_list[node][adjacency_node]
                        self._add_node_and_edge(node, adjacency_node, weight)

    def _add_node_and_edge(self, s, d, weight):
        if s not in self.nodes:
            self.nodes.add(s)
            self.num_nodes += 1

        if d not in self.nodes:
            self.nodes.add(d)
            self.num_nodes += 1

        self.edges.add((s, d, weight))
        self.num_edges += 1
```

```
class ShortestPath:
    def __init__(self, graph):
        self.graph = graph
        self.dijkstra_tree = {}
        self.bellman_ford_tree = {}
        self.floyd_warshall_p = {}

    def print_solution(self, algorithm=None):
        if algorithm == "dijkstra":
            print("*** Dijkstra Solution ***")
            for v, u in self.dijkstra_tree.items():
                print("{0} -> {1}".format(u, v))
        elif algorithm == "bellman_ford":
            print("*** Bellman Ford Solution ***")
            for v, u in self.bellman_ford_tree.items():
                print("{0} -> {1}".format(u, v))
        else:
            raise ValueError()

    def print_floyd_warshall_path(self, q, r):
        if self.floyd_warshall_p[q][r] == 0:
            print(r, end=" ")
        else:
            self.print_floyd_warshall_path(q, self.floyd_warshall_p[q][r])
            self.print_floyd_warshall_path(self.floyd_warshall_p[q][r], r)

    def dijkstra(self, start_node='1'):
        S = set()
        d = {}
        for node in self.graph.nodes:
            d[node] = sys.maxsize
        d[start_node] = 0

        while len(S) != len(self.graph.nodes):
                # 이곳에 코딩을 추가하세요. (약 8~10라인)
            V_minus_S = self.graph.nodes - S

            node = self.extract_min(V_minus_S, d) # 가장 작은 node 구함
            S.add(node) # S 집합에 node 추가
            for i in graph.adjacency_list[node]: # node의 인접한 정점들을 한 번씩
방문
                if i in V_minus_S and graph.adjacency_list[node][i] + d[node] < d
```

[i]:

```
    def extract_min(self, V_minus_S, d):
        min = sys.maxsize
        selected_node = None

        # 이곳에 코딩을 추가하세요. (약 5~7라인)

        return selected_node

    def bellman_ford(self, start_node='1'):
        d = {}
        for node in self.graph.nodes:
            d[node] = sys.maxsize
        d[start_node] = 0

        # 이곳에 코딩을 추가하세요. (약 5~7라인)


        return self.check_negative_cycle(d)

    def check_negative_cycle(self, d):
        is_ok = True

        # 이곳에 코딩을 추가하세요. (약 5~7라인)


        return is_ok

    def floyd_warshall(self):
        d = {}
        for node in self.graph.nodes:
            d[node] = {}
            self.floyd_warshall_p[node] = {}

        for i in self.graph.nodes:
            for j in self.graph.nodes:
                if j in self.graph.adjacency_list[i]:
                    d[i][j] = self.graph.adjacency_list[i][j]
                else:
                    d[i][j] = sys.maxsize

                self.floyd_warshall_p[i][j] = 0
```

```
        # 이곳에 코딩을 추가하세요. (약 6~8라인)


if __name__ == "__main__":
    ##########################################################
    ##########################################################
    ##                dijkstra algorithm test                 ##
    ##########################################################
    ##########################################################
    dijkstra_adjacency_list = {
        '1': {'2': 8, '3': 9, '4': 11},
        '2': {'5': 10},
        '3': {'2': 6, '5': 2, '4': 3},
        '4': {'6': 9, '7': 8},
        '5': {'8': 2},
        '6': {'3': 12, '8': 5},
        '7': {'6': 7},
        '8': {'7': 4}
    }

    graph = Graph(adjacency_list=dijkstra_adjacency_list, directed=True)
    print("[Graph] number of nodes: {0}, number of edges: {1}".format(graph.num
_nodes, graph.num_edges))
    print(graph.edges)
    sp = ShortestPath(graph=graph)
    sp.dijkstra(start_node='1')
    sp.print_solution(algorithm="dijkstra")

    print()

    ##########################################################
    ##########################################################
    ##          bellman-ford algorithm test - 1             ##
    ##########################################################
    ##########################################################
    bellman_ford_adjacency_list = {
        '1': {'2': 8, '3': 9, '4': 11},
        '2': {'5': 10},
        '3': {'2': -15, '5': 1, '4': 3},
        '4': {'6': 9, '7': 8},
        '5': {'8': 2},
        '6': {'3': 12, '8': 5},
        '7': {'6': -7},
```

```
        '8': {'7': 4}
    }

    graph = Graph(adjacency_list=bellman_ford_adjacency_list, directed=True)
    print("[Graph] number of nodes: {0}, number of edges: {1}".format(graph.num
_nodes, graph.num_edges))
    print(graph.edges)
    sp = ShortestPath(graph=graph)
    is_ok = sp.bellman_ford(start_node='1')
    if is_ok:
        sp.print_solution(algorithm="bellman_ford")

    print()

    ############################################################
    ############################################################
    ##           bellman-ford algorithm test - 2              ##
    ############################################################
    ############################################################
    bellman_ford_adjacency_list_with_negative_cycle = {
        '1': {'2': 8, '3': 9, '4': 11},
        '2': {'5': 10},
        '3': {'2': -15, '5': 1, '4': 3},
        '4': {'6': 9, '7': 8},
        '5': {'8': 2},
        '6': {'3': -12, '8': 5},
        '7': {'6': -7},
        '8': {'7': 4}
    }

    graph = Graph(adjacency_list=bellman_ford_adjacency_list_with_negative_cycl
e, directed=True)
    print("[Graph] number of nodes: {0}, number of edges: {1}".format(graph.num
_nodes, graph.num_edges))
    print(graph.edges)
    sp = ShortestPath(graph=graph)
    is_ok = sp.bellman_ford(start_node='1')
    if is_ok:
        sp.print_solution(algorithm="bellman_ford")

    print()

    ############################################################
    ############################################################
```

```
    ##              floyd-warshall algorithm test            ##
    ###########################################################
    ###########################################################
    floyd_warshall_adjacency_list = {
        '1': {'2': 1, '4': 1, '5': 5},
        '2': {'1': 9, '3': 3, '4': 2},
        '3': {'4': 4},
        '4': {'3': 2, '5': 3},
        '5': {'1': 3}
    }

    graph = Graph(adjacency_list=floyd_warshall_adjacency_list, directed=True)
    print("[Graph] number of nodes: {0}, number of edges: {1}".format(graph.num
_nodes, graph.num_edges))
    print(graph.edges)
    sp = ShortestPath(graph=graph)
    is_ok = sp.floyd_warshall()

    #SOURCE: 2, DESTINATION: 1
    print('2', end=" ")
    sp.print_floyd_warshall_path(q='2', r='1')
    print()

    # SOURCE: 5, DESTINATION: 3
    print('5', end=" ")
    sp.print_floyd_warshall_path(q='5', r='3')
    print()

    # SOURCE: 2, DESTINATION: 4
    print('2', end=" ")
    sp.print_floyd_warshall_path(q='2', r='4')
    print()
```

[문제 3] 연쇄 행렬들의 곱인 A1 × A2 × A3 × A4 × A5를 계산하는 최적 순서와 그 비용을 구하라.

• 주어진 행렬들의 크기는 다음과 같다.

A_1	×	A_2	×	A_3	×	A_4	×	A_5
10 × 4		4 × 5		5 × 20		20 × 2		2 × 50

풀이) $A_i, A_{i+1}, \cdots, A_j$ 를 곱하는 최소비용을 $m[i,j]$라고 하면,

$$m[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min\{m[i,k] + m[k+1,j] + P_{i-1}P_kP_j\} & \text{if } i<j \end{cases} \quad (i \le k \le j-1)$$

을 만족한다.

주어진식 $A_1 \times A_2 \times A_3 \times A_4 \times A_5$ 을 행렬 그래프를 이용해 풀면 다음과같다.
(10×4, 4×5, 5×20, 20×2, 2×50)

i \ j	1	2	3	4	5
1	0	①	③	⑥	⑩
2	0	0	②	⑤	⑨
3	0	0	0	④	⑧
4	0	0	0	0	⑦
5	0	0	0	0	0

① $m[1,2]$, 즉 $A_1 \times A_2$의 최소비용은,
$10 \times 4 \times 5 = 200$ 이다.
따라서, ①은 200이다.

② $m[2,3]$, 즉 $A_2 \times A_3$의 최소비용은,
$4 \times 5 \times 20 = 400$이다.
따라서 ②는 400이다.

③ $m[1,3]$, 즉 $A_1 \times \cdots \times A_3$의 비용은 총 2가지 경우의 수가 나올수있다
$(A_1A_2)A_3$ 이거나, $A_1(A_2A_3)$ 중 하나인데,
A_1A_2는 ①, A_2A_3는 ②에서 구하였으므로
①, ②중 더 작은 값을 선택해 계산하면 된다.
따라서, ① < ② 이므로 ①을 선택하여 계산한다.
즉, $(A_1A_2)A_3$ 이므로, 10×5 와 5×20을 계산하면,
$10 \times 5 \times 20 = 1000$ 이다.
이값에 ①의 값을 누적하여 더해면 ③은 $1000 + 200 = 1200$ 이다.

④ m[3,4], 즉 $A_3 \times A_4$의 최소비용은,
5×20×2 = 200 이다.
따라서 ④는 200 이다.

⑤ m[2,4], 즉, $A_2 \times \cdots \times A_4$의 최소비용은,
②>④ 이므로, ④를 선택하여 계산한다.
$A_2(A_3A_4)$에서 A_2는 4×5, A_3A_4는 5×20이므로,
4×5×2 = 400이고,
④에 40을 더한값이 ⑤가되므로
200+40=240.
따라서 ⑤는 240이다.

⑥ m[1,4], 즉 $A_1 \times \cdots \times A_4$의 최소비용은,
③>⑤ 이므로, ⑤를 선택하여 계산한다
$A_1(A_2(A_3A_4))$에서 A_1은 10×4, $A_2(A_3A_4)$는 4×20이므로,
10×4×2=800이고, ⑤에 80을 누적하여 더한 값이 ⑥이되므로,
따라서 ⑥은 240+80=320이다.

⑦ m[4,5], 즉 $A_4 \times A_5$의 최소비용은,
20×2×50 = 2000 이다.
따라서 ⑦은 2000 이다.

⑧ m[3,5], 즉 $A_3 \times \cdots \times A_5$의 최소비용은,
④<⑦ 이므로, ④를 선택하여 계산한다.
$(A_3A_4)A_5$에서 A_3A_4는 5×20이고, A_5는 2×50이므로,
5×2×50 = 500이고, ④에 500을 누적하여 더한값이 ⑧이되므로,
따라서 ⑧은 200+500=700 이다.

⑨ m[2,5], 즉 $A_2 \times \cdots \times A_5$의 최소비용은,
⑤<⑧ 이므로, ⑤를 선택하여 계산한다.
$(A_2(A_3A_4))A_5$에서 $A_2(A_3A_4)$는 4×2이고, A_5는 2×50이므로,
4×2×50=400이고, ⑤에 400을 누적하여 더한값이 ⑨이므로,
따라서, ⑨는 240+400=640이다.

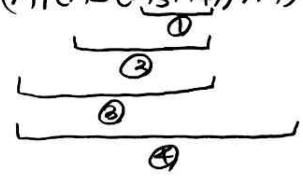
⑩ m[1,5], 즉 $A_1 \times \cdots \times A_5$의 최소비용은,
⑥<⑨ 이므로, ⑥을 선택하여 계산한다.
$(A_1(A_2(A_3A_4)))A_5$에서 $(A_1(A_2(A_3A_4)))$는 10×2이고, A_5는 2×50이므로,
10×2×50=1000이고, ⑥에 1000을 누적하여 더한값이 ⑩이므로,
따라서 ⑩는 320+1000=1320이다.

따라서 그래프로 나타내면 다음과 같다.

i \ j	1	2	3	4	5
1	0	200	1200	320	1320
2	0	0	400	240	640
3	0	0	0	200	700
4	0	0	0	0	2000
5	0	0	0	0	0

따라서 $A_1 \times A_2 \times A_3 \times A_4 \times A_5$의 최적순서와 최소비용은 다음과 같다.

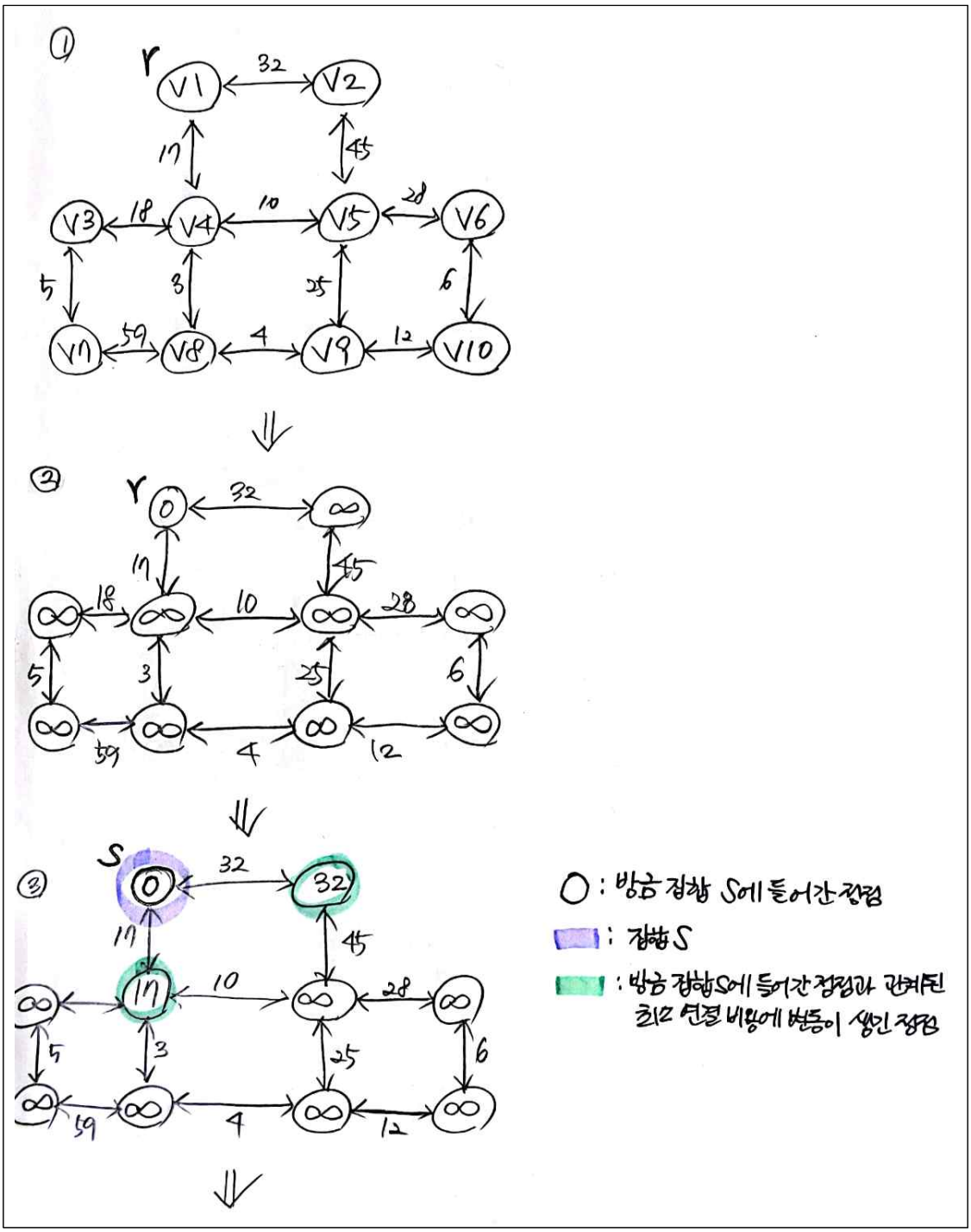
최적순서: $(A_1(A_2(A_3A_4)))A_5$

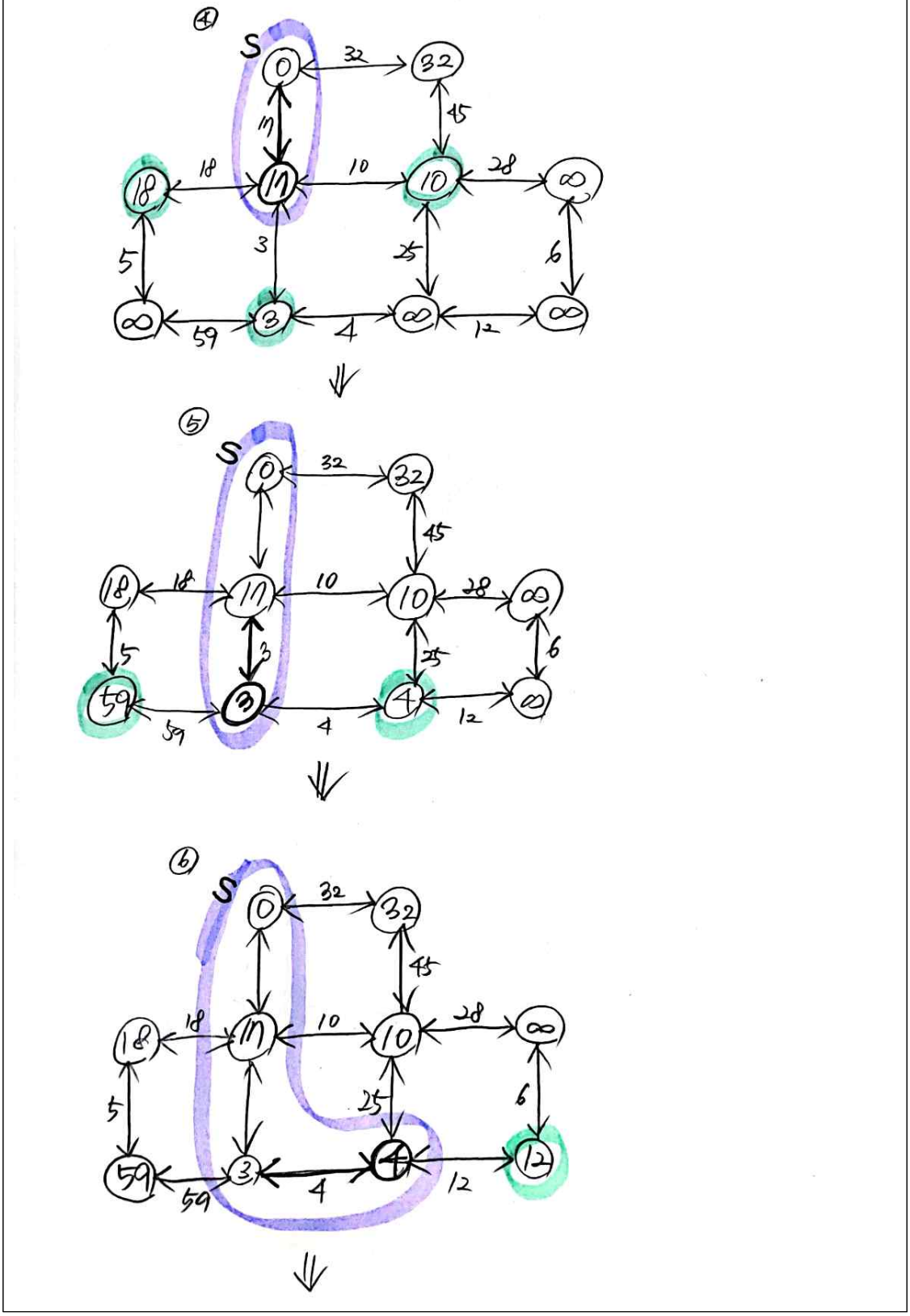


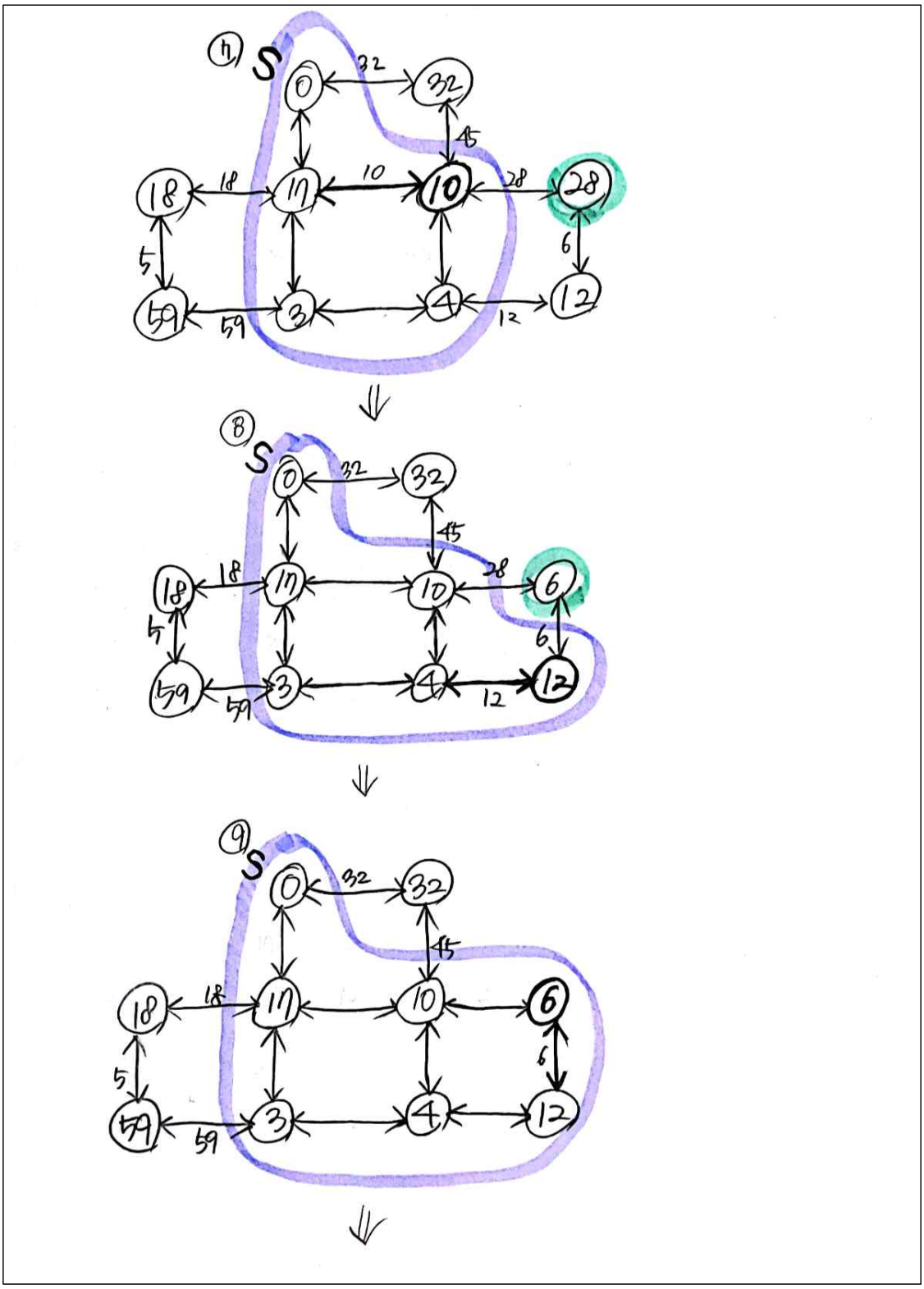
최소비용: m[1,5] = 1320

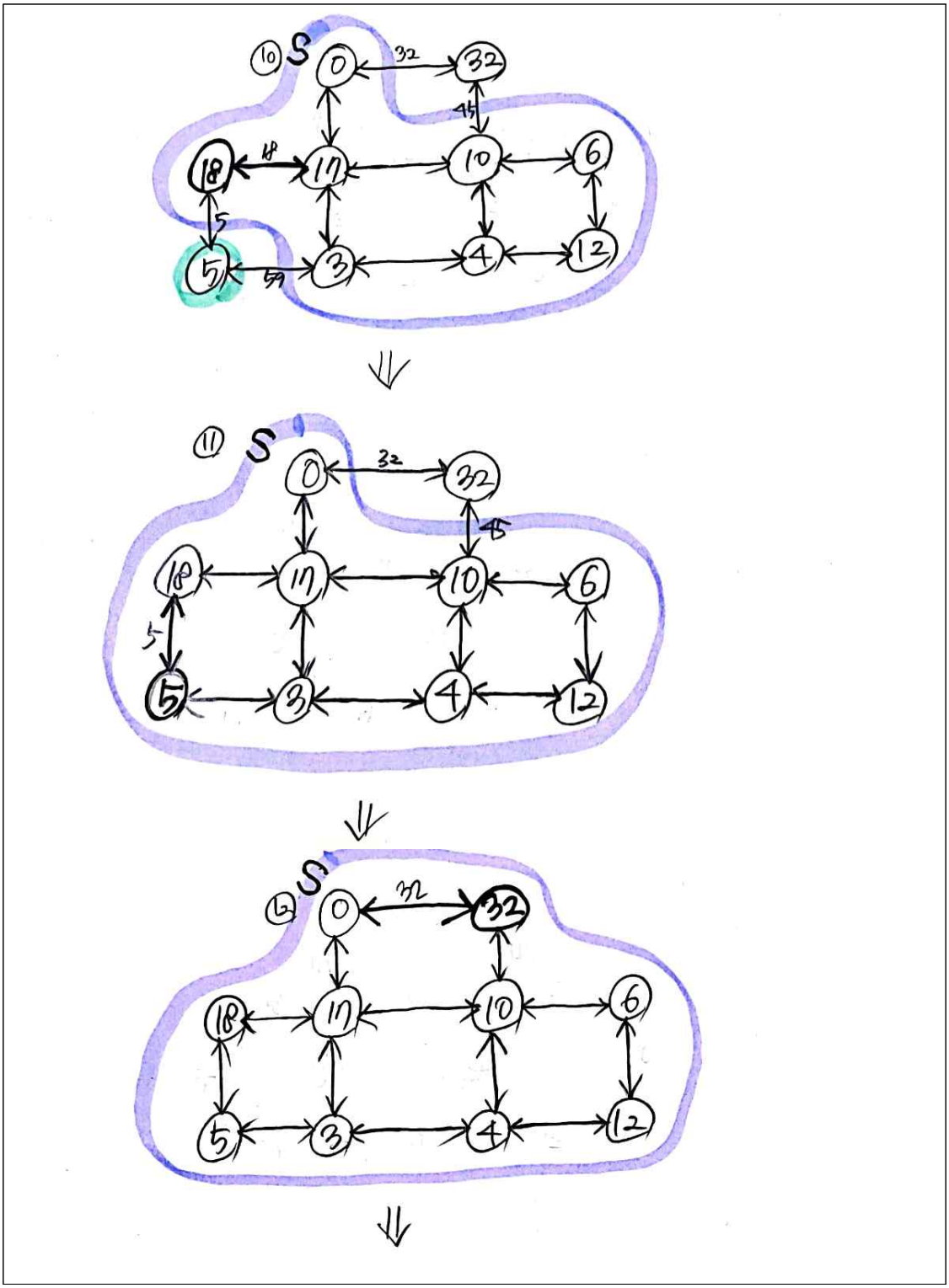
[문제 4]

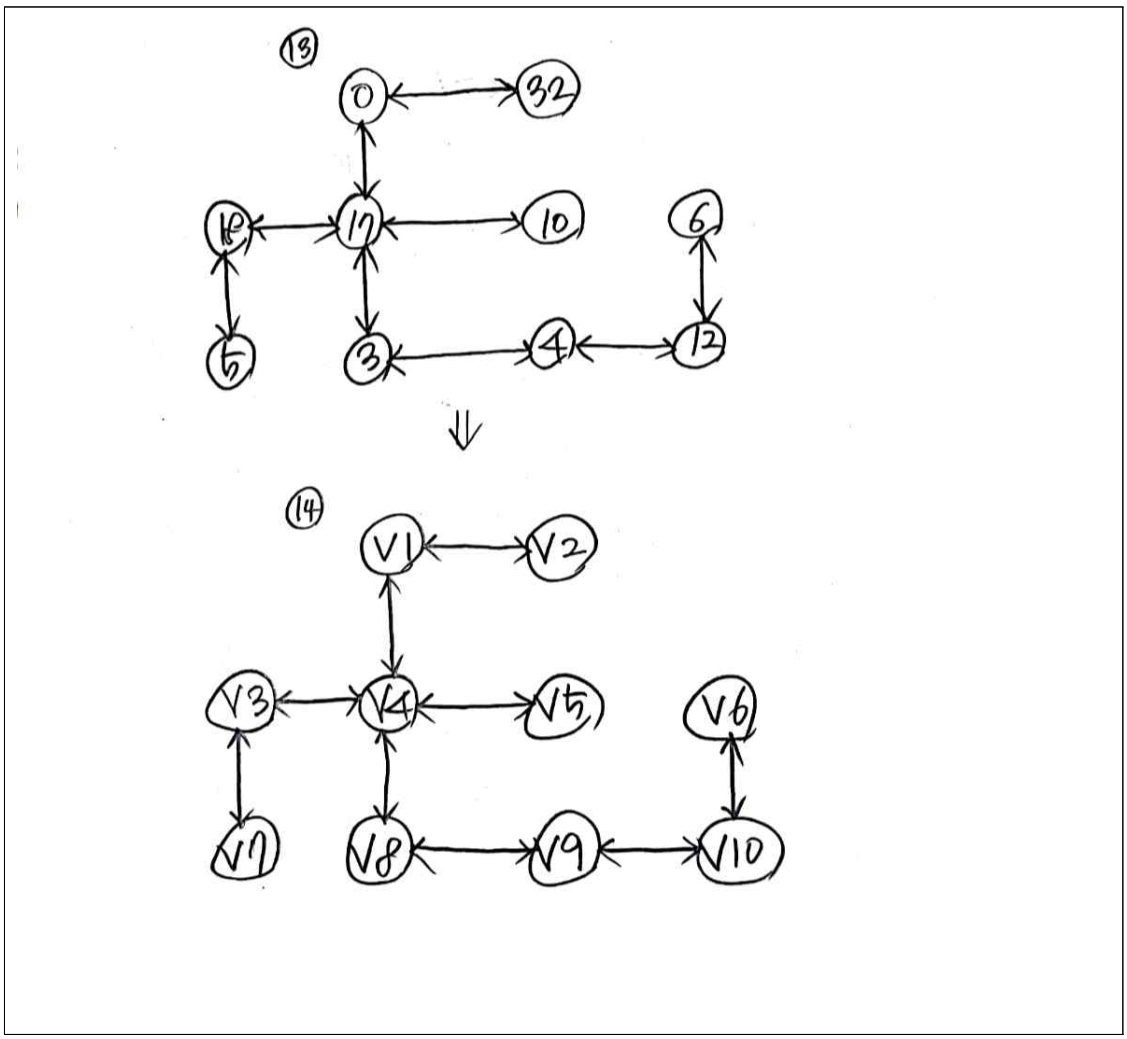
[문제 4-1] Prim 알고리즘을 이용하여 위 그래프의 최소비용 신장 트리를 구하는 과정을 교재 [그림 10-16]과 유사한 형태로 그려 제시하시오.



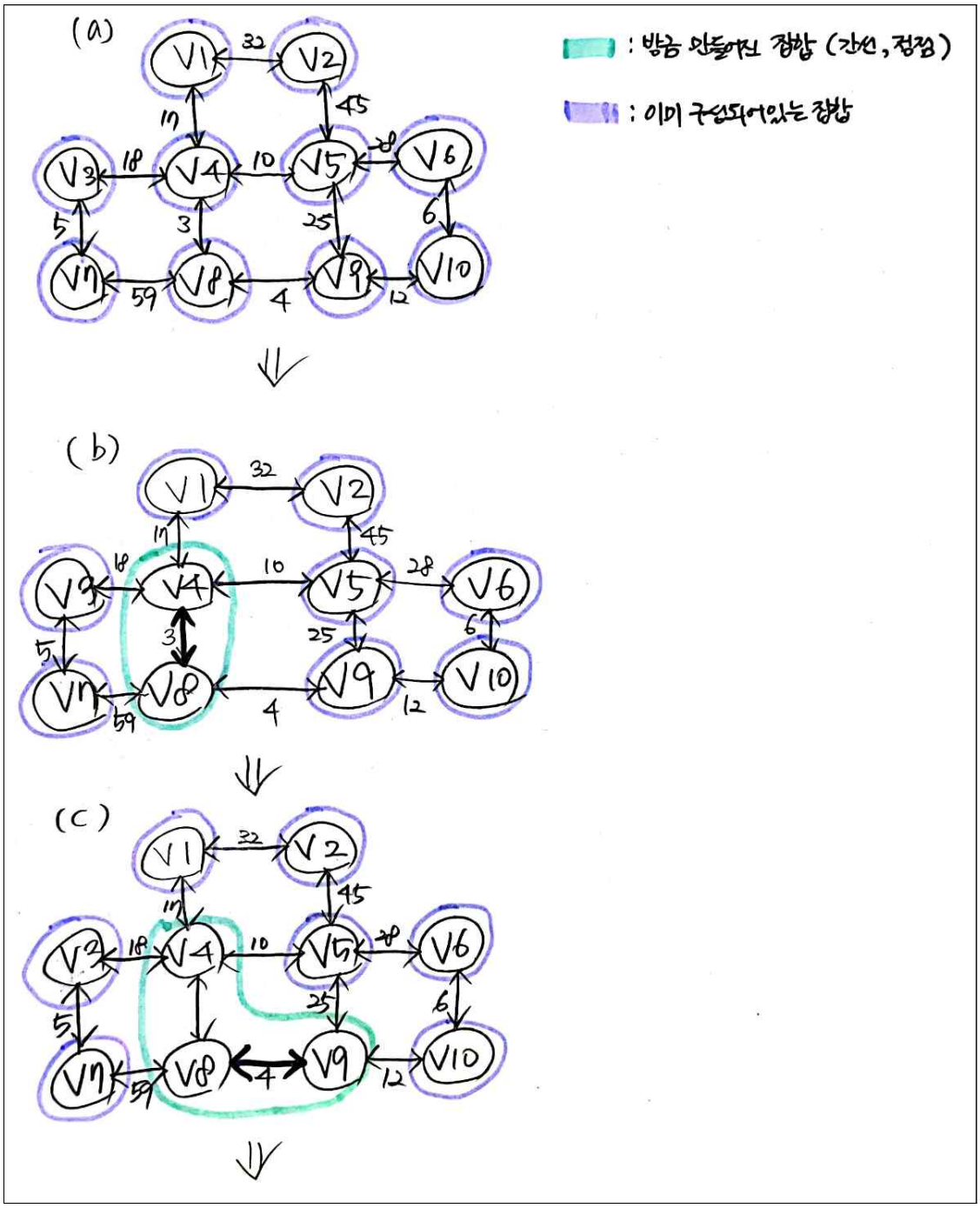

④
S
0
32
32
11
45
18
18
10
28
∞
5
3
25
6
∞
59
3
4
∞
12
∞
⑤
S
0
32
32
45
18
18
11
10
10
28
∞
5
3
25
6
59
3
4
4
12
∞
59
⑥
S
0
32
32
45
18
18
11
10
10
28
∞
5
25
6
59
3
4
4
12
59

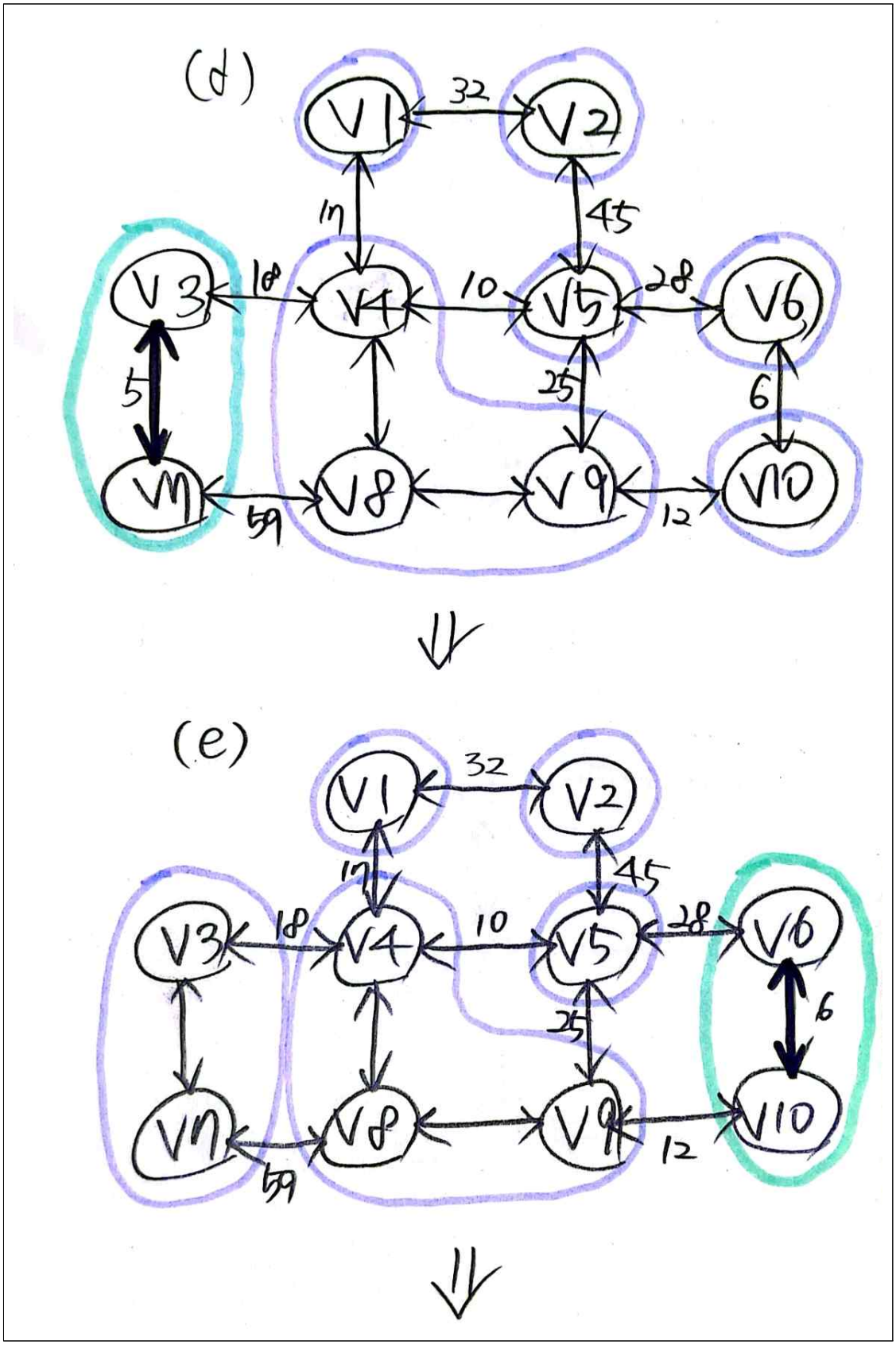

⑦
S
0
32
32
45
18
18
17
10
10
28
28
5
6
59
59
3
4
12
12
⑧
S
0
32
32
45
18
18
17
10
28
6
5
6
59
59
3
4
12
12
⑨
S
0
32
32
45
18
18
17
10
6
5
6
59
59
3
4
12

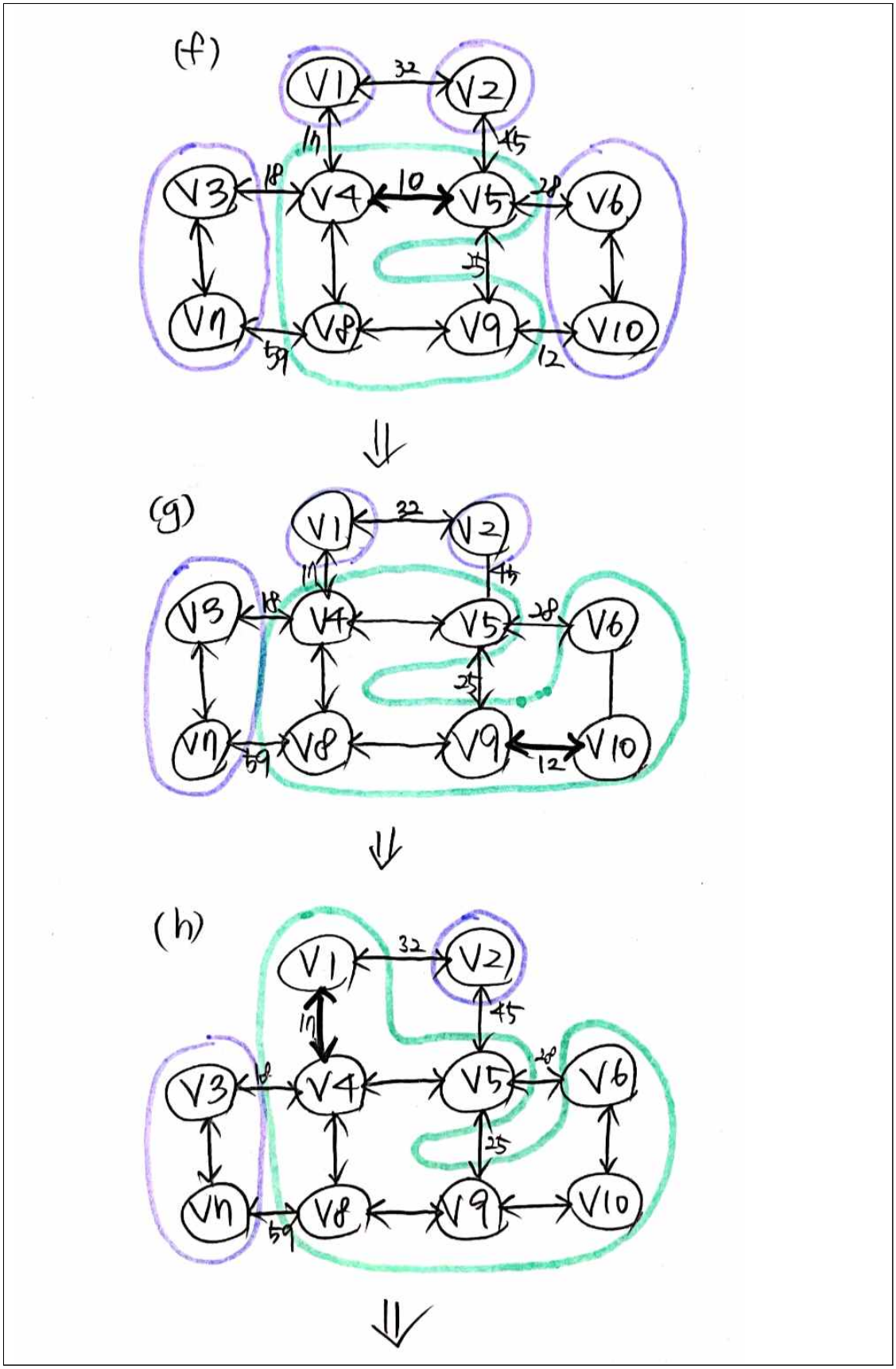

⑩ S
0
32
32
45
18
18
11
10
6
5
5
5
3
4
12
⑪ S
0
32
32
45
18
11
10
6
5
5
3
4
12
⑫ S
0
32
32
18
11
10
6
5
3
4
12

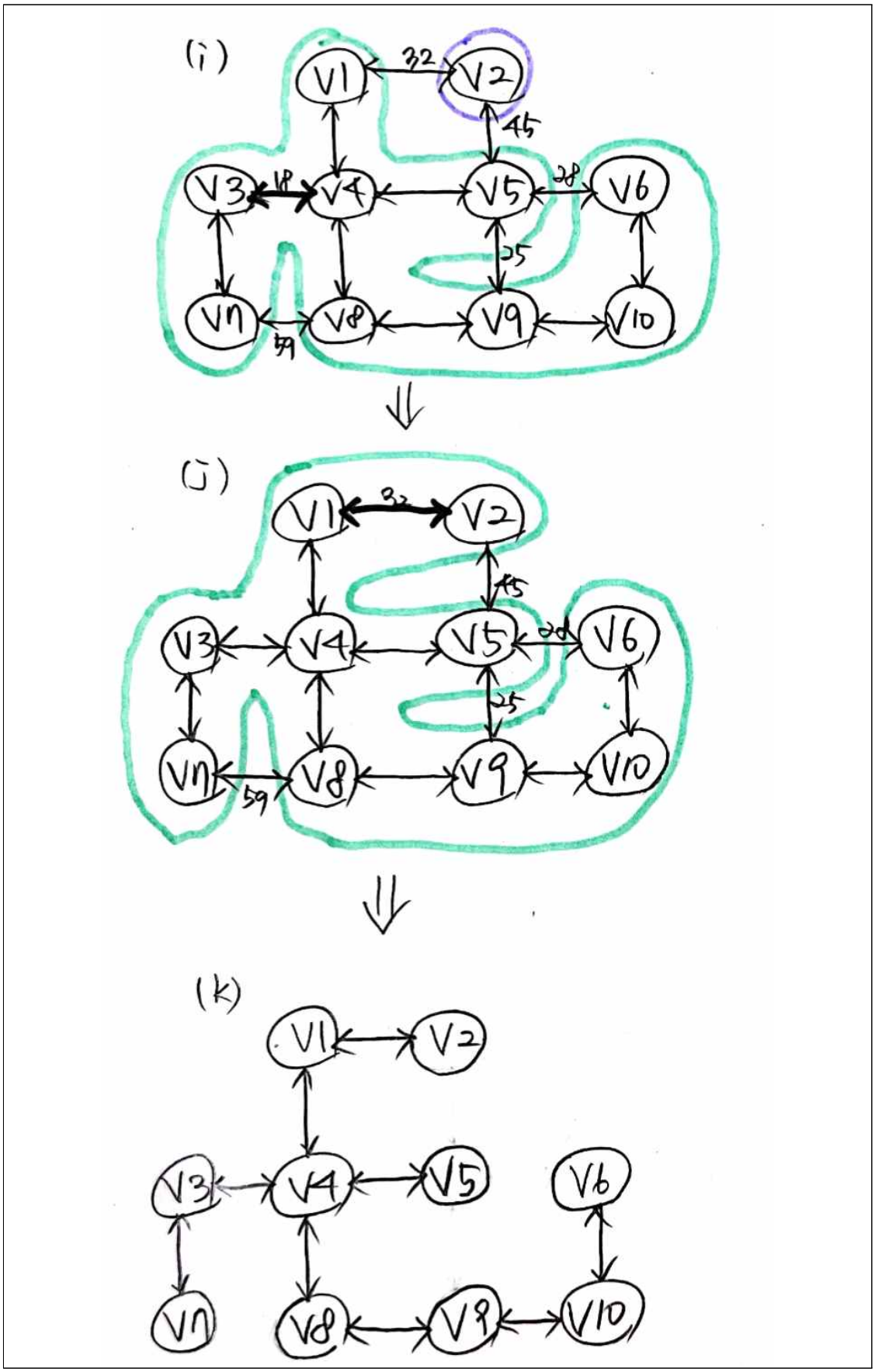

13
0
32
18
17
10
6
5
3
4
12
14
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10

[문제 4-2] Kruskal 알고리즘을 이용하여 위 그래프의 최소비용 신장 트리를 구하는 과정을 교재 [그림 10-17]과 유사한 형태로 그려 제시하시오.




(d)
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
32
17
45
18
10
28
5
25
6
59
12
(e)
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
32
17
45
18
10
28
6
25
59
12


(f)
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
32
17
45
18
10
28
35
59
12
(g)
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
32
17
45
18
28
25
59
12
(h)
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
32
17
45
25
59

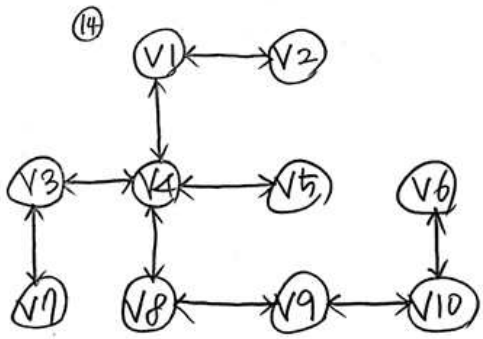

(i)
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
32
45
18
28
25
59
(j)
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
32
45
28
25
59
(k)
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10

[문제 4-3] 앞선 [문제 1]에서 작성한 최소 비용 신장 트리 알고리즘 파이썬 코드를 이용하여 위 [4-1] 및 [4-2]에 제시한 해답을 검증하시오.

파이썬 코드 결과

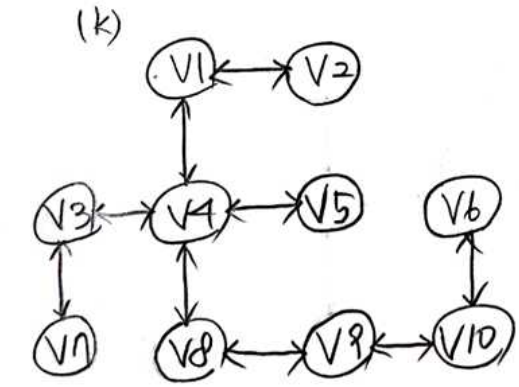
```
[Graph] number of nodes: 10, number of edges: 13
{('4', '8', 3), ('8', '9', 4), ('5', '6', 28), ('9', '10', 12), ('3', '7', 5), ('2', '5', 45), ('3
', '4', 18), ('1', '4', 17), ('4', '5', 10), ('7', '8', 59), ('1', '2', 32), ('6', '10', 6), ('5',
 '9', 25)}
*** Prim Solution ***
1 - 2
1 - 4
4 - 3
4 - 5
4 - 8
3 - 7
8 - 9
9 - 10
10 - 6
*** Kruskal Solution ***
4 - 8
8 - 9
3 - 7
6 - 10
4 - 5
9 - 10
1 - 4
3 - 4
1 - 2
```

[4-1] Prim 알고리즘



파이썬 코드와 비교해보았을 때 연결되어있는 간선이 모두 동일한 것을 확인할 수 있따.

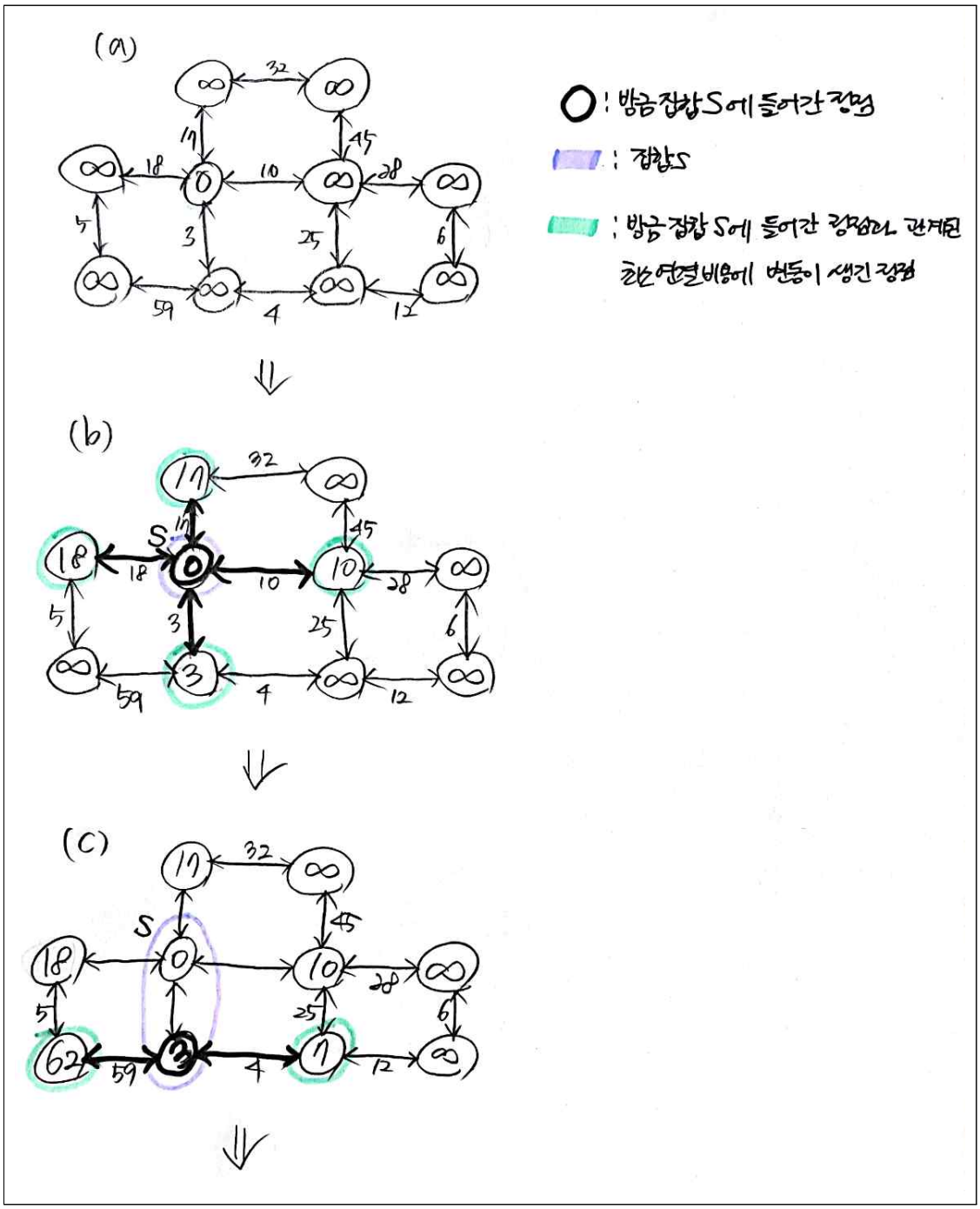
[4-2] Kruskal 알고리즘



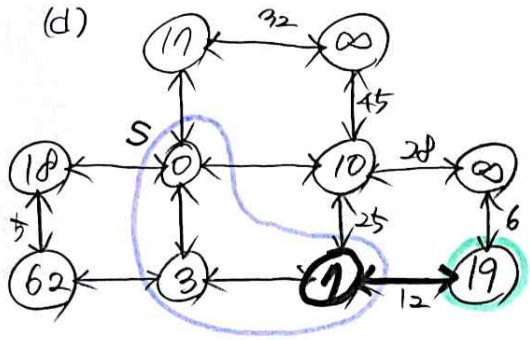
파이썬 코드와 비교해보았을 때 연결되어있는 간선이 모두 동일한 것을 확인할 수 있따.

[문제 5]

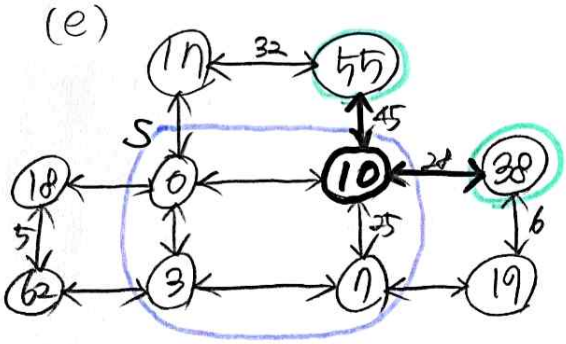
[문제 5-1] Dijkstra 알고리즘을 이용하여 위 그래프(앞선 [문제 4와 동일한 그래프)에서 정점 $v4$에서 다른 모든 정점으로 가는 최단경로를 구하는 과정을 교재 [그림 10-23]과 유사한 형태로 그려 제시하시오. 여기서 각 무향 간선은 같은 가중치를 가진 2개의 쌍방향 간선을 나타낸다고 가정하시오.



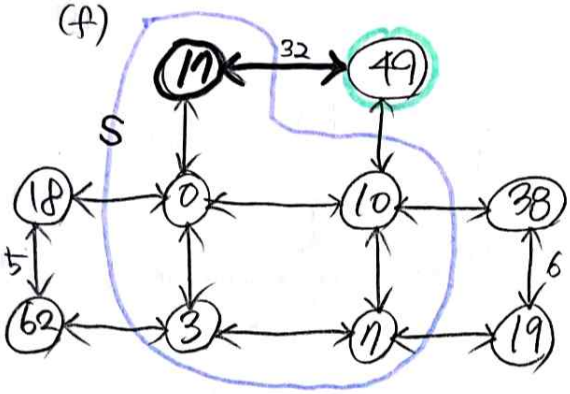
(d)

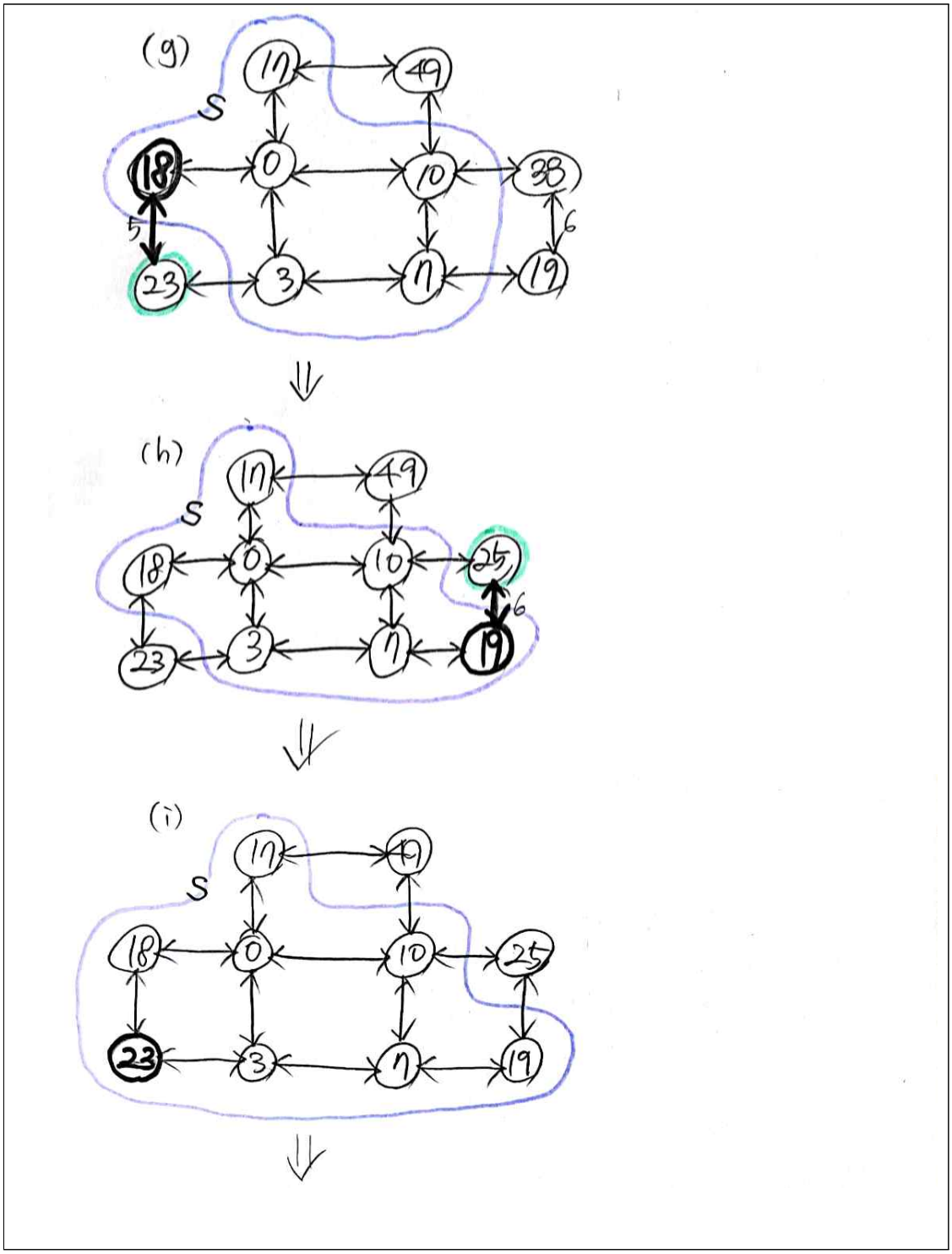

17
32
∞
45
S
18
0
10
28
∞
5
25
6
62
3
7
19
12

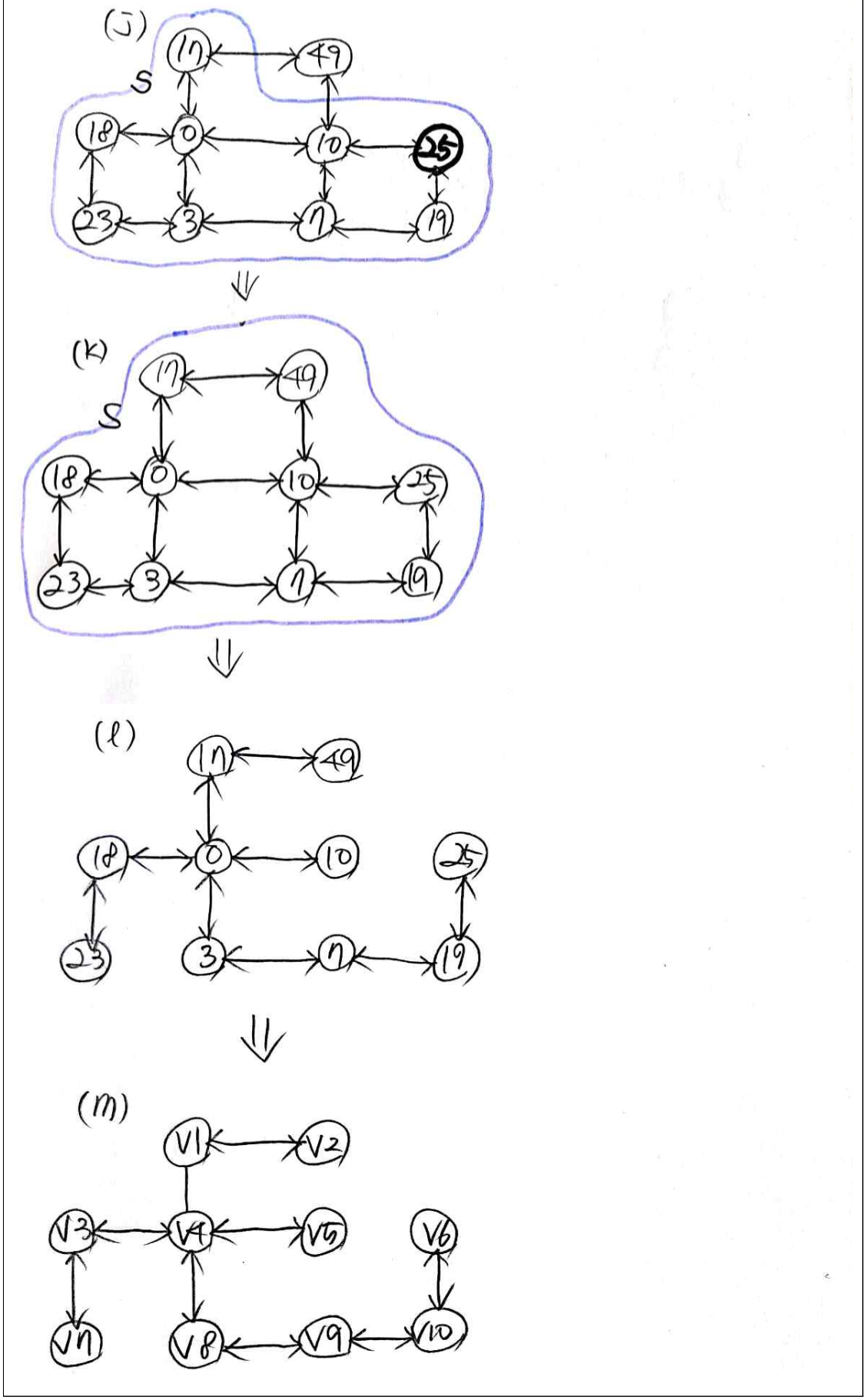
(e)


17
32
55
45
S
18
0
10
28
38
5
25
6
62
3
7
19

(f)


17
32
49
S
18
0
10
38
5
6
62
3
7
19


(g)
S
17
49
18
0
10
38
5
6
23
3
7
19
(h)
S
17
49
18
0
10
25
6
23
3
7
19
(i)
S
17
49
18
0
10
25
23
3
7
19


(j)
S
17
49
18
0
10
25
23
3
7
19
(k)
S
17
49
18
0
10
25
23
3
7
19
(l)
17
49
18
0
10
25
23
3
7
19
(m)
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10

[문제 5-2] 앞선 [문제 2]에서 작성한 Dijkstra 알고리즘 파이썬 코드를 이용하여 위 [5-1]에 제시한 해답을 검증하시오.

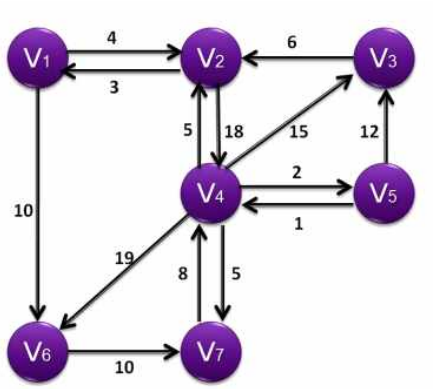
파이썬 코드 결과

[5-1]

[문제 6]

[6-1] 오른쪽 주어진 그래프에서 Floyd-Warshall 알고리즘에서 ***dijk*** 와 ***pijk***가 만들어지는 과정을 그림으로 제시하시오.

• ***dijk*** 와 ***pijk*** 각각 2차원 행렬로 간주하고 총 7 단계별로 2차원 행렬 각각 1개씩 차례로 제시하시오.



D(0)

i\j	1	2	3	4	5	6	7
1	0	4	∞	∞	∞	10	∞
2	3	0	∞	18	∞	∞	∞
3	∞	6	0	∞	∞	∞	∞
4	∞	5	15	0	2	19	5
5	∞	∞	12	1	0	∞	∞
6	∞	∞	∞	∞	∞	0	10
7	∞	∞	∞	8	∞	∞	0

P(0)

i\j	1	2	3	4	5	6	7
1	0	1	0	0	0	1	0
2	2	0	0	2	0	0	0
3	0	3	0	0	0	0	0
4	0	4	4	0	4	4	4
5	0	0	5	5	0	0	0
6	0	0	0	0	0	0	6
7	0	0	0	7	0	0	0

D(1)

i\j	1	2	3	4	5	6	7
1	0	4	∞	∞	∞	10	∞
2	3	0	∞	18	∞	13	∞
3	∞	6	0	∞	∞	∞	∞
4	∞	5	15	0	2	19	5
5	∞	∞	12	1	0	∞	∞
6	∞	∞	∞	∞	∞	0	10
7	∞	∞	∞	8	∞	∞	0

P(1)

i\j	1	2	3	4	5	6	7
1	0	1	0	0	0	1	0
2	2	0	0	2	0	1	0
3	0	3	0	0	0	0	0
4	0	4	4	0	4	4	4
5	0	0	5	5	0	0	0
6	0	0	0	0	0	0	6
7	0	0	0	7	0	0	0

D(2)

i\j	1	2	3	4	5	6	7
1	0	4	∞	22	∞	10	∞
2	3	0	∞	18	∞	13	∞
3	9	6	0	24	∞	19	∞
4	8	5	15	0	2	18	5
5	∞	∞	12	1	0	∞	∞
6	∞	∞	∞	∞	∞	0	10
7	∞	∞	∞	8	∞	∞	0

P(2)

i\j	1	2	3	4	5	6	7
1	0	1	0	2	0	1	0
2	2	0	0	2	0	1	0
3	2	3	0	2	0	1	0
4	2	4	4	0	4	1	4
5	0	0	5	5	0	0	0
6	0	0	0	0	0	0	6
7	0	0	0	7	0	0	0

D(3)

i\j	1	2	3	4	5	6	7
1	0	4	∞	22	∞	10	∞
2	3	0	∞	18	∞	13	∞
3	9	6	0	24	∞	19	∞
4	8	5	15	0	2	18	5
5	21	18	12	1	0	31	∞
6	∞	∞	∞	∞	∞	0	10
7	∞	∞	∞	8	∞	∞	0

P(3)

i\j	1	2	3	4	5	6	7
1	0	1	0	2	0	1	0
2	2	0	0	2	0	1	0
3	2	3	0	2	0	1	0
4	2	4	4	0	4	1	4
5	2	3	5	5	0	1	0
6	0	0	0	0	0	0	6
7	0	0	0	7	0	0	0

D(4)

i\j	1	2	3	4	5	6	7
1	0	4	37	22	24	10	27
2	3	0	33	18	20	13	23
3	9	6	0	24	26	19	29
4	8	5	15	0	2	18	5
5	9	6	12	1	0	19	6
6	∞	∞	∞	∞	∞	0	10
7	16	13	23	8	10	26	0

P(4)

i\j	1	2	3	4	5	6	7
1	0	1	4	2	4	1	4
2	2	0	4	2	4	1	4
3	2	3	0	2	4	1	4
4	2	4	4	0	4	1	4
5	2	4	5	5	0	1	4
6	0	0	0	0	0	0	6
7	2	4	4	7	4	1	0

D(5)

i\j	1	2	3	4	5	6	7
1	0	4	36	22	24	10	27
2	3	0	32	18	20	13	23
3	9	6	0	24	26	19	29
4	8	5	14	0	2	18	5
5	9	6	12	1	0	19	6
6	∞	∞	∞	∞	∞	0	10
7	16	13	22	8	10	26	0

P(5)

i\j	1	2	3	4	5	6	7
1	0	1	5	2	4	1	4
2	2	0	5	2	4	1	4
3	2	3	0	2	4	1	4
4	2	4	5	0	4	1	4
5	2	4	5	5	0	1	4
6	0	0	0	0	0	0	6
7	2	4	5	7	4	1	0

D(6)

i\j	1	2	3	4	5	6	7
1	0	4	36	22	24	10	20
2	3	0	32	18	20	13	23
3	9	6	0	24	26	19	29
4	8	5	14	0	2	18	5
5	9	6	12	1	0	19	6
6	∞	∞	∞	∞	∞	0	10
7	16	13	22	8	10	26	0

P(6)

i\j	1	2	3	4	5	6	7
1	0	1	5	2	4	1	6
2	2	0	5	2	4	1	4
3	2	3	0	2	4	1	4
4	2	4	5	0	4	1	4
5	2	4	5	5	0	1	4
6	0	0	0	0	0	0	6
7	2	4	5	7	4	1	0

D(7)

i\j	1	2	3	4	5	6	7
1	0	4	36	22	24	10	20
2	3	0	32	18	20	13	23
3	9	6	0	24	26	19	29
4	8	5	14	0	2	18	5
5	9	6	12	1	0	19	6
6	26	23	32	18	20	0	10
7	16	13	22	8	10	26	0

P(7)

i\j	1	2	3	4	5	6	7
1	0	1	5	2	4	1	6
2	2	0	5	2	4	1	4
3	2	3	0	2	4	1	4
4	2	4	5	0	4	1	4
5	2	4	5	5	0	1	4
6	2	4	5	7	4	0	6
7	2	4	5	7	4	1	0

[6-2] [3-1]에서 제시한 $pijk$의 마지막 행렬을 이용하여 다음 2가지 경우에 대한 경로를 풀이과정과 함께 제시하시오

• Case1] 출발: $v1$, 도착: $v5$

case1) 출발 V1, 도착 V5이므로,

P_{15}^{k}에 대하여 [6-1]에서 제시한 행렬을 통해 경로를 분석하면 다음과 같다.

pijk

i\j	1	2	3	4	5	6	7
1	0	1	5	2	4	1	6
2	2	0	5	2	4	1	4
3	2	3	0	2	4	1	4
4	2	4	5	0	4	1	4
5	2	4	5	5	0	1	4
6	2	4	5	7	4	0	6
7	2	4	5	7	4	1	0

위 행렬을 P[i][j]라고 표현했을때,

출발점이 V1, 도착점이 V5이므로 P[1][5]라고 표현할수 있다.

이때, 위행렬에서 P[1][5] = 4 인데,

P[1][5]=4 라는 의미는 V1이 V5로 가기 위해 반드시 V4를 거쳐가야한다는 뜻이다.

따라서 다음 행선지는 P[1][4]가 된다.

P[1][4]= 2이므로 V1이 V4로 가기위해서는 반드시 V2를 거쳐야한다.

따라서 다음행선지는 P[1][2]가 된다.

P[1][2] = 1이므로 V1이 V2로 가기위해서는 반드시 V1를 거쳐야한다,

P[1][1]은 자기 자신의 경로를 나타내는 것이므로 P[1][1]=0 이다.

즉, 경로 탐색이 끝났다고 볼수있다.

따라서 V1에서 V5로 가는 최단경로는 다음과 같다.

$\langle V1 \rightarrow V2 \rightarrow V4 \rightarrow V5 \rangle$

• Case2] 출발: $v3$, 도착: $v6$

(case2) 출발 V3, 도착 V6이므로,

P_{36}^{k}에 대하여 [6-1]에서 제시한 방법을 통해 경로를 분석하면 다음과 같다.

pijk

i\j	1	2	3	4	5	6	7
1	0	1	5	2	4	1	6
2	2	0	5	2	4	1	4
3	2	3	0	2	4	1	4
4	2	4	5	0	4	1	4
5	2	4	5	5	0	1	4
6	2	4	5	7	4	0	6
7	2	4	5	7	4	1	0

위 행렬을 P[i][j]라고 표현했을 때,
출발점이 V3, 도착점이 V6이므로 P[3][6]라고 표현할 수 있다.
이때, 위 행렬에서 P[3][6] = 1 인데,
P[3][6] = 1 이라는 의미는 V3이 V6으로 가기위해 반드시 V1을 거쳐야한다는 뜻이다.
따라서 다음 행선지는 P[3][1]이 된다.
P[3][1] = 2 이므로 V3이 V1로 가기위해서는 반드시 V2를 거쳐야한다.
따라서 다음 행선지는 P[3][2]가 된다.
P[3][2] = 3 이므로 V3이 V2로 가기위해서는 반드시 V3을 거쳐야한다.
P[3][3]은 자기 자신의 경로를 나타내는 것이므로 P[3][3] = 0 이다.
즉, 경로탐색이 끝났다고 볼 수 있다.
따라서 V3에서 V6으로 가는 최단경로는 다음과 같다.

$\langle V3 \rightarrow V2 \rightarrow V1 \rightarrow V6 \rangle$

[6-3] 앞선 [문제 2]에서 작성한 Floyd-Warshall 알고리즘 파이썬 코드를 이용하여 위 [6-2]에 제시한 해답을 검증하시오.

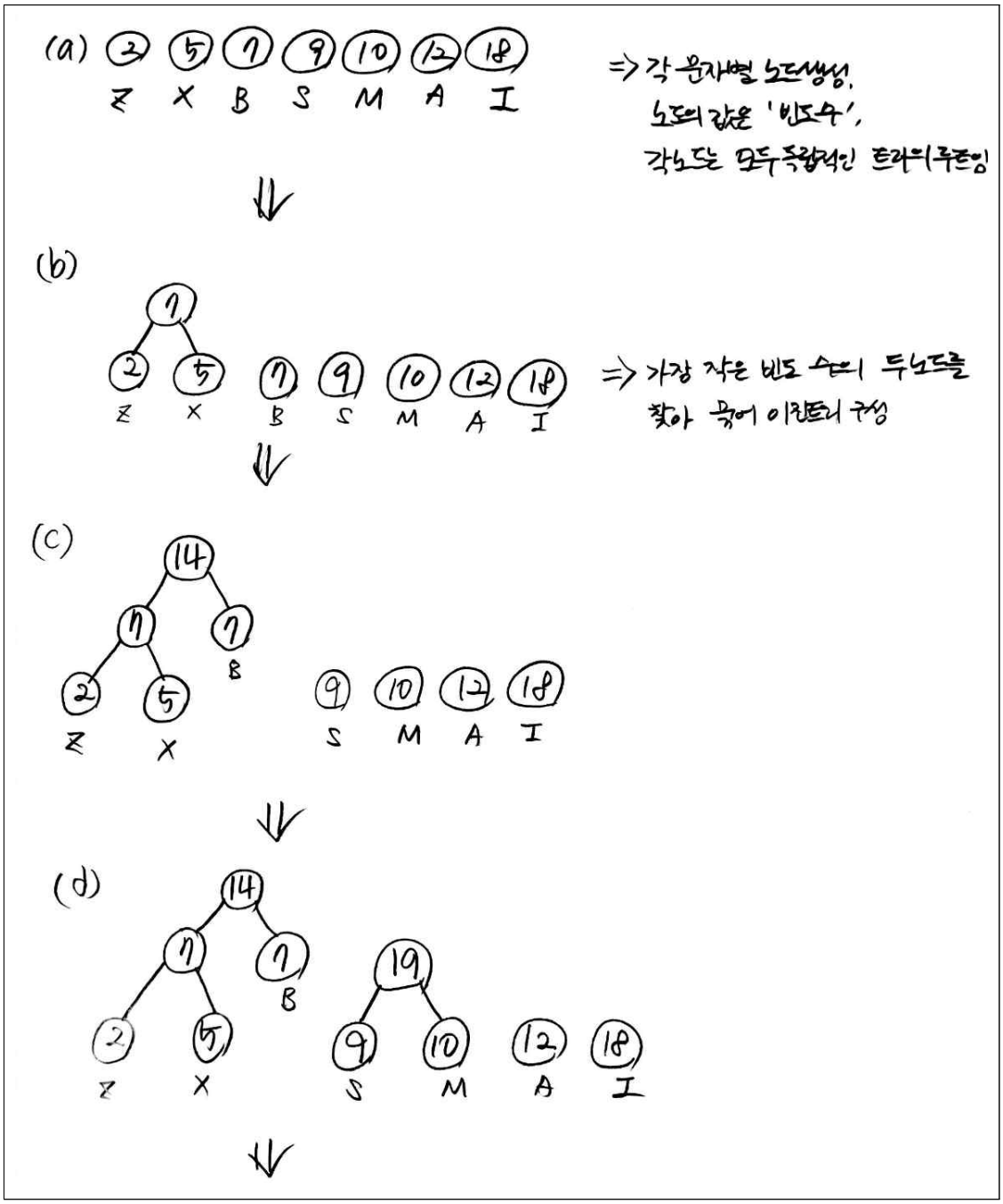
파이썬 코드 결과

[6-2]

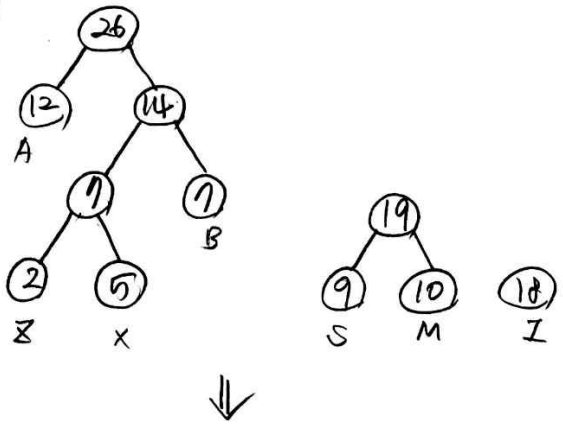
[문제 7]

[문제 7-1] 허프만의 알고리즘을 사용하여 다음 표에 있는 글자들에 대한 최적 이진 트리를 구축하는 과정을 제시하시오.

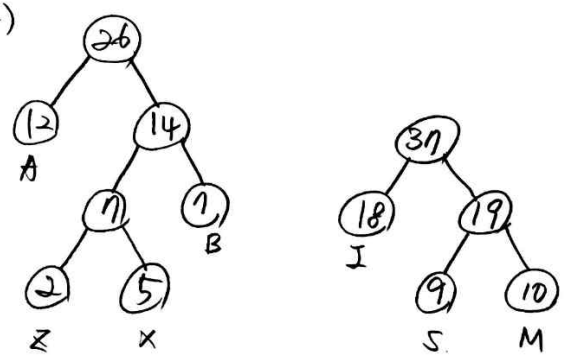
글자	A	B	I	M	S	X	Z
빈도수	12	7	18	10	9	5	2



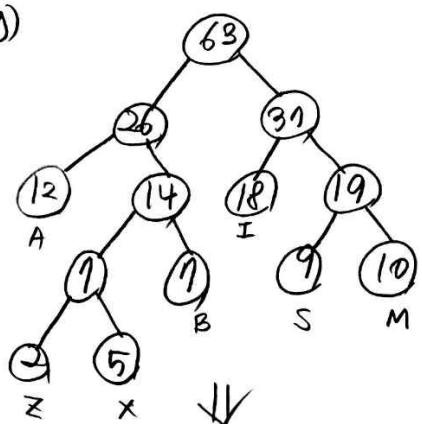
(e)


26
12
14
A
7
7
B
2
5
Z
X
19
9
10
18
S
M
I

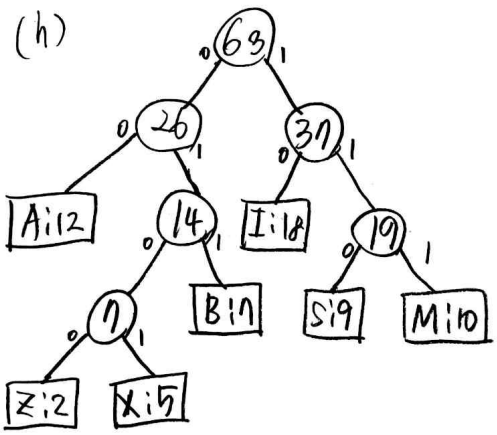
(f)


26
12
14
A
7
7
B
2
5
Z
X
31
18
19
I
9
10
S
M

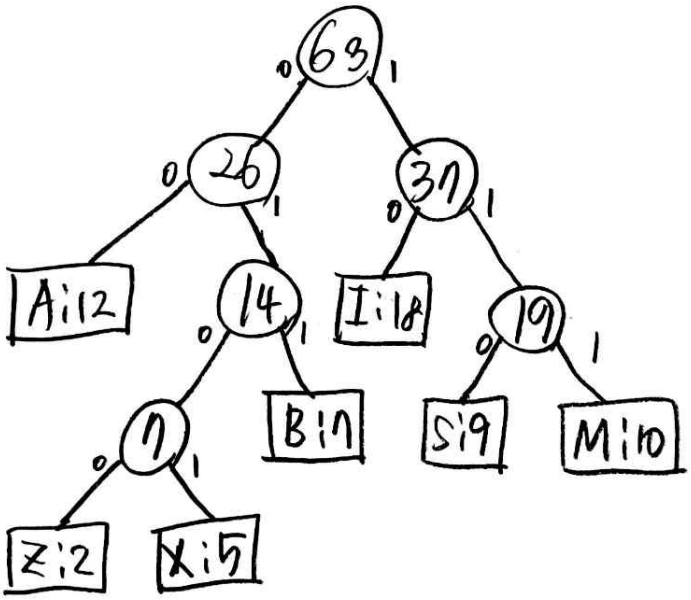
(g)


63
26
31
12
14
18
19
A
I
7
7
9
10
B
S
M
2
5
Z
X

(h)


63
0
1
26
31
A:12
14
I:18
19
7
B:7
S:9
M:10
Z:2
X:5

[문제 7-2] 위 [7-1]에서 제시한 최적 이진 트리를 기반으로 주어진 각 문자에 대한 최적 전치 코드(허프만 코드)를 제시하시오.



최적 이진 트리

문자	허프만코드
A	00
B	011
I	10
M	111
S	110
X	0101
Z	0100

결론

이번 과제에서 코드 구현하는 것이 생각보다 많이 어려웠다. 미리 교수님께서 짜주신 코드에서 빈 라인을 채우는 방식이라 소스 코드를 분석하는 데에 있어서 시간이 오래 걸렸다. 또 그림 그리는 문제가 많아서 시간이 더 오래 걸렸던 것 같다. 이번 과제는 문제가 많고 어려웠지만 그래도 항상 과제를 하면서 시험공부도 할 수 있어서 좋았다. 아직 Prim, Kruskal, Dijkstra, Bellman-ford, floyd-warshall 알고리즘을 정확히 이해하지 못한 것 같다. 다른 과목도 같이 병행해야 하기 때문에 시간 투자를 충분히 못한 것 같아 아쉬웠다. 방학 때 알고리즘 복습하면서 한 번 더 스스로 구현해보는 시간을 가져야겠다고 생각했다.