REPORT

숙명여자대학교
SOOKMYUNG WOMEN'S UNIVERSITY

2024-1학기 자료구조 과제

과목명 : 자료구조

담당교수 : 유석종

학과 : 컴퓨터과학전공

학번 : 2312282

이름 : 임다희

제출일 : 2024/05/28

ch01.링크드 최대힙

```
classNode:
__init__(self,item):
        self.data=item
        self.parent=None
        self.rlink=None
        self.llink=None

class LinkedList:
        def__init__(self):
                self.head=Node(0)
                self.head.parent=None
                self.head.rlink=self.head
                self.head.llink=self.head
 def add(self,item):
    node=Node(item)
    self.head.data+=1
    count=self.head.data
    if self.empty():
        node.llink=self.head
        node.rlink=self.head
        self.head.rlink=node
        self.head.llink=node
    else:
        node.llink=self.head.rlink
        node.rlink=self.head
        self.head.rlink.rlink=node
        self.head.rlink=node

        temp=self.head.llink
        y=1
        while y<count//2:
            temp=temp.rlink
            y+=1
```

Node 클래스에 자신의 부모를 가리키는 self.parent 요소를,

LinkedList 클래스에 해당 부모로의 연결을 추가한다.

def add(self,item) 함수에서는 노드 추가를 실행한다. self.head.data는 전체 노드의 개수를 나타내며 노드가 하나 추가될 때마다 1을 더한다.

리스트가 비어있을 경우에는 새로 추가되는 노드의 좌,우 링크를 헤드에 연결하고 헤드의 좌,우 링크 또한 노드에 연결한다. 비어있지 않을 경우에는 새로 추가되는 노드의 우측 링크를 헤드의 우측 링크가 가리키는 요소에 연결하고. 좌측 링크는 헤드에 연결한다. 헤드의 우측 링크가 가르키는 요소(마지막 노드)의 우측 링크에 노드를 추가하고. 새로운 마지막 노드로 헤드의 우측 링크가 노드를 가리키게 한다.

temp=self.head.llink (루트노드) 라고 하고 y를 1의 값에서 출발해 1씩 더해가며 y가 전체 노드 개수를 2로 나눈 몫보다 크지 않을 때 temp를 temp의 우측 링크 값으로 한칸씩 옮겨간다. 이때 최종적으로 구해진 temp가 node의 부모가 된다.

self.head.rlink(최댓값)을 다시 새로운 temp (tempp)로 지정하고 child라는 변수가 2의 값에서 시작해 1씩 더해가며 child가 전체 노드의 개수보다 크지 않을 때, 현재 마지막 노드의 자리에 위치하고 있는 tempp의 값을 그 부모의 값과 비교하여 tempp의 값이 더 크다면 부모와 위치를 교환한다. 바뀐 위치에서도 while문을 이용해 계속해서 부모의 값과 자신의 값을 비교하며 자신의 값이 더 크다면 부모와 위치를 바꾼다. 이러한 과정을 루트 노드를 제외한 모든 하위 노드들에 대하여 실행한다.

```
            node.parent=temp


            child=2
            tempp=self.head.rlink
            while child<=self.head.data:
                while tempp.parent.data<tempp.data:
                    a=tempp.parent.data
                    tempp.parent.data=tempp.data
                    tempp.data=a
                tempp=tempp.llink
                child+=1
    def delete(self):
        self.head.llink.data=self.head.rlink.data
        self.head.rlink.rlink=None
        self.head.rlink=self.head.rlink.llink
        self.head.rlink.rlink.llink=None
        self.head.rlink.rlink=self.head
        self.head.data-=1


        child=2
        tempp=self.head.rlink
        while child<=self.head.data:
            while tempp.parent.data<tempp.data:
                a=tempp.parent.data
                tempp.parent.data=tempp.data
                tempp.data=a
            tempp=tempp.llink
            child+=1
    def postorder(self):
        postorderlst=[]
        x=self.head.llink


        child=2
        temp=self.head.llink.rlink
```

루트 노드의 값을 삭제하는 delete(self)기능을 만든다.

루트 노드 위치의 값(self.head.llink.data)를 마지막 노드 위치의 값으로 변경하고. 마지막 노드와 헤드 노드간의 연결을 None으로 바꾸어 없앤다. 헤드에는 기존 마지막 노드의 왼쪽 링크로 연결된 노드를 새로운 마지막 노드로 연결해 준다. 전체 노드의 개수인 self.head.data에서는 노드가 한개 삭제되었으므로 1을 뺀다.

기존 헤드노드가 삭제되었고 임시로 마지막 노드가 헤드 노드 위치에 자리하고 있으므로 하위 노드들의 부모자식 관계에도 변동이 있다. add 기능에서 자신의 부모 자리에 있는 노드 값이 자신의 값보다 작다면 서로 자리를 바꾸었던 기능을 똑같이 반복하여 최종적인 노드들의 위치를 결정한다.

최대힙의 모든 노드 값을 postorder로 출력하는 기능을 만든다.

후위 탐색이 맨 처음으로 시작되는 위치를 찾기 위해 루트 노드에서부터 자신(x)을 부모 노드로 가지는 왼쪽 자식(temp)를 찾기를 반복한다. 가장 마지막 층의 맨 왼쪽 노드에 도달하면 해당 과정은 종료하고, 해당 노드를 후위 탐색 결과 리스트 postorderlst[]에 추가한다.

루트 노드의 오른쪽 링크 노드(왼쪽 자식) temp가 방금 과정에서 찾은 탐색 시작 위치 노드와 같은 부모를 가진 형제이면서 후위 탐색 결과 리스트에 포함되어있지 않은지를 확인한다. 만일 그렇다면 temp를 부모 노드로 가지는 노드가 있는지 확인하고 좌측 자식 노드, 우측 자식 노드 순으로 postorderlst에 추가한다. 해당 과정이 완료되면 postorderlst에 temp 노드 또한 추가한다. 시작 위치 노드가 부모를 가진다면 그 부모를 새로운 시작 위치로 지정하여 해당 과정을 반복하고, 부모를 가지지 않는 루트 노드의 위치까지 시작 위치가 옮겨갔다면 과정을 종료한다.

```
            while child<=self.head.data:
                if x.data==temp.parent.data and temp.data not in postorderlst:
                    x=temp
                temp=temp.rlink
                child+=1
            postorderlst.append(x.data)
            while True:
                child=2
                temp=self.head.llink.rlink
                while child<=self.head.data:
                    if x.parent==temp.parent and temp.data not in postorderlst:
                        num=2
                        tempp=self.head.llink.rlink
                        while num<=self.head.data:
                            if tempp.parent==temp and tempp.data not in postorderlst:
                                postorderlst.append(tempp.data)
                            num+=1
                            tempp=tempp.rlink
                        postorderlst.append(temp.data)
                    temp=temp.rlink
                    child+=1
                if x.parent==None:
                    break
                postorderlst.append(x.parent.data)
                x=x.parent
lst=LinkedList()
for item in [10,20,30,40,56,35,60,70,85]:
    print('Add',item,end=' ')
    lst.add(item)
    lst.view()

lst.postorder()
```

출력 결과

```
137             for x in postorderlst:
138                 print(x, end=' ')
139
140 lst=LinkedList()
141 for item in [10,20,30,40,56,35,60,70,85]:
142     print('Add',item,end=' ')
143     lst.add(item)
144     lst.view()
145
146 lst.postorder()
```

문제 출력 디버그 콘솔 **터미널** 포트

```
hon311\python.exe c:\Users\a0108\.vscode\extensions\ms
ive\바탕 화면\공부(24-1)\0518\ch01.py" "
Add 10 [ 10 ] H=10 R=10 I=1
Add 20 [ 20 10 ] H=20 R=10 I=2
Add 30 [ 30 10 20 ] H=30 R=20 I=3
Add 40 [ 40 30 20 10 ] H=40 R=10 I=4
Add 56 [ 56 40 20 10 30 ] H=56 R=30 I=5
Add 35 [ 56 40 35 10 30 20 ] H=56 R=20 I=6
Add 60 [ 60 40 56 10 30 20 35 ] H=60 R=35 I=7
Add 70 [ 70 60 56 40 30 20 35 10 ] H=70 R=10 I=8
Add 85 [ 85 70 56 60 30 20 35 10 40 ] H=85 R=40 I=9
10 40 60 30 70 20 35 56 85
c:\Users\a0108\OneDrive\바탕 화면\공부(24-1)\0518>
```

추가 출력 예시

```
141 for item in [10,20,5,12,40,80]:
142     print('Add',item,end=' ')
143     lst.add(item)
144     lst.view()
145
146 lst.postorder()
```

문제 20 출력 디버그 콘솔 **터미널** 포트

```
\python.exe c:\Users\a0108\.vscode\extensio
면\공부(24-1)\0518\ch01.py" "
Add 10 [ 10 ] H=10 R=10 I=1
Add 20 [ 20 10 ] H=20 R=10 I=2
Add 5 [ 20 10 5 ] H=20 R=5 I=3
Add 12 [ 20 12 5 10 ] H=20 R=10 I=4
Add 40 [ 40 20 5 10 12 ] H=40 R=12 I=5
Add 80 [ 80 20 40 10 12 5 ] H=80 R=5 I=6
10 12 20 5 40 80
```

```
140 lst=LinkedList()
141 for item in [24,7,80,3,64,15,50,10,42,18]:
142     print('Add',item,end=' ')
143     lst.add(item)
144     lst.view()
145
146 lst.postorder()
```

문제 20 출력 디버그 콘솔 **터미널** 포트

```
Add 80 [ 80 7 24 ] H=80 R=24 I=3
Add 3 [ 80 7 24 3 ] H=80 R=3 I=4
Add 64 [ 80 64 24 3 7 ] H=80 R=7 I=5
Add 15 [ 80 64 24 3 7 15 ] H=80 R=15 I=6
Add 50 [ 80 64 50 3 7 15 24 ] H=80 R=24 I=7
Add 10 [ 80 64 50 10 7 15 24 3 ] H=80 R=3 I=8
Add 42 [ 80 64 50 42 7 15 24 3 10 ] H=80 R=10 I=9
Add 18 [ 80 64 50 42 18 15 24 3 10 7 ] H=80 R=7 I=10
3 10 42 7 18 64 15 24 50 80
```

ch02. 최소비용 신장트리

```
class Graph:
    def __init__(self):
        self.graph={}
        self.v_list={}
        self.edge=[]
        self.visitedList=[]
        self.total=0

 def create(self,v,data,weight): #인접리스트 추가
        node=Node(data)
        if v not in self.graph:
            self.graph[v]=[]
        self.graph[v].append((node,weight))

    def sort(self,item_list):
        temp=[]
        for item in item_list:
            for v1,v2,cost in network:
                if (v1==item or v2==item) and ((v1,v2,cost) not in self.edge and (v1,v2,cost) not in self.visitedList):
                    temp.append((cost,v1,v2))
                    temp.sort()
        return (temp[0][0],temp[0][1],temp[0][2])

    def find(self,v2):
        for v1, lst in self.v_list.items():
            if v2==v1:return v1
            if v2 in self.v_list[v1]:return v1
        return -1

    def union(self,s1,s2):
        if s1<s2:
            self.v_list[s1].append(s2)
            self.v_list[s1].extend(self.v_list[s2])
            del self.v_list[s2]
        else:
            self.v_list[s2].append(s1)
            self.v_list[s2].extend(self.v_list[s1])
            del self.v_list[s1]

    def prim(self):
        for v1,v2,cost in network:
            if v1 not in self.v_list:
                self.v_list[v1]=[]
            if v2 not in self.v_list:
                self.v_list[v2]=[]
        print('set list=',self.v_list)

        start=int(input("시작 노드 입력"))
        start_list=[]
```

graph 클래스에 그래프를 나타내는 self.graph, 원소들의 집합을 나타내는 self.v_list, 그래프의 간선을 나타내는 self.edge, 방문한 간선을 나타내는 visitedlist, 최소비용 신장트리의 총 길이를 나타내는 self. total을 추가한다.

create 기능으로 graph에 어떤 정점이 포함되어 있는지 확인하고, 포함되어 있지 않다면 해당 정점(시작점)의 인접 리스트를 새로 추가한다. 포함되어 있다면 인접 리스트에 새로운 요소를 추가한다.

sort 기능으로 리스트를 인수로 받아 해당 리스트의 원소 중 네트워크에서 시작점이나 도착점으로 활용되는 것이 있다면 해당 네트워크 연결이 그래프의 간선에 포함되어 있는지, 그리고 방문한 간선 리스트에 포함되어 있는지 확인한다. 해당 원소가 특정 간선에서 시작점이나 도착점으로 활용되면서 그 간선이 그래프 간선이나 방문 간선 리스트에 포함되지 않는다면 이를 temp 리스트에 가중치가 맨 앞으로 오도록 추가하고, temp 리스트를 sort으로 정렬하여 가중치 순으로 정렬되도록 한다. return 값으로는 정렬된 temp 리스트의 0번째 원소의 가중치, 출발점, 도착점을 지정한다.

find 기능으로는 어떤 원소가 어떤 집합에 속해있는지 찾을 수 있도록 한다.

union 기능으로는 집합끼리 서로 합칠 수 있다.

prim 알고리즘을 구현하면

: 네트워크에 존재하는 모든 정점들을 집합에 추가한다.

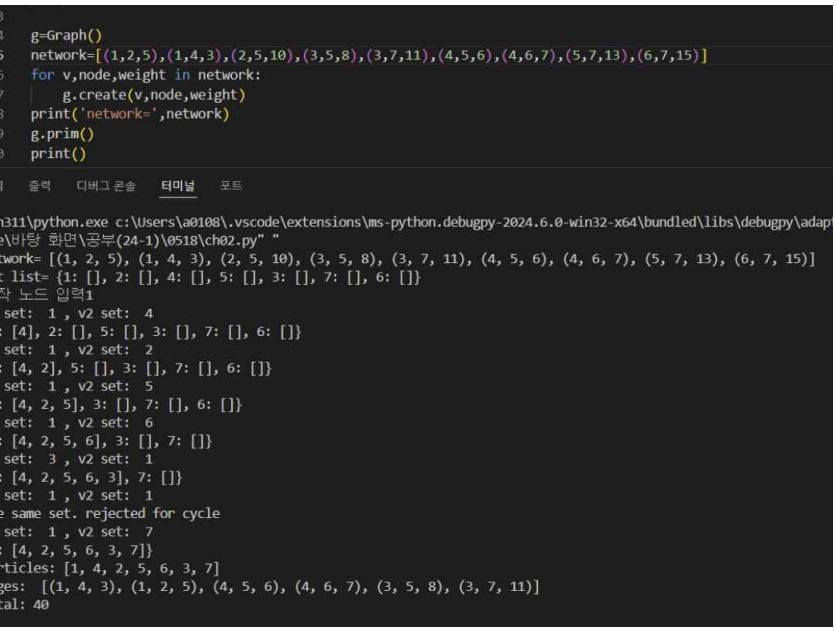
>>사용자에게 시작 노드를 숫자로 입력받고 이를 start list 에 추가한다.

n: 정점의 개수라고 할 때 n-1개의 간선이 최소 신장 비용 트리에 추가되면 종료되는 조건을 while문을 이용해 만든다. start list에 있는 원소와 연결된 간선 중 최소값을 가지는 간선을 찾아 해당 간선의 시작점과 도착점이 속한 집합을 찾는다. 속한 집합이 서로 같다면 사이클을 형성하기에 해당 간선을 최소 비용 신장 트리의 간선으로 추가해서는 안된다. 서로 다를 경우에는 해당 간선을 최소 비용 신장 트리의 간선으로 추가하고, total(트리의 총 길이)에 간선의 비용을 더한다. 방금 구한 시작점과 도착점이 start list에 포함되어 있지 않다면 start list에 추가하고, visited list에 방금 구한 간선을 추가한다. 위 과정을 n-1개의 간선이 최소 신장 비용 트리에 추가될 때까지 반복하고, 트리의 총 간선 비용의 합과 간선 추가 순서, 트리의 간선들을 출력한다.

```
        start_list.append(start)
        length=len(self.v_list)

            if length-1==len(self.edge):
                break
            x=self.sort(start_list)

            s1=self.find(x[1])
            s2=self.find(x[2])
            print('v1 set: ',s1,',','v2 set: ',s2)
            if s1==s2:
                print("the same set. rejected for cycle")

            else:
                self.edge.append((x[1],x[2],x[0]))
                self.total+=x[0]
                self.union(s1,s2)
                print(self.v_list)

                if x[1] not in start_list:
                    start_list.append(x[1])

                if x[2] not in start_list:
                    start_list.append(x[2])

            self.visitedList.append((x[1],x[2],x[0]))
        print("verticles:" ,start_list)
        print("edges: ", self.edge)
        print("total:" ,self.total)

g=Graph()
network=[(1,2,5),(1,4,3),(2,5,10),(3,5,8),(3,7,11),(4,5,6),(4,6,7),(5,7,13),(6,7,15)]
for v,node,weight in network:
    g.create(v,node,weight)
print('network=',network)
g.prim()
print()
```

다른 그래프로 테스트한 경우(교재 227쪽 9.24)

```
network= [(1, 5, 6), (1, 6, 8), (2, 3, 17), (2, 6, 9), (3, 4, 5), (3, 7, 15), (3, 8, 3), (4, 8, 4), (5, 6, 7), (6, 7, 10)]
set list= {1: [], 5: [], 6: [], 2: [], 3: [], 4: [], 7: [], 8: []}
시작 노드 입력6
v1 set:  5 , v2 set:  6
{1: [], 5: [6], 2: [], 3: [], 4: [], 7: [], 8: []}
v1 set:  1 , v2 set:  5
{1: [5, 6], 2: [], 3: [], 4: [], 7: [], 8: []}
v1 set:  1 , v2 set:  1
the same set. rejected for cycle
v1 set:  2 , v2 set:  1
{1: [5, 6, 2], 3: [], 4: [], 7: [], 8: []}
v1 set:  1 , v2 set:  7
{1: [5, 6, 2, 7], 3: [], 4: [], 8: []}
v1 set:  3 , v2 set:  1
{1: [5, 6, 2, 7, 3], 4: [], 8: []}
v1 set:  1 , v2 set:  8
{1: [5, 6, 2, 7, 3, 8], 4: []}
v1 set:  4 , v2 set:  1
{1: [5, 6, 2, 7, 3, 8, 4]}
verticles: [6, 5, 1, 2, 7, 3, 8, 4]
edges:  [(5, 6, 7), (1, 5, 6), (2, 6, 9), (6, 7, 10), (3, 7, 15), (3, 8, 3), (4, 8, 4)]
total: 54
```



ch03. Floyd-Warshall 알고리즘 구현

```
class ShortestPath:
    def __init__(self):
        self.graph={}
        self.visit=[]
        self.dist=[]
        self.prev=[]

    def Floyd_Warshall(self):
        for x in range(len(self.graph)):
            self.dist.append([])
            for y in range(len(self.graph)):
                self.dist[x].append("inf")

        for x in range(len(self.graph)):
            self.prev.append([])
            for y in range(len(self.graph)):
                self.prev[x].append([])

        for x in range(len(self.graph)):
            for y in range(len(self.graph)):
                if x==y:
                    self.dist[x][y]=0

                else:
                    if x in self.graph:
                        for node in self.graph[x]:
                            vertex=node.data
                            cost=node.weight
                            if y==vertex:
                                self.dist[x][y]=cost
        print("dist: A-1")
        self.printdist()

        for k in range(len(self.graph)):
            for i in range(len(self.graph)):
                for j in range(len(self.graph)):
                    if self.dist[i][j]!="inf" and self.dist[i][k]!="inf" and self.dist[k][j]!="inf":
                        if self.dist[i][j]>self.dist[i][k]+self.dist[k][j]:
                            self.dist[i][j]=self.dist[i][k]+self.dist[k][j]
                            self.prev[i][j]=k

                    elif self.dist[i][j]=="inf" and self.dist[i][k]!="inf" and self.dist[k][j]!="inf":
                        self.dist[i][j]=self.dist[i][k]+self.dist[k][j]
                        self.prev[i][j]=k

            print("dist: A",k)
            self.printdist()

        for x in range(len(self.graph)):
            for y in range(len(self.graph)):
                self.showFW(x,y)
```

class ShortestPath에 그래프의 인접리스트를 나타내는 self.graph, 인접행렬을 나타내는 self.dist, 경유 노드를 표시하는 self.prev, 두 정점간 최종 최단 경로를 표시하는 self.visit을 추가한다.

플로이드 워셜 기능을 구현한다. 인접행렬 self.dist, 경유 노드 리스트 self.prev를 행, 열 모두 노드의 개수만큼의 크기를 가진 행렬로 표현한다. self.dist의 모든 원소의 초기값을 inf로 지정하고, 대각행렬의 원소값을 0으로 지정한다. 대각행렬이 아닌 원소 중 인접리스트에 존재하는 원소들과 간선의 value를 확인하고 해당 값을 대각행렬에 추가한다.

i: 시작점, j: 종점, k: 경유 노드 라고 하고 self.dist[i][j]가 (i에서 j까지의 거리) 가 다른 지점을 경유하는 경우보다 크면서 self.dist[i][j]가 inf가 아닌 경우(경로가 존재하지 않는 경우)에는 self.dist[i][j]를 self.dist[i][k]+self.dist[k][j]의 값으로 교체하고, k를 경유했음을 self.prev에 k를 추가하여 표시한다.

self.dist[i][j]가 inf이면서 경유하는 경로의 길이가 inf가 아닌 경우에도 위와 같이 self.dist[i][j]의 값을 교체한다.

showFW를 통해 노드간 최종 경로를 구한다. self.prev[x][y]에 원소가 존재하면(x,y 사이에 경유 루트가 존재하면) 해당 원소를 self.visit[x][y]에 추가한다. 이를 완료하면 x,prev[x][y] 사이에 경유 노드가 존재하는지. prev[x][y],y 사이에 경유 노드가 존재하는지 재귀적으로 다시 함수를 호출하여 확인한다.

```
def showFW(self,x,y):
    if self.prev[x][y]!=[]:

        a=self.showFW(x,self.prev[x][y])
        b=self.showFW(self.prev[x][y],y)

        if a not in self.visit[x][y] and a!=None:
            self.visit[x][y].append(a)

        if self.prev[x][y] not in self.visit[x][y]:
            self.visit[x][y].append(self.prev[x][y])

        if b not in self.visit[x][y] and b!=None:
            self.visit[x][y].append(b)

        return self.prev[x][y]

g=ShortestPath()
for start,dest,weight in [(0,1,8),(0,3,1),(1,2,1),(2,0,4),(3,1,2),(3,2,9)]:
    g.create(start,dest,weight)
g.createpath()
g.Floyd_Warshall()
```

```
83                  if x!=y:
84                      print("s~d: ", x, y,", path [", x, end=' ')
85                      for z in range(len(self.visit[x][y])):
문제  출력  디버그 콘솔  터미널  포트

c:\Users\a0108\OneDrive\바탕 화면\공부(24-1)\0518>
c:\Users\a0108\OneDrive\바탕 화면\공부(24-1)\0518> c: && cd "c:\Users\a0108\O
hon311\python.exe c:\Users\a0108\.vscode\extensions\ms-python.debugpy-2024.6.
ive\바탕 화면\공부(24-1)\0518\ch03_최종.py" "
dist: A-1
0 : 0 8 inf 1
1 : inf 0 1 inf
2 : 4 inf 0 inf
3 : inf 2 9 0
```

```
dist: A 0
0 : 0 8 inf 1
1 : inf 0 1 inf
2 : 4 12 0 5
3 : inf 2 9 0

dist: A 1
0 : 0 8 9 1
1 : inf 0 1 inf
2 : 4 12 0 5
3 : inf 2 3 0

dist: A 2
0 : 0 8 9 1
1 : 5 0 1 6
2 : 4 12 0 5
3 : 7 2 3 0

dist: A 3
0 : 0 3 4 1
1 : 5 0 1 6
2 : 4 7 0 5
3 : 7 2 3 0

s~d:  0 1 , path [ 0 3 1 ] len = 3
s~d:  0 2 , path [ 0 3 1 2 ] len = 4
s~d:  0 3 , path [ 0 3 ] len = 1
s~d:  1 0 , path [ 1 2 0 ] len = 5
s~d:  1 2 , path [ 1 2 ] len = 1
s~d:  1 3 , path [ 1 2 0 3 ] len = 6
s~d:  2 0 , path [ 2 0 ] len = 4
s~d:  2 1 , path [ 2 0 3 1 ] len = 7
s~d:  2 3 , path [ 2 0 3 ] len = 5
s~d:  3 0 , path [ 3 1 2 0 ] len = 7
s~d:  3 1 , path [ 3 1 ] len = 2
s~d:  3 2 , path [ 3 1 2 ] len = 3

c:\Users\a0108\OneDrive\바탕 화면\공부(24-1)\0518>
```

그림 10.2의 출력예시

```
6   class ShortestPath:
92      def showFW(self,x,y):
107             return self.prev[x][y
108
109 g=ShortestPath()
110 for start,dest,weight in [(0,1,8)
111     g.create(start,dest,weight)
112 g.createpath()
문제  출력  디버그 콘솔  터미널  포트

dist: A-1
0 : 0 8 3 6
1 : inf 0 4 1
2 : inf inf 0 inf
3 : 7 inf 2 0

dist: A 0
0 : 0 8 3 6
1 : inf 0 4 1
2 : inf inf 0 inf
3 : 7 15 2 0

dist: A 1
0 : 0 8 3 6
1 : inf 0 4 1
2 : inf inf 0 inf
3 : 7 15 2 0

dist: A 2
0 : 0 8 3 6
1 : inf 0 4 1
2 : inf inf 0 inf
3 : 7 15 2 0

dist: A 3
0 : 0 8 3 6
1 : 8 0 3 1
2 : inf inf 0 inf
3 : 7 15 2 0
```

연습문제10.3의 출력예시

```
s~d:  0 1 , path [ 0 1 ] len = 8
s~d:  0 2 , path [ 0 2 ] len = 3
s~d:  0 3 , path [ 0 3 ] len = 6
s~d:  1 0 , path [ 1 3 0 ] len = 8
s~d:  1 2 , path [ 1 3 2 ] len = 3
s~d:  1 3 , path [ 1 3 ] len = 1
s~d:  2 0 , path [ 2 0 ] len = inf
s~d:  2 1 , path [ 2 1 ] len = inf
s~d:  2 3 , path [ 2 3 ] len = inf
s~d:  3 0 , path [ 3 0 ] len = 7
s~d:  3 1 , path [ 3 0 1 ] len = 15
s~d:  3 2 , path [ 3 2 ] len = 2

c:\Users\a0108\OneDrive\바탕 화면\공부(24-1)\0518>
```

```
106
107             return self.prev[x][y]
108
109 g=ShortestPath()
110 for start,dest,weight in [(0,1,11),(0,2,5),(1,0,2),(1,2,8),(2,1,3)]:
111     g.create(start,dest,weight)
문제  출력  디버그 콘솔  터미널  포트

ive\바탕 화면\공부(24-1)\0518\ch03_최종.py" "
dist: A-1
0 : 0 11 5
1 : 2 0 8
2 : inf 3 0

dist: A 0
0 : 0 11 5
1 : 2 0 7
2 : inf 3 0

dist: A 1
0 : 0 11 5
1 : 2 0 7
2 : 5 3 0

dist: A 2
0 : 0 8 5
1 : 2 0 7
2 : 5 3 0

s~d:  0 1 , path [ 0 2 1 ] len = 8
s~d:  0 2 , path [ 0 2 ] len = 5
s~d:  1 0 , path [ 1 0 ] len = 2
s~d:  1 2 , path [ 1 0 2 ] len = 7
s~d:  2 0 , path [ 2 1 0 ] len = 5
s~d:  2 1 , path [ 2 1 ] len = 3
```

연습문제10.2의 출력예시

ch04.위상 정렬

```
class Graph:
    def __init__(self):
        self.graph={}
        self.queue=[]
        self.indegree={}
        self.route=[]

    def create(self,data,v):
        node=Node(data)
        if v not in self.graph:
            self.graph[v]=[]
        if data not in self.graph:
            self.graph[data]=[]
        self.graph[v].append(node)

    def setIndegree(self):
        for x in self.graph:
            self.indegree[x]=len(self.graph[x])

    def topologicalSort(self):
        self.setIndegree()
        for x in self.indegree:
            if self.indegree[x]==0:
                self.addq(x)
                print("큐에 ",x,"추가" )
        self.showq()
        while self.queue:
            vertex=self.deleteq()
            print("큐에서 ", vertex, "제거")
            self.route.append(vertex)

            a=len(self.queue)
            print()
            self.showIndegree()
            self.showq()
            for key in self.graph:
                if vertex in [node.data for node in self.graph[key]]:
                    self.indegree[key]-=1
                    if self.indegree[key]==0:
                        self.addq(key)
                        print("큐에 ",key,"추가" )
            b=len(self.queue)

            if a!=b:
             print()
             self.showIndegree()
             self.showq()
        print("모든 정점 방문, 위상정렬 종료.")
        print("위상 정렬 순서: ", end=' ')
        for x in self.route:
            print(x, end=' ')
```

Graph 클래스에 초기 인접 리스트를 나타내는 self.graph, 위상 정렬에 쓰이는 큐를 나타내는 self. queue, 각 노드의 진입차수를 나타내는 self.indegree, 위상 정렬 순서를 나타내는 self.route를 지정한다.

create 로 도착 지점 노드와 출발 지점 노드를 경로로 받아 인접 리스트 self.graph를 완성하고, setIndegree로 self.graph의 각 키(x=도착 지점 노드 번호)에 해당하는 value의 값을 받아 어떤 노드로 향하는 간선의 개수(각 노드의 진입차수)를 구한다.

topologicalSort 에서 위상정렬을 실행한다. 각 노드들의 진입차수를 self.setIndegree를 통해 구하고, self.indgree[x]==0인 노드가 있는지 검사해 진입차수가 0인 노드가 존재하면 해당 노드를 큐에 추가한다.

큐가 비어있지 않으면 해당 큐의 0번째 요소를 제거하고, 해당 요소를 위상정렬 순서 리스트에 추가한다.

만일 제거한 요소와 연결되는 간선이 있다면 해당 간선과 연결된 노드의 진입간선의 수를 1만큼 줄인다. 해당 과정 완료 후 진입간선의 수가 0인 노드가 생긴다면 그 노드를 다시 큐에 추가한다.

큐에 어떤 원소도 존재하지 않을 때까지 해당 과정을 반복하고, 큐가 완전히 비면 위상정렬을 종료한다.

```
g=Graph()
for v,node in [(0,2),(1,2),(2,4),(4,6),(6,9),(4,7),(7,10),(3,5),(5,4),(5,8)]:
    g.create(v,node)
g.show()
g.showIndegree()
g.topologicalSort()
```

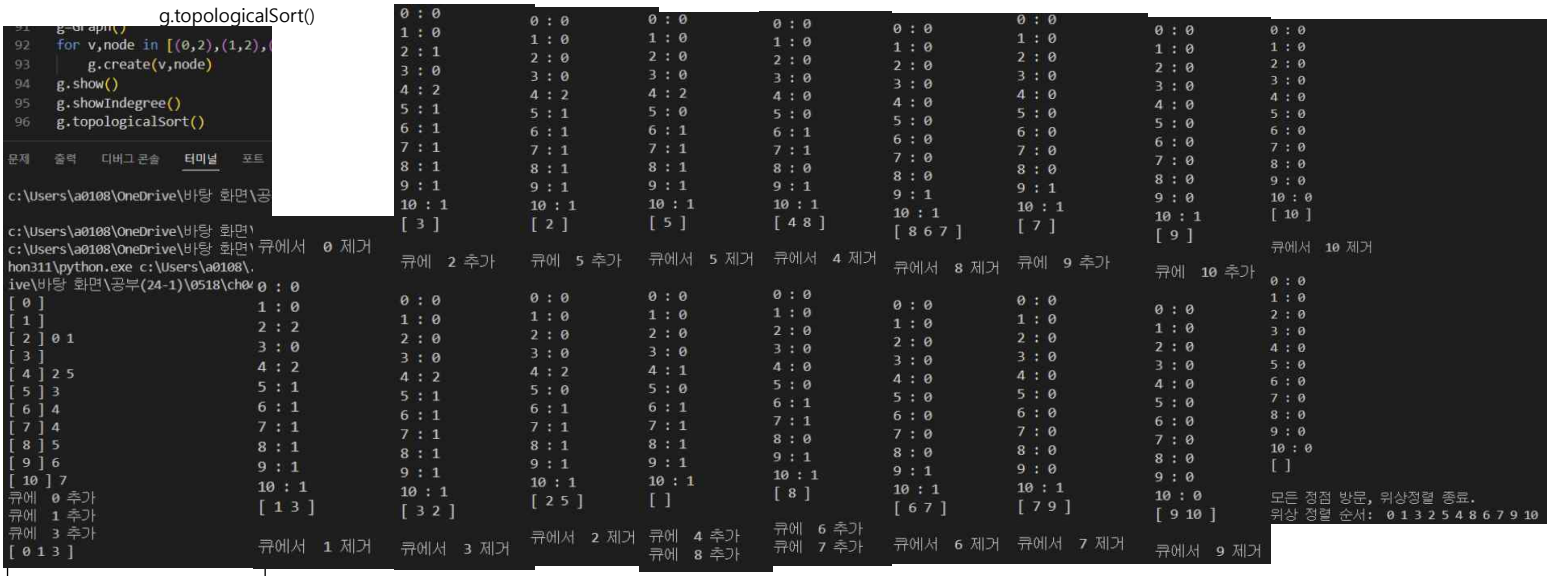
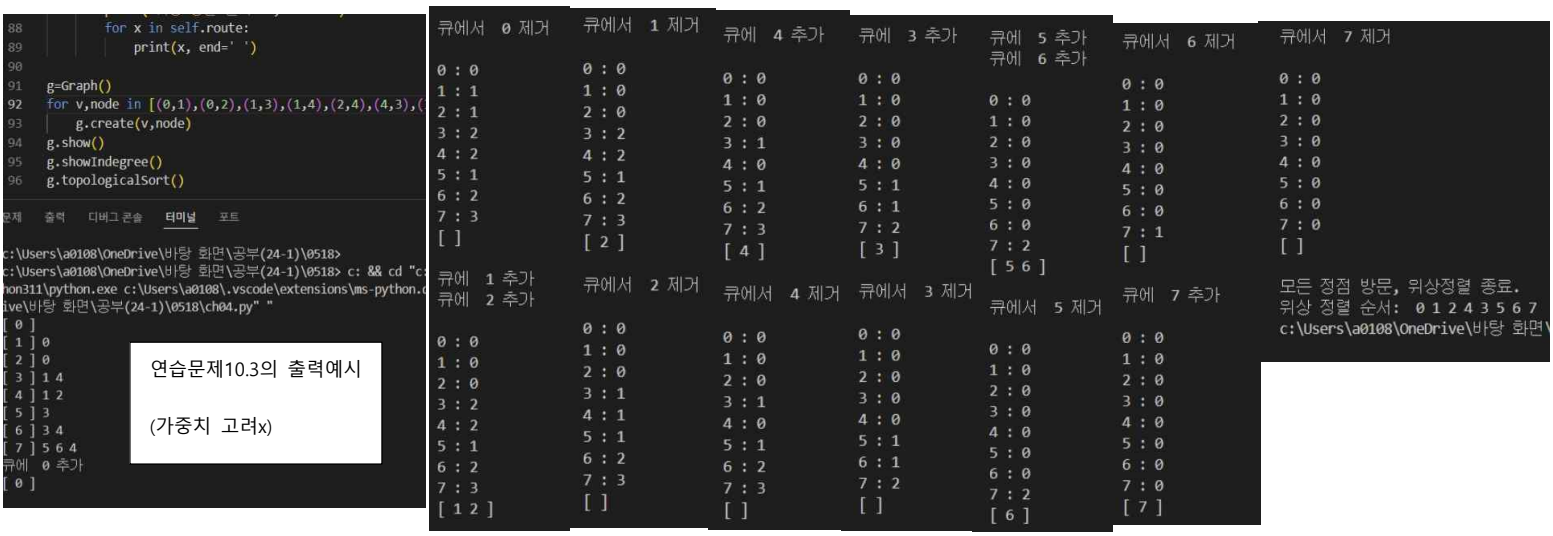


그림 10.3의 출력예시



연습문제10.3의 출력예시

(가중치 고려x)

문제	제목	미완성 기능	배점	본인평가점수
1	링크드 최대힙	없음	25	25
2	최소비용신장트리(Prim)	없음	25	25
3	Floyd-Warshall 최단 경로 출력	없음	30	30
4	위상 정렬	없음	20	20
총점			100	100