<1번문제>
연결리스트에 최대 힙을 표현하는 linked max heap을 구현하시오.
다음 예시의 입력값들을 하나씩 추가하여 최대힙을 구성하여 출력하고, 최대값(루트)를 하나씩 삭제하는 과정을 보이시오.
<예시>
- 7-6 예시 [10, 20, 30, 40, 56, 35, 60, 70, 85]
- 연습문제 7-1 입력값 [25,30,17,14,49,66,23,39]
- 193쪽, 예제 입력값 [15,20,8,30,18,48,35]
<기능>
1. 노드 추가 : 마지막 노드 다음 위치에 추가한 후 최대힙을 재구성한다.
(중복 값이 없는 경우에만 추가)
2. 노드 삭제(루트 삭제) : 루트 삭제 후 마지막 노드가 대신한 후 힙 재구성한다.
3. 노드 추가 또는 삭제 연산 후 최종 최대힙의 모든 노드 값 postorder로 출력한다

-풀이과정 및 핵심 코드
큐를 이용함

```
def heap_up(self,child): #힙 추가한 뒤 밑에서 올라가며 재구성
    while child.parent and child.parent!=self.head and child.data>child.parent.data:
        child.data,child.parent.data=child.parent.data,child.data
        child=child.parent
```

```
def heap_down(self,parent):  #루트 제거한뒤 위에서 내려오며 재구성
    while parent and parent!=self.head:#부모가 헤드노드이면 탐색할수없음
        left=parent.llink
        right=parent.rlink
        max_child=None
        if left and left != self.head and right and right != self.head: # 더 큰 자식을 선택하여 부모와 비교한다
            max_child = left if left.data > right.data else right
        elif left and left != self.head:
            max_child = left
        elif right and right != self.head:
            max_child = right
        else:
            break

        if parent.data < max_child.data: #부모<자식일 경우swqp
            parent.data, max_child.data = max_child.data, parent.data
            parent = max_child
        else:
            break
```

```
queue.append(self.head.llink) #힙은 완전 이진트리의 성질을 만족해야 하므로 비어있는 llink rlink가 있다면 바로 그곳에 추가하면 됨
while queue:
    current=queue.pop(0)
    if not current.llink: #llink빈 경우
        current.llink=new_node
        new_node.parent=current
        self.head.rlink=new_node
        break
    elif not current.rlink: #rlink빈 경우
        current.rlink=new_node
        new_node.parent=current
        self.head.rlink=new_node
        break
    else:
        queue.append(current.llink) #두 자식노드가 모두 채워져 있으면 자식도 큐에 추가하고 다음 노드로 반복해서 진행
        queue.append(current.rlink)
```

(노드 추가하는 함수. head.rlink를 이용한다)

(노드 제거하는 함수. 부모-자식간 연결 끊고 self.head.rlink의 연결도 변경하면 된다)

```
self.head.llink.data,self.head.rlink.data=self.head.rlink.data,self.head.llink.data #swap
queue=[self.head.llink] #루트노드부터 시작
prev=parent=None
while queue: #부모&직전노드 찾기
    current=queue.pop(0)
    if current.llink:
            queue.append(current.llink)
    if current.rlink:
            queue.append(current.rlink)
    if current.llink ==self.head.rlink or current.rlink==self.head.rlink: #현재노드의 자식 중 하나가 마지막 노드라면 현재노드가 바로 부모임
        parent=current
        prev=queue[-2] if len(queue) >= 2 else self.head.llink #부모의 자식 존재하므로 마지막 노드가 큐에 추가됨->그 직전이 새로운 마지막 노드가 됨
        break
```

```
if parent:
    if parent.llink==self.head.rlink: #부모->자식 끊기
        parent.llink=None
    elif parent.rlink==self.head.rlink:
        parent.rlink=None
self.head.rlink.parent=None #자식->부모 끊기
del self.head.rlink
self.head.rlink=prev#새로운 마지막 노드
self.head.data-=1
self.heap_down(self.head.llink)
self.view(self.head.llink)
print()
```

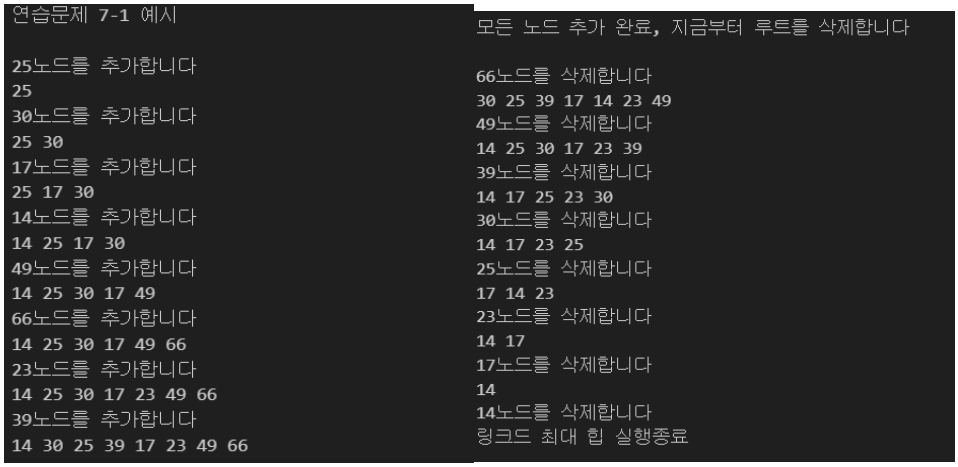
-실행 화면

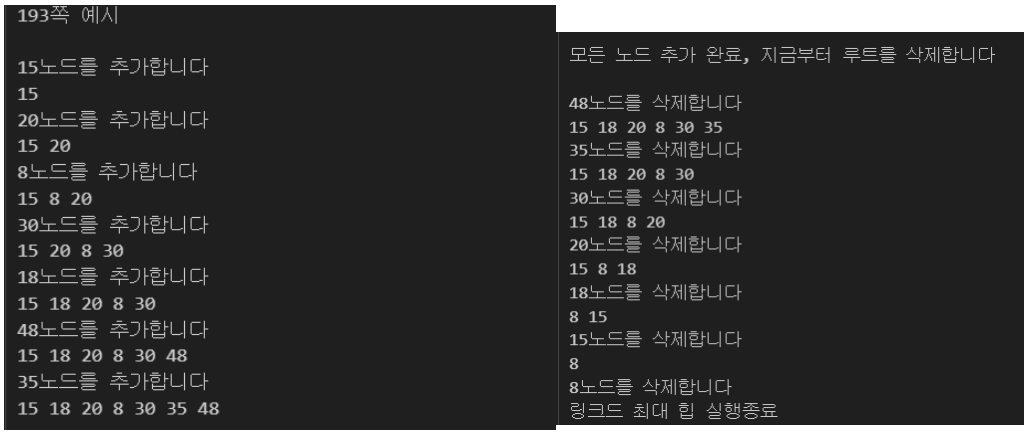
```
7-6 예시

10노드를 추가합니다
10
20노드를 추가합니다
10 20
30노드를 추가합니다
10 20 30
40노드를 추가합니다
10 30 20 40
56노드를 추가합니다
10 30 40 20 56
35노드를 추가합니다
10 30 40 20 35 56
60노드를 추가합니다
10 30 40 20 35 56 60
70노드를 추가합니다
10 40 30 60 20 35 56 70
85노드를 추가합니다
10 40 60 30 70 20 35 56 85
```

```
모든 노드 추가 완료, 지금부터 루트를 삭제합니다

85노드를 삭제합니다
10 40 30 60 20 35 56 70
70노드를 삭제합니다
10 30 40 20 35 56 60
60노드를 삭제합니다
10 30 40 20 35 56
56노드를 삭제합니다
10 20 30 35 40
40노드를 삭제합니다
10 30 20 35
35노드를 삭제합니다
10 20 30
30노드를 삭제합니다
10 20
20노드를 삭제합니다
10
10노드를 삭제합니다
링크드 최대 힙 실행종료
```





<2번문제>

(문제2) (9-9) 최소비용 신장트리 (Prim 방법)

프로그램 9.3(Kruskal 방법)을 수정하여 Prim의 방법으로 최소 비용 신장 트리를 탐색하는 프로그

램을 작성하시오.

- 연습문제 9-1, 9-2, 9-3의 예시 그래프를 사용하여 작성한 프로그램을 테스트한다.
- 탐색 시작 노드를 사용자로부터 입력 받는다. (시작 노드가 변경되는 경우도 처리할 것)
- 최소 비용 신장 트리에 간선이 추가되는 순서대로 출력한다

-풀이과정 및 핵심 코드

```
def prim(self,start):
    self.v_list=[start] #신장 트리에 시작정점 추가
    self.total=0
    min_cost=10000

    while len(self.edge)<len(self.graph)-1: #간선=정점개수-1
        tmp=[]
        for u in self.v_list: #기존 신장트리의 각 정점을 방문하며 인접 리스트를 탐색
            for node, cost in self.graph[u]:
                v=node.data #노드 이름으로 검색
                if v not in self.v_list: #아직 미방문 정점일 경우에 추가
                    tmp.append((cost,u,v))

        tmp.sort()   #가중치 낮은 순서로 정렬
        min_cost,from_v,to_v=tmp[0] #최소 비용을 가지는 간선,시작정점,도착정점 저장하고
        self.v_list.append(to_v) #신장트리 확장
        self.edge.append((from_v,to_v,min_cost)) #간선 추가
        self.total+=min_cost #총 비용 추가
        print(f"간선 추가: {from_v} - {to_v} (비용 {min_cost})")
```

-실행화면

```
연습문제 9-1
network= [(1, 3, 15), (1, 2, 8), (1, 5, 10), (3, 4, 11), (3, 5, 12), (2, 4, 14), (2, 7, 17), (4, 6, 16), (4, 5, 6), (7, 6, 9)]
6 정점에서 시작합니다
간선 추가: 6 - 7 (비용 9)
간선 추가: 6 - 4 (비용 16)
간선 추가: 4 - 5 (비용 6)
간선 추가: 5 - 1 (비용 10)
간선 추가: 1 - 2 (비용 8)
간선 추가: 4 - 3 (비용 11)

간선: [(6, 7, 9), (6, 4, 16), (4, 5, 6), (5, 1, 10), (1, 2, 8), (4, 3, 11)]
정점 리스트 [6, 7, 4, 5, 1, 2, 3]
최소 비용 신장 트리의 총 비용: 60
```

```
연습문제 9-2
network= [(1, 2, 5), (1, 4, 3), (2, 5, 10), (4, 5, 6), (5, 3, 8), (4, 6, 7), (5, 7, 13), (6, 7, 15), (3, 7, 11)]
4 정점에서 시작합니다
간선 추가: 4 - 1 (비용 3)
간선 추가: 1 - 2 (비용 5)
간선 추가: 4 - 5 (비용 6)
간선 추가: 4 - 6 (비용 7)
간선 추가: 5 - 3 (비용 8)
간선 추가: 3 - 7 (비용 11)

간선: [(4, 1, 3), (1, 2, 5), (4, 5, 6), (4, 6, 7), (5, 3, 8), (3, 7, 11)]
정점 리스트 [4, 1, 2, 5, 6, 3, 7]
최소 비용 신장 트리의 총 비용: 40
```

```
연습문제 9-3
network= [(0, 1, 6), (0, 2, 1), (1, 3, 10), (1, 4, 1), (2, 4, 2), (3, 4, 3), (3, 5, 4), (3, 6, 5), (4, 6, 8), (4, 7, 9), (5, 7, 12), (6, 7, 7)]
2 정점에서 시작합니다
간선 추가: 2 - 0 (비용 1)
간선 추가: 2 - 4 (비용 2)
간선 추가: 4 - 1 (비용 1)
간선 추가: 4 - 3 (비용 3)
간선 추가: 3 - 5 (비용 4)
간선 추가: 3 - 6 (비용 5)
간선 추가: 6 - 7 (비용 7)

간선: [(2, 0, 1), (2, 4, 2), (4, 1, 1), (4, 3, 3), (3, 5, 4), (3, 6, 5), (6, 7, 7)]
정점 리스트 [2, 0, 4, 1, 3, 5, 6, 7]
최소 비용 신장 트리의 총 비용: 23
```

<3번문제>
문제3) Floyd-Warshall 알고리즘을 구현하시오. (30점)
구현 프로그램은 각 비용행렬과 각 정점 간 경로를 출력해야 한다. (아래의 조건을 만족해야 하며,
그림 10.2와 연습문제 10-2, 10-3의 그래프로 각각 테스트할 것)
(조건)
비용행렬(dist)과 동일한 크기의 p배열(prev)을 단계별로 출력한다.
최종 비용행렬에서 두 정점간 최단 경로(정점들의 시퀀스)를 재귀적으로 역추적한다.
* 경로 (a, b)에 대한 p배열에 경유 정점 k1이 존재하면 경로는 (a, k1, b)이 된다.

-풀이과정 및 핵심코드

```python
def create_matrix(edges): #A-1 생성(초기상태)
    print("dist: A-1")
    print("---------------")
    vertices = set() # 집합을 이용해 정점 목록 만든다.(겹침방지)
    for edge in edges:
        start, end, cost = edge
        vertices.add(start)
        vertices.add(end)
    vertices = sorted(list(vertices))  # 인덱스 고정
    index={v:i for i,v in enumerate(vertices)} #a:0, b:1 ...형태
    inf=float('inf')
    n=len(vertices) #정점개수
    dist=[[inf]*n for _ in range(n)]
    prev=[[None]*n for _ in range(n)]
    for i in range(n):
        dist[i][i]=0
    for start,end,cost in edges: #직접 연결된 정점들(초기상태이므로)
        i,j=index[start],index[end]
        dist[i][j]=cost #비용
        prev[i][j] = None #경유점 없음
```

```python
def update(dist,prev,n): #경유 하는 경우에 A-1을 갱신하는 함수이다
    for k in range(n):          # 경유 노드
        print("\ndist: A",k)
        print("---------------")
        for i in range(n):      # 출발 노드
            print(i,":",end=' ')
            for j in range(n): # 도착 노드
                if dist[i][j]>dist[i][k]+dist[k][j]: #최소비용을 유지하며 갱신
                    dist[i][j] = dist[i][k] + dist[k][j]
                    prev[i][j]=k #i에서 j로 가는 동안 k를 경유할 경우 p 배열 추가
```

```python
def path(prev,s,d): #경유 정점을 경로에 추가하는 함수
    if prev[s][d]==None: #경유지 없음
        return []
    else:
        k=prev[s][d]
        return path(prev,s,k)+[k]+path(prev,k,d) #[시작~경유]경로 + 경유 정점 + [경유~도착]경로
```

-실행화면

그림 10.2

연습문제 10-2

```
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
```

```
<최단 경로>
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
```

연습문제 10-3

```
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

<최단 경로>
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
```

<4번문제>

위상 정렬 (20점)

방향 그래프의 간선 정보를 입력 받아 그래프의 인접 리스트를 생성하고 위상 정렬을 수행하시오.

(그림 10.3의 예시 그래프 및 추가 방향 그래프에 대해서 테스트를 수행한다)

(* 프로그램 9.2를 활용하고 이에 기반하여 위상 정렬 코드를 작성할 것)

(조건)

- 초기 인접 리스트를 출력한다.
- 위상 정렬 수행 과정에서 큐의 상태 및 각 정점의 진입 차수 정보를 단계별로 갱신하여 출력한다.
- 위상 정렬 순서를 출력한다.
- 그래프에 사이클이 있는 예시와 없는 예시를 각각 테스트할 것

-풀이과정 및 핵심코드

```
초기 인접 리스트
0:[2]
1:[2]
2:[4]
3:[5]
4:[6, 7]
5:[4, 8]
6:[9]
7:[10]
8:[]
9:[]
10:[]
```

```
현재 큐: [0, 1, 3]
진입 차수: {0: 0, 2: 2, 1: 0, 4: 2, 3: 0, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1}

현재 큐: [1, 3]
진입 차수: {0: 0, 2: 1, 1: 0, 4: 2, 3: 0, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1}

현재 큐: [3, 2]
진입 차수: {0: 0, 2: 0, 1: 0, 4: 2, 3: 0, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1}

현재 큐: [2, 5]
진입 차수: {0: 0, 2: 0, 1: 0, 4: 2, 3: 0, 5: 0, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1}

현재 큐: [5]
진입 차수: {0: 0, 2: 0, 1: 0, 4: 1, 3: 0, 5: 0, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1}

현재 큐: [4, 8]
진입 차수: {0: 0, 2: 0, 1: 0, 4: 0, 3: 0, 5: 0, 6: 1, 7: 1, 8: 0, 9: 1, 10: 1}

현재 큐: [8, 6, 7]
진입 차수: {0: 0, 2: 0, 1: 0, 4: 0, 3: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 1}

현재 큐: [6, 7]
진입 차수: {0: 0, 2: 0, 1: 0, 4: 0, 3: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 1}

현재 큐: [7, 9]
진입 차수: {0: 0, 2: 0, 1: 0, 4: 0, 3: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 1}

현재 큐: [9, 10]
진입 차수: {0: 0, 2: 0, 1: 0, 4: 0, 3: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0}
```

```
현재 큐: [10]
진입 차수: {0: 0, 2: 0, 1: 0, 4: 0, 3: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0}

위상 정렬 결과: [0, 1, 3, 2, 5, 4, 8, 6, 7, 9, 10]


<사이클 있는 그래프 예시>

초기 인접 리스트
0:[1]
1:[2]
2:[3]
3:[1]


현재 큐: [0]
진입 차수: {0: 0, 1: 2, 2: 1, 3: 1}

사이클이 존재하여 위상 정렬이 불가능합니다.
```