# 자료구조 과제 #5

214823 박종현

## 조건

- ✓ 집합을 클래스로 구현
- ✓ 생성 시 공집합으로 생성
- ✓ `add` – 집합에 원소 추가

```
A 집합의 원소를 입력 :
2 6 14 9 9 1 3 5 -3
B 집합의 원소를 입력 :
1 3 9 1 8 5 2 -2
```

- ✓ `search` – 집합에 원소가 있는지 확인 후 `T` 혹은 `F` 출력
- ✓ `union` – 합집합 연산

```python
a.union(b)
```

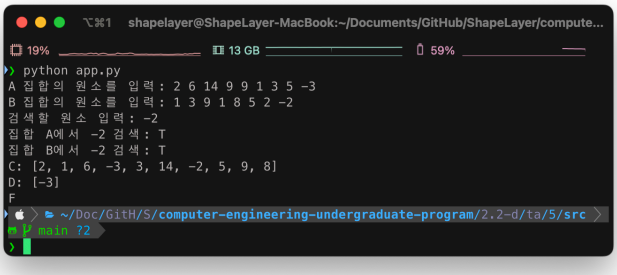
- ✓ `difference` – 차집합 연산

```python
a.difference(b)
```

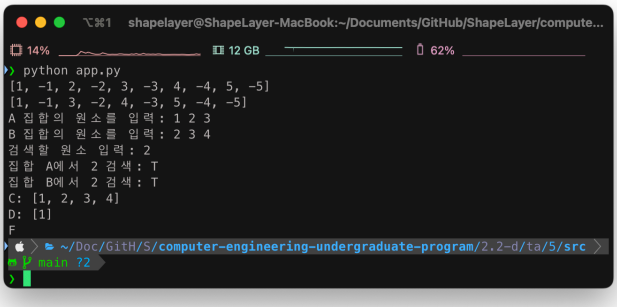
- ✓ `is_empty` – 공집합 여부

```python
c.difference(0).is_empty()
```

## /out/tc1.png 제시 테스트 케이스 입력 결과




## /out/tc1.png 자체 테스트 케이스 입력 결과




* 리스트를 출력하는 첫 두 줄은 자체 테스트 코드에 의해 출력됨.

## /src/app.py 프로그램 소스

```python
from typing import TypeVar
from collections import deque

T = TypeVar('T')

class TreeNode:
    def __init__(self, value: T):
        self.value: T = value
        self.parent: TreeNode = None
```

```
10          self.left: TreeNode = None
11          self.right: TreeNode = None
12
13      def __str__(self) -> str:
14          return f'TreeNode ({self.value}): ' + f'parent({self.parent.value if self.parent else self.parent}), left({self.left.value if self.left else self.left}), right({self.right.value if self.right else self.right})'
15
16  class Tree:
17      def __init__(self):
18          self.entry: TreeNode[T] = None
19
20      def add(self, value: T):
21          node = TreeNode(value)
22          finding = self.search(value)
23          if not finding:
24              self.entry = node
25              return
26
27          if finding.value == value:
28              return
29          if finding.value > value:
30              finding.left = node
31          elif finding.value < value:
32              finding.right = node
33          node.parent = finding
34
35      def remove(self, value: T):
36          finding = self.search(value)
37          if finding.value != value:
38              return
39
40          s: deque[TreeNode[T]] = deque()
41          now = finding
42          while True:
43              if now.left:
44                  s.append(now)
45                  now = now.left
46              elif now.right:
47                  s.append(now)
48                  now = now.right
49              else:
50                  break
51          if s:
52              now = s.pop()
53          while s:
54              next = s.pop()
55              if now is next.left:
56                  now.right = next.right
57              elif now is next.right:
58                  now.left = next.left
59              now = next
60
61          now = now
62          if now.left:
63              now = now.left
```

```
64            elif now.right:
65                now = now.right
66            else:
67                now = None
68
69            parent = finding.parent
70            if not parent:
71                if parent:
72                    now.parent = None
73                self.entry = now
74            else:
75                if finding is parent.left:
76                    parent.left = now
77                elif finding is parent.right:
78                    parent.right = now
79            if now:
80                now.parent = finding.parent
81            del finding
82
83        def search(self, value: T) -> TreeNode:
84            if self.is_empty():
85                return None
86            now = self.entry
87            while True:
88                if now.value > value:
89                    next = now.left
90                    if not next:
91                        return now
92                    now = next
93                elif now.value < value:
94                    next = now.right
95                    if not next:
96                        return now
97                    now = next
98                else:
99                    return now
100
101       def is_empty(self):
102           return self.entry == None
103
104       def union(self, other: 'Tree[T]') -> 'Tree[T]':
105           _new: Tree[T] = Tree()
106           [_new.add(each) for each in self.inorder_traverse()]
107           [_new.add(each) for each in other.inorder_traverse()]
108           return _new
109
110       def difference(self, other: 'Tree[T]') -> 'Tree[T]':
111           _new: Tree[T] = Tree()
112           [_new.add(each) for each in self.inorder_traverse()]
113           for each in other.inorder_traverse():
114               _new.remove(each)
115           return _new
116
117       def inorder_traverse(self) -> list[T]:
118           q: deque[TreeNode[T]] = deque()
```

```
119         result: list[T] = []
120         q.append(self.entry)
121         while q:
122             now = q.popleft()
123             result.append(now.value)
124             if now.left:
125                 q.append(now.left)
126             if now.right:
127                 q.append(now.right)
128         return result
129 
130 def example():
131     a: Tree[int] = Tree()
132     b: Tree[int] = Tree()
133     [a.add(each) for each in map(int, input('A 집합의 원소를 입력: ').split())]
134     [b.add(each) for each in map(int, input('B 집합의 원소를 입력: ').split())]
135 
136     finding = int(input('검색할 원소 입력: '))
137     print(f'집합 A에서 {finding} 검색:', 'T' if a.search(finding) else 'F')
138     print(f'집합 B에서 {finding} 검색:', 'T' if a.search(finding) else 'F')
139 
140     c = a.union(b)
141     d = a.difference(b)
142 
143     print(f'C: {c.inorder_traverse()}')
144     print(f'D: {d.inorder_traverse()}')
145     print('T' if c.difference(d).is_empty() else 'F')
146 
147 def manual_test():
148     tree: Tree[int] = Tree()
149     for i in range(10):
150         tree.add((i // 2 + 1) * (1 + -2 * (i % 2)))
151     print(tree.inorder_traverse())
152     tree.remove(2)
153     print(tree.inorder_traverse())
154     assert tree.difference(tree).is_empty()
155 
156 if __name__ == '__main__':
157     manual_test()
158     example()
159 
```