Symbol Table

Symbol Table이 무엇이고, 어떻게 구현하며, 어떤 경우에 어떻게 활용하는지 이해

01. 예습자료 & 퀴즈 주요 내용 복습
02. 2-3 Tree 활용한 Symbol Table 구현 (기본 BST보다 더 효율적인 Symbol Table 구현 알아보기)
03. BST 활용한 2-3 Tree 구현 (LLRB, Left-Leaning Red Black BST)
04. 1-D Range Search (symbol table 기능 추가)
05. Line-segment Intersection (symbol table & priority queue 활용 예)
06. 실습: Line-segment Intersection 구현

실습 목표: Sweep-line 알고리즘 구현

- Symbol Table과 Priority Queue를 적재적소에 활용해 보기

프로그램 구현 조건

- sweep line algorithm 수행하는 함수 구현

  def sweepLine(**segments**):

- 입력 **segments**: 수평선, 수직선의 리스트로 각 선은 **Segment 클래스 객체**
- 반환 값
  - 모든 교차점의 리스트 반환
  - 리스트의 각 교차점: 교차점 만드는 (수평선, 수직선)의 tuple로 표현 (즉 **두 Segment 객체의 tuple**)
  - Sweep-line 알고리즘이 찾는 순서대로 리스트에 담음

결과를 print하지 말고 반환하세요.

- 이번 시간에 제공한 코드 SweepLine.py에 위 함수 추가해 제출
  - SweepLine.py에 포함된 Segment 클래스는 반드시 사용해야 함
  - 작은 x좌표 순으로 처리하기 위해 위 파일에 이미 import된 PriorityQueue 클래스 사용 (**정렬 사용 시 감점**)
  - 또한 Symbol Table로는 위 파일에 이미 import된 LLRB 클래스 사용 (**dictionary 사용 시 감점**)

프로그램 구현 조건

- 최종 결과물로 SweepLine.py 파일 하나만 제출하며
- 이 파일과 수업에서 제공한 RedBlackBST.py 만으로 코드가 동작해야 함

- import는 원래 SweepLine.py 파일에서 import하던 2개 패키지 외에는 추가로 할 수 없음 (queue.PriorityQueue, RedBlackBST.LLRB)

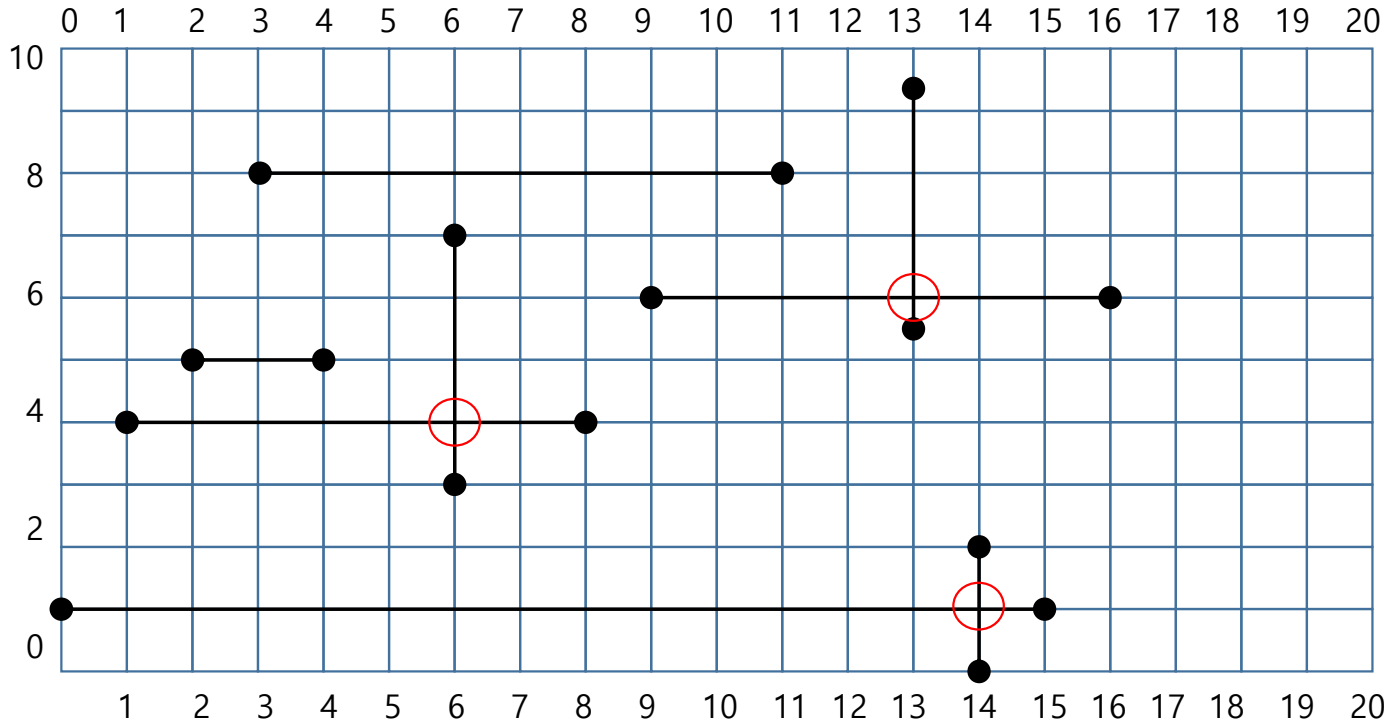
- SweepLine.py 내에 이미 구현되어 있는 코드는 채점에 사용되는 코드이니 제거하거나 수정하지 말 것
- __main__ 아래의 채점 코드는 각 테스트 케이스에 대해 P/F 혹은 Pass/Fail을 출력하니 검증에 활용하세요.

- 정확도 테스트만 있고 속도 테스트는 별도로 없으나
- 정확도 테스트 케이스 각각에 대해 5초 이상의 시간이 걸리면 통과하지 못한 것으로 봅니다.
- 올바르게 코드를 작성했다면 모든 테스트 케이스에 대해 거의 즉답이 나옵니다.

프로그램 입출력 예


```
intersections = sweepLine([Segment(0,1,15,1), Segment(14,0,14,2), Segment(1,4,8,4), Segment(6,3,6,7),\
        Segment(2,5,4,5), Segment(3,8,11,8), Segment(9,6,16,6), Segment(13,5.5,13,9.5)])
print(intersections)
```

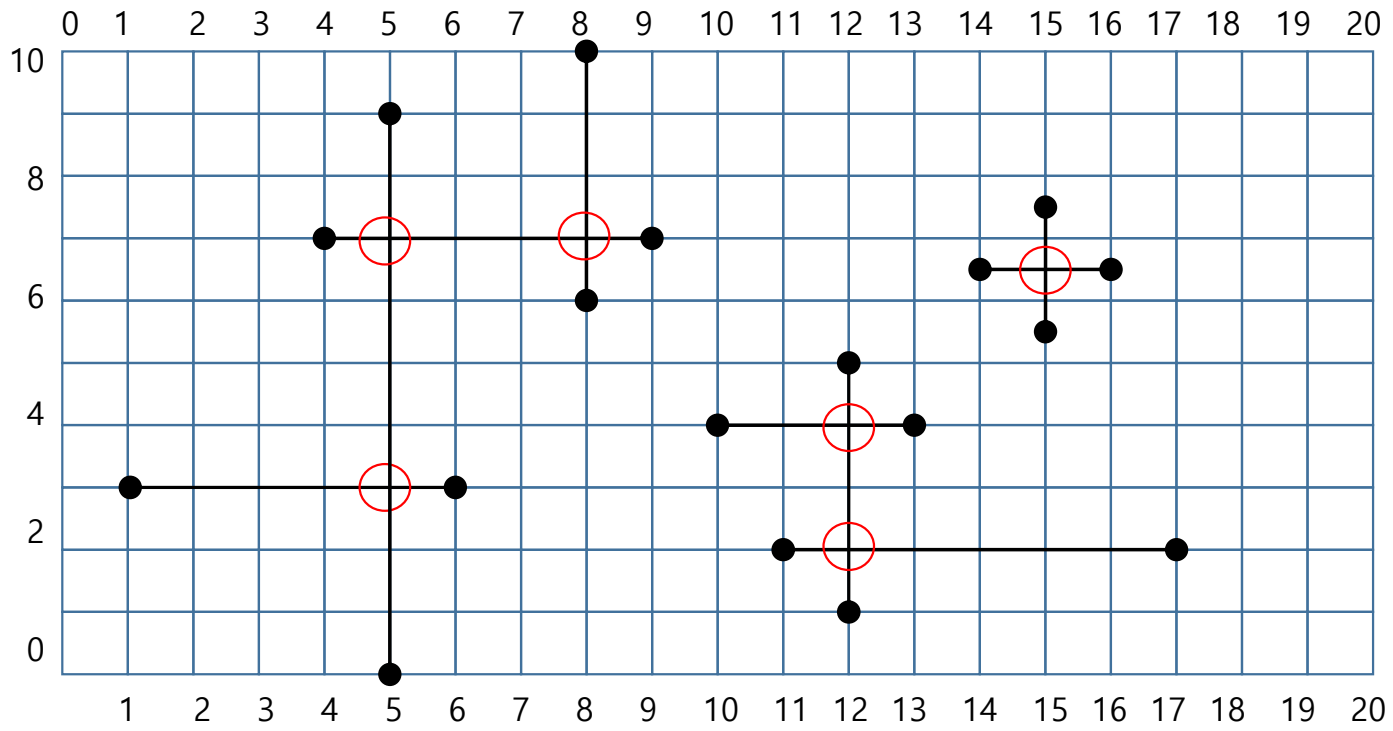
```
# 실행 결과: 3개 교차점
[((1,4)--(8,4), (6,3)--(6,7)), ((9,6)--(16,6), (13,5.5)--(13,9.5)), ((0,1)--(15,1), (14,0)--(14,2))]
```



프로그램 입출력 예


```
intersections = sweepLine([Segment(1,3,6,3), Segment(5,0,5,9), Segment(4,7,9,7), Segment(8,6,8,10),\
        Segment(10,4,13,4), Segment(11,2,17,2), Segment(12,1,12,5), Segment(15,5.5,15,7.5), Segment(14,6.5,16,6.5)])
print(intersections)
```

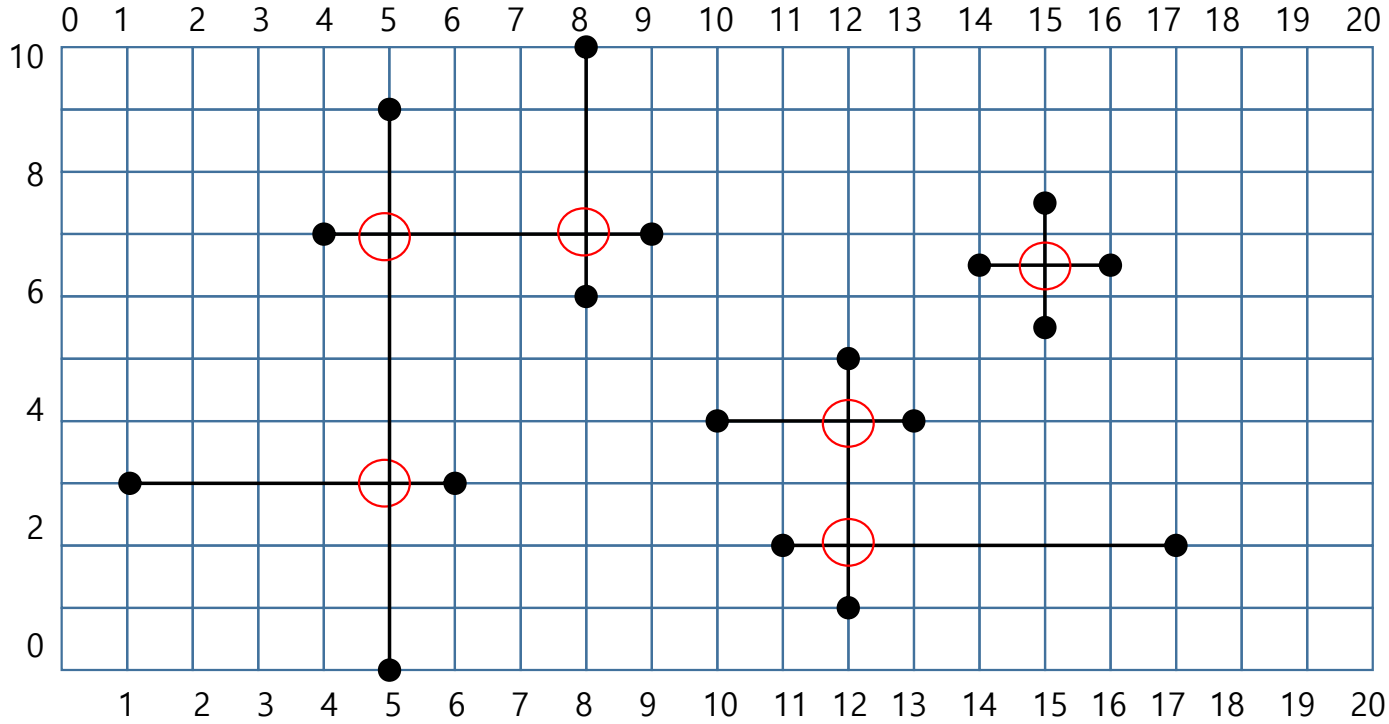
```
# 실행 결과: 6개 교차점
[((1,3)--(6,3), (5,0)--(5,9)), ((4,7)--(9,7), (5,0)--(5,9)), ((4,7)--(9,7), (8,6)--(8,10)), ((11,2)--(17,2), (12,1)--
(12,5)), ((10,4)--(13,4), (12,1)--(12,5)), ((14,6.5)--(16,6.5), (15,5.5)--(15,7.5))]
```



그 외 유의사항: 프로그램 입력 조건

- x, y좌표는 임의의 실수로, **모두 서로 다름** (같은 경우는 입력으로 들어오지 않음)

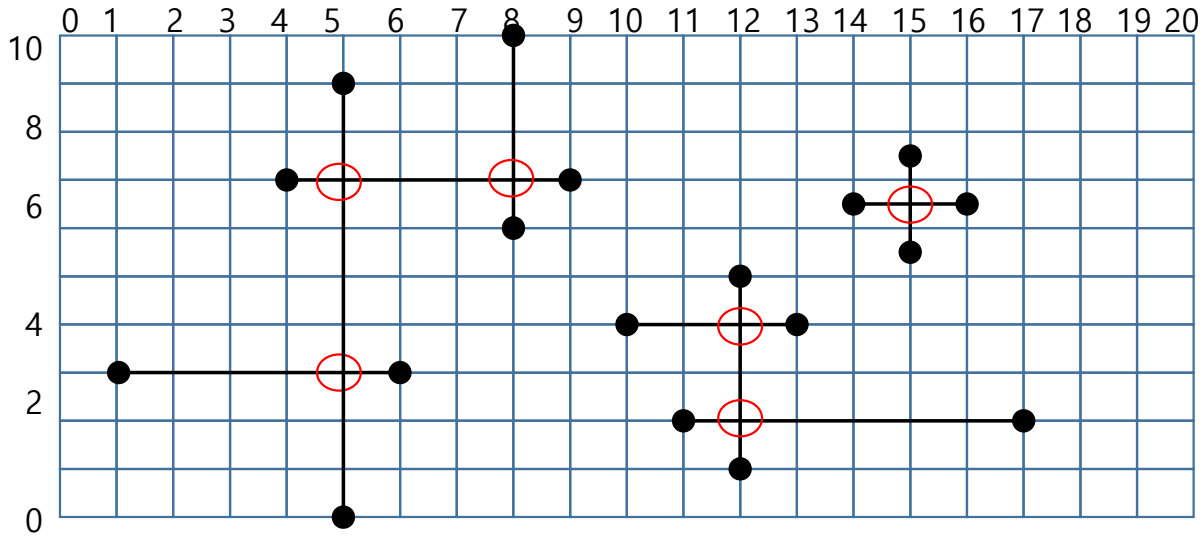
```
intersections = sweepLine([Segment(1,3,6,3), Segment(5,0,5,9), Segment(4,7,9,7), Segment(8,6,8,10),\
        Segment(10,4,13,4), Segment(11,2,17,2), Segment(12,1,12,5), Segment(15,5.5,15,7.5), Segment(14,6.5,16,6.5)])
```



그 외 유의사항: sweepLine()이 반환하는 교차하는 Segment 쌍(2-tuple)의 리스트

- 각 Segment 쌍(2-tuple)은 (수평선 객체, 수직선 객체) 순으로 담기
- 교점의 x좌표가 작은 경우부터 넣기
  - sweep-line 알고리즘을 x좌표 작은 순서로(왼쪽 → 오른쪽 방향으로) 적용하면 자연스럽게 이렇게 됨
- x좌표 같은 교점 여럿이면, y좌표 작은 것부터 넣기
  - range search가 정렬된 순서로 결과를 반환하므로, 그 순서대로 사용하면 자연스럽게 이렇게 됨

```
# 실행 결과: 6개 교차점
[((1,3)--(6,3), (5,0)--(5,9)), ((4,7)--(9,7), (5,0)--(5,9)), ((4,7)--(9,7), (8,6)--(8,10)), ((11,2)--(17,2), (12,1)--
(12,5)), ((10,4)--(13,4), (12,1)--(12,5)), ((14,6.5)--(16,6.5), (15,5.5)--(15,7.5))]
```



구현된 API 정리 Segment Class – 한 line 나타내는 클래스 (수직선 or 수평선)

수업 자료에 포함된 SweepLine.py 내에 구현된 클래스

```
# 이미 구현된 클래스로 각 함수의 의미 이해하고 사용하기
class Segment:
    def __init__(self, x1, y1, x2, y2): # Segment 생성자. (x1,y1)과 (x2,y2)를 잇는 직선 생성
        # 이들은 멤버 변수 self.x1, self.y1, self.x2, self.y2에 저장됨
        # 수직선인 경우 (x1 == x2): y1, y2 중 더 작은 값이 self.y1에, 더 큰 값이 self.y2에 저장됨
        # 수평선인 경우 (y1 == y2): x1, x2 중 더 작은 값이 self.x1에, 더 큰 값이 self.x2에 저장됨

    def isHorizontal(self): # Segment 객체가 수평선이면 True를, 그렇지 않으면 False 반환
    def isVertical(self): # Segment 객체가 수직선이면 True를, 그렇지 않으면 False 반환

    def __str__(self): # Segment 객체에 저장된 내용을 string 형태로 반환. 출력, debugging에 사용
                       # 반환하는 스트링: "(x1,y1)--(x2,y2)"

    def __eq__(self): # == 사용해 두 Segment 객체가 같음을 확인할 때 호출됨. Grading, debugging에 사용
```

Python 기본 제공 PQ 클래스: minPQ이나, tuple로 key 지정해 다른 우선순위 지정 가능

MinPQ 제공 함수	설명
PriorityQueue()	Constructor(생성자). MinPQ 객체 생성
put(k)	원소 k를 PQ에 추가
get()	가장 작은 원소를 제거하며 반환
qsize()	PQ에 저장된 원소의 수를 반환

```
from queue import PriorityQueue
pq1 = PriorityQueue()
pq1.put(('A',100))
pq1.put(('B',20))
pq1.put(('C',150))
while pq.qsize() > 0:
    print(pq.get())
================================
('A',100)
('B',20)
('C',150)
```

알파벳을 key로 minPQ

```
from queue import PriorityQueue
pq1 = PriorityQueue()
pq1.put((100,'A'))
pq1.put((20,'B'))
pq1.put((150,'C'))
while pq.qsize() > 0:
    print(pq.get())
================================
(20,'B')
(100,'A')
(150,'C')
```

숫자를 key로 minPQ

Tuple 추가할 때는 tuple을 괄호()로 싸는 것 유의

```
from queue import PriorityQueue
pq1 = PriorityQueue()
pq1.put((-100,'A'))
pq1.put((-20,'B'))
pq1.put((-150,'C'))
while pq.qsize() > 0:
    a = pq.get()
    print((a[1], -a[0]))
================================
('C',150)
('A',100)
('B',20)
```

숫자를 key로 maxPQ

구현된 API 정리: LLRB Class - Left-Leaning Red-Black BST 나타내는 클래스

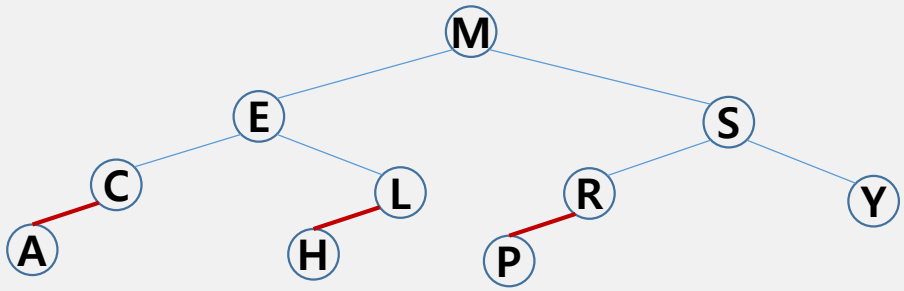
수업 자료에 포함된 RedBlackBST.py 내에 구현된 클래스

```
class LLRB:
    # 이미 구현된 함수
    def __init__(self): # LLRB 생성자. 비어 있는 LLRB 생성

    def put(self, key, value): # (key, value) 쌍을 BST에 추가. key가 이미 존재한다면 value만 update

    def get(self, key): # key에 대응되는 value 찾아 반환
    def contains(self, key): # key가 저장되어 있다면 True를, 그렇지 않다면 False 반환

    def delete(self, key): # key 및 대응되는 value를 BST에서 제거

    def rank(self, key): # key보다 작은 key의 개수 반환

    def inorder(self): # BST에 저장된 key를 정렬된 순서로 리스트에 저장해 반환
    def levelorder(self): # BST에 저장된 key를 Top→Bottom 순서로(깊이(level) 순으로) 리스트에 저장해 반환
    … # 그 외 함수는 위 함수들의 구현에 내부적으로 사용되었거나, 본 과제와 직접적으로 연관 없음

    def rangeCount(self, lo, hi): # lo~hi 범위에 속한 key의 개수 반환
    def rangeSearch(self, lo, hi): # lo~hi 범위에 속한 key를 정렬된 순서로 리스트에 저장해 반환
```

이 페이지에 있는 멤버함수만으로 필요한 기능을 다 구현할 수 있으며, 그 외 함수는 호출하지 마세요.

LLRB 클래스의 활용 예


```
bst2 = LLRB()
bst2.put("S",1)
bst2.put("E",2)
bst2.put("Y",3)
bst2.put("A",4)
bst2.put("R",5)
bst2.put("C",6)
bst2.put("H",7)
bst2.put("M",8)
bst2.put("L",9)
bst2.put("P",10)
print(bst2.rank("H"))
print(bst2.select(4))
print("level order", bst2.levelorder())
print("inorder",bst2.inorder())
print("range count", bst2.rangeCount("F", "T"))
print("range search", bst2.rangeSearch("F", "T"))
print("range count", bst2.rangeCount("B", "I"))
print("range search", bst2.rangeSearch("B", "I"))
print("range count", bst2.rangeCount("C", "H"))
print("range search", bst2.rangeSearch("C", "H"))
print("range count", bst2.rangeCount("J", "R"))
print("range search", bst2.rangeSearch("J", "R"))
```




```
# 실행 결과
3
L
level order ['M', 'E', 'S', 'C', 'L', 'R', 'Y', 'A', 'H', 'P']
inorder ['A', 'C', 'E', 'H', 'L', 'M', 'P', 'R', 'S', 'Y']
range count 6
range search ['H', 'L', 'M', 'P', 'R', 'S']
range count 3
range search ['C', 'E', 'H']
range count 3
range search ['C', 'E', 'H']
range count 4
range search ['L', 'M', 'P', 'R']
```

구현할 API 정리: sweepLine()의 수도코드

```
# 구현해야 할 함수. Segment 객체의 리스트를 입력으로 받아, 교차하는 Segment 쌍(2-tuple)의 리스트를 반환
def sweepLine(segments)
    # queue.PriorityQueue 클래스 사용해 minPQ 객체 생성
    #
    # minPQ에 담는 원소는 segments 리스트에 담긴 Segment 객체이며,
    # 특히 x축을 기준으로 정렬되도록 아래와 같은 2-tuple 형태로 담을 것
    # (Segment 객체의 x 좌표, Segment 객체)
    #
    # 그 후에는 while loop을 사용해 다음을 반복
    #     minPQ에서 가장 작은 원소를 get(). 이 원소는 2-tuple일 것이며 e라 하겠음
    #     If e가 수평선이라면: e의 y 좌표를 LLRB에 추가하거나 혹은 제거
    #     If e가 수직선이라면: e와 교차하는 수평선이 LLRB에 있다면 교차점을 결과 리스트에 추가
    #
    # 결과 리스트 반환
```

정리: Symbol Table 무엇이고, 어떻게 활용하며, 어떤 경우 활용하는지 이해

- Symbol Table: (key, value) 쌍 저장하며, key를 검색어로 value 찾는 것이 주 기능
- put(), get() 및 get()을 확장한 여러 작업을 ~logN 시간에 효율적으로 구현하기 위해 **Binary Search Tree 구조** 사용 (왼쪽 자식 < 노드 < 오른쪽 자식)
  - Tree 구조 + 왼쪽<가운데<오른쪽 조건 ➔ Key 값에 따른 효율적인 추가, 탐색 가능하게 함
- 깊이를 ~logN으로 제한하기 위해 2-3 tree로 변형
- 2-3 tree를 정확하고 효율적으로 구현하기 위해 BST 코드를 재활용하며, 특히 LLRB (Left-Leaning Red-Black) Tree 형태로 표현

- BST에 기반한 Symbol Table은 put(), get() 뿐 아니라, get()을 확장한 **다양한 탐색 작업**을 **효율적으로 수행** 가능하므로 데이터베이스 구현의 기본의 됨
- Range count, range search, nearest neighbor, floor, ceiling, rank, select, min, max, …

중간고사 일정 (lms 공지사항 '2025-2학기 일정' 참조)

- 10.21(화) 10:00~11:00, 14:00~15:00: 중간고사 공지 및 질의응답
  - lms에서 시험 관련 유의사항을 읽어보고 **질문이 있는 경우만 IT5-234에서** 하세요. 예습/퀴즈는 없습니다.
- 10.28(화) **오후 6~9시**: 중간고사 @ IT5-224(오전 분반), IT5-225(오후 분반)
  - 10.28(화) 저녁에는 종합설계프로젝트2 화요일 분반 발표가 없음 다시 확인
- 11.4(화) 수업 있음. 수업 전날까지 예습/퀴즈 있음

실습 과제 (프로그래밍) 유의사항

- **개별 평가**이므로 코드는 꼭 각자 **스스로 작성**하세요.
- 본인이 제출한 코드의 의미를 알아야 하고, 또한 다시 작성할 수 있어야 합니다.
- 이에 대해 매주 무작위로 기존에 제출한 과제를 스스로 작성했는지 확인합니다.

- 중간시험과 기말시험에는 실기 시험이 있습니다.
- 시험은 closed book이므로 시험 중에는 AI의 도움을 받을 수 없으며, 수업 자료와 코드를 참조하며 코드를 작성할 수도 없습니다. 따라서 매주 실습 과제를 할 때 스스로 직접 작성해 보는 연습을 해두세요.

실습 문제 풀이 & 질의응답

- 작성한 코드는 lms > 강의실 > 오늘 수업 > 실습 과제 제출함에 제출
- 제출 마감: **내일 11:59pm까지**
- **제출하면 기본 점수가 있지만 제출하지 않으면 0점이므로, 마감 전에** 작성한 코드를 **꼭 제출**하세요.

- **실습 종료 시간까지는 실습실에서 질문을 받습니다.**
- **일찍 퇴실할 때는 출석을 확인**하고 퇴실하세요.

- 실습 과제 **채점 관련 질문은 튜터에게** 해주세요.
- **다음 시간도 수업 전날까지 예습 & 퀴즈 완료**해 주세요.

공지사항: lms에 과제물을 올릴 때 기존에 올린 파일과 같은 이름의 파일 올리면 -1, -2 등이 붙는데, 이때문에 감점되지는 않습니다. 또한 수업 자료도 변경되면 마찬가지로 숫자가 붙어 있으니 변경 여부 확인해 보세요 (첫 버전에서 약간씩 수정이 있습니다. 특히 실습 과제 요건 변경 있는지 꼭 확인).