Data Structure_Assignment#2

- **기초학부 201911189 한현영**

#A4 - Linked List (5+1 points)

#A4-1(1 point)
singly_linked_list의 경우, x와 y를 swap하는 과정에서 x의 previous node와 y의 previous node를 모두 알아야 한다. 이 과정에서 각각의 previous node를 찾기 위해 O(n) time이 소요된다. 반면 doubly_linked_list의 경우 x와 y의 previous 노트 탐색을 O(1) time에 수행 가능하다.(모든 node가 previous_node_field와 next_node_field를 갖고 있기 때문) 따라서 singly_linked_list의 작업 소요 시간이 더 길다.

```
#A4-2(1 point)
from positional_list import PositionalList
class personal_PL(PositionalList):
    def __reversed__(self):
        cursor = self.last()
        while cursor is not None:
            yield cursor.element()
            cursor = self.before(cursor)
```

```
#A4-3(1 point)
from positional_list import PositionalList
class personal_PL(PositionalList):
    def add_last(self, elem):
        if is_empty(self) == True: #리스트가 비어있으면
            return self.add_first(elem) #header 노드 뒤에 elem 삽입
        else:
            last = self.last()
            return self.add_after(last, elem)

    def add_before(self, pos, elem):
        if is_empty(self) == True:
            return self.add_first(elem)
        else:
            before_node = self.before(self.before(pos)) #pos 이전의 이전 node를
before_node에 할당
            return self.add_after(before_node, elem) #before_node의 다음 node에
elem 삽입
```

```
#A4-4(1 point)
from positional_list import PositionalList
class personal_PL(PositionalList):
    def max(self):
        k = 0
        for i in L.__iter__():
            max_elem = k
            if k < i:
                max_elem = i
        return max_elem
```

```
#A4-5(1 point)
from positional_list import PositionalList
class personal_PL(PositionalList):
    def find(self, elem):
        for i in L.__iter__():
            if elem == i:
                return self._make_position(i)
            else:
                return None
```

```
#A4-Bonus(1 point)
from positional_list import PositionalList
class personal_PL(PositionalList):
    walk = self.first()
    def find(self, elem):
        if self.is_empty():
            raise EmptyError('The list is empty')
        else:
            if self.first().element() == elem:
                return self._make_position(first)
            else:
                walk = self.after(walk)
                if self.element() == elem:
                    return self._make_position(walk)
                else:
                    self.find(self.element())
```

#A5 - Trees (7 points)

#A5-1(1 point)

1. /user/rt/courses/
2. /user/rt/courses/, cs016/, homeworks/, programs/, cs252/, projects/, papers/, /demos/
3. cs016/ has 9 decendants.
4. It has only 1 ancestor.(//user/rt/courses/)
5. Its sibilings are grades/ and programs/.
6. papers/ and demos/
7. 4
8. 5
9.

#A5-2 (1 point)

```
     E
    / \
   X   N
  / \
 A   U
/ \
M   F
```

```python
#A5-3 (2 points)
def root():
    for node in T:
        if f(node) == 0:
            return node

def is_root(p):
    return f(p) == 0

def parent(p):
    if f(p) % 2 == 0:
        for node in T:
            return T[(f(p))/2-1]
    elif f(p) % 2 == 1:
        for node in T:
            return T[(f(p)-1)/2]

def left(p):
    for node in T:
        if node == parent(T[2*f(p)+1]):
            return node
        else:
            return None

def right(p):
    for node in T:
        if node == parent(T[2*f(p)+2]):
            return node
        else:
            return None

def is_leaf(p):
    if left(p) != None or right(p) != None:
        False
    else:
        True
```

```
#A5-4 (3 points)
from linked_binary_tre import LinkedBinaryTree
class personal_LBP(LinkedBinaryTree):
    def _delete_subtree(self, p):
        while self.num_children(p) != 0:
            if self.right(p) != None:
                self._delete_subtree(self.right(p))
            self._delete(self.right(p))
            elif self.left(p) != None:
                self._delete_subtree(self.left(p))
            self._delete(self.left(p))

running time의 경우, O(n) time이 소요된다.
```

#A6 - Priority Queues, Heaps (8+2 points)

#A6-1 (1 point)
heap에서의 remove_min 실행의 시간복잡도는 O(log(n))이다. 그런데, 문제에서 removing smallest elements의 횟수를 log(n)이라 지정하였으므로, 총 시간복잡도는 O((log(n))^2)이다.

#A6-2 (1 point)

1. remove_min()
   In PQ: [(1, D), (4, B), (5, A), (7, F)] ---> (1, D)
2. remove_min()
   In PQ: [(3, j), (4, B), (5, A), (6, L), (7, F)] ---> (3, j)
3. remove_min()
   In PQ: [(4, B), (5, A), (6, L), (7, F)] ---> (4, B)
4. remove_min()
   In PQ: [(5, A), (6, L), (7, F), (8, G)] ---> (5, A)
5. remove_min()
   In PQ: [(2, H), (6, L), (7, F), (8, G)] ---> (2, H)
6. remove_min()
   In PQ: [(6, L), (7, F), (8, G)] ---> (6, L)

Result: (1, D), (3, j), (4, B), (5, A), (2, H), (6, L)

#A6-3 (1 point)

- Priority Queue
  future event를 추가하고, time stamp가 가장 낮은 것부터 extract 해야한다. 따라서 key값의 우선순위를 비교하여 pop하는 priority queue의 구조와 흡사하다.

#A6-4 (1 point)

Input: Sequence: (22, 15, 36, 44, 10, 3, 9, 13, 29, 25) Queue: ()
Phase1
a. S: (15, 36, 44, 10, 3, 9, 13, 29, 25)   Q: (22)
b. S: (36, 44, 10, 3, 9, 13, 29, 25)   Q: (22, 15)

...
j. S: () Q: (22, 15, 36, 44, 10, 3, 9, 13, 29, 25)

Phase2
a. S: (3) Q: (22, 15, 36, 44, 10, 9, 13, 29, 25)
b. S: (3, 9) Q: (22, 15, 36, 44, 10, 13, 29, 25)
c. S: (3, 9, 10) Q: (22, 15, 36, 44, 13, 29, 25)
d. S: (3, 9, 10, 13) Q: (22, 15, 36, 44, 29, 25)
e. S: (3, 9, 10, 13, 15) Q: (22, 36, 44, 29, 25)
f. S: (3, 9, 10, 13, 15, 22) Q: (36, 44, 29, 25)
g. S: (3, 9, 10, 13, 15, 22, 25) Q: (36, 44, 29)
h. S: (3, 9, 10, 13, 15, 22, 25, 29) Q: (36, 44)
i. S: (3, 9, 10, 13, 15, 22, 25, 29, 36) Q: (44)
j. S: (3, 9, 10, 13, 15, 22, 25, 29, 36, 44) Q: ()

#A6-5 (1 point)

Input: Sequence: (22, 15, 36, 44, 10, 3, 9, 13, 29, 25) Priority Queue: ()
Phase1

a. S: (15, 36, 44, 10, 3, 9, 13, 29, 25) Q: (22)
b. S: (36, 44, 10, 3, 9, 13, 29, 25) Q: (15, 22)
c. S: (44, 10, 3, 9, 13, 29, 25) Q: (15, 22, 36)
d. S: (10, 3, 9, 13, 29, 25) Q: (15, 22, 36, 44)
e. S: (3, 9, 13, 29, 25) Q: (10, 15, 22, 36, 44)
f. S: (9, 13, 29, 25) Q: (3, 10, 15, 22, 36, 44)
g. S: (13, 29, 25) Q: (3, 9, 10, 15, 22, 36, 44)
h. S: (29, 25) Q: (3, 9, 10, 13, 15, 22, 36, 44)
i. S: (25) Q: (3, 9, 10, 13, 15, 22, 29, 36, 44)
j. S: () Q: (3, 9, 10, 13, 15, 22, 25, 29, 35, 44)

Phadse2
a. S: (3) Q: (9, 10, 13, 15, 22, 25, 29, 35, 44)
b. S: (3, 9) Q: (10, 13, 15, 22, 25, 29, 35, 44)
...
j. S: (3, 9, 10, 13, 15, 22, 25, 29, 35, 44) Q: ()

```
#A6-6 (3 points)
from heap_priority_queue import HeapPriorityQueue
class heapq(HeapPriorityQueue):
    def heappushpop(self, elem):
        self.add(elem[0], elem[1])
        return self.remove_min() #downheap의 경우 remove_min에서 자동으로 수행

    def heapreplace(self, elem):
        show = self.remove_min() #조건에 맞추어 새로운 element가 add되기 전 최솟값을 
추출
        self.add(elem[0], elem[1])
        return show
```

```
#A6-Bonus (2 points)
```