문제에 나온 용어나 공식에 대해서 질문할 수 있습니다. 시험 시간은 총 100 분이며 퇴실 가능시간은 50 분간 후 입니다. 퇴실가능 시간 이후 풀이를 마친 학생은 답안지를 제출하고 퇴실할 수 있습니다.

Name: _____ Student ID: _____ Class: _____

담당교수: 김종규 _____

---

1. (15 points) Big-O 와 관련한 다음 질문에 답하시오.

   (a) 알고리즘의 복잡도가 다음과 같이 주어져 있을 때, 알고리즘이 가장 빠르게 수행되는 복잡도에서 가장 느리게 수행되는 복잡도의 순으로 기술하시오. $O(1), O(n \log n), O(n), O(n^2), O(\log n), O(2^n), O(n!)$

   Answer:
   $O(1), O(\log n), O(n), O(n \log n), O(n^2), O(2^n), O(n!)$

   (b) 다음에서 `print 1` 이 수행되는 횟수는? Big-O 로 나타내고 그 이유를 간단히 설명하시오. (recursive function 임에 주의)

```
def Ones(n):
  if n == 0:
    print 1
  else
    for i = 1 to n
      Ones(n-1)
```

   Answer:

$$T(0) = 1$$

$$T(n) = nT(n-1)$$

$$n! = n(n-1)(n-2)(n-3)\cdots 1$$

$$\therefore O(n!)$$

2. (20 points) 어떤 시스템에 있는 n 개의 데이터는 이미 정렬되어 array `A` 에 들어있다고 가정하자.

   (a) 이 배열에서 주어진 데이터를 찾는데 소요되는 최선의 알고리즘으로 수행한다고 가정하고 그 복잡도를 Big-O 로 나타내면 얼마인가? 그 이유를 간단히 설명하시오.

   Answer:
   이미 정렬되어 있으므로 binary search algorithm 을 적용하면 $O(\lg n)$ 에 수행된다.

(b) 이 시스템에는 새로운 데이터가 들어오는 일은 거의 없다. 그래서 많아야 $\lg n$ 개만큼이 추가로 들어올 것이라고 안전하게 예상할 수 있다. 이렇게 새롭게 들어온 데이터를 `A` 와 별도로 sorting 되지 않은 linked list `B` 에 넣는다고 가정하면 새로운 데이터를 넣는데 걸리는 시간은 얼마인가? 그 이유를 간단히 설명하시오.

Answer:
linked list 에 추가하는 것이므로 $O(1)$

(c) 새로운 데이터의 총 수가 $\lg n$ 미만이라고 할 때 이 시스템에서 데이터를 찾기 위해서는 `A` 와 `B` 모두를 검색하여야 한다. 소요되는 총 시간을 Big-O 로 나타내면 얼마인가? 그 이유를 간단히 설명하시오.

Answer:
`A` 에서 찾는 시간 $O(\lg n)$, `B` 에서 찾는 시간 $O(\lg n)$
$\therefore O(\lg n)$

(d) balanced tree 를 사용하는 것과 비교해서 Big-O 의 관점에서 이익이 있는가? 그 이유를 간단히 설명하시오.

Answer:
search 의 경우: 없다. balanced tree: $O(\lg(n + \lg n))$ For large n, without loss of generality

$$\lg(n + \lg n) \leq \lg(n \times \lg n)$$

$$O(\lg(n + \lg n)) = O(\lg(n \times \lg n) = O(\lg n + \lg\lg n) = O(\lg n)$$

$O(\lg(n + \lg n) \approx O(\lg n)$ 으로 설명한 경우 (1 점 감점)
설명이 없는 경우 (3 점 감점)
insert 의 경우도 설명한 경우 추가 1 점: $O(\lg n) \rightarrow O(1)$ 약간의 이익이 있다.

3. (20 points) Abstract data type DSet 에 새로운 값을 넣는 것을 `dset_insert()` 특정한 값을 찾는 것을 `dset_search()`, 그리고 찾은 값을 삭제하는 것을 `dset_delete()` 라고 정의하자. 예를 들어 다음과 같이 사용할 수 있다.

```
DSet d
dset_insert(d, 10)
n = dset_search(d, 10)
dset_delete(d, n)
```

이를 구현하기 위하여 다음과 같은 내용을 조사하였다.

A. Stack
B. Queue
C. Heap
D. Array
E. Singly linked list
F. Doubly linked list
G. Binary search tree

H. Red-black tree

(a) 각 연산의 시간 비용 (Big-O) 에 제약이 없어서 원하는 기능을 제공하기만 하면 어떤 것이든 사용할 수 있다고 가정하자. 이 때 조사된 내용 중 DSet 을 구현할 때 사용하기에 적합하지 않은 것은? 그 이유를 간단히 설명하시오.

Answer:
A (stack), B (queue)
stack 과 queue 은 abstract data type 으로 임의의 search 를 지원하지 않는다. 따라서 delete 도 구현될 수 없다.

(b) 만일 다른 모든 연산의 수행시간에는 제약이 없고 `dset_insert()` 만 $O(1)$ 에 수행되어야 한다는 조건이 있다면 조사된 내용중 어떤 것이 적합한가? 가능한 방법을 모두 선택하고 그 이유를 간단히 설명하시오.

Answer:
D (array), E (singly linked list), F (doubly linked list)
Array 의 경우 마지막을 기억하고 있으면 되고, linked list 의 경우 insert 연산은 $O(1)$ 이다.

(c) 만일 다른 모든 연산의 수행시간에는 제약이 없고 단 두 개의 연산, 즉 `dset_insert()` 와 `dset_delete()` 가 모두 $O(1)$ 에 수행되어야 한다는 조건이 있다면 조사된 내용중 어떤 것이 적합한가? 가능한 방법을 모두 선택하고 그 이유를 간단히 설명하시오.

Answer:
F (doubly linked list)
임의의 위치에서 $O(1)$ 의 delete 를 지원하는 자료구조는 doubly linked list 밖에 없다.

(d) 만일 어떤 연산이 많이 수행될 지 사전에 알 수 없기 때문에 최악의 경우를 상정하여 DSet 을 구현하여야 한다. 최악의 경우에도 최선의 성능을 내기 위해서는 조사된 내용중 어떤 것을 활용하는 것이 적합한가? 가능한 방법을 모두 선택하고 그 이유를 간단히 설명하시오.

Answer:
G (red-black tree)
Red-black tree 는 insert, delete, search 모두 $O(\lg n)$ 으로 균형있게 수행될 수 있다.

(e) 만일 DSet 에 현재까지 insert 한 값 중 가장 큰 (혹은 가장 작은) 값을 찾아 제거하는데 소요되는 시간이 작아야 한다면 가장 적합한 데이터 구조는 무엇인가? *그 이유를 간단히 설명하시오.*

Answer:
C (heap)
Heap 은 가장 작은 (혹은 큰) 값을 찾는데 $O(1)$ 의 시간이 소요되고 이를 제거하는데 $O(\lg n)$ 의 시간이 소요된다.

4. (10 points) `x` 가 binary search tree 의 한 노드일 때, left, right, key 값을 다음과 같이 표시한다고 하자. 여기서 `print(a,b,...)` 는 `a,b,...` 의 내용을 출력하는 함수이다.

```
test(x):
  left = x.left
  right = x.right
```

```
    key = x.key
    if x.left == NIL:
      print(x.key, "No more left child")
    else
      return test(x.left)
```

(a) 이 binary search tree 의 inorder traversal 알고리즘을 recursion 을 이용하여 작성하시오.

```
inorder(x):
+-------------------------------------------------------------+
|                                                             |
|                                                             |
|                                                             |
|                                                             |
|                                                             |
|                                                             |
+-------------------------------------------------------------+
```

Answer:

```
inorder(x):
|--- ---+--- ---+--- ---+--- ---+--- ---+--- ---+--- ---+--- ---|
    if x == NIL:
      return;
    else:
      inorder(x.left);
      print(x.key);
      inorder(x.right);
|--- ---+--- ---+--- ---+--- ---+--- ---+--- ---+--- ---+--- ---|
```

(b) 이 binary search tree 에서 주어진 값 `k` 와 일치하는 key 값 `x.key` 를 갖고 있는 노드를 찾는 함수 `search()` 를 *recursion 을 사용하지* 말고 iteration 만으로 작성하시오.

```
Tree-search(x,k):
+-------------------------------------------------------------+
|                                                             |
|                                                             |
|                                                             |
|                                                             |
|                                                             |
|                                                             |
+-------------------------------------------------------------+
```

Answer:

```
Tree-search(x,k):
+-------------------------------------------------------------+
```

```
  while x != NIL and k != x.key:
    if k < x.key:
      x = x.left
    else:
      x = x.right
    end while
+-------------------------------------------------------------+
```

5. (20 points) 다음과 같은 binary search tree 가 주어져 있다고 가정하자

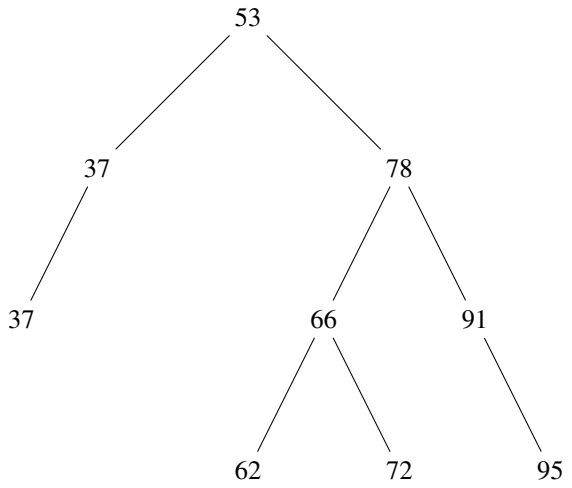

그림 1: Binary search tree

(a) 각 노드에 적절한 색을 칠하면 black height 가 2 인 정당한 red-black tree 가 될 수 있다. 이 트리를 그리시오. 만일 색을 칠하는 것이 불편한 경우 red 는 rectangle 로 black 은 circle 로 표시하시오.

Answer:
그림 2 참조.

(b) 이 트리에 red-black tree insert 알고리즘을 사용하여 60 을 insert 하고 결과로 생성된 tree 를 그리시오.

Answer:
red-black tree 조건만 맞춘 경우 2 점 감점
step 1: 60 은 62 의 왼쪽으로 들어간다. 그림 3 참조.

step 2: Case 1 에 해당 (z's uncle is red) → color change. z=66 그림 4 참조.

step 3: Case 2 에 해당 (zig-zag) → straighten (without color change). z=78 그림 5 참조.

step 4: Case 3 에 해당 (skew) → rotate (with color change). z=78 그림 6 참조.

6. (15 points) 다음 그래프는 각 edge 를 통과하는데 소요되는 비용을 표시한 weighted graph 이다. A 로 부터 시작하는 각각의 vertex 에 도착하는데 소요되는 비용과 경로 (path) 는 Dijkstra 알고리즘을 적용하여 구할 수 있다. Dijkstra 알고리즘을 적용하여 찾은 비용과 경로 정보를 다음 표에 작성하시오.

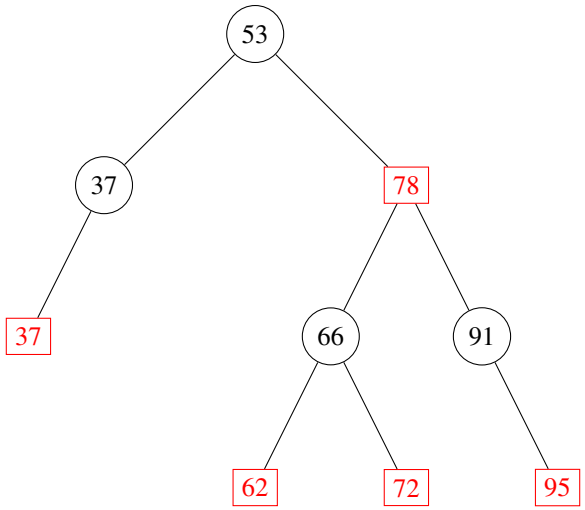

그림 2: Red-black tree

Vertex	Cost (d)	Pred (π)
A		
B		
C		
D		
E		
F		
G		

Answer:

다음과 같은 과정을 거친다. 최종상태만 기술하면 정답인정. 표 1 참조

End of exam. Thank you

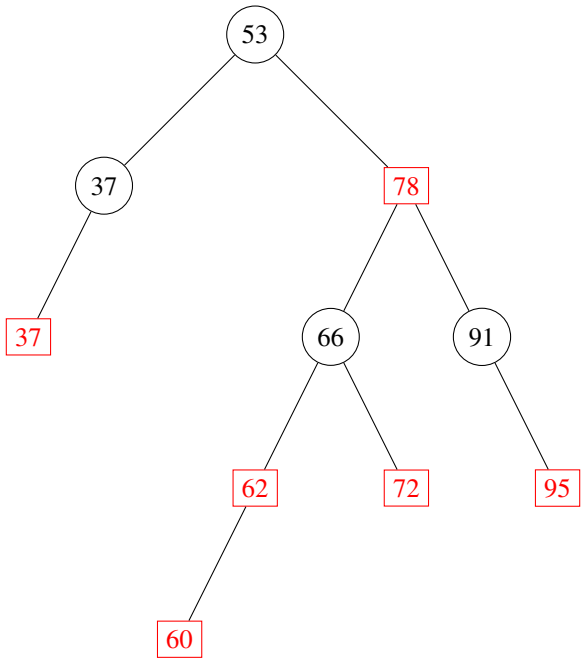

그림 3: Red-black tree (insert 60, step 1)

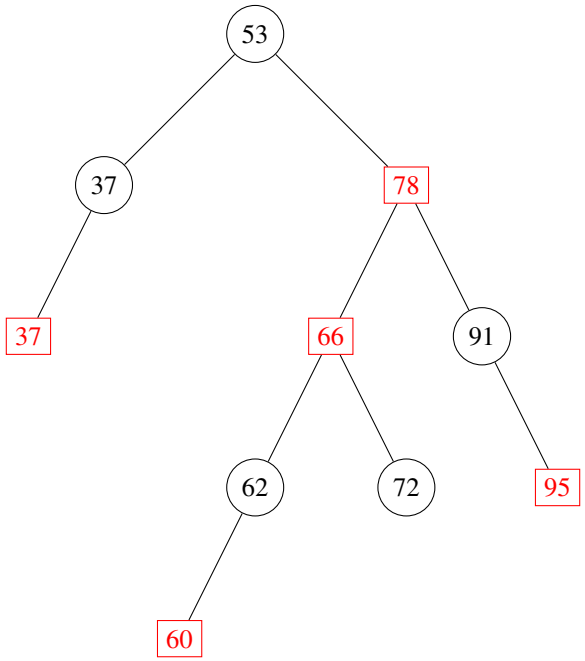

그림 4: Red-black tree (insert 60, step 1)

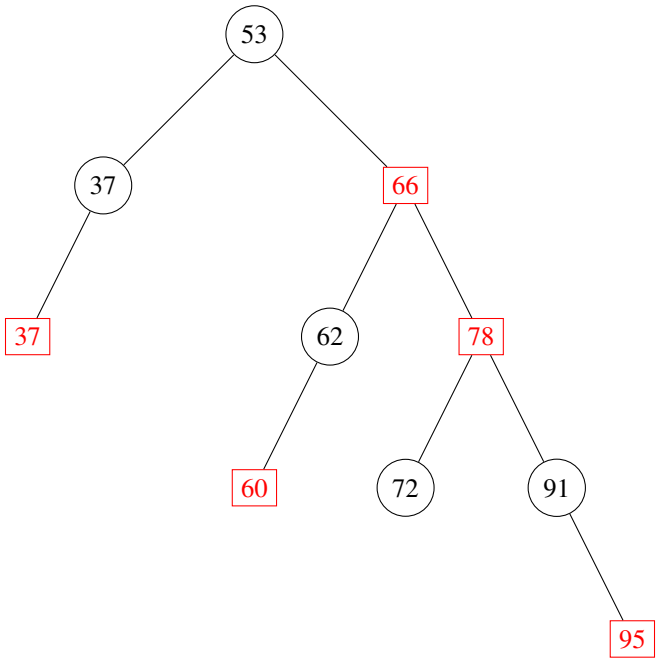

그림 5: Red-black tree (insert 60, step 3)

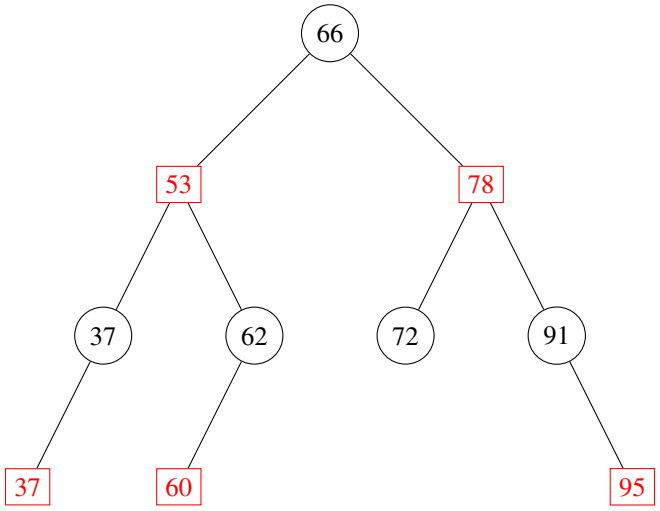

그림 6: Red-black tree (insert 60, step 3)

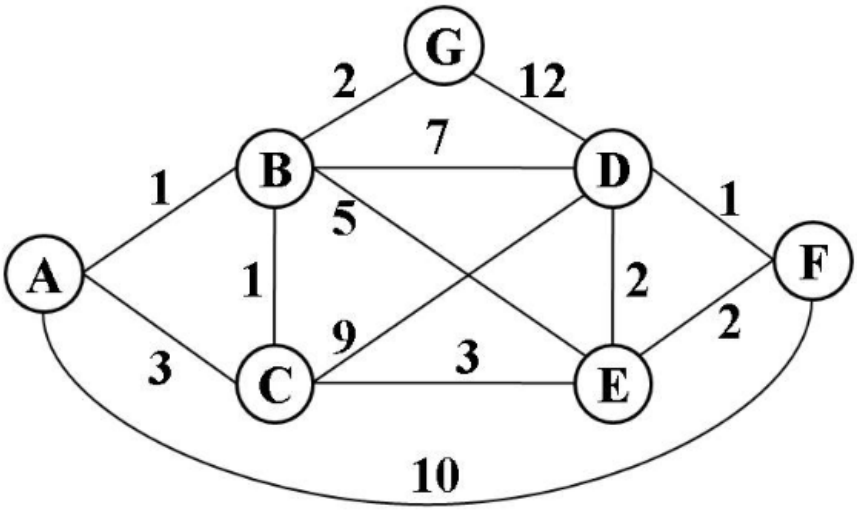

그림 7: Dijkstra problem

Vertex	Cost (d)	Pred (π)
A	0	
B	1	A
C	$3 \rightarrow 2$	$A \rightarrow B$
D	$8 \rightarrow 7$	$B \rightarrow E$
E	$6 \rightarrow 5$	$B \rightarrow C$
F	$10 \rightarrow 7$	$A \rightarrow E$
G	3	B

표 1: Dijkstra