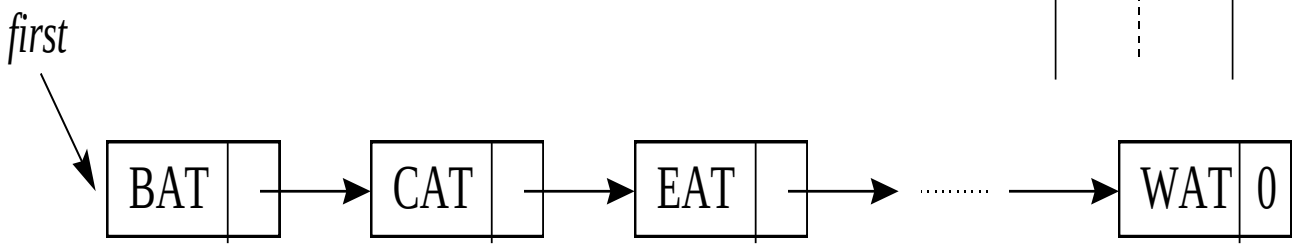
기본적인 자료구조

- 연결리스트
- 스택과 큐
- 트리 – 히프와 이진탐색트리

1. 연결리스트(체인)

- 순서 리스트를 연결 포인터를 사용하여 표현하는 것

	data	*link*
1	HAT	15
2		
3	CAT	4
4	EAT	9
5		
6		
7	WAT	0
8	BAT	3
9	FAT	1
10		
11	VAT	7



제네릭 클래스(Generic Class)

- 다양한 종류의 데이터를 처리할 수 있는 방법으로 C++의 템플리트 개념
- 자바 버전 1.5부터 추가됨
- 2가지 방법

  (1) 타입매개변수 <Type> 을 선언

  (2) 다형성을 이용: 모든 종류의 클래스를 받을 수 있는 Object 클래스로 선언

노드 클래스

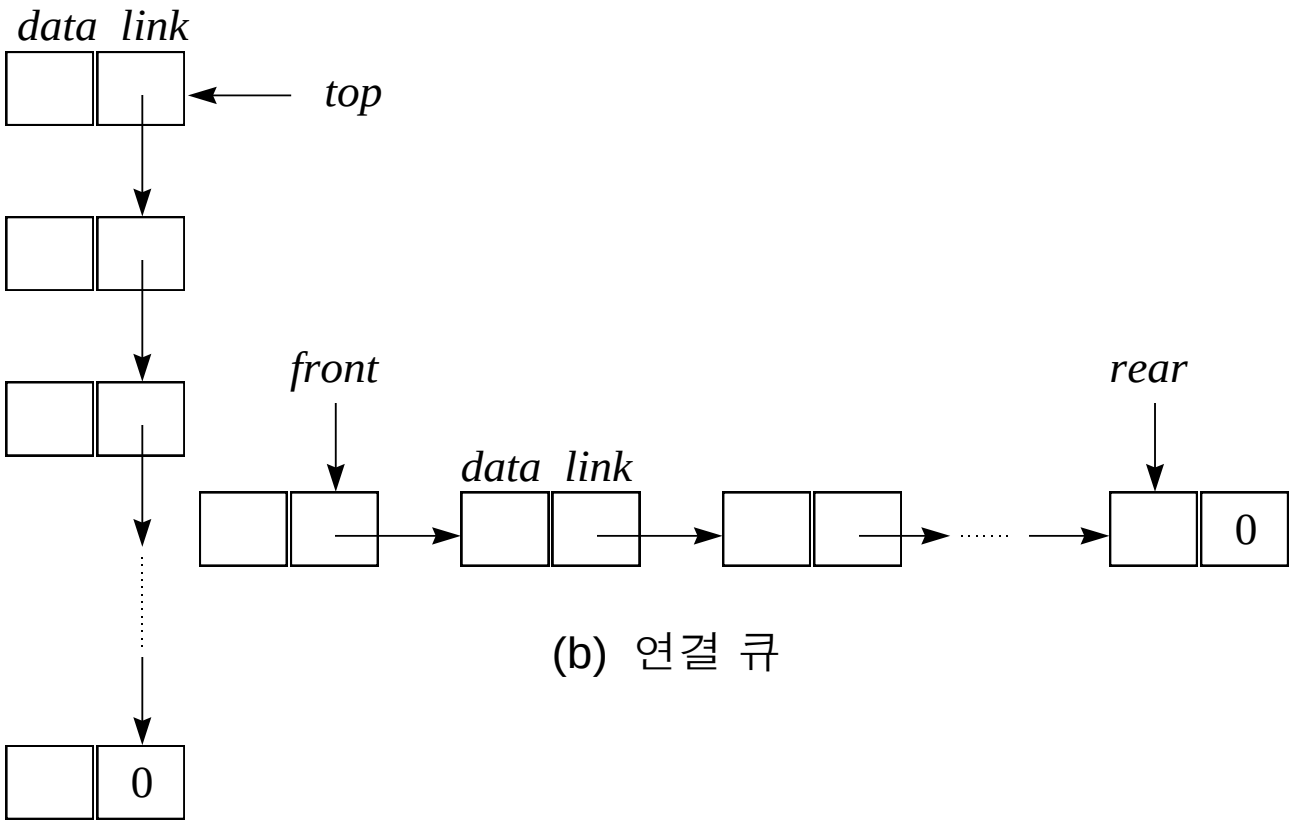
```
public class ChainNode <Type> {
   Type element;
   ChainNode <Type>  next;

   // package visible constructors
   ChainNode() {}

   ChainNode(Type element)
      {this.element = element;}

   ChainNode(Type element, ChainNode <Type>  next)
      {this.element = element;
       this.next = next;}
}
```

체인 클래스(lec3-1)

```
public class Chain <Type2>{
   protected  ChainNode<Type2>  firstNode;   // 첫 노드에 대한 포인터
   protected  int  size;                     // 노드의 수
   public Chain(int initialCapacity) ;    // 생성자
   public Chain() ;                          // 생성자
   public boolean isEmpty() ;
   public int size() ;
   void checkIndex(int  index) ;
   public Type2 get(int index) ;
   public int indexOf(Type2 theElement) ;
   public Type2 remove(int index) ;
   public void add(int index, Type2  theElement) ;
   public String toString() ;                // 리스트 노드의 element 값을 문자열로  변환
```

2. 연결 스택과 큐

- 스택과 큐를 연결 리스트로 구현하는 것.



(b) 연결 큐

(a) 연결 스택

연결 스택 클래스(lec3-2)

```
public class Stack <Type2>
{
   protected ChainNode<Type2> topNode ;  // 스택의 top
   public boolean empty();
   public Type2 peek();                        // top 원소를 반환
   public void push(Type2 theObject);  // 삽입
   public Type2 pop();                         // 삭제
}
```

연결 큐 클래스(lec3-3)

```
public class Queue <Type2>
{
   protected ChainNode <Type2> front ;   // 큐의 front
   protected ChainNode <Type2> rear ;    // 큐의 rear
   public boolean isEmpty();
   public Type2 getFrontElement();  // 맨 앞 원소
   public Type2 getRearElement();   // 맨 뒤 원소
   public void put(Type2 theObject);    // 삽입
   public Type2 remove();                // 삭제
}
```

3. 히프

(1) 우선 순위 큐

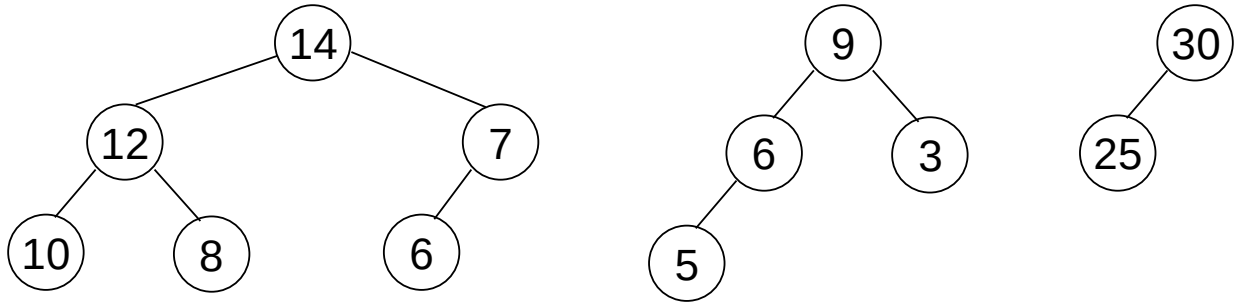
- 삽입은 일반적인 큐와 같으나 삭제는 우선 순위가 가장 높은 것이 먼저 삭제된다.
- 최대(최소) 히프는 최대(최소) 우선 순위 큐를 구현하는데 사용된다.

(2) 최대 히프의 정의

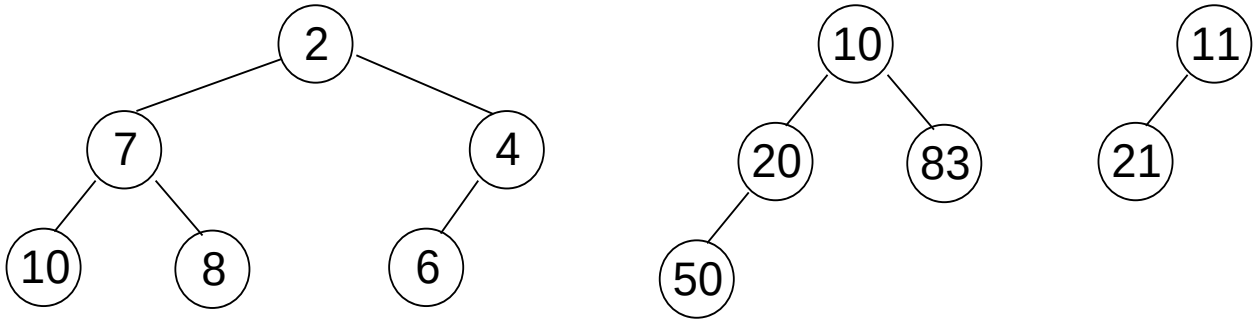
- 최대 트리: 각 노드의 키값이 그 자식의 키값보다 작지 않은 트리
- 최소 트리: 각 노드의 키값이 그 자식의 키값보다 크지 않은 트리
- 최대 히프: 최대 트리이며 완전 이진 트리
- 최소 히프: 최소 트리이며 완전 이진 트리

최대 히프와 최소 히프의 예

- 최대 히프



- 최소 히프



최대 히프 클래스(lec3-4)

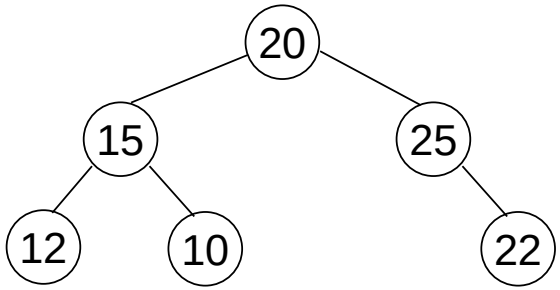
- 다형성 이용(Comparable 이용)

```
public class MaxHeap {
   Comparable [] heap;  // 히프를 위한 일차원 배열
   int size;            // 히프에 있는 원소 수를 저장
   public boolean isEmpty();
   public int size();   // 히프의 원소 수
   public Comparable getMax();  // 최대값을 반환
   public void put(Comparable theObject);  // 삽입
   public Comparable removeMax(); // 최대값 삭제
}
```

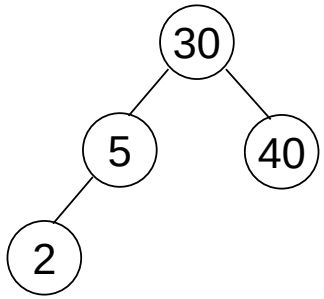
4. 이진탐색트리(binary search tree)

(1) 정의

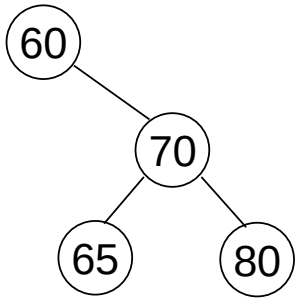
- 임의의 한 노드를 삽입, 삭제, 탐색하는 데 유리하다.
- 정의: **(i)** 모든 원소는 키를 가지며 동일한 키가 없다.
  (ii) 왼쪽 서브트리에 있는 키들은 루트의 킷값보다 작다.
  (iii) 오른쪽 서브트리에 있는 키들을 루트의 킷값보다 크다.
  (iv) 왼쪽, 오른쪽 서브트리도 이진탐색트리이다.



(a)

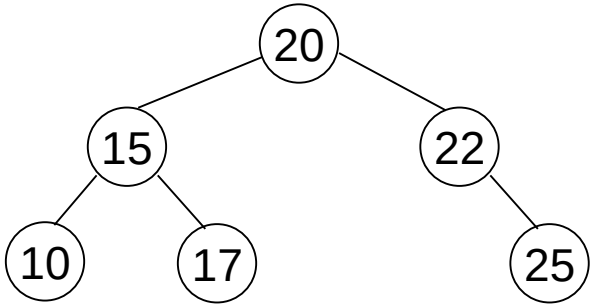


(b)



(c)

(2) 탐색



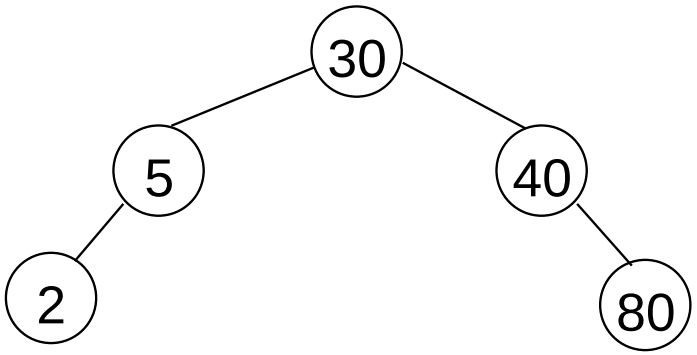
key = 22 의 탐색

- 방법: key = root : found
  key < root : search left subtree
  key > root : search right subtree

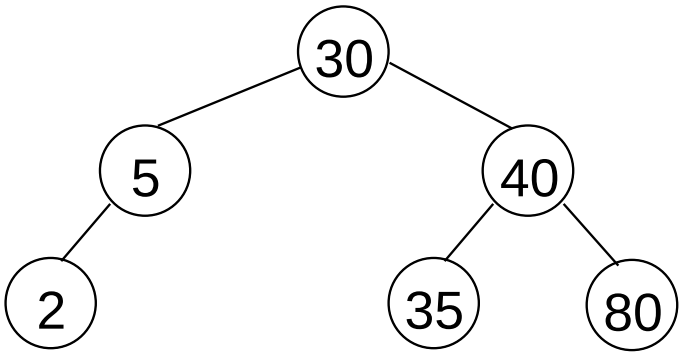
- key = 24 와 같이 존재하지 않는 키의 탐색

(3) 삽입

- 동일한 키값을 가진 원소가 있는가를 확인한 후, 없으면 그 지점에 삽입한다.



(a) 80을 삽입



(b) 35를 삽입

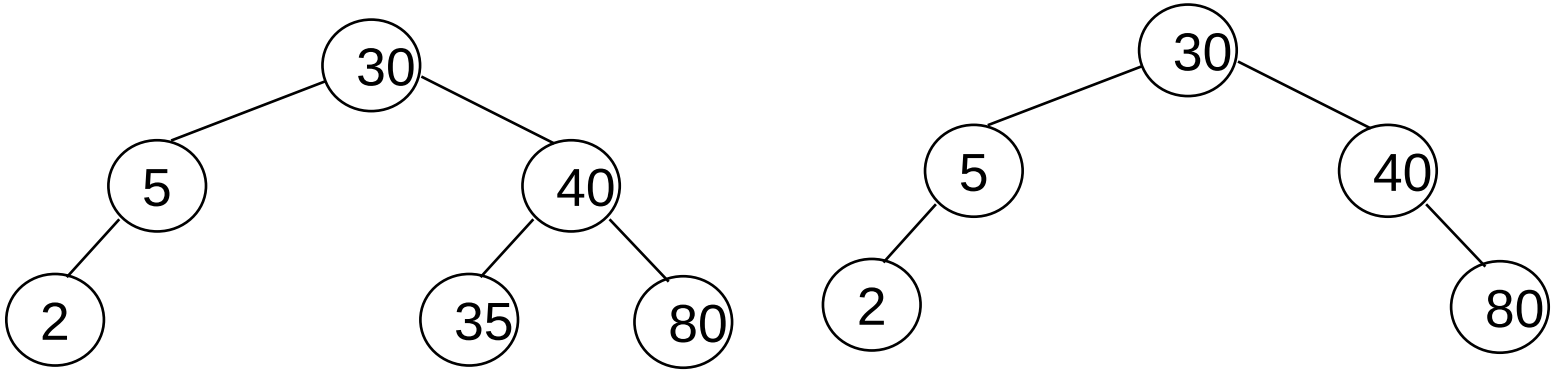
- 시간복잡도 O(h)

(4) 삭제

- 3가지 경우:

(i) 리프노드의 삭제: 부모의 자식 필드에 null 을 삽입

예: 35의 삭제

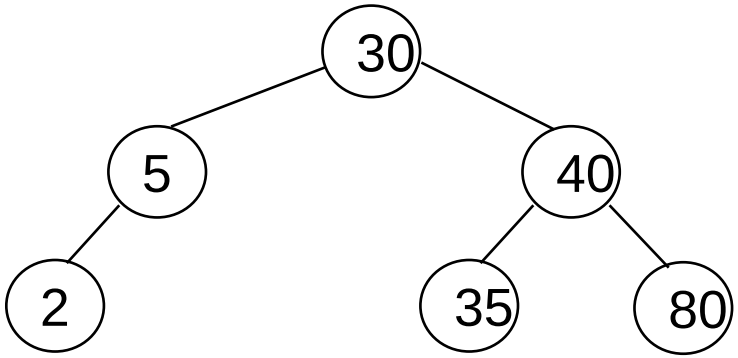


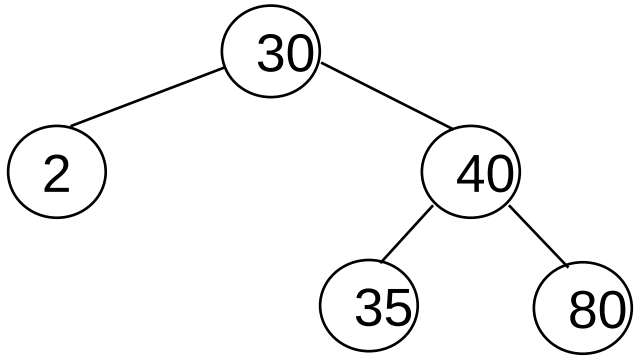
삭제(하나의 자식을 갖는 노드)

(ii) 하나의 자식을 갖는 비단말노드의 삭제:

삭제된 노드의 자식을 삭제된 노드의 자리에 대체

예: 5의 삭제





삭제(두 개의 자식을 갖는 노드)

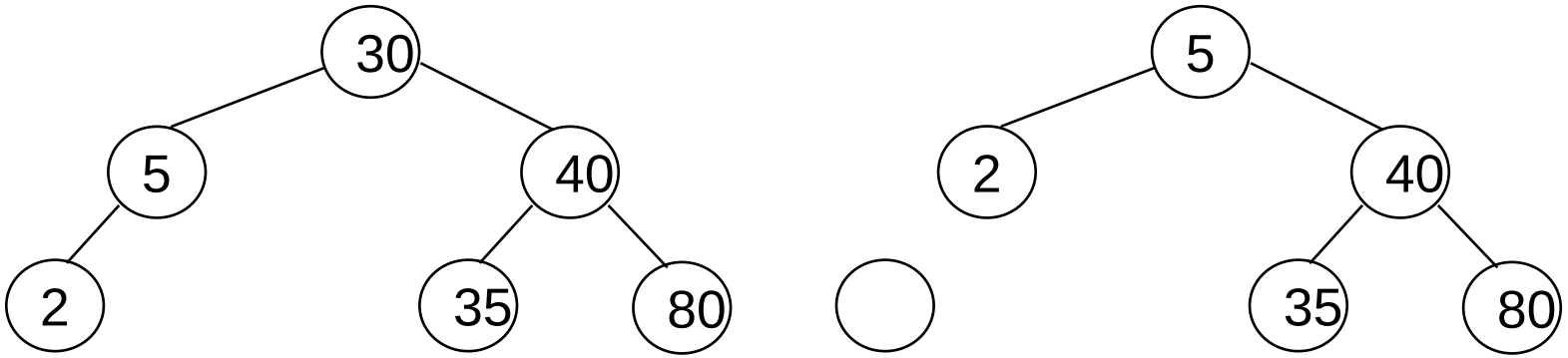
(iii) 두 개의 자식을 갖는 비단말노드의 삭제:

왼쪽 서브트리에서 가장 큰 노드로 대체하거나

오른쪽 서브트리에서 가장 작은 노드로 대체한다.

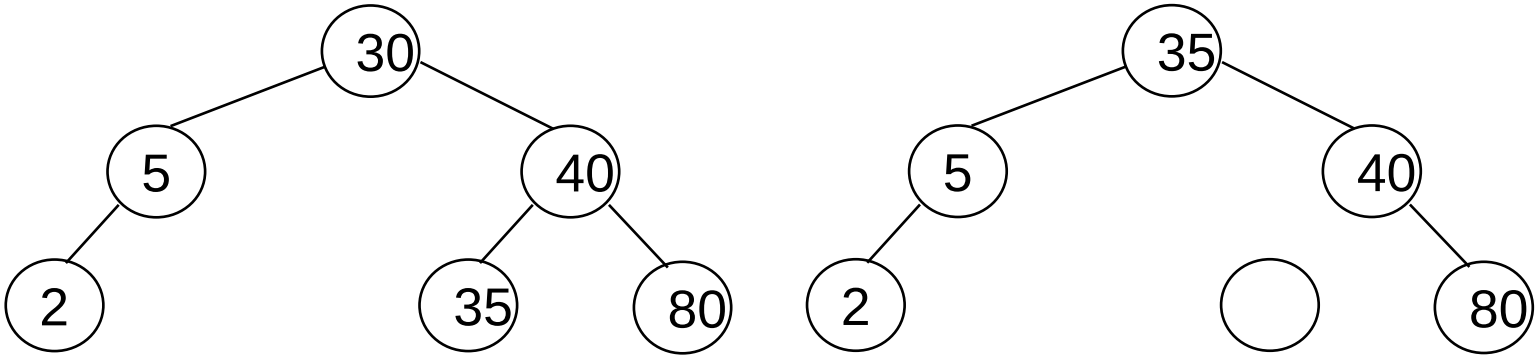
이때 대체되는 노드의 삭제는 위의 (i) 또는 (ii)의 경우에 해당한다.

예: 30의 삭제



왼쪽 서브트리에서 가장 큰 노드로 대체

삭제(두 개의 자식을 갖는 노드)



오른쪽 서브트리에서 가장 작은 노드로 대체

이진탐색트리 클래스(lec3-5)

```
public class BinaryTreeNode {    // 하나의 노드 정의
    Object element;
    BinaryTreeNode leftChild;    // leftsubtree
    BinaryTreeNode rightChild;   // right subtree
    public BinaryTreeNode() {}
    public BinaryTreeNode(ObjecttheElement)
            {element = theElement;}
    public BinaryTreeNode(Object theElement,
                          BinaryTreeNode theleftChild,
                          BinaryTreeNode therightChild) {
        element = theElement;
        leftChild = theleftChild;
        rightChild = therightChild;
    }
```

클래스 정의

```
public class BinarySearchTree {
    protected BinaryTreeNode root ; // 이진탐색트리의 루트

    public BinaryTreeNode get(Object theKey)   // 탐색
    public Object put(Object theKey)   // 삽입
    public Object remove(Object theKey) // 삭제
    public void ascend()  // 중위순회
}
```