표현방법

배열, 포인터

# 8. 트리

계층적인 자료의 표현에 적합한 자료구조

탐색트리, 우선순위큐를 위한 힙트리, 인공지능 문제에서 다루는 결정 트리

Node와 edge를 이용해서, 사이클을 허용하지 않도록 구성한 데이터구조


## 용어

단말노드 ⇒ 자식노드가 없는 노드, 자식이 있으면 비단말노드

노드의 차수 ⇒ 현재 노드가 가지고 있는 자식의 수

트리의 차수 ⇒ 트리가 가지고 있는 노드의 차수 중에서 가장 큰 차수

레벨 ⇒ 트리의 각 층에 번호를 매기는 것 (루트의 레벨은 1이고 한 층씩 내려갈수록 1씩 증가)

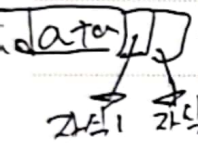
트리의 높이 ⇒ 트리가 가지고 있는 최대 레벨

포레스트 ⇒ 트리들의 집합

## 일반트리

일반트리 표현 방법

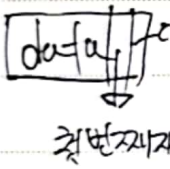
- 연결된 그래프로서의 노드로 표현




- 노드 정보를 저장하는 데이터필드와 자식노드를 가리키는 여러개의 링크를 갖는 노드

- 일반트리: 임의의 개수의 자식을 가질 수 있는 트리

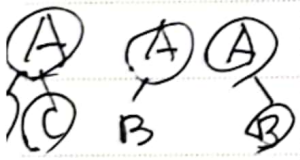
- 두 가지 링크를 갖도록 하는 방법

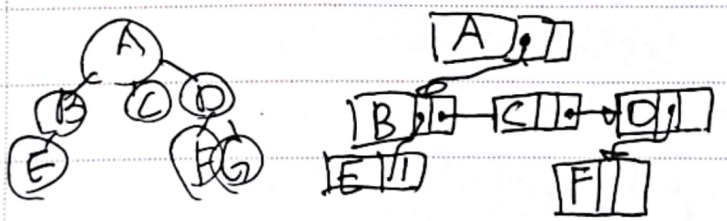



- 하나는 첫번째 자식을 가리키고, 다른하나는 다음형제를 가리키게함

- 임의의 일반트리 표현 가능하지만, 표현이 복잡하고 성능이 안좋음

기울구조




## 이진트리

- 공집합이거나, 루트와 왼쪽서브트리, 오른쪽 서브트리로 구성된 노드들의 집합

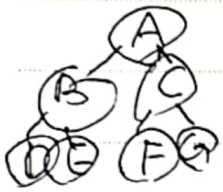
이진트리의 서브트리들은 모두 이진트리여야 함

모든 노드의 차수가 2이하, 최대 2개까지 갖을 수 있음

자식들 사이에 순서 존재, 왼쪽노드는 해당노드보다 작은값, 오른쪽노드는 해당노드보다 큰값

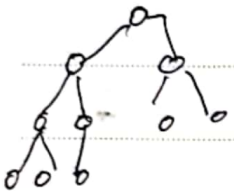
# 이진 트리의 종류

## 포화 이진 트리




- 포화 이진 트리: 트리의 각 레벨에 노드가 꽉 차 있는 이진 트리
  - 포화 이진 트리에서는 각 노드에 순서대로 번호를 붙일 수 있음
  - 높이 k인 포화 이진 트리의 전체 노드 개수: $2^{1-1}+2^{2-1}+2^{3-1}+\cdots 2^{k-1}$

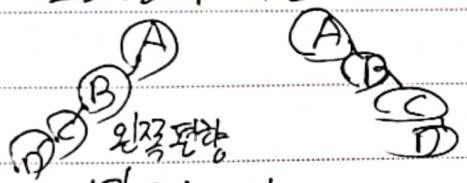
## 완전 이진 트리



- 완전 이진 트리: 높이가 k인 트리에서 레벨 1부터 k-1까지는 노드가 모두 채워져 있고 마지막 레벨에서는 노드가 왼쪽부터 오른쪽으로 차례대로 있는 이진 트리
  - 마지막 레벨은 꽉 차 있지 않아도 되지만 중간에 빈곳이 있으면 안 됨

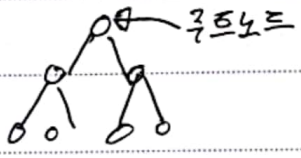
예) 힙

- 편향 이진 트리
  - 높이가 k인 트리에서 k+1개의 노드를 가지면서 모든 노드가 왼쪽이나 오른쪽 중 한 방향으로만 서브트리를 가지고 있는 트리

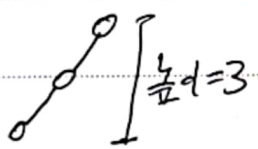


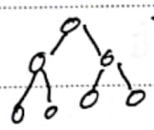

## 이진 트리의 성질

- 노드 개수가 n개이면 간선의 개수는 n-1



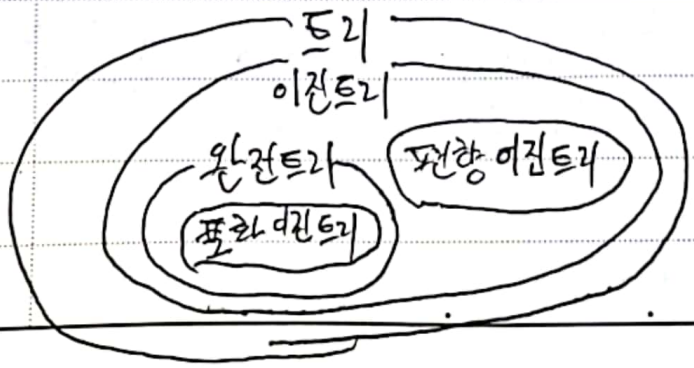

- 높이가 h이면 노드의 개수는 h이상 $2^h-1$개 이하의 노드를 가짐

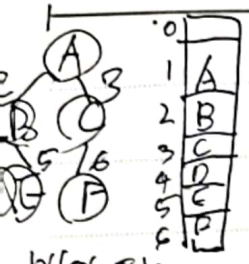



- n개의 노드의 이진 트리 높이는 $\log_2(n+1)$ 이상이고 n이하

## 종류






배열 표현

표현방법

〈배열 표현법〉

• 배열로 표현하기 위하여 인덱스를 사용

1) 트리의 높이를 구해 배열을 할당

2) 포화 이진 트리의 번호를 인덱스로 사용하여 배열에 노드들을 저장

• 트리가 불균형 하다면 메모리낭비가 심함


노드 i의 인덱스를 안다면 다른 노드의 인덱스를 구할 수 있음

배열 표현법은 간단

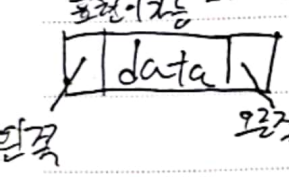
기억공간의 낭비, 표현할 수 있는 높이가 배열의 크기에 따라 제한됨

부모노드 인덱스 = i/2

왼쪽자신 = 2i

오른쪽자신 = 2i+1

링크가 두개만 있으면 표현이 가능




〈링크 표현법〉

• 연결된 구조로 이진 트리 표현

• 왼쪽 링크와 오른쪽링크는 각각 왼쪽자식, 오른쪽자식을 가리킴

연산

트리의모든 노드를 방문하는 방법: 순회

• 트리에 속해 있는 모든노드를 한번씩 방문하는것

• 선형 자료구조는 순회가 단순

• 트리는 다양한 방법이있음

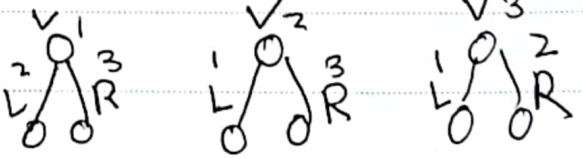
기본순회 3가지

→ 왼쪽서브트리, 오른쪽서브트리를 어떤순서로 방문하느냐로 나눔

1) 전위 VLR

2) 중위 LVR

3) 후위 LRV



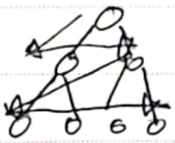

각 서브트리도 이진트리이므로 동일한 순회방법이 적용

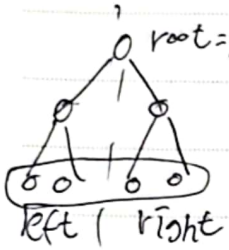
## 레벨 순회

• 노드를 레벨 순으로 검사하는 순회방법

1) 큐를 사용해구현

2) 순환을 사용하지 않음






〈트리의 전체 노드개수 구하기〉

• 모든 노드들을 한번씩 방문하여야함 → 순환사용

○ 왼쪽서브트리의 노드수 + 오른쪽서브트리 노드수 + 루트노드의수(1)

○ 후위 순회방식의 순환호출

〈단말 노드 개수 구하기〉

○ 왼쪽 자식과 오른쪽 자식이 모두 None인노드

○ 순환을 사용

○ 두 서브트리의 단말 노드개수를 계산하고, 이를 반환

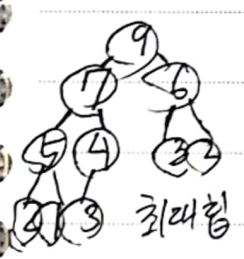
〈트리의 높이 구하기〉

○ 순환을 사용

○ 후위순회 코드를 사용

• 루트 노드를 포함한 현재의 트리의 높이는 좌우 서브트리의 높이 중에서 더 높은값에 1을 더한 결정트리

→ 여러 단계의 복잡한 조건을 갖는 문제에 대해 조건과 해결방법을 트리형태로 나타낸것




〈힙트리〉

• "더미"와 모습이 비슷한 완전이진트리 기반의 자료 구조

• 가장 큰 또는 작은 값을 빠르게 찾아내기를 만들어진 자료 구조

○ 최대힙, 최소힙


최대힙 : 부모노드의 키값이 자식노드의 키값보다 크거나 같은 완전이진트리 부모 >= 자식

최소힙 : 키값이 자식노드의 키값보다 작거나 같은 완전이진트리 부모 <= 자식

<힙의 삽입 연산>

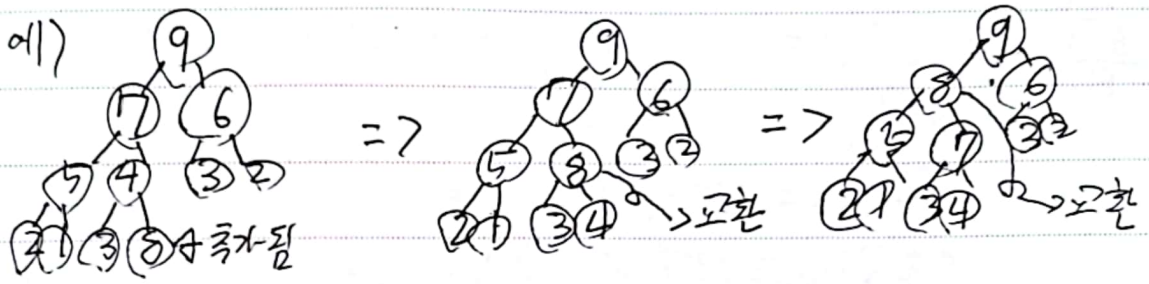
시간복잡도 O(logn)

• 삽입 연산은 힙의 속성 특성(최대, 최소힙)과 트리의 형태적 특성(완전이진트리) 반드시 유지

1) 새로운 항목이 들어오면 힙의 마지막 노드의 다음 위치에 삽입
(완전 이진 트리 조건 만족하지만 힙의 속성은 만족하지 못함)

2) 마지막에 삽입된 노드를, 부모 노드들과 교환해서 힙의 속성 특징을 만족시켜줄 교환을 처리 최대 트리의 높이만큼의 이동이 필요

예)



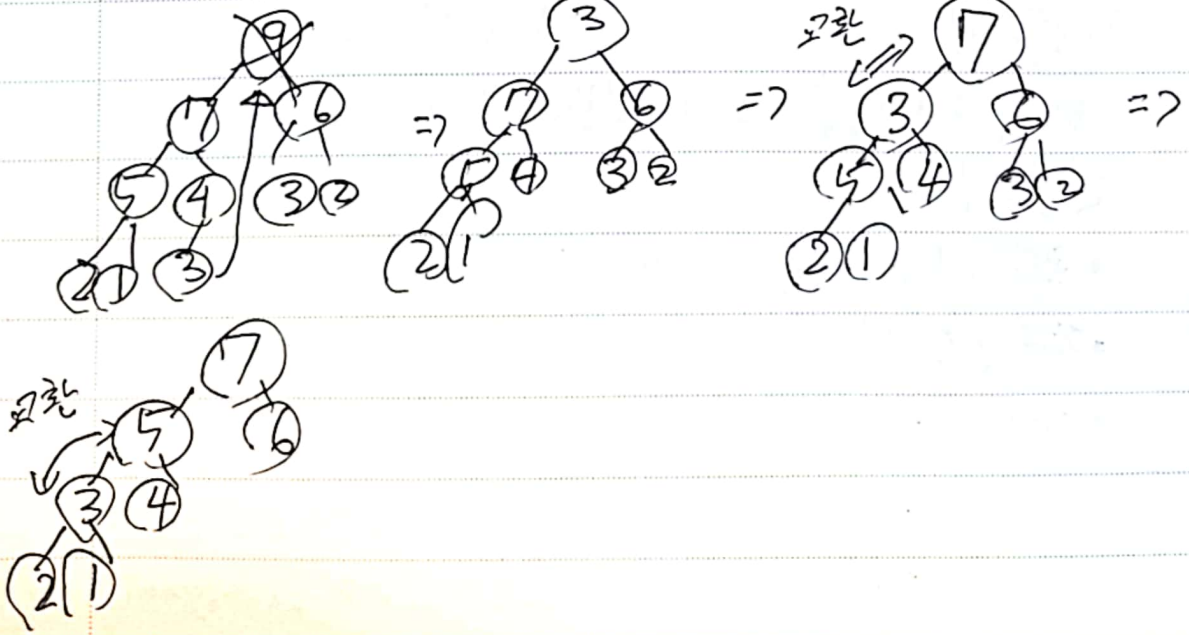

<힙의 삭제 연산>

시간복잡도 O(logn) Down-heap

힙에서 삭제 연산은 루트노드를 꺼내며 힙의 성질 유지하도록 하는것

1) 루트노드에 마지막 노드를 올림, 완전이진 트리 조건은 만족하지만 힙의 속성은 만족X

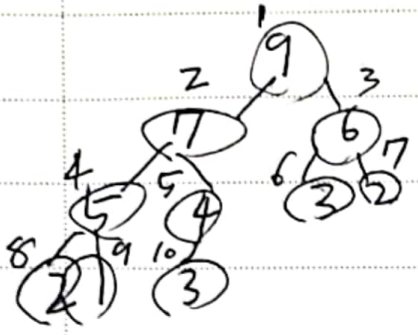
2) 루트노드를 자식과 비교하여 자식이 더 크면 교환, 이때 자식들 중에서 더 큰 자식이 교환되게 됨. 이 과정에서 자식이 없거나 자식이 더 작을 때까지 반복

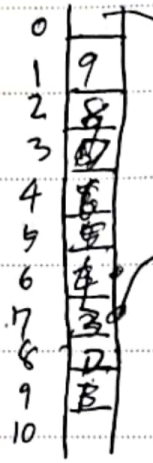


배열을 이용한 힙의 표현

힙을 보통 배열을 이용하여 구현

- 완전이진트리 → 각 노드에 번호를 붙임 → 배열의 인덱스






부모의 인덱스 = 자식의 인덱스 / 2

왼쪽 자식의 인덱스 = 부모의 인덱스 * 2

오른쪽 자식의 인덱스 = 부모의 인덱스 * 2 +

완전이진트리이므로 중간에 빈 칸이 없음

우선순위 큐 구현 : 힙

- 힙을 이용한 우선순위 큐의 구현은 가장 좋은 방법

〈삽입 연산〉

- 맨 마지막 항목으로 추가하기 때문에 시간복잡도는 O(1)
- 배열을 정렬시키려면 우선순위에 따라 삽입 위치 찾아야 하기 때문에 시간복잡도 O(
- 힙을 이용하면 삽입연산의 시간복잡도 → O($\log_2 n$)

〈삭제 연산〉

- 배열이 정렬되어 있지 않으면 삭제가 반복되므로 → 우선순위 높은 항목 찾아야 하기에 시간복잡도 O(n
- 정렬된 배열이라면 단순히 맨 앞, 우선순위가 가장 높은 항목은 맨 끝에 있기 때문에 시간복잡도 O(1
- 힙을 이용하면 시간복잡도 O($\log_2 n$)

〈탐색 연산〉

- 정렬되지 않은 배열을 이용하면 우선순위가 높은 항목을 찾아야 하므로 시간복잡도 O(n)
- 정렬된 배열이라면 맨 마지막 항목 반환하면 되므로 시간복잡도 O(1)
- 힙은 루트를 반환하면 되므로 시간복잡도 O(1)

허프만 코드

- 허프만 알고리즘에 의해 생성된 최적 이진코드
- 이진 트리는 각 글자의 빈도가 알려져 있는 메시지의 내용을 압축하는데 사용될 수 있음
- 이런 종류의 이진트리 → 허프만 코딩 트리

허프만 코드

- 아스키(ASCII) 코드는 모든 문자를 동일한 비트수로 표현 + 압축의 관점에서는 효율적이지 못함
- 자주 사용하는 문제에는 적은 비트수를, 그렇지 않은 문자에는 많은 비트수를 부여 → 효율적임

1) 모든 문자를 7비트로 표현 868bit 고정길이 코드

2) E는 2비트, Z는 20비트로 표현 266비트 가변길이 코드

고정길이 코드와 가변길이 코드 비교

- 텍스트 A~J로만 이루어진 빈도수
- 10가지 문자 표현하기 위해 고정길이 코드 사용하면 최소 4비트필요 ($2^4$=16가지로 표현가능)
- 가변길이 코드 사용하면 글자마다 코드의 비트수가 달라짐
- 모든 가변길이 코드가 다른 코드의 첫 부분이 아니어야 함

생성방법

1. 각 문자별로 노드 생성, 노드의 값은 빈도수. 각 노드는 모두 독립적인 트리의 루트
2. 가장 작은 빈도수의 두 노드를 골라 묶어 이진 트리 구성. 루트의 값은 자식 노드 값의 합
3. 남은 트리에서 가장 작은 빈도수의 루트 2개 골라 묶어 이진 트리 구성
4. 남은 트리에 대해서도 동일한 처리
5. 마지막으로 남은 트리를 묶고, 최종 허프만 트리 1개가 됨
6. 마지막으로 코드 할당. 트리에서 왼쪽 간선은 비트 1, 오른쪽 0