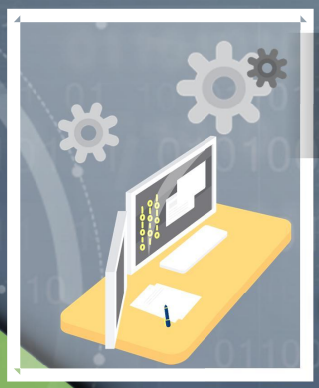
알고리즘과 문제해결

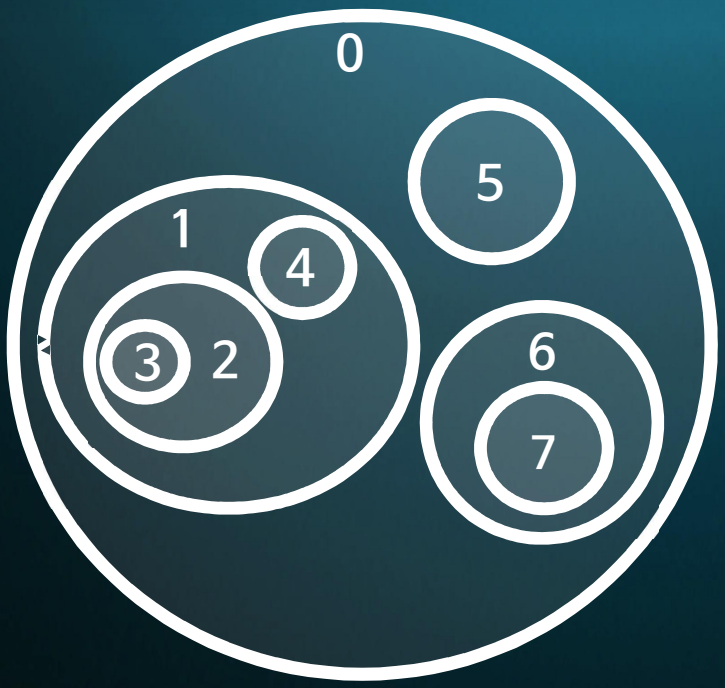


1차시
트리 개념

학습목표

❶ 트리의 개념을 이해 할 수 있다.

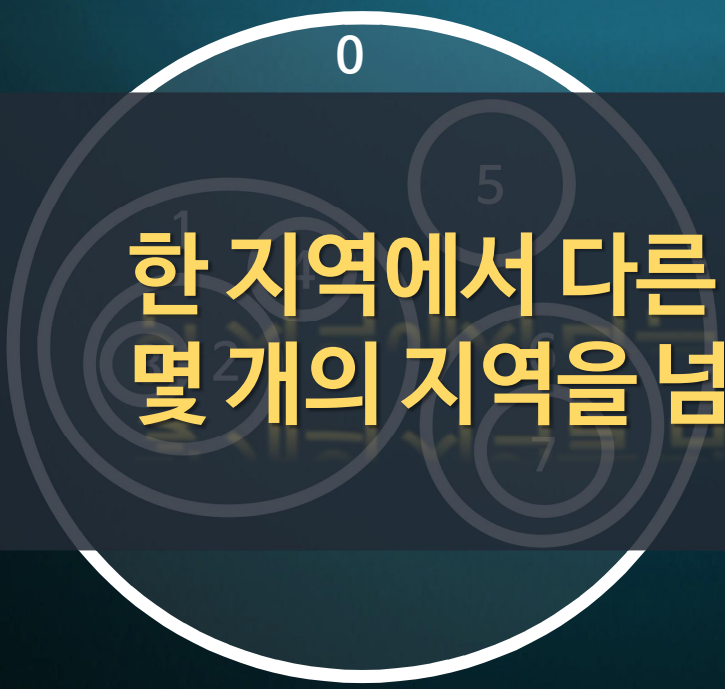
1 트리 사례



성으로 지역이 구분

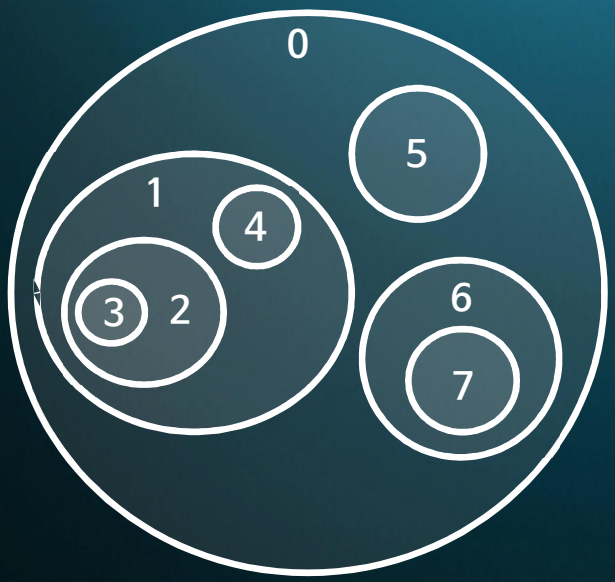
성안에 성이 있음

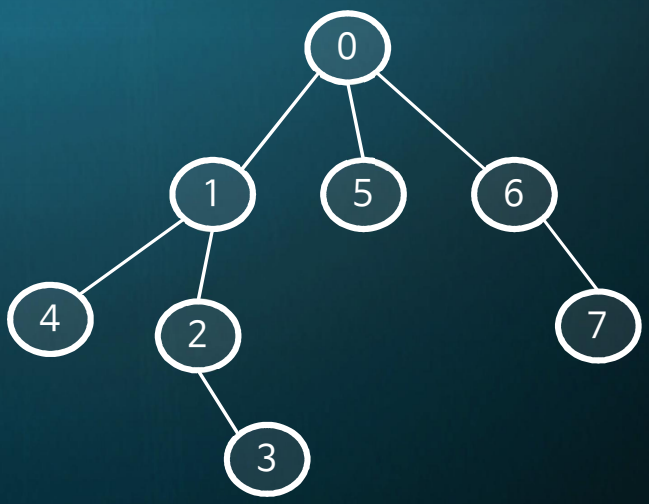
1 트리 사례



한 지역에서 다른 지역으로 갈 때 몇 개의 지역을 넘어야 하는가?

1 트리 사례





계층적 특징을 이용하면
트리로 표현 가능

2 트리 개요

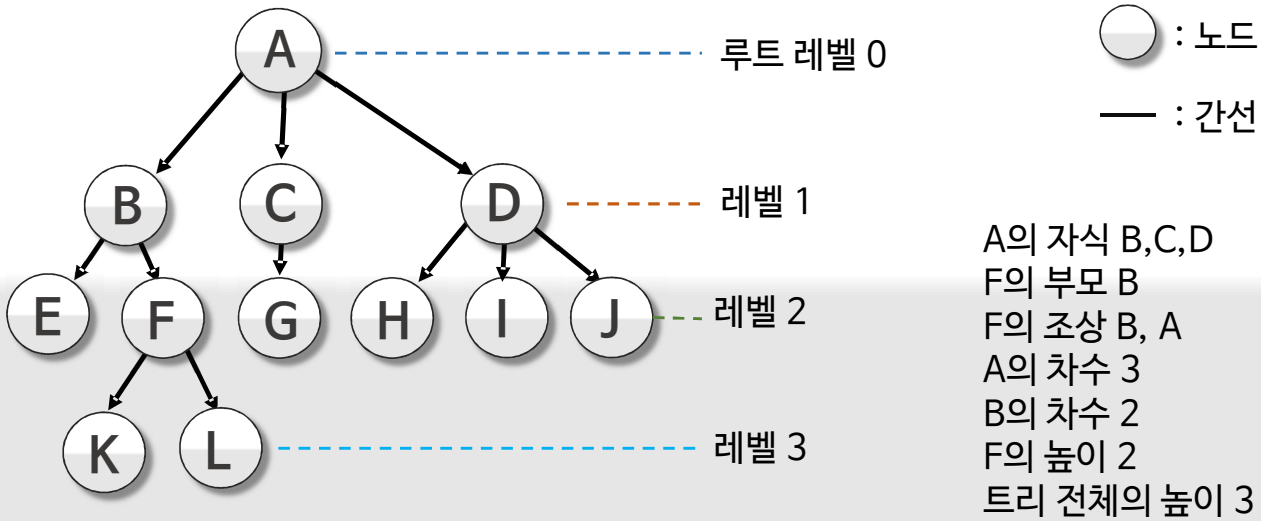
트리의 필요성

- 선형으로 표현하기 어려운 자료
- 자료 간의 상하위 관계나 포함 관계가 존재

트리 Tree

- 하나의 개념이 상위 개념에서 하위로 가지를 쳐나가는 모습
- 계층적 구조를 갖는 자료를 표현

2 트리 개요



2 트리 개요

트리의 특성

- 현실 세계의 개념을 추상화하여 표현
- 탐색형 자료 구조로도 유용 ➔ 빠른 자료 탐색 가능

3 트리 활용

트리의 문제점

- 트리의 좌우 균형이 맞지 않으면 비효율적
- 새로운 노드를 추가, 삭제 할 때 트리 스스로 균형 조정
- B-Tree, AVL Tree, Red-Black Tree 등

B-Tree

- 많은 수의 자식을 가질 수 있도록 일반화
- 하나의 레벨에 많은 자료 저장 ➔ 트리 높이가 낮아짐
- 트리의 균형을 자동으로 맞추는 로직

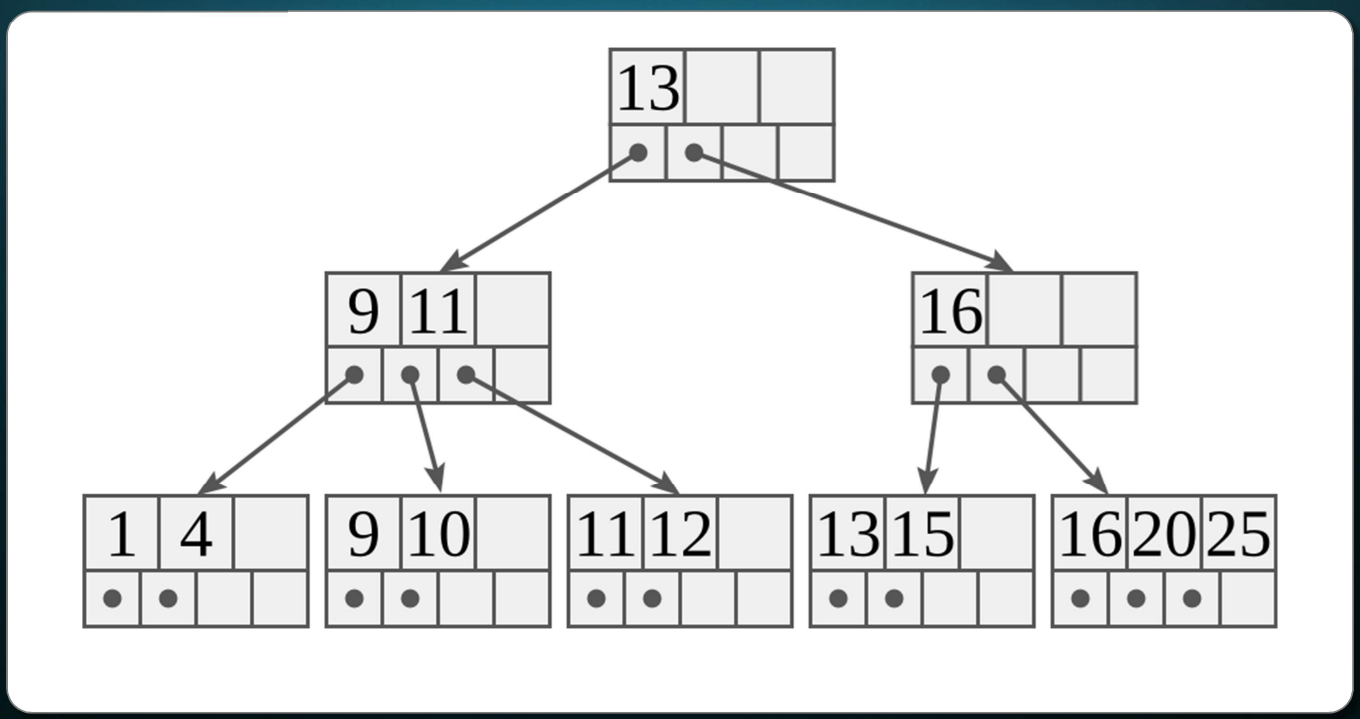
B-Tree 규칙

- 노드의 자료수가 N, 자식의 수는 N+1
- 노드의 자료는 정렬된 상태
- Root 노드는 적어도 2개 이상의 자식을 가져야 함

B-Tree 특성

- 하나의 노드에 많은 데이터 ➔ 대량데이터 처리에 적합
- 대량 데이터 처리 ➔ 메모리보다는 HDD, SSD 활용
- DB 인덱스 저장 방법으로 활용 (주로 B+ Tree)

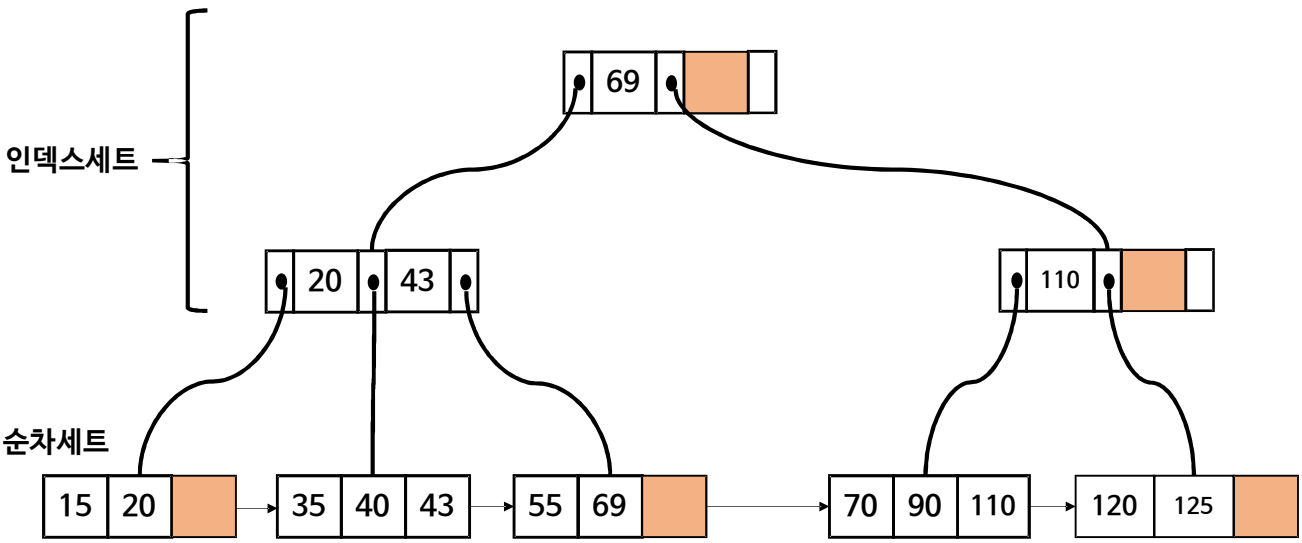
3 트리 활용



B+Tree

- B-Tree는 순차 접근이 어려움 (순회작업 어려움)
- 데이터는 leaf에만 저장, 이외 노드는 인덱스
- Leaf노드는 연결리스트로 서로 연결

3 트리 활용

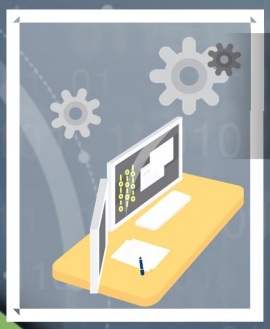


핵심 키워드 이 시간을 통해 꼭 기억해야하는 '핵심 키워드'입니다.

1 트리

참고문헌

- https://youtu.be/NI9wYuVIYcA
- http://ddmix.blogspot.com/2015/01/cppalgo-18-b-tree-search.html
- https://pixabay.com/
- http://12bme.tistory.com/138



2차시

트리 순회

학습목표

❶ 트리 순회의 개념을 이해하고,
소스코드로 구현 할 수 있다.

트리 순회 Tree traversal

- 트리 구조에서 각각의 노드를 정확히 한 번만 체계적 방법으로 방문

- 전위, 중위, 후위 순회

➡ **이진 트리 포함 모든 트리에 일반화**

1 트리 순회 개념

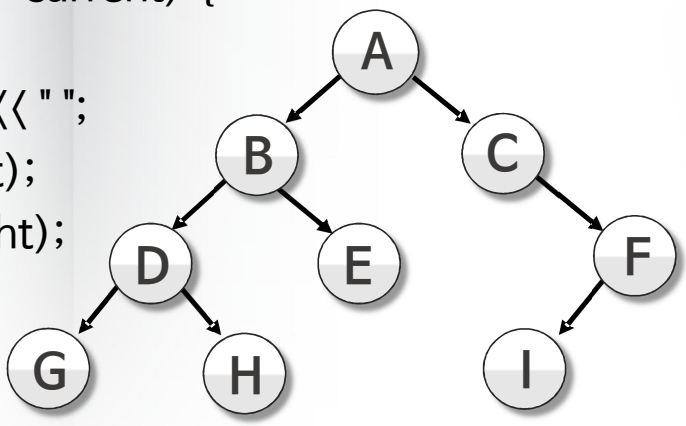
전위 순회 Preorder traversal

❶ Root를 방문한다.
❷ 왼쪽 서브 트리를 방문한다.
❸ 오른쪽 서브 트리를 방문한다.

- Root를 가장 먼저 방문, 깊이 우선 순회

전위 순회 소스코드

```
void preorder(TreeNode<T>* current) {
    if (current != null) {
        cout << current->data << " ";
        preorder(current->left);
        preorder(current->right);
    }
}
```



A→B→D→G→H→E→C→F→I

1 트리 순회 개념

중위 순회 Inorder traversal

❶ 왼쪽 서브 트리를 방문한다.

❷ Root를 방문한다.

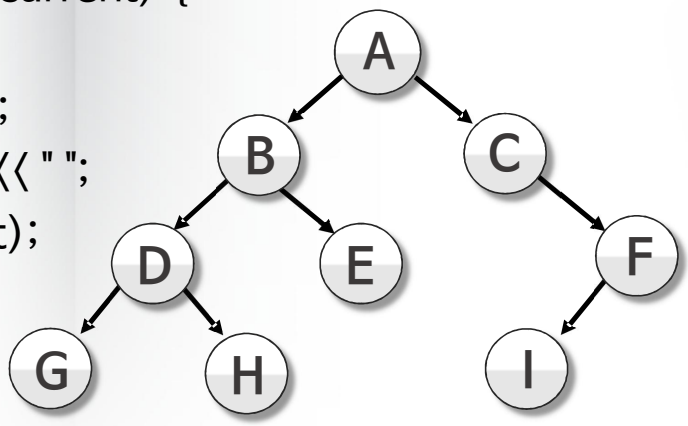
❸ 오른쪽 서브 트리를 방문한다.

- 왼쪽 서브 트리를 가장 먼저 방문, 대칭 순회
  Root는 중간에 방문

1 트리 순회 개념

중위 순회 소스코드

```
void inorder(TreeNode<T>* current) {
    if (current != null) {
        inorder(current->left);
        cout << current->data << " ";
        inorder(current->right);
    }
}
```



G→D→H→B→E→A→C→I→F

1 트리 순회 개념

후위 순회 Postorder traversal

❶ 왼쪽 서브 트리를 방문한다.

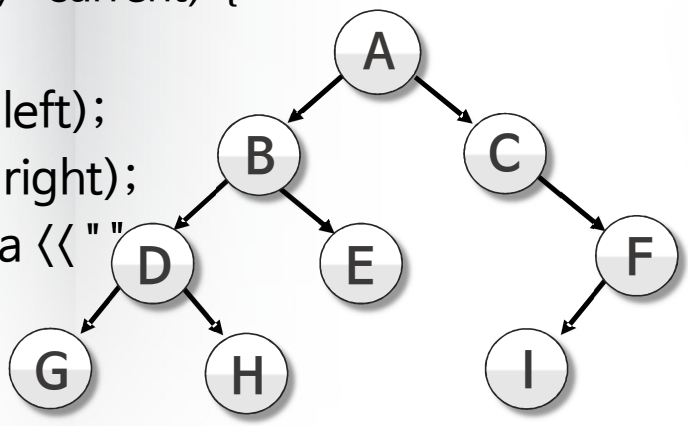
❷ 오른쪽 서브 트리를 방문한다.

❸ Root를 방문한다.

- Root를 가장 나중에 방문

후위 순회 소스코드

```
void postorder(TreeNode<T>* current) {
    if (current != null) {
        postorder(current->left);
        postorder(current->right);
        cout << current->data << " ";
    }
}
```



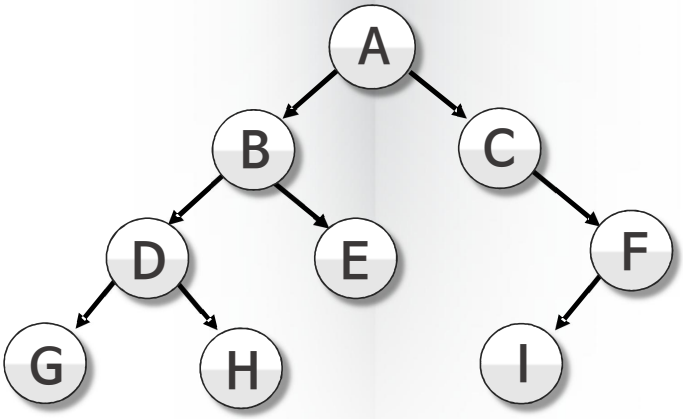
G→H→D→E→B→I→F→C→A

1 트리 순회 개념

레벨 순회 Level order traversal

- 모든 노드를 낮은 레벨부터 순회
- 너비 우선 순회

레벨 순회



A→B→C→D→E→F→G→H→I

1 트리 순회 개념

깊이 구하기

- 자식 노드를 재귀적으로 방문 (+1씩 증가)
- 왼쪽, 오른쪽 자식 노드 중에 큰 값을 반환

1 트리 순회 개념

깊이 구하기 소스코드

```
void maxDepth(TreeNode<T>* current) {
  if (current = null) return;
  int l_depth = maxDepth(current->left);
  int r_depth = maxDepth(current->right);
  return max(l_depth, r_depth) +1;
}
```

2 트리 순회 구현

트리 순회 소스코드

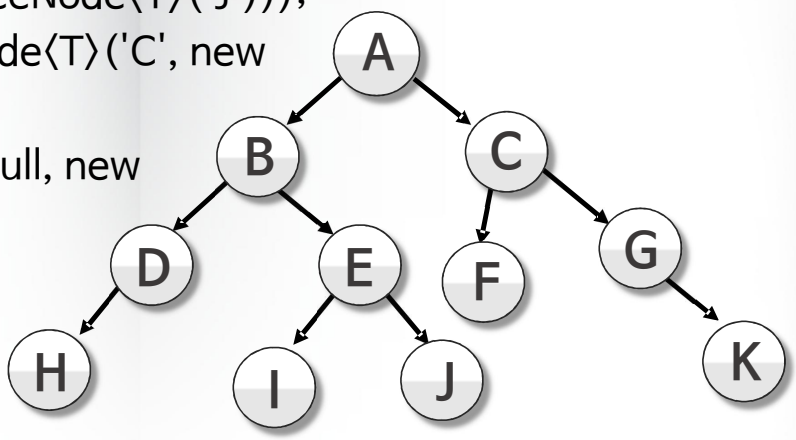
```
template <typename T>class TreeNode {   friend class Tree<T>;
private:
  T data;
  TreeNode* left;
  TreeNode* right;
public:
  TreeNode(T data = 0, TreeNode* left = null, TreeNode* right = null) {
    this->data = data;
    this->left = left;
    this->right = right;   }};
```

2 트리 순회 구현

```
template <typename T>class Tree {
private:
  TreeNode<T>* root;
public:
  Tree(T data = 0) {
    root = new TreeNode<T>(data);   }
   void buildTree() {
```

2 트리 순회 구현

```
root->left = new TreeNode<T>('B', new
TreeNode<T>('D', new TreeNode<T>('H')),
  new TreeNode<T>('E', new
TreeNode<T>('I'), new TreeNode<T>('J')));
root->right = new TreeNode<T>('C', new
TreeNode<T>('F'),
  new TreeNode<T>('G', null, new
TreeNode<T>('K')));   }
```



2 트리 순회 구현

트리 순회 소스코드

```
TreeNode<T>* getRoot() {  return root;  }
void visit(TreeNode<T>* current) {
  cout << current->data << " ";  }
void preorder(TreeNode<T>* current) {
  if (current != null) {
    visit(current);
    preorder(current->left);
    preorder(current->right);     }  }
void inorder(TreeNode<T>* current) {   중략   }
void postorder(TreeNode<T>* current) {   중략   }
```

2 트리 순회 구현

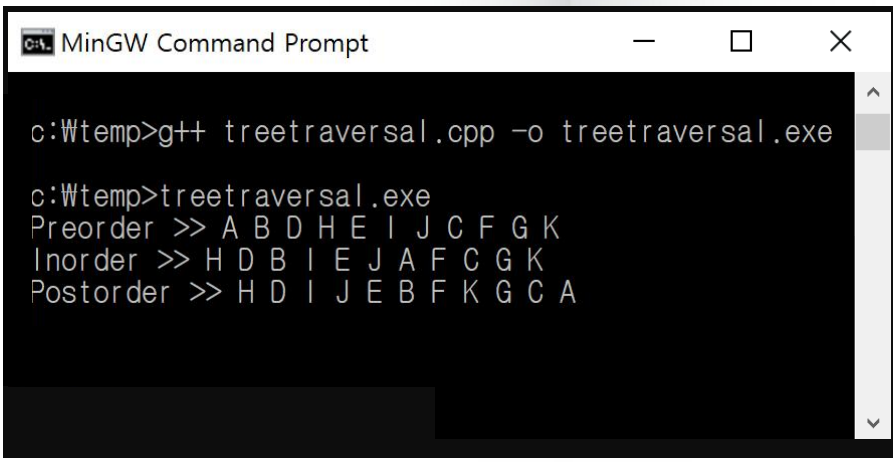
트리 순회 소스코드

```
int main() {
  Tree<char> tree('A');
  tree.buildTree();
  cout << "Preorder >> ";
  tree.preorder(tree.getRoot());  cout << endl;
  cout << "Inorder >> ";
  tree.inorder(tree.getRoot());   cout << endl;
  cout << "Postorder >> ";
  tree.postorder(tree.getRoot());   cout << endl;
}
```

2 트리 순회 구현

트리 순회 소스코드

컴파일 하기 (윈도우용 gcc or Linux 활용)
g++ treetraversal.cpp -o treetraversal.exe

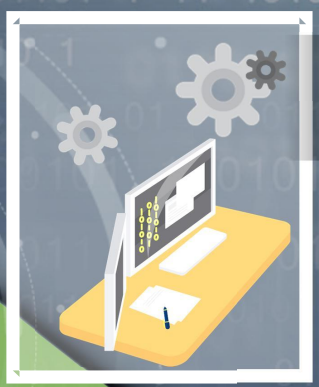


핵심 키워드 이 시간을 통해 꼭 기억해야하는 '핵심 키워드'입니다.

❶ 트리 순회

참고문헌

- https://dojang.io/mod/page/view.php?id=337



3차시

이진 검색

학습목표

❶ **이진검색 트리의 개념**을 이해하고,
소스코드로 구현 할 수 있다.

1 이진검색 트리 개념

검색 트리 Search Tree

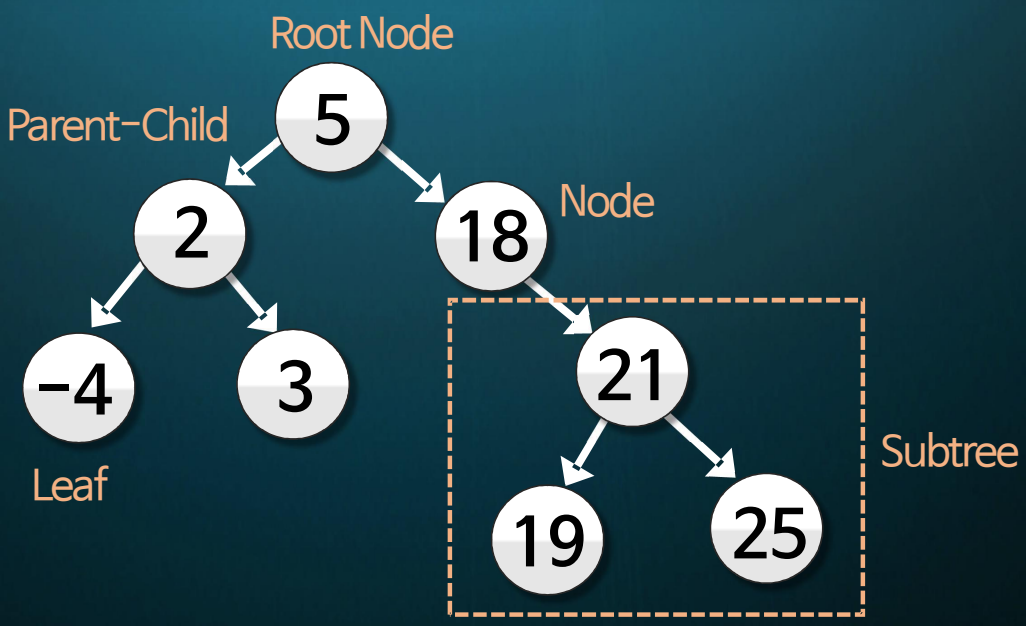
- 자료들 담는 컨테이너
- 자료들을 일정한 순서에 따라 정렬하여 저장

➡ 빠른 탐색이 가능 (이진 탐색)

이진검색 트리 Binary search tree

- 모든 노드의 키는 유일
- 왼쪽 서브 트리의 키들은 루트의 키보다 작음
  오른쪽 서브 트리의 키들은 루트의 키보다 큼
- 왼쪽과 오른쪽 서브 트리도 이진 탐색 트리

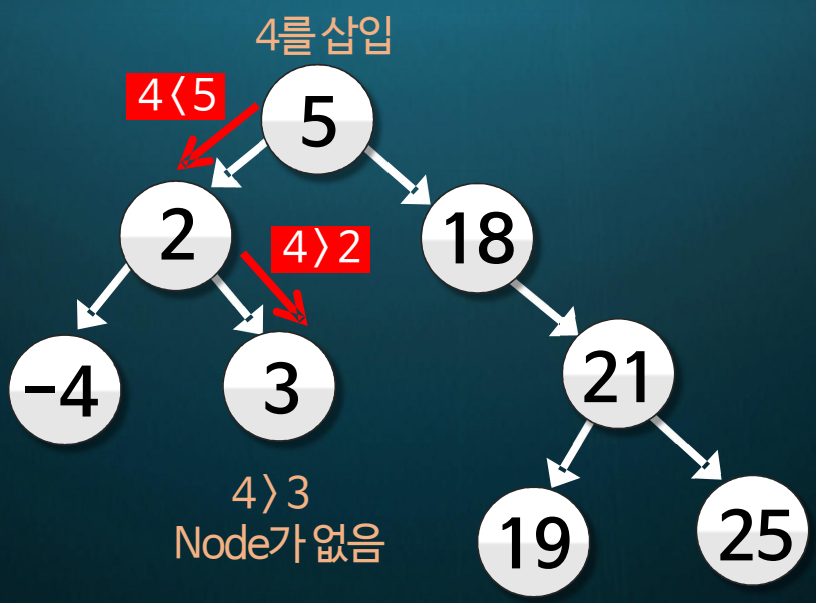
1 이진검색 트리 개념

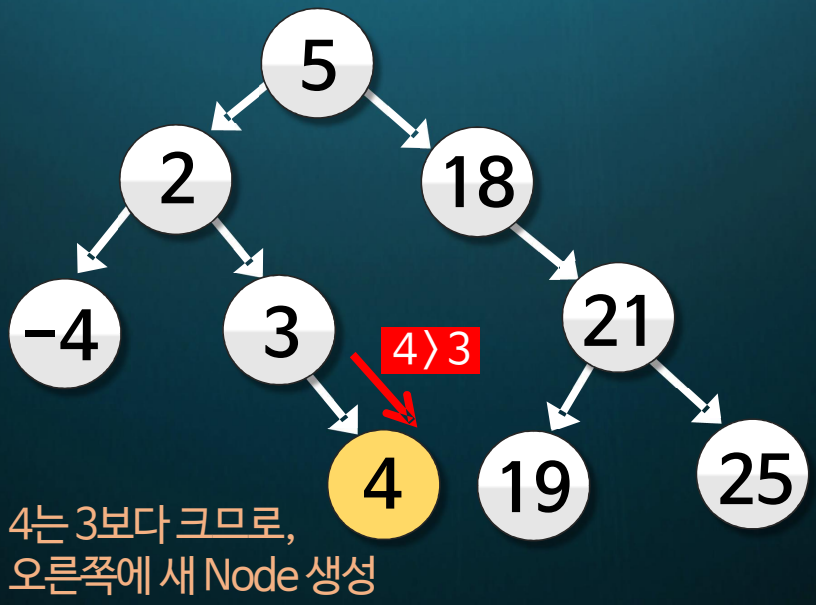


노드 추가

- 이진 탐색을 통해 추가될 위치 확인
- Node의 왼쪽 Child에 작은 값
  Node의 오른쪽 Child에 큰 값
- 탐색 결과 NULL이 되는 위치에 새로운 값 추가

1 이진검색 트리 개념





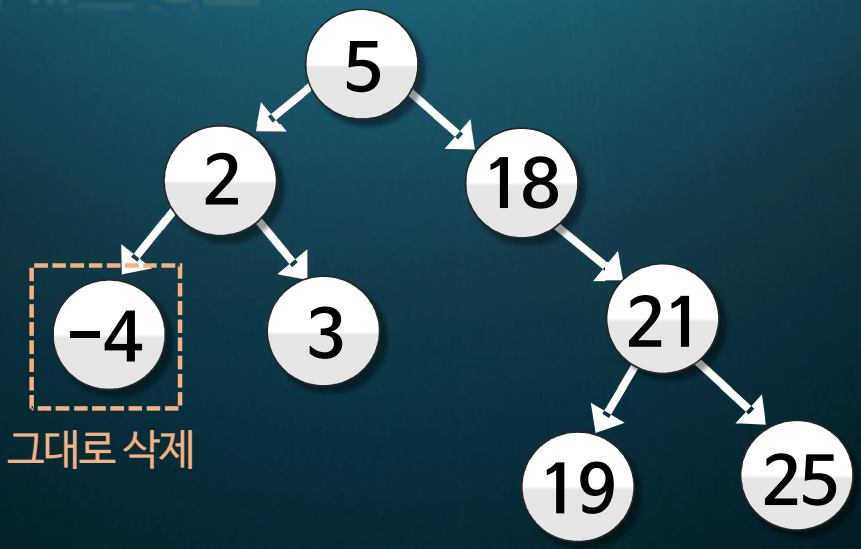
1 이진검색 트리 개념

노드 삭제

- 삭제할 노드의 Child가 없을 때, 하나일 때, 두개일 때
- 3가지 경우 마다 다른 알고리즘으로 삭제

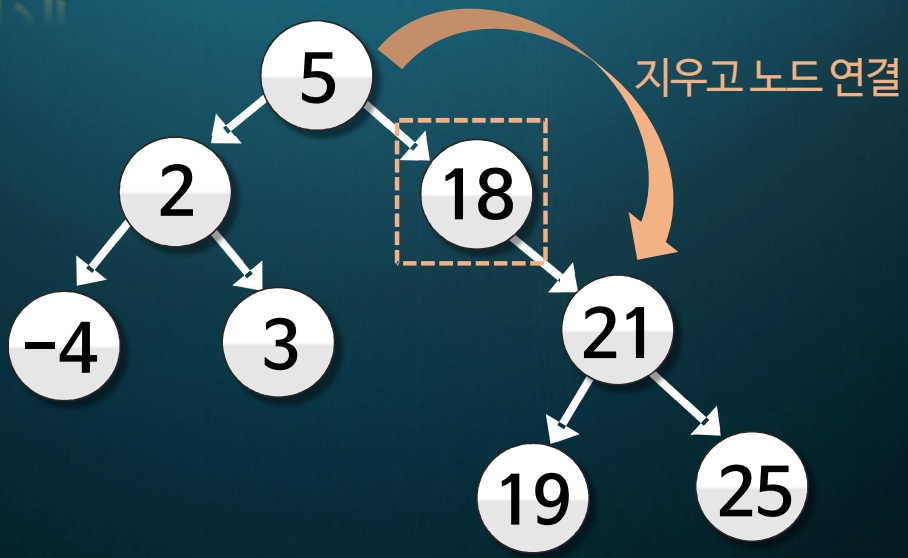
1 이진검색 트리 개념

1 Child가 없는 경우



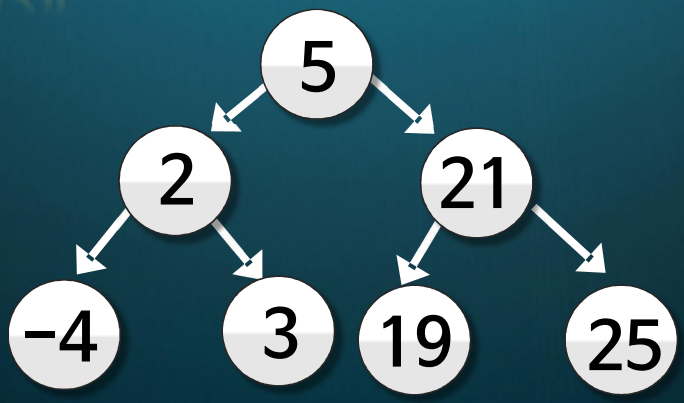
1 이진검색 트리 개념

2 Child가 1개



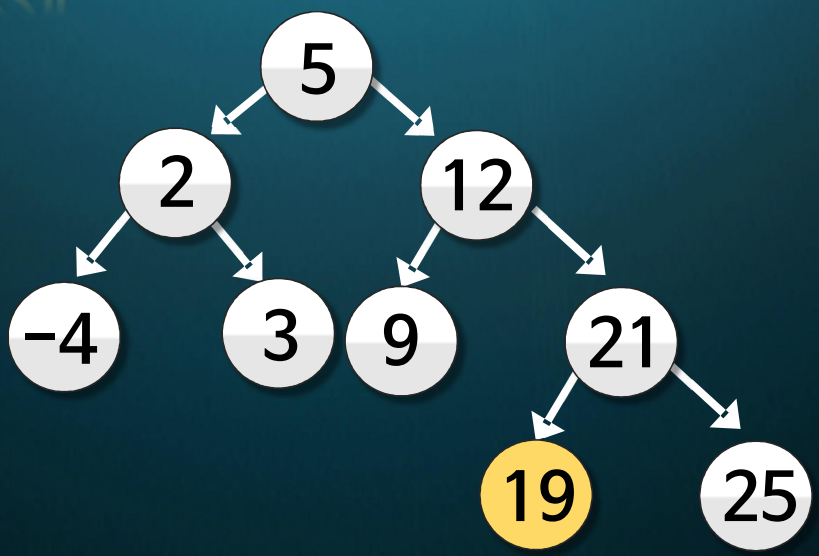
1 이진검색 트리 개념

2 Child가 1개



1 이진검색 트리 개념

3 Child가 2개



12의 오른쪽에서 최소값 선택

3 Child가 2개

1 이진검색 트리 개념

시간복잡도

- 추가, 삭제, 탐색 모두 ➔ $\log_2 N$
- 자료만 잘 정리되어 있으면 효율적

2 이진검색 트리 구현

이진검색 트리 구현 소스코드

```
void tree::recursiveInsert(treeNode *&node, int val) {
  //탐색하고 있는 노드가 NULL이면 새로 만든다(삽입)
  if (node == NULL)  node = new treeNode(val);
  //삽입할 값이 노드의 값보다 크면 오른쪽 Child를 탐색
  else if (val > node->value)
    recursiveInsert(node->right, val);
 //삽입할 값이 노드의 값보다 작으면 왼쪽 Child를 탐색
  else if (val < node->value)
    recursiveInsert(node->left, val); }
```

2 이진검색 트리 구현

```
void tree::recursiveRemove(treeNode *& node, int val) {
  treeNode *removal = new treeNode(0);
  //찾지못했을 경우 리턴해서 함수종료
  if (node == NULL)  return;
  //삭제 대상 탐색
  else if (val > node->value)
    recursiveRemove(node->right, val);
  else if (val < node->value)
    recursiveRemove(node->left, val);
```

2 이진검색 트리 구현

```
//삭제할 노드를 찾았음(이제 3가지 경우로 나눌수 있다)
 else {
//경우 1: 노드의 Child가 없을때
  if (node->left == NULL && node->right == NULL){
    delete node;
    node = NULL;
  }
```

2 이진검색 트리 구현

```
  //경우 2-1: 노드의 Child가 오른쪽 하나일 때
  else if (node->left == NULL) {
    removal = node;
    node = node->right;
    delete removal;     }
  //경우 2-2: 노드의 Child가 왼쪽 하나일 때
  else if (node->right == NULL) {
    removal = node;
    node = node->left;
    delete removal;     }
```

2 이진검색 트리 구현

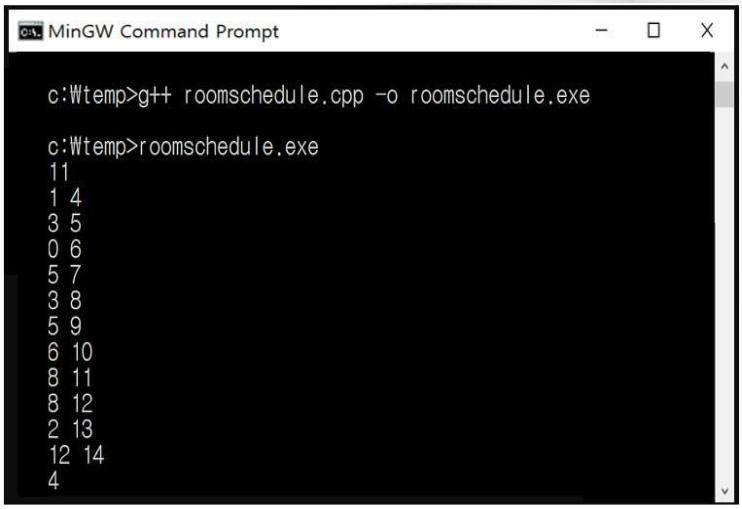
```
//경우 3: 노드의 Child가 두개일 때
else {   removal = node->right;
//노드의 오른쪽 Subtree에서 최소값을 가진 노드를 찾는다.
  while (removal->left != NULL)
    removal = removal->left;
  int minVal = removal->value;
//최소값 노드를 삭제한다
  recursiveRemove(root, minVal);
//백업한 최소값을 "삭제된 값을 가진 노드"에 덮어쓴다
  node->value = minVal;}
```

2 이진검색 트리 구현

```
int main() {
  tree tree;
  tree.insert(5);  tree.insert(2);  tree.insert(12);
  tree.insert(-4);  tree.insert(3);  tree.insert(9);
  tree.insert(21);  tree.insert(19);  tree.insert(25);
  tree.preorderPrint();
//4를 추가
  tree.insert(4);   tree.preorderPrint();
//12를 삭제
  tree.remove(12);     tree.preorderPrint();          }
```

2 이진검색 트리 구현

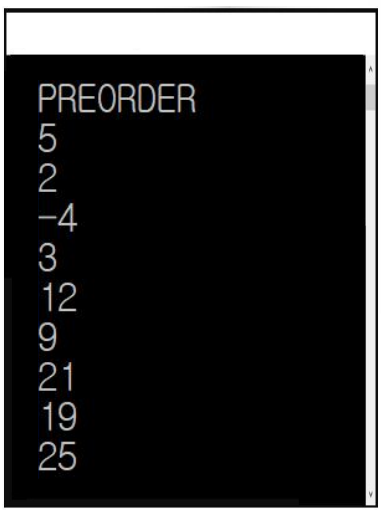
실행결과

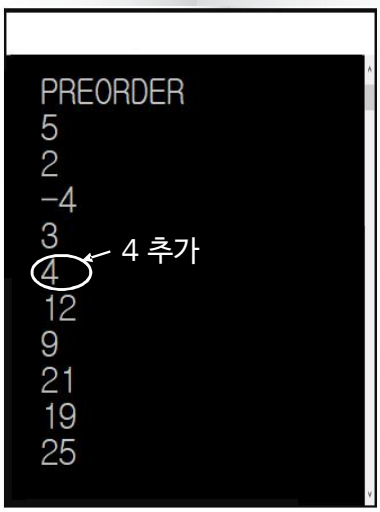


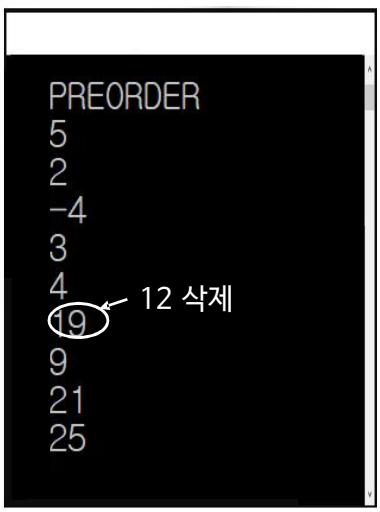
컴파일 하기
(윈도우용 gcc or Linux 활용)
g++ binarytree.cpp -o binarytree.exe

2 이진검색 트리 구현

실행결과







핵심 키워드 이 시간을 통해 꼭 기억해야하는 '핵심 키워드'입니다.

❶ 이진검색 트리

확장하기 1 **B-Tree & B+ Tree**

http://wangin9.tistory.com/entry/B-tree-B-tree

확장하기 2 **이진 트리 레벨 순회 구현**

http://yonssa.tistory.com/entry/%EC%9D%B4%EC%A7%84%ED%8A%B8%EB%A6%AC%EB%A0%88%EB%B2%A8-%EC%88%9C%ED%9A%8CLevel-Traversal

확장하기 3 **이진 탐색 트리 설계 및 사용**

http://ehpub.co.kr/ 7-3-%EC%9D%B4%EC%A7%84-%ED%83%90%EC%83%89-%ED%8A%B8%EB%A6%AC-%EC%84%A4%EA%B3%84-%EB%B0%8F-%EC%82%AC%EC%9A%A9/

참고문헌

- http://wangin9.tistory.com/entry/B-tree-B-tree
- http://yonssa.tistory.com/entry/%EC%9D%B4%EC%A7%84-%ED%8A%B8%EB%A6%AC%EB%A0%88%EB%B2%A8-%EC%88%9C%ED%9A%8CLevel-Traversal
- http://ehpub.co.kr/7-3-%EC%9D%B4%EC%A7%84-%ED%83%90%EC%83%89-%ED%8A%B8%EB%A6%AC-%EC%84%A4%EA%B3%84-%EB%B0%8F-%EC%82%AC%EC%9A%A9/
- http://mintnlatte.tistory.com/457