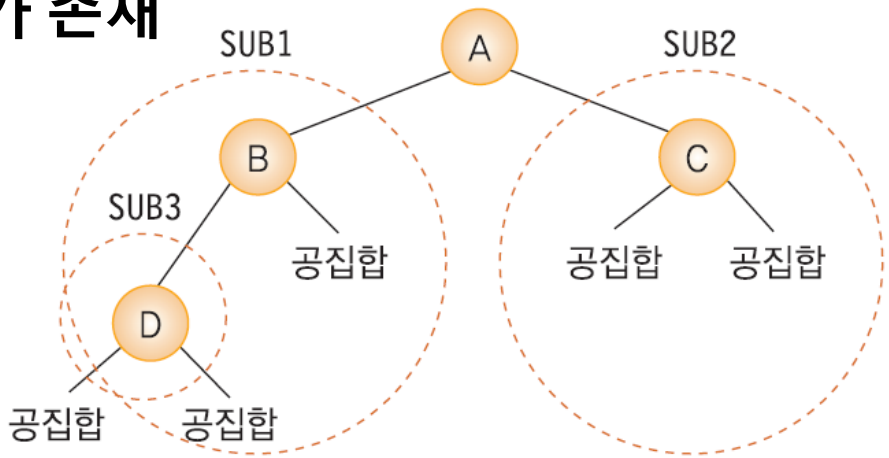
9 주차 실습

실습 - (partice_17.c)

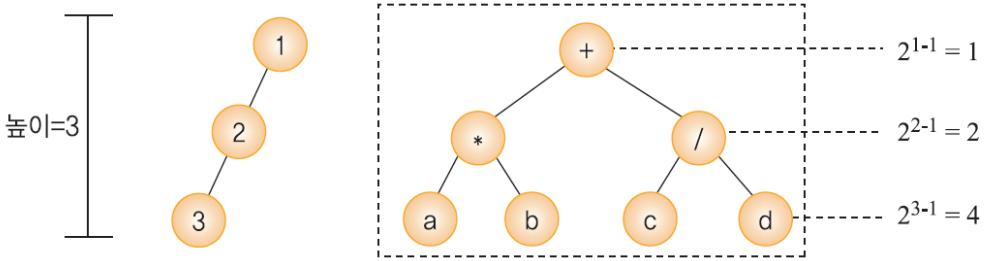
이진 트리

- 모든 노드가 2개의 서브 트리를 가지고 있는 트리
  - 최대 2개까지의 자식 노드가 존재
  - 모든 노드의 차수가 2 이하
  - 서브 트리간 순서 존재



- 노드의 개수가 n 이면
  간선의 개수는 n-1

- 높이가 x인 이진 트리의 경우
  최소 x개의 노드를 가짐 and 최대 $2^h - 1$개의 노드를 가짐



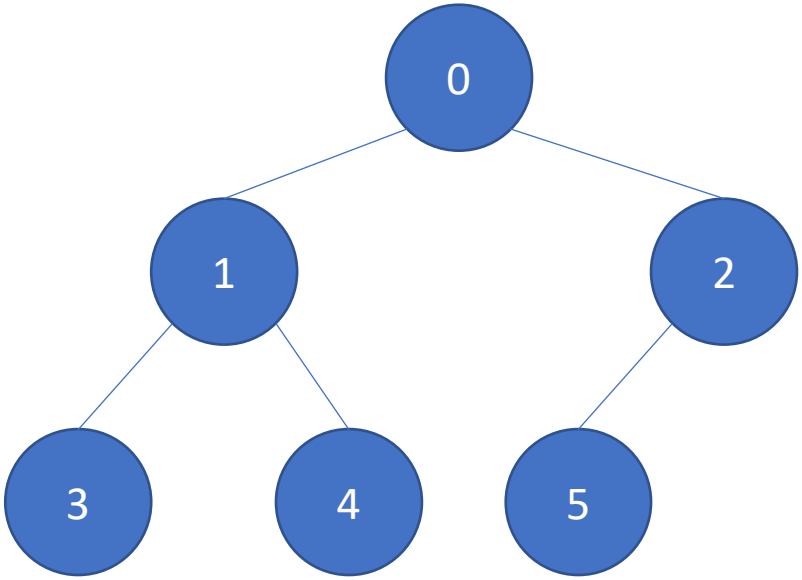
최소 노드 개수 = 3

최대 노드 개수 = $2^{1-1} + 2^{2-1} + 2^{3-1} = 1 + 2 + 4 = 7$

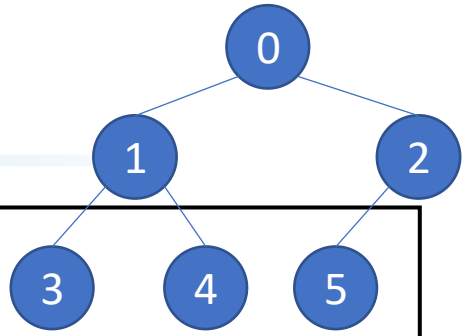
실습 - (partice_17.c)

이진 트리

- 전위, 중위, 후위



실습 - (partice_17.c)



이진 트리

```c
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>

typedef struct TreeNode {
int data;
struct TreeNode* left, * right;
} TreeNode;
TreeNode n1 = { 5, NULL, NULL };
TreeNode n2 = { 4, NULL, NULL };
TreeNode n3 = { 3, NULL, NULL };
TreeNode n4 = { 2, &n1, NULL };
TreeNode n5 = { 1, &n3,  &n2 };
TreeNode n6 = { 0, &n5,  &n6 };
TreeNode* root = &n6;

void inorder(TreeNode* root) {
  if (root) {
    inorder(root->left);
    printf("[%d] ", root->data);
    inorder(root->right);
  }
}
```

```c
void preorder(TreeNode* root) {
  if (root != NULL) {
    printf("[%d] ", root->data);
    preorder(root->left);
    preorder(root->right);
  }
}
void postorder(TreeNode* root) {
  if (root != NULL) {
    postorder(root->left);
    postorder(root->right);
    printf("[%d] ", root->data);
  }
}
int main(void) {
TreeNode n1 = { 5, NULL, NULL };TreeNode n2 = { 4, NULL, NULL };
TreeNode n3 = { 3, NULL, NULL };TreeNode n4 = { 2, &n1,  NULL };
TreeNode n5 = { 1, &n3,  &n2 };TreeNode n6 = { 0, &n5,  &n4 };
TreeNode* root = &n6;

printf("중위 순회");    inorder(root);
printf("₩n");
printf("전위 순회");    preorder(root);
printf("₩n");
printf("후위 순회");    postorder(root);
printf("₩n");
return 0;
}
```

실습 (partice_18.c)

이진 탐색 트리

```c
#include <stdio.h>
#include <stdlib.h>
typedef int element;
typedef struct TreeNode {
   element key;
   struct TreeNode *left, *right;
} TreeNode;
TreeNode * new_node(int item){
   TreeNode * temp = (TreeNode *)malloc(sizeof(TreeNode));
   temp->key = item;
   temp->left = temp->right = NULL;
   return temp;
}
TreeNode* min_value_node(TreeNode* node) {
   TreeNode* current = node;
   while (current->left != NULL) {
      current = current->left;
   }
   return current;
}
TreeNode * insert_node(TreeNode * node, int key){
   if (node == NULL)
      return new_node(key);
   if (key < node->key)
      node->left = insert_node(node->left, key);
   else if (key > node->key)
      node->right = insert_node(node->right, key);
   return node;
}
```

```c
TreeNode* delete_node(TreeNode* root, int key){
   if (root == NULL)
      return root;
   if (key < root->key)
      root->left = delete_node(root->left, key);
   else if (key > root->key)
      root->right = delete_node(root->right, key);
   else {
      if (root->left == NULL) {
         TreeNode* temp = root->right;
         free(root);
         return temp;
      }
      else if (root->right == NULL) {
         TreeNode* temp = root->left;
         free(root);
         return temp;
      }
      TreeNode* temp = min_value_node(root->right);
      root->key = temp->key;
      root->right = delete_node(root->right, temp->key);
   }
   return root;
}
```

실습 (partice_18.c)

이진 탐색 트리

```
TreeNode* search(TreeNode* node, int key) {
…
}
int main(void){
  TreeNode* root = NULL;
  root = insert_node(root, 30);
  root = insert_node(root, 20);
  root = insert_node(root, 10);
  root = insert_node(root, 40);
  root = insert_node(root, 50);
  root = insert_node(root, 60);
  if (search(root, 30) != NULL)
    printf("이진 탐색 트리에서 30을 발견함 Wn");
  else
    printf("이진 탐색 트리에서 30을 발견못함 Wn");
  return 0;
}
```

- 반복적인 탐색
  1. 노드를 받는다. (검색할 키와 함께)
  2. 노드가 NULL이 아닐 때 반복
     1. 탐색 하려는 키가 현재 노드 키와 같으면 return 한다.
     2. 탐색 하려는 키가 현재 노드 키보다 작으면 left를 접근한다.
     3. 탐색 하려는 키가 현재 노드 키보다 크면 right를 접근한다.

- 순환적인 탐색
  1. 노드를 받는다. (검색할 키와 함께)
  2. 탐색하려는 키가 현재 노드 키와 같으면 node를 return 한다.
  3. 탐색하려는 키가 현재 노드 키보다 작으면 search 함수를 다시 부르는데 이 때, left 노드를 파라미터 값으로 보내준다.
  4. 탐색하려는 키가 현재 노드 키보다 크면 search 함수를 다시 부르는데 이 때, right 노드를 파라미터 값으로 보내준다.

실습

이진 트리 분석

- 탐색, 삽입, 삭제 연산의 평균 시간 복잡도: O(h) * 높이가 h라고 했을 경우
- 노드의 최대 값 $2^h - 1$
- $n = 2^h - 1$ 만약, 한쪽으로 치우칠 경우 O(n) 이 되어버림
- $2^h = n + 1$
- $h = log_2(n + 1)$
- 결론 O($log_2 n$)

실습 - (partice_19.c)

이진 탐색 트리를 활용한 전화번호부

- 위 만들어 놓은 이진 탐색 트리를 일부 수정하여
- 친구 이름 / 전화번호를 저장할 수 있는 전화번호부를 만들어보자 (배열 변수)
- 5페이지 소스코드 (이진 탐색 트리) 수정하여 작성 (string.h 추가)
- Search /inorder 함수도 추가
  (위 내용을 모두 한 학생은 FILE에 최종 저장하는 것도 해보기 바람)

```c
#include <stdio.h>
#include <stdlib.h>
typedef int element;
typedef struct TreeNode {
    element key;
    struct TreeNode *left, *right;
} TreeNode;
TreeNode * new_node(int item){
    TreeNode * temp = (TreeNode *)malloc(sizeof(TreeNode));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}
TreeNode* min_value_node(TreeNode* node) {
    TreeNode* current = node;
    while (current->left != NULL) {
        current = current->left;
    }
    return current;
}
TreeNode * insert_node(TreeNode * node, int key){
    if (node == NULL)
        return new_node(key);
    if (key < node->key)
        node->left = insert_node(node->left, key);
    else if (key > node->key)
        node->right = insert_node(node->right, key);
    return node;
}
```

```c
TreeNode* delete_node(TreeNode* root, int key){
    if (root == NULL)
        return root;
    if (key < root->key)
        root->left = delete_node(root->left, key);
    else if (key > root->key)
        root->right = delete_node(root->right, key);
    else {
        if (root->left == NULL) {
            TreeNode* temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL) {
            TreeNode* temp = root->left;
            free(root);
            return temp;
        }
        TreeNode* temp = min_value_node(root->right);
        root->key = temp->key;
        root->right = delete_node(root->right, temp->key);
    }
    return root;
}
```