자료구조 (Data Structure)
12주차: 이진 탐색 트리

지난 시간 내용

- 이진탐색트리: 왼쪽 자손 < 노드 < 오른쪽 자손
- 데이터 찾기: 왼쪽 또는 오른쪽 자손에서
- 데이터 추가: 자손의 끝까지 찾아서 그 자리에
- 데이터 삭제: 적절한 자손을 부모에 연결

오늘의 목차

- 출력하기
- 이진트리 출력하기
- 이진탐색트리에서 노드 찾기

출력하기 목차

- printf로 "Hello, tree!" 출력하기
- 문자 배열에 "Hello, bst!"를 저장하고, 출력하기
- 이차원 문자 배열에 'H', 'I', '!'를 저장하고, 출력하기

출력 - Hello, tree

- printf로 문자열을 출력할 수 있다

출력 - Hello, tree

```
#include <stdio.h>

int main()
{
        printf("Hello, tree!\n");
        return 0;
}
```

출력 - 문자 배열

- 문자열: 문자의 배열

출력 - 문자 배열

```
int main()
{       /* 이전 내용에 이어서 */

        char str[20] = "Hello, tree!";
        printf("%s\n", str);
        return 0;
}
```

출력 - 문자 배열

- 문자열의 끝은 '\0'으로 표시한다

출력 - 문자 배열

```
int main()
{       /* 이전 내용에 이어서 */

        for (int i = 0; i < 20; i++) {
                printf("%c(%d)\n", str[i], str[i]);
        }
        return 0;
}
```

출력 - 2차원 문자 배열

- 이차원 문자 배열로 문자열 여러개를 쓸 수 있다

출력 - 2차원 문자 배열

```
#define WIDTH 32
#define HEIGHT 9

char texts[HEIGHT][WIDTH + 1];
```

출력 - 2차원 문자 배열

```
void reset_texts()
{
	for (int i = 0; i < HEIGHT; i++) {
		for (int j = 0; j < WIDTH; j++) {
			texts[i][j] = ' ';
		}
		texts[i][WIDTH] = '\0';
	}
}
```

출력 - 2차원 문자 배열

```
void print_texts()
{
        for (int i = 0; i < HEIGHT; i++)
                printf("%s\n", texts[i]);
}
```

출력 - 2차원 문자 배열

```
int main()
{       /* 이전 내용에 이어서 */
        reset_texts();
        texts[0][0] = 'H';
        texts[1][1] = 'I';
        texts[2][2] = '!';
        print_texts();
        return 0;
}
```

출력 - 2차원 문자 배열

```
int main()
{       /* 이전 내용에 이어서 */
        texts[0][0] = ' ';
        texts[1][1] = ' ';
        texts[2][2] = ' ';
        print_texts();
        return 0;
}
```

출력하기 요약

- 이차원 배열 texts에 출력할 내용을 편집한다

이진트리 출력하기 목차

- 이진 트리 노드 구조체
- 루트 출력하기
- 루트가 아닌 노드 출력하기

트리 출력 - 노드 구조체

- 트리 노드에 데이터, 왼쪽 자식, 오른쪽 자식이 있다

트리 출력 - 노드 구조체

```
struct tree_node {
        int data;
        struct tree_node *left;
        struct tree_node *right;
};
```

트리 출력 - 노드 구조체

```
void print_node(struct tree_node *node)
{
        char data = '0' + node->data;
        char l = node->left ? '0' + node->left->data : '.';
        char r = node->right ? '0' + node->right->data : '.';
        printf("node[%c] (%c, %c)\n", data, left, right);
}
```

트리 출력 - 노드 구조체

```
int main()
{       /* 이전 내용에 이어서 */
        struct tree_node nodes[10];
        for (int i = 0; i < 10; i++) {
                nodes[i].data = i;
                nodes[i].left = nodes[i].right = NULL;
        }
        print_tree(&nodes[0]);
        return 0;
}
```

트리 출력 - 루트

- 루트 노드를 출력할 때는 화면 가운데에 출력한다

트리 출력 - 루트

```
void print_tree(struct tree_node *root)
{
        int x = WIDTH / 2;
        int y = 0;
        if (root)
                texts[y][x] = '0' + root->data;
        else
                texts[y][x] = '.';
        print_texts();
}
```

트리 출력 - 비루트

- 루트가 아닌 노드를 출력할 때는,

  - 노드의 위치를 계산한다

  - 부모로부터 연결선을 출력한다

  - 데이터를 출력한다

  - 자식들을 재귀호출한다

트리 출력 - 비루트

```
void print_tree_visit(struct tree_node *node, int level,
int level_index)
{
        if (level >= (HEIGHT + 1) / 2)
                return;

        int y_edge = 2 * level - 1;
        int y_node = 2 * level
```

트리 출력 - 비루트

```
int level_capacity = 1 << level;
int level_margin = WIDTH / level_capacity / 2;

int x = WIDTH * level_index / level_capacity
        + level_margin

texts[y_node][x] = '0' + node->data;
```

트리 출력 - 비루트

```
if (level_index % 2 == 0) {
        x += level_margin / 2;
        texts[y_edge][x] = '/';
}
else {
        x -= level_margin / 2;
        texts[y_edge][x] = '\\';
}
```

트리 출력 - 비루트

```
    if (node->left) {
        print_tree_visit(node->left, level + 1,
            level_index * 2);
    }
    if (node->right) {
        print_tree_visit(node->right, level + 1,
            level_index * 2 + 1);
    }
}
```

트리 출력 - 비루트

```
void print_tree(struct tree_node *root)
{       /* 이전 내용에 이어서 */
        if (root->left) {
                print_tree_visit(root->left, 1, 0);
        }
        if (root->right) {
                print_tree_visit(root->right, 1, 1);
        }
        print_texts();
}
```

트리 출력 - 비루트

```
int main()
{       /* 이전 내용에 이어서 */
        for (int i = 0; i < 9; i++) {
                nodes[i].right = &nodes[i + 1];
        }
        print_tree(&nodes[0]);
        reset_texts();

        return 0;
}
```

이진트리 출력하기 요약

- 이진 트리 노드 구조체에 data, left, right가 있다
- print_tree 함수로 트리를 출력한다

노드 찾기 목차

- 균형 잡힌 이진 탐색 트리 만들기
- 찾기 함수 만들기

노드 찾기 - make_bst

- 정렬된 배열을 bst로 만들때,
- 전체 구간의 가운데가 루트다
- 루트의 왼쪽 자식은 왼쪽 구간의 가운데다
- 루트의 오른쪽 자식은 오른쪽 구간의 가운데다
- 재귀호출한다

노드 찾기 - make_bst

```
struct interval {
        int l;
        int r;
};

void set_childs_mid(struct tree_node *node_array,
struct interval *range);
```

노드 찾기 - make_bst

```
struct tree_node *balanced_bst_from_sorted_nums (
struct tree_node *nodes, int *nums, int size)
{
        for (int i = 0; i < size; i++) {
                nodes[i].data = nums[i];
                nodes[i].left = NULL;
                nodes[i].right = NULL;
        }
```

노드 찾기 - make_bst

```
    struct interval range = {0, size - 1};
    set_childs_mid(nodes, &range);

    int mid = (range.l + range.r + 1) / 2;
    return &nodes[mid];
}
```

노드 찾기 - make_bst

```
void set_childs_mid(struct tree_node *nodes,
struct interval *range)
{
        int mid = (range->l + range->r + 1) / 2;

        struct interval left_range = {range->l, mid - 1};
        struct interval right_range = {mid + 1, range->r};
```

노드 찾기 - make_bst

```
if (left_range.l <= left_range.r) {
        int left_mid
                = (left_range.l + left_range.r + 1) / 2;
        nodes[mid].left = &nodes[left_mid];
        set_childs_mid(nodes, &left_range);
}
```

노드 찾기 - make_bst

```
        if (right_range.l <= right_range.r) {
                int right_mid
                        = (right_range.l + right_range.r + 1) /2;
                nodes[mid].right = &nodes[right_mid];
                set_childs_mid(nodes, &right_range);
        }
}
```

노드 찾기 - make_bst

```
int main()
{       /* 이전 내용에 이어서 */
        for (int i = 0; i < 9; i++) {
                nodes[i].data = -1;
                nodes[i].left = NULL;
                nodes[i].right = NULL;
        }
        int nums[10];
        for (int i = 0; i < 10; i++)
                nums[i] = i;
```

노드 찾기 - make_bst

```
        struct tree_node *root
                = balanced_bst_from_sorted_nums(
                        nodes, nums, 10);

        print_tree(root)
        return 0;
}
```

노드 찾기 - search

- 노드가 없으면 NULL을 반환한다
- key == node->data면 노드를 반환한다
- key < node->data면 왼쪽 자손에서 찾는다
- key > node->data면 오른쪽 자손에서 찾는다

노드 찾기 - search

```
struct tree_node *search(struct tree_node n, int key)
{
        if (!n) return NULL;
        if (key == n->data)
                return n;
        else if (key < n->data)
                return search(n->left, key);
        else
                return search(n->right, key);
}
```

노드 찾기 - search

```
int main()
{       /* 이전 내용에 이어서 */
        struct tree_node *seven = search(root, 7);
        printf("search node 7... ");
        print_node(seven);

        return 0;
}
```

노드 찾기 요약

- 정렬된 구간의 가운데를 이어서 bst를 만든다
- 왼쪽 또는 오른쪽을 따라 search를 한다

오늘 한 내용

- 전역 변수로 출력할 내용을 저장한다
- 좌표를 계산해서 이진트리를 출력한다
- 방향을 선택해서 이진탐색트리의 노드를 찾는다