자료구조 (Data Structure)
5주차: 트리

지난 시간 요약

- 트리: 루트로부터 부모 자식 관계로 연결된 구조
- 이진 트리: 자식 노드가 최대 2개인 트리
- 이진 트리 예제: 수식 트리
- 이진 트리 노드: left, right, value
- 트리 방문: Preorder, Inorder, Postorder

오늘 목표

1) 이진 트리 및 방문 구현

2) 예제: 방문 순서로 이진 트리 재구성

3) 예제: 수식 트리

이진 트리 및 방문 구현

- 이진 트리 복습

- 이진 트리 노드 타입
- 이진 트리 타입

- 이진 트리 방문
- 이진 트리 삭제하기

이진 트리

- 자식 노드가 최대 2개인 트리

  - root: 트리의 시작점 (타입: 이진 트리 노드)

이진 트리 노드 타입

- 데이터와 자식 노드 2개 정보의 묶음

  - left: 왼쪽 자식 노드 주소

  - right: 오른쪽 자식 노드 주소

  - value: 저장된 데이터

이진 트리 노드 타입

```
struct bi_node {
        struct bi_node *left;
        struct bi_node *right;
        char value;
};
```

이진 트리 노드 타입

```
struct bi_node *new_bi_node(
        char v, struct bi_node *l, struct bi_node *r)
{
        struct bi_node *node
                = malloc(sizeof(struct bi_node));
        node->left = l;
        node->right = r;
        node->value = v;
        return node;
}
```

이진 트리 타입

```
struct bi_root {
        struct bi_node *root;
};
```

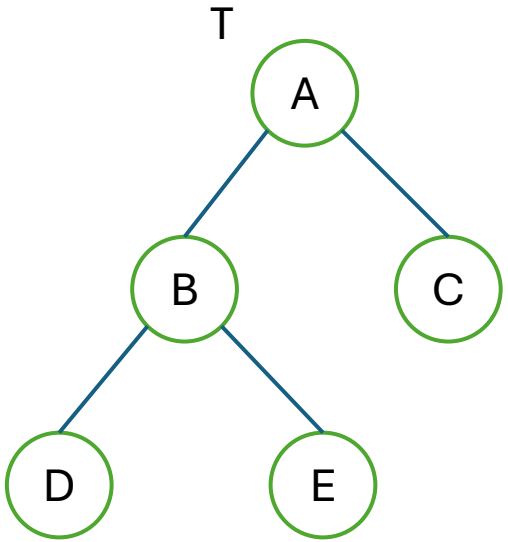
이진 트리 타입

```
struct bi_root *new_bi_root(struct bi_node *root)
{
        struct bi_root * tree
                = malloc(sizeof(struct bi_root));

        tree->root = root;
        return tree;
}
```

이진 트리 예시

- 다음과 같은 이진 트리를 만드는 함수를 써보자



깊이 우선 이진 트리 방문

함수 TRAVERSE(node):

- 만약: node != NULL이면

    pre_task(node->value)

    TRAVERSE(node->left)

    in_task(node->value)

    TRAVERSE(node->right)

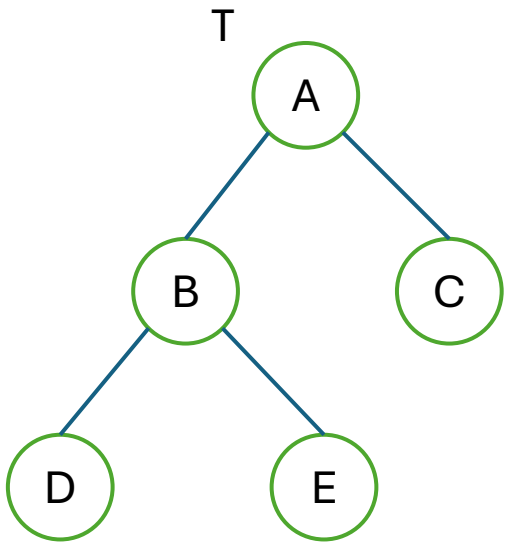
    post_task(node->value)

깊이 우선 이진 트리 방문

```
void traverse(struct bi_node *node)
{
        if (node != NULL) {
                pre_task(node->value);
                traverse(node->left);
                in_task(node->value);
                traverse(node->right);
                post_task(node->value);
        }
}
```

이진 트리 방문시 작업 타이밍

- 다음 트리에 대한 traverse(root)에서
  pre-, in-, post-order 작업이 실행되는 순서는?



이진 트리 삭제하기

- 트리 삭제시 노드를 free할 자연스러운 타이밍은?

(힌트: 먼저 변수를 다 쓰고 free 한다)

이진 트리 삭제하기

함수 DELETE(node):

- 만약: node != NULL이면

  DELETE(node->left)

  DELETE(node->right)

  free(node)

이진 트리 삭제하기

```
void bi_node_delete(struct bi_node *node)
{
        if (node != NULL) {
                bi_node_delete(node->left);
                bi_node_delete(node->right);
                free(node);
        }
}
```

이진 트리 삭제하기

트리 루트를 삭제하는 함수를 구현하면?

```
void bi_tree_delete(struct bi_root *tree)
{


}
```

이진 트리 및 방문 구현 요약

- 이진 트리 - 자식 노드가 2개까지 가능한 트리

- 이진 트리 노드 타입 - left, right, value
- 이진 트리 타입 - root

- 이진 트리 방문 - 순서 preorder, inorder, postorder
- 이진 트리 삭제하기 - postorder로

예제: 방문 순서로 이진 트리 재구성

- 트리 재구성 퀴즈
- 트리 재구성 전략
  - 부분 리스트
  - 데이터 찾기
  - 부분 트리 찾기
- 트리 재구성 c 코드

트리 재구성 퀴즈

- preorder: A B D E C
- inorder: D B E A C
- Original tree???

트리 재구성 전략

함수 rebuild(preorder, inorder):

- 만약: preorder가 비어있으면, 반환: NULL

- root_val = RETRIEVE(preorder, 1)
- in_pivot = inorder에서 root_val의 순서

- in_l = inorder 부분리스트(inorder 시작, in_pivot - 1)
- in_r = inorder 부분리스트(in_pivot + 1, inorder 끝)

트리 재구성 전략

- pre_l = preorder에서 in_l에 해당하는 부분리스트
- pre_r = preorder에서 in_r에 해당하는 부분리스트

- left = rebuild(pre_l, in_l)
- right = rebuild(pre_r, in_r)

- 반환: 새 노드(root_val, left, right)

트리 재구성 전략

- preorder: A B D E C
- inorder: D B E A C
- rebuild(preorder, inorder) 실행 과정은?

트리 재구성 전략 - 부분 리스트

- preorder, inorder를 저장하기에 적합한 자료구조는?

트리 재구성 전략 - 부분 리스트

```
struct view {
        const char *data;
        int start;
        int end;
        int size;
};
```

트리 재구성 전략 - 부분 리스트

```
struct view *new_view(const char* arr, int from, int to)
{
        struct view *obj = malloc(sizeof(struct view));
        obj->data = arr;
        obj->start = from;
        obj->end = to;
        obj.size = to - from + 1;
        return obj;
}
```

트리 재구성 전략 - 부분 리스트

- preorder: A B D E C
- preorder의 3 - 5번에 해당하는 부분 리스트를 new_view 함수로 만들려면?

트리 재구성 전략 - 데이터 찾기

- preorder: A B D E C
- preorder에서 B의 순서를 찾는 방법은?

트리 재구성 전략 - 데이터 찾기

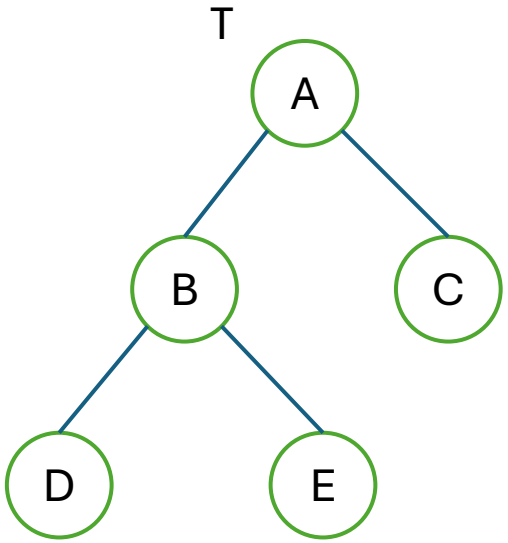
```
int find_index(struct view *range, char value)
{
        for (int i = range->start; i <= range->end; i++) {
                if (view->range[i] == value)
                        return i;
        }
        return -1;
}
```

트리 재구성 전략 - 데이터 찾기

- preorder: A B D E C
- find_index(preorder, 'D')의 실행 과정은?

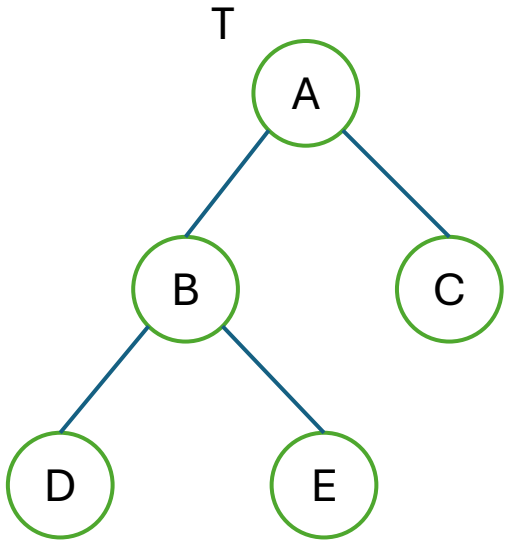
트리 재구성 전략 - 부분 트리 찾기

- inorder: D B E A C
- inorder에서 루트의 왼쪽에 해당하는 부분리스트를 구하면?



트리 재구성 전략 - 부분 트리 찾기

- preorder: A B D E C
- preorder에서 부분 트리 B에 해당하는 부분리스트를 구하면?



트리 재구성 전략 - 부분 트리 찾기

- preorder: A B D E C
- preorder에서 D B E로 이뤄진 부분리스트를 구하면? (단, D B E는 부분 트리를 이룬다)

트리 재구성 전략 - 부분 트리 찾기

```
struct view *find_subrange(struct view *range,
                           struct view *set)
{
        int is_in_set[256] = {0};

        for (int i = set->start; i <= set->end; i++) {
                int key = (unsigned char)set->data[i];
                is_in_set[key] = 1;
        }
```

트리 재구성 전략 - 부분 트리 찾기

```
        int i = range->start;
        while (i <= range->end) {
                int key = (unsigned char)range->data[i];
                if (is_in_set[key])
                        break;
                i++;
        }
        return new_view(range->data, i, i + set->size - 1);
}
```

트리 재구성 전략 - 부분 트리 찾기

- preorder: A B D E C
- subrange_set: D B E
- find_subrange(preorder, subrange_set)의 실행 과정은?

트리 재구성 C 코드

- rebuild(preorder, inorder) 함수를 C로 구현하면?

트리 재구성 C 코드

```
struct bi_node *rebuild(struct view *pre,
                        struct view *in)
{

        if (pre->size == 0 || in->size == 0) return NULL;

        int root_val = pre->data[pre->start]);
        int in_pivot = find_index(in, root_val);
```

트리 재구성 C 코드

```
struct view *in_l = new_view(in->data,
                             in_start,
                             in->pivot - 1);

struct view *in_r = new_view(in->data,
                             in_pivot + 1,
                             in->end);
```

트리 재구성 C 코드

```
struct view *pre_l = find_range(pre, in_l);
struct view *pre_r = find_range(pre, in_r);

struct bi_node *left = rebuild(pre_l, in_l);
struct bi_node *right = rebuild(pre_r, in_r);
```

트리 재구성 C 코드

```
        free(in_l);
        free(in_r);
        free(pre_l);
        free(pre_r);

        return create_bi_node(root_val, left, right);
}
```

트리 재구성 C 코드

- preorder: A B D E C
- inorder: D B E A C
- rebuild(preorder, inorder)의 실행과정은?

예제: 방문 순서로 이진 트리 재구성 요약

- 트리 재구성 퀴즈
- 트리 재구성 전략: 재귀적으로 왼쪽/오른쪽 구분
  - 부분 리스트: 리스트의 일부 구간
  - 데이터 찾기
  - 부분 트리 찾기
- 트리 재구성 C 코드

다음 시간 예고

- 이진 트리의 특수한 케이스

예제: 수식 트리

- 수식 트리 퀴즈
- 수식 트리 생성
  - 수식 트리 노드 타입
  - C코드
- 수식 트리 계산

수식 트리 퀴즈

- 1 + 2 * 3을 수식 트리로 만들고, 값을 계산하면?

수식 트리 생성

함수 expr_tree(수식 노드 큐 Q, 처리중인 우선순위 p)

- node = DEQUEUE(Q)
- 반복: size(Q) > 0인 동안

    op = FRONT(Q)

    만약: 우선순위(op) ≤ p이면 반복 종료

    DEQUEUE(Q)

    op->left = node

    op->right = expr_tree(Q, 우선순위(op))

    node = op
- 반환: node

수식 트리 생성 - 노드 타입

- 수식 트리 노드의 데이터에 저장할 정보는?

수식 트리 생성 - 노드 타입

- 숫자 또는 연산자가 저장된다

  - type: 숫자, 더하기, 곱하기 구분
  - value: 계산 결과를 저장
  - precedence:

    더하기는 1
    곱하기는 2
    숫자는 -1

수식 트리 생성 - 노드 타입

```
struct expr_node {
        struct expr_node *l;        // 왼쪽 자식 노드
        struct expr_node * r;       // 오른쪽 자식 노드
        char type;                  // 'i': 숫자, '+': 덧셈, '*': 곱셈
        int value;                  // 결과 값
        int precedence;             // 연산자 우선순위
};
```

수식 트리 생성 - C 코드

- 수식 노드 큐를 수식 트리로 만드는 함수를 C 코드로 써보면?

수식 트리 생성 - C 코드

```
struct expr_node *expr_tree(struct expr_q *q,
                            int cur_prec)
{
        struct expr_node *node = expr_q_dequeue(q);

        while (!expr_q_empty(q)) {
                struct expr_node *op = expr_q_front(q);
                if (op->precedence <= cur_prec)
                        break;
```

수식 트리 생성 - C 코드

```
            expr_q_dequeue(q);
            op->l = node;
            op->r = expr_tree(q, op->precedence);
            node = op;
        }
        return node;
}
```

수식 트리 생성 - C 코드

- expr_q: [1, '+', 2, '*', 3, '*', 4, '+', 5]
- expr_tree(expr_q, 0)이 실행되는 과정은?

수식 트리 계산

- 수식 트리를 계산할 때 pre-, in-, post-order 중 적절한 작업 타이밍은?

수식 트리 계산

- 함수 post_task(struct expr_node *expr)에서 할 일로 적절한 것은?

수식 트리 계산

함수 post_task(struct expr_node *expr):

- 스위치(expr->type):
- 케이스 '+':

  expr->value = expr->l->value + expr->r->value;

  break;
- 케이스 '*':

  expr->value = expr->l_value * expr->r->value;

  break;
- 케이스 'i':

  break;

수식 트리 계산

```
void expr_post_task(struct expr_node *e)
{
        if (e->type == '+')
                e->value = e->l->value + e->r->value;
        else if (e->type == '*')
                e->value = e->l->value * e->r->value;
}
```

수식 트리 계산

```
void expr_eval(struct expr_node *e)
{
        if (e->l)
                expr_eval(e->l);
        if (e->r)
                expr_eval(e->r);

        expr_post_task(e)
}
```

수식 트리 계산

- 다음 수식 트리의 루트에서 expr_eval을 실행했을 때 실행되는 과정은?