

자료구조를 구조하자

2019.11.03

32153180 이상민

32162436 신창우 32163006 이건욱

32164420 조정민 32164959 허전진

학습 내용

(3) 최대 힙에서의 삽입

최대 힙의 삽입 함수

```

void insertMaxHeapType()
{
    int MaxHeapType* insertMaxHeapType()
    {
        int key;
        cout << "삽입할 값을 입력하세요: ";
        cin >> key;
        if (key > 0)
        {
            // 삽입할 경우 0이 아닌 값을 입력한 경우
            // 삽입할 위치를 찾는다.
            int i = 1;
            while (i <= size)
            {
                if (i < 2 * i && arr[i] < arr[2 * i])
                    i = 2 * i;
                else if (i < 2 * i + 1 && arr[i] < arr[2 * i + 1])
                    i = 2 * i + 1;
                else
                    break;
            }
            // 삽입할 위치를 찾았다.
            arr[i] = key;
            // 삽입할 위치를 찾았다.
            // 삽입할 위치를 찾았다.
        }
    }
}
    
```

5.7 이진탐색트리(binary search tree)

(1) 정의

- 임의의 리프노드를 삽입, 삭제, 탐색하는 데 사용된다.
- 최대 (이) 모든 원소는 리프노드에서 도달할 수 있다.
- (1) 왼쪽 서브트리에 있는 리프노드는 부모의 키보다 작거나 같다.
- (2) 오른쪽 서브트리에 있는 리프노드는 부모의 키보다 작거나 같다.
- (3) 삽입, 삭제, 또는 모든 서브트리에 이진탐색트리이다.

(2) 탐색

(3) 삽입

탐색 함수 (반복문 사용)

```

template <class Type>
void BinarySearchTree::insert(Type key)
{
    if (key < 0) return;
    if (key > 0)
    {
        // 삽입할 위치를 찾는다.
        int i = 1;
        while (i <= size)
        {
            if (i < 2 * i && arr[i] < arr[2 * i])
                i = 2 * i;
            else if (i < 2 * i + 1 && arr[i] < arr[2 * i + 1])
                i = 2 * i + 1;
            else
                break;
        }
        // 삽입할 위치를 찾았다.
        arr[i] = key;
        // 삽입할 위치를 찾았다.
        // 삽입할 위치를 찾았다.
    }
}
    
```

최대 힙의 삭제 함수

```

void deleteMaxHeapType()
{
    int deleteMaxHeapType()
    {
        int key;
        cout << "삭제할 값을 입력하세요: ";
        cin >> key;
        if (key > 0)
        {
            // 삭제할 경우 0이 아닌 값을 입력한 경우
            // 삭제할 위치를 찾는다.
            int i = 1;
            while (i <= size)
            {
                if (i < 2 * i && arr[i] < arr[2 * i])
                    i = 2 * i;
                else if (i < 2 * i + 1 && arr[i] < arr[2 * i + 1])
                    i = 2 * i + 1;
                else
                    break;
            }
            // 삭제할 위치를 찾았다.
            arr[i] = key;
            // 삭제할 위치를 찾았다.
            // 삭제할 위치를 찾았다.
        }
    }
}
    
```

신창우

학습 내용

제 5 장 트리(계속)

5.6 힙 트리와 이진 탐색 트리

이진 탐색 트리의 특징

- 삽입과 삭제의 경우 루트에서 새로운 노드를 삽입하거나 삭제하는 과정에서 노드의 위치를 변경해야 한다.
- 힙 트리는 완전 이진 트리와 유사한 구조를 가진다.
- 힙 트리는 완전 이진 트리와 유사한 구조를 가진다.
- 힙 트리는 완전 이진 트리와 유사한 구조를 가진다.
- 힙 트리는 완전 이진 트리와 유사한 구조를 가진다.
- 힙 트리는 완전 이진 트리와 유사한 구조를 가진다.
- 힙 트리는 완전 이진 트리와 유사한 구조를 가진다.

최대 힙과 최소 힙의 예

최대 힙: 루트 노드가 자식 노드보다 큰 값을 가진다.

최소 힙: 루트 노드가 자식 노드보다 작은 값을 가진다.

최대 힙의 표현과 연산

최대 힙의 표현

- 배열을 사용하여 표현할 수 있다.
- 삽입과 삭제의 경우 루트에서 새로운 노드를 삽입하거나 삭제하는 과정에서 노드의 위치를 변경해야 한다.
- 힙 트리는 완전 이진 트리와 유사한 구조를 가진다.
- 힙 트리는 완전 이진 트리와 유사한 구조를 가진다.
- 힙 트리는 완전 이진 트리와 유사한 구조를 가진다.
- 힙 트리는 완전 이진 트리와 유사한 구조를 가진다.
- 힙 트리는 완전 이진 트리와 유사한 구조를 가진다.

이진 탐색 트리(Binary Search Tree)

이진 탐색 트리의 특징

- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.

(2) 삽입

삽입의 과정

- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.

삽입 함수(이진 탐색 트리)

삽입 함수의 구현

```

int insert(Node* root, int data)
{
    if (root == NULL)
        return new Node(data);
    if (data < root->data)
        root->left = insert(root->left, data);
    else
        root->right = insert(root->right, data);
    return root;
}
    
```

(3) 삽입

삽입의 과정

- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.

삽입 함수

삽입 함수의 구현

```

Node* insert(Node* root, int data)
{
    if (root == NULL)
        return new Node(data);
    if (data < root->data)
        root->left = insert(root->left, data);
    else
        root->right = insert(root->right, data);
    return root;
}
    
```

(4) 삭제

삭제의 과정

- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.

삭제(두 개의 자식을 갖는 노드)

삭제의 과정

- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.

이진 탐색 트리의 높이

이진 탐색 트리의 높이

- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.

삭제(하나의 자식을 갖는 노드)

삭제의 과정

- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.

삭제(두 개의 자식을 갖는 노드)

삭제의 과정

- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.

레포트 #3

레포트 #3의 내용

- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.

이진 탐색 트리의 높이

이진 탐색 트리의 높이

- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.
- 루트 노드가 자식 노드보다 작은 값을 가진다.
- 루트 노드가 자식 노드보다 큰 값을 가진다.

학습 내용

삭제: $O(n)$ $O(1)$ $O(\log n)$
 삽입: $O(1)$ $O(n)$ $O(\log n)$ } - heap

5.6 힙

(1) 우선 순위 큐

- 삽입은 일반적인 큐와 같으나 삭제는 우선 순위가 가장 높은 것이 먼저 삭제된다.
- 최대(최소) 힙은 최대(최소) 우선 순위 큐를 구현하는데 사용된다.

(2) 최대 힙의 정의

- 최대 트리: 각 노드의 키값이 그 자식의 키값보다 작지 않은 트리
- 최소 트리: 각 노드의 키값이 그 자식의 키값보다 크지 않은 트리
- 최대 힙: 최대 트리이며 완전 이진 트리
- 최소 힙: 최소 트리이며 완전 이진 트리

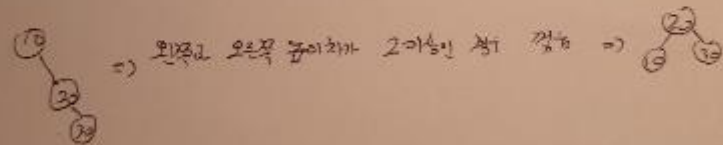
2

최대 힙의 표현과 연산

이진탐색트리의 높이

- worst case: height = n
- average case: height = $O(\log n)$
- 최악의 경우에도 height = $O(\log n)$ 이 되는 트리
 → 균형 탐색 트리 (balanced search tree)
- 탐색, 삽입, 삭제를 $O(\log n)$ 시간에 할 수 있다.
- 예) AVL 트리

10 20 30 40 50



조정민

학습 내용

(3) 최대 힙에서의 삽입

최대 힙의 삽입 함수

```

void insertHeap(TNode* root, TNode* newNode) {
    // 1. 새로운 노드를 힙의 끝에 추가
    newNode->parent = root;
    root->rchild = newNode;
    // 2. 최대 힙 성질 유지하기 위해 재배열
    while (newNode != root) {
        if (newNode->data < newNode->parent->data) {
            // 부모와 자식 교환
            swap(newNode, newNode->parent);
            newNode = newNode->parent;
        } else {
            break;
        }
    }
}
    
```

(4) 최대 힙에서의 삭제

최대 힙의 삭제 함수

```

void deleteHeap(TNode* root) {
    // 1. 루트 노드를 삭제하고, 마지막 노드를 루트로 대체
    TNode* lastNode = findLastNode(root);
    lastNode->parent = root->parent;
    root->parent->rchild = lastNode;
    // 2. 새로운 루트에 대해 최대 힙 성질 유지
    while (root != root->parent) {
        if (root->data < root->parent->data) {
            swap(root, root->parent);
            root = root->parent;
        } else {
            break;
        }
    }
}
    
```

삽입 함수

(4) 삭제

삭제(하나의 자식을 갖는 노드)

삭제(두 개의 자식을 갖는 노드)

5.7 이진 탐색 트리(binary search tree)

(2) 탐색

탐색 함수(반복문 사용)

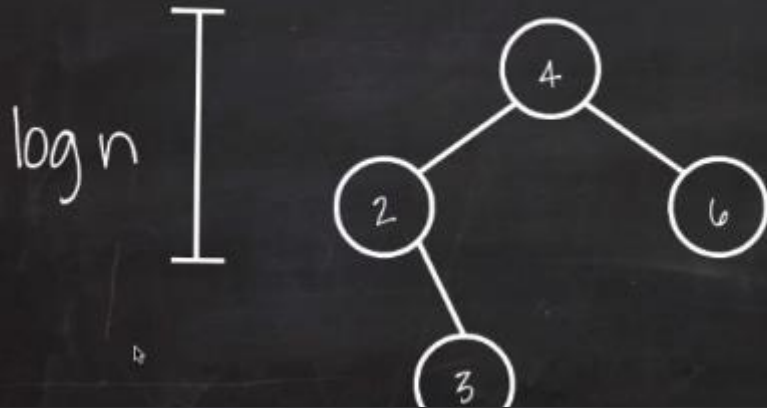
(3) 삽입

점진적 트리

허전진

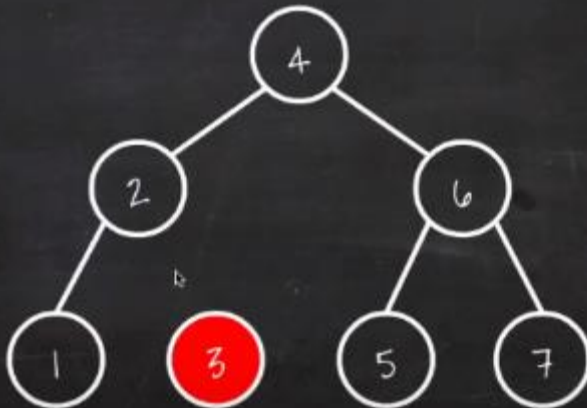
참고자료

BST Insertion



BST Deletion

1. no child
2. one child
3. 2 children



<https://www.youtube.com/watch?v=xxADG17SveY>

<https://www.youtube.com/watch?v=MbJsVpX-Y4Y>

6. 이진 탐색 트리 (6/12)

- 검색의 예
- 키 값이 22인 노드 검색

