2025-2 자료구조 기말고사 (20241519 조예성)

자료구조 두 형태: 연결된 구조, 배열 구조
- 삽입 삭제시 이동이 많은 구조: 배열 구조
- 구현이 복잡하지만 중간 삽입의 효율이 좋은 구조: 연결된 구조

연결된 구조에서 하나의 상자를 뜻하는 것: Node

주소를 저장하는 변수: Link

노드의 시작: Head Pointer

노드 종류 3개: 단순 연결 노드, 이중 연결 노드, 원형 연결 노드

노드 속 2개: DataField, LinkField

이중연결 구조는 바로 전 노드 찾기가 쉽다: **O**, X

비효율 적인 정렬 3종류: 선택, 삽입, 버블

정렬 순서 2개: 오름차순, 내림차순

복잡하지만 효율성이 좋은 정렬: 퀵, 힙, 병합, 기수 텀

메인 메모리에서만 정렬: 내부 정렬

외부 저장소 활용 정렬: 외부 정렬

같은 값들을 정렬했을 때 위치가 같은지 여부: 안전성

추가적인 메모리 공간 활용 X: 제자리 정렬

가장 작은 값을 선택하여 이동하는 알고리즘: 선택정렬
- 장점: 알고리즘 간단, 제자리 정렬, 이동 횟수 일정
- 단점: 안전성 X, 비효율적
- 복잡도: $O(n^2)$

순서대로 반복하며 더 큰/작은 값을 사이에 끼워 넣는 정렬: 삽입정렬
- 복잡도 최악: $O(n^2)$
- 복잡도 최선: $O(n)$

인접 레코드 비교 후 크기가 순서대로 아니면 교환하는 정렬: 버블 정렬
- 복잡도 최악: $O(n^2)$
- 복잡도 최선: $O(n)$

리스트 [8,3,4,9,7]을 버블 정렬을 할 때, pass1의 결과는? [3,4,8,7,9]

탐색은 레코드의 집합에서 원하는 (키)를 갖는 (레코드)를 찾는 과정

탐색 위한 구조 2개: Map, Dictionary

처음부터 하나씩 찾는 알고리즘: 순차 탐색
- 크기 제공이 필요하다: **O**, X
- 시간 복잡도: $O(n)$

반씩 찾는 알고리즘: 이진 탐색
- 사전 (정렬) 이 되어 있어야 함.
- 찾는 값이 없을 경우: low high 뒤집힘
- 분리한 조건: 삽입 삭제 많으면
- 시간복잡도: $O(log_2 n)$

탐색키 위치 예측하여 탐색: 보간 탐색
- 시간복잡도: $O(log_2 n)$
- 탐색하는 (값)과 (위치)가 (비례)한다는 과정
- 탐색 위치 = $low + (high - low) * k - \frac{A[low]}{A[high]-A[low]}$
- 위 식을 코드로: middle = int(low+(high-low) * (key-A[low] / (A[high]-A[low]))
- 정렬 필요: **O,** X

키값에 연산을 적용하여 저장 위치를 계산하는 것: 해싱
- 키값 저장 위치 계산 함수: 해시 함수
- 계산된 위치에 레코드가 저장된 테이블: 해시 테이블
- 이상적 시간복잡도: $O(1)$

슬롯 -> 버킷 -> 테이블

탐색키를 해시 함수를 통하여 처리하면: 해시 주소

버킷이 충분하지 않은 경우: 충돌
- 충돌의 원인인 키: 동의어
- 충돌 상황 다른 용어: 오버플로

좋은 해시함수의 조건: 충돌 최소, 주소 고르게 분포, 계산 빠름

가장 일반적인 방법으로 나머지 연산 사용: 제산 함수

비트별 XOR 같은 bool 연산 이용: 폴딩 함수
- 종류 2개: 이동 폴딩, 경계폴딩

임의의 위치의 k개의 비트를 이용하는 방법: 비트 추출 방법

숫자 특징을 알고 있을 때 활용 가능한 방법: 숫자 분석 방법

오버플로시 그 항목을 다른 위치주소에 저장: 개방 주소법

문자인 경우 하는 방법과 함수: ASCII 코드, ord()

하나의 위치에 여러 항목 저장: 체이닝

개방 조사법 대표적 방법: 선형조사법, 이차 조사법, 이중 해싱법

오버플로 발생시 다음 버킷 빈 슬롯 찾는 방법: 선형 조사법
- 한번 충돌 발생한 위치에 항목 집중 현상: 군집화 현상

선형 조사법의 종료 조건: 레코드 발견, 레코드 빈 버킷 발견, 모든 버킷 검사

삭제시 (별도 표기) 필요

하나의 버킷에 여러 레코드 저장할 수 있도록: 체이닝

해싱의 시간 복잡도: (최선 O(1), 최악 O(n))

계층적 관계 자료의 표현에 유용한 자료구조: 트리
- 트리에서 값의 명칭: 노드
- 자식 없는 노드: 단말 노드
- 어떠한 노드의 자식 수: 차수
- 트리의 각 층의 층수: 레벨
- 트리의 최대 층수: 트리의 높이
- 트리의 집합: 포리스트(forest)
- 노드 연결하는 선: 에지, 간선
- 동일한 부모 를 가진 노드: 형제 노드
- 어떤 노드에서 루트 노드까지의 경로상에 있는 모든 노드: 조상 노드
- 어떤 노드 하위 연결된 노드: 자손 노드

일반적 트리에서의 링크 개수: 노드의 차수 개수

모든 노드가 2개의 서브트리를 갖는 트리: 이진트리

트리 종류 크게 2개: 자유 트리, 루트를 가진 트리

각 레벨에 노드가 꽉 차있는 이진트리: 포화 이진 트리
- 개수: $2^n - 1$

마지막 레벨에서는 왼쪽부터 오른쪽으로 순서대로 차있는 트리: 완전 이진 트리

노드가 n개일 때 간선 개수: n-1

높이가 n인 이진트리의 최소, 최대 노드 수: n ~ 2^n-1

n개의 노드를 가진 이진트리의 최소, 최대 높이: $[\log_2(n+1)] \geq n$

배열로 구연한 이진트리에서 노도 i의 부모 인덱스: i/2

노드 i의 왼쪽 자식 노드 인덱스: 2i
- 오른쪽 자식 노드 인덱스: 2i+1

연결된 구조로 표현된 이진트리에 화살표 없는 링크의 값: None

트리 속 모든 노드 방문: 순회

순회 종류 3개: 전위순회, 중위 순회, 후위 순회
- 루트노드가 V이고 각 자식이 L,R 일때 전위: VLR, 중위: LVR, 후위순회: LRV

각 노드를 레벨 순으로 검사하는 방법: 레벨 순회
- 활용 자료구조: 큐

노드 개수 구하는 방법: 왼쪽 서브트리 노드 수 + 오른쪽 서브트리 노드 수 + 1
- 어떠한 순회: 후위순회

이진트리 응용: 모스코드

가장 큰(또는 작은)값을 빠르게 찾아내도록 만들어진 자료구조: 힙
- 힙 조건: 완전이진트리
- 힙 종류: 최대 힙, 최소 힙
- 힙 삽입 명칭: 시프트 업, 업힙
  - 시간 복잡도: $O(\log_2 n)$
- 힙 삽입 명칭: 시프트 다운, 다운 힙
  - 시간 복잡도: $O(\log_2 n)$

힙 연산의 최악의 상황: 최소가 Top에 삽입되었을 때

힙 저장 효과적 자료구조: 배열

단일 연결 리스트에서 리스트의 Head(시작)에 새로운 노드를 삽입하는 연산의 시간 복잡도는? O(1)

단일 연결 리스트에서 k번째 인덱스에 있는 요소를 검색(Access)하는 연산의 최악 시간 복잡도는? O(n)

Tail 포인터가 없는 단일 연결 리스트에서 리스트의 Tail(끝) 노드를 삭제하는 연산의 최악 시간

복잡도는? O(n)

단일 연결 리스트에서 특정 값을 가진 노드를 찾은 후 그 뒤에 새 노드를 삽입하는 연산의 전체 시간 복잡도는? O(n)

이진탐색트리의 삽입, 삭제, 탐색 연산의 시간 복잡도는: O(log n)

이진탐색트리의 왼쪽은 루트키보다: 작다

이진탐색트리의 오른쪽은 루트키보다: 크다

이진탐색트리는 완전이진트리여야 한다: O, **X**

키 이용시 시간복잡도: O(log n)

값 이용시 시간복잡도: O(n)

이진 탐색 트리의 삽입위치: 루트노드

탐색 과정: 탐색 시도 -> 없으면 없는 위치 삽입
- 값이 있다면: 삽입 x

이진탐색트리는 중복을 허용한다: O, **X**

이진탐색트리의 삭제 조건 3개: 노드가 단말 노드, 노드에 자식이 한 개, 노드의 자식이 2개

이진탐색트리의 연산: search_bst, search_value_bst, search_max_bst, search_min_bst, insert_bst delete_bst