정렬이란?

데이터를 순서대로 재배열하는것

가장 기본적이고 중요한 알고리즘

비교할수 있는 모든 속성들을 정렬의 기준이 될수 있음

출력은 입력을 재배열하여 만든순열

오름차순 or 내림차순 등등

동일한 문제에대해 다양한 알고리즘이 고안될수 있음을 이해하고, 각 알고리즘간

성능비교를 통해 알고리즘 성능 분석에 대해서도 이해할 수 있음

$h(k) = k \bmod M$ 이렇게도

NO.

year month day ( )

가장기본적인 해시함수

재산관리 ₿

DevOps

11. 정렬과 탐색 (해싱 까지)

정렬 ┌ 선택
　　 ├ 삽입
　　 └ Bubble 버블

탐색 ┌ 순차
　　 ├ 정렬
　　 └ 해싱 (겟좀) O(1)

레코드

- 정렬시켜야 될 상태
  → 여러가지 필드로 이루어짐
  → 정렬키: 정렬의 기준이 되는 필드

레코드 필드 ... 정렬키 ... 필드

→ 정렬: 레코드들을 키의 순서로 재배열하는것

정렬 알고리즘의 종류

- 정렬장소에 따른 분류
  → 내부정렬: 모든 데이터가 메인 메모리에 올라와 있는
  → 외부정렬: 보조기억장치에 여럿의 데이터가 있고, 일부데이터만 메모리에 올려 정렬하는 방법

대용량 자료를 정렬하기 위해 사용

- 단순하지만 비효율적인 방법의 정렬
  → 삽입, 선택, 버블 정렬 등
- 복잡하지만 효율적인 방법의 정렬
  → 퀵, 힙, 병합, 기수정렬 등
- 키값의 비교를 사용하지 않는 정렬: 기수, 카운팅 정렬 등
- 안정성에 따른 정렬
  → 안정성: 입력데이터에 동일한 값을 갖는 레코드가 여러개 존재할 때, 정렬후에도 상대적 위치가 바뀌지 않는것
  → 삽입, 버블, 병합정렬 등

선택정렬 ┐ 기본적인 알고리즘
삽입정렬 ├ 오름차 리스트에서 가장 작은 숫자를 선택하여 왼쪽 리스트의 맨뒤로 이동하는 개념 반복
버블정렬 ┘

선택정렬

→ 오른쪽 리스트에서 가장 작은 숫자를 선택해 왼쪽리스트의 맨 뒤로 이동하는 작업을 반복

1 주어진 데이터중, 최소값을 찾음

2 해당 최소값을 데이터 맨 앞에 위치한 값과 교체함

3 맨 앞의 위치를 뺀 나머지 데이터를 동일한 방법으로 반복함

복잡도 $n(n-1)/2 = O(n^2)$

- 삽입정렬

- 정렬되어 있는 부분에 새로운 레코드를 올바른 위치에 삽입하는 과정 반복

많은 이동 필요
(↔ 선택정렬 방법 정렬되어 있으면 매우 효율적)

특징

- 삽입 정렬은 두 번째 인덱스 부터시작
- 해당 인덱스(key값) 앞에 있는 데이터(B)부터 비교해서 key값이 더 작으면 B값을 뒤(오른쪽)로 이동
- 더 큰 key값이 더 큰 데이터를 만날 때까지 반복, 큰데이터를 만난 위치바로 뒤에 key값을 이동

버블정렬

- 인접한 두 수를 비교하여 큰 수를 뒤로 보내는 간단한 정렬 알고리즘

비교횟수 (최상, 평균, 최악)

집합 자료구조 특징

- 집합 원소의 중복 허용하지않고 원소들 사이에 순서가 없음
- 이번 장에서는 정렬 등 사용하지 않음
- 집합 원소들이 정렬되어있으면 집합의 비교나 합집합, 교집합, 차집합등은 훨씬 효율적

최악
역순으로 정렬된 경우
→ 이동횟수 = 3× 비교횟수

<삽입연산>

→ 중복체크 없이 마지막 위치에 중복허용 연산실시

→ 삽입할 위치를 먼저 찾아냄 → 정렬된 상태 유지

이미 정렬된 경우
=> 이동횟수 = 0
평균의 경우 $O(n^2)$

(비교연산)

- 두 집합의 원소의 개수가 같아야 같은 집합이됨
- 집합이 정렬되어 있으므로 순서대로 같은 원소를 가져와야함 → 가장 작은 원소부터 하나씩 끝까지 서로 같아야 같은 집합

레코드의 이동과 같은 연산은 비교연산보다 더 많은 시간이소요

합집합 연산 union

가장작은 원소를 부터 비교하여 더 작은 원소를 새로운 집합에 넣고 그 집합의 인덱스를 증가시킴

만약 두 집합의 현재원소가 같으면 하나만을 넣음, 인덱스는 모두 증가시킴

한쪽 집합이 모두 처리되면 나머지 한쪽에 남은 모든 원소를 순서대로 새 집합에 넣음

시간복잡도 O(n)

탐색

- 테이블에서 원하는 탐색키를 가진 레코드를 찾는 방법

맵 아 딕셔너리

탐색을 위한 자료구조이며, 맵은 키의 중복허용X, 딕셔너리는 키의 중복허용

엔트리, 또는 키를 가진 레코드의 집합

엔트리

키와 쌍으로 이루어짐

키 : 영어 단어와 같은 레코드를 구별할 수 있는 탐색키

값 : 단어의 의미와 같이 탐색키와 관련된 값

기본적인 탐색 알고리즘

순차탐색, 이진탐색, 보간탐색

순차탐색 (데이터가 담겨있는 리스트를 앞에서 부터 하나씩 비교하여 원하는 데이터를 찾는 방법
정렬되지않은 배열에 적용가능

이진탐색 (정렬된 배열의 탐색에 적합
배열의 중앙에 있는 값을 조사하여 찾고자하는 항목의 왼쪽 또는 오른쪽부분에 배열원소가 있는지 알아내어 탐색의 범위를 줄여나감

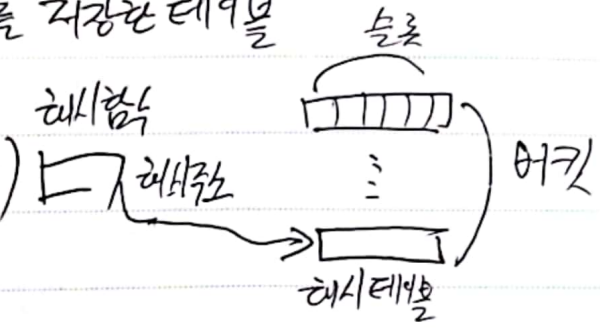
보간탐색 - 탐색키가 존재할 위치를 예측하여 탐색
- 리스트를 불균등하게 분할하여 탐색

〈해싱〉
키 값을 해시함수 라는 수식에 대입시켜 계산한 후 나온 결과를 주소로 사용하여 값에 접근하게하는 방법

〈해시함수〉
키 값을 값이 저장되는 주소값으로 바꾸기 위한 식

〈해시 테이블〉
해시함수에 의해 계산된 위치에 레코드를 저장한 테이블

〈버킷과 슬롯〉
◦ 해시테이블은 M개의 버킷으로 구성
하나의 버킷은 S개의 슬롯을 가짐
◦ 하나의 슬롯에는 하나의 레코드저장



특징
단 한번의 접근으로 찾을수 있음
자료의 삽입과 탐색을 탐색키를 인덱스로 생각하고 그 위치에 저장, 읽기
적절한 해싱 함수 구현이 중요

해싱 충돌의 문제발생(주소의 중복)
∨ 서로 다른 키가 해시 함수에 의해 같은 주소로 계산되는 상황
이것을 충돌이라고 하고, 충돌을 일으키는 키들을 동의어라고 함

문제
오버플로 : 버킷에 여러개의 슬롯을 더 이상 저장 할수없는, 넘쳐나는 현상
이상적인 해싱 : 충돌이 절대 일어나지 않는 해싱
⊖ 해시 테이블의 크기를 충분히 키워야 됨
⊖ 충돌과 오버플로 조정을 필요로 함
⊛ 선형조사에 의한 오버플로 처리
↳ 해싱함수로 계산된 버킷에 빈슬롯이 없으면 그 다음 버킷에서 빈슬롯을 찾는 방식

주소 + 0 비어 있는 공간을 찾을것
선형조사 법에는 삽입, 탐색, 삭제연산 존재
(탐색연산)
탐색키가 입력되면 해시 주소 계산, 해당 주소에 같은 키의 레코드가 존재하면 탐색성공
같은 키의 레코드가 없다면 계속 다음 버킷검사
(삭제연산)
탐색 연산을 통해 해당값을 찾고 삭제
선형 조사법에서 항목이 삭제되면 탐색이 불가능해질 수있음
(군집화)
충돌 발생 위치에서 항목들이 집중되는 현상
선형조사법은 간단하지만 오버플로 발생 → 탐색효율저하
완화 방법
이차조사법 : 충돌이 발생하면 다음에 조사할 위치를 결정
○군집화 현상을 완화시킬 수 있으나 2차 군집화 현상초래
이중해싱법 : 충돌이 발생했을때 원래 해시함수와 다른 별개의 해시함수를 이용하는 방법
항목들을 해시테이블에 보다 균일하게 분포시킬 수 있으므로 효과적인 방법

충돌이 발생하면 해시함수값에
1 또는 i^2 을더함

해시함수 값이 같더라도 탐색키가 다르면
서로 다른 순서를 갖게됨 → 2차군집화 회피

체이닝에 의한 오버플로 처리
○체이닝 : 연결리스트로 버킷에 여러개의 레코드를 저장할수 있도록하는 방법
체이닝은 연결리스트로 구현가능
해시함수
좋은해시 함수의조건
- 충돌이 적어야함
- 함수값이 테이블의 주소 영역 안에서 고르게 분포되어야한다.
- 계산이 빨라야함

(제산함수, 폴딩함수, 숫자분석 방법, 비트추출함수, 탐색키가 문자열인 경우)

제산함수 ─ $h(k) = k \% M$
└ M은 소수를 선택

폴딩함수 ┌ 레코드 키값을 여러 부분으로 나눈후 각 부분의 값을 더하거나 XOR한
└ 값을 주소로 삼는 방식

이동폴딩함수 ─ 각 부분의 값을 계산하기 위해 마지막을 제외한 모든 부분을 이동시켜 최하위 비트(LSB)가 마지막 부분(더한)의 자리와 일치하도록 오른쪽 끝을 맞추어 버킷주소(인덱스)로 사용하는 방법

경계폴딩함수 ─ 원래의 키(key)값을 나눈후 나누어진 각 부분의 경계선을 종이
접듯이 접어 역으로 정렬한 다음 같은 자리에 위치한 수를 더한 값을
└ 버킷주소(인덱스)로 사용함

중간제곱함수 ─ 키(key)값을 제곱한 후 결과의 중간부분에 있는 몇 비트만을 선택
└ 하여 버킷주소(인덱스)로 사용

비트추출방법 ┌ 해시 테이블의 크기가 2^k일때 키값을 이진비트로 놓고 임의의 위치에
있는 비트들을 추출하여 주소로 사용하는 방법
충돌이 발생할 가능성이 높으므로 테이블의 일부에 주소가 편중되지 않도록
└ 키 값들의 비트들을 미리 분석하여 사용

숫자분석방법 ┌ 키값을 이루고 있는 각 자릿수의 분포를 분석하여 해시주소로 사용하는 방법
파일의 키값이 이미 알려진 정적파일인 경우에 유용하며, 삽입과 삭제가
└ 빈번히 발생하는 경우에는 비효율적

탐색키가 문자열인경우

(각 문자에 정수를 대응시켜 주소를 삼는 방식
각 문자의 아스키코드나 유니코드 값을 이용 a~z, → 1~26

성능비교 해싱의 적재밀도

해싱의 성능을 분석하기 위해 해시 테이블이 얼마나 채워져 있는가를 나타냄
적재밀도는 충돌해결 방법에 따라 달라짐

$$\alpha = \frac{\text{저장된 항목의 개수}}{\text{해싱 테이블 버킷의 개수}} = \frac{n}{M}$$

성능비교

방법		탐색	삽입	삭제
순차탐색		$O(n)$	$O(1)$	$O(n)$
이진 탐색		$O(\log_2 n)$	$O(n)$	$O(n)$
이진탐색	균형트리	$O(\log_2 n)$	$O(\log_2 n)$	$O(\log_2 n)$
	경사트리	$O(n)$	$O(n)$	$O(n)$
해싱	최선	$O(1)$	$O(1)$	$O(1)$
	최악	$O(n)$	$O(n)$	$O(n)$