7. 정렬과 탐색

정렬

가장 기본적이고 중요한 알고리즘
비교할 수있는 모든 속성들은 정렬의 기준이될
효율은 알고리즘을 재배열하여 만든 순열
오름차순과 내림차순 존재
동일한 목적에 대해 다양한 알고리즘이 고안될수있음을 이해하고
알고리즘간 성능 비교를통해 알고리즘성능 분석을 이해할수있음

레코드

- 정렬되어야 할 대상
  - ∨ 여러개의 필드로 이루어짐
  - ∨ 정렬 키 : 정렬의 기준이되는 필드
- 정렬 알고리즘의 종류

내부 정렬
외부정렬
삽입, 선택 버블정렬
퀵, 힙, 병합, 기수정렬등
기수 카운팅 정렬
삽입, 버블병합 정렬 등

선택 정렬

· 모든 리스트에서 가장 작은 숫자를 선택

```
def selection_sort(A):
    n= len(A)
    for i in range(n-1):
        least = i;
        for j in range(i+1,n):
            if(A[j] < A[least]):
                least = j
        A[i], A[least] = A[least], A[i]
```

1, 주어진 데이터 중 최소값을 찾음
2, 해당 최소값을 데이터 맨 앞에 위치한 값과 교체한
3, 맨 앞의 위치를 뺀 나머지 데이터를 동일한 방법으로 반복함

시간복잡도 $O(n^2)$

삽입정렬

· 정렬되어 있는 부분에 새로운 레코드를 올바른 위치에 삽입하는 과정반복

정렬되있는 부분에 새로운 레코드를 올바른 위치에 삽입하는 과정반복
1, 삽입정렬은 두번째 인덱스부터 시작
2, 해당 인덱스 (값) 앞에 있는 데이터(B)부터 비교해서 key값이 더 작으면 B값을 뒤 인덱스로 복사
3, 이를 key값이 더 큰 데이터를 만날때까지 반복, 그리고 큰데이터를 만난 위치 바로뒤에 key값을 이동

이미정렬 / 역순정렬

최선의 경우 $O(n)$, 최악의 경우 On^2

특징

✓ 많은 이동필요 ⇒ 레코드가 큰 경우불리
✓ 안정된 정렬 방법
✓ 대부분 정렬되어 있으면 매우 효율적

산함수 ☆☆☆ 가장일반적인 해쉬함수 $h(k) = k \bmod M$

버블정렬
인접한 두수를 비교하여 두수를 두고보내는 간단한 정렬 알고리즘

- 비교횟수 모두일때 $O(n^2)$
  이동횟수
  ✓ 역순으로 정렬된경우 (최악)
  ✓ 이미정렬된 경우 (최선의 경우)
  ✓ 평균의 경우 : $O(n^2)$
- 레코드의 이동과다
  ✓ 이동연산≫비교연산 보다 더 많은 시간이 소요

11.3 정렬응용 : 집합 다시보기

집합 자료구조 수정
- 집합 : 원소의 중복 허용하지 않고 원소들 사이에 순서가 없음
- 이번 장에서는 정렬을 이용해 구현
- 집합원소들이 정렬되어 있으면 집합의 비교나 합집합, 교집합, 차집합등을 훨씬 효율적

- 삽입연산
  - 중복검사 먼저실행
  - 삽입위치를 먼저 정함 → 정렬된 상태유지

- 비교연산
  - 두집합의 원소의 개수가 같아야 같은 집합이 됨
  - 집합이 정렬되어 있으므로 순서대로 같은원소를 가져야함 → 가장작은 원소부터 하나씩 끝까지 전부 같아야 같은 집합

합집합연산: union
· 가장작은 원소들로부터 비교하여 더작은 원소를 새로운 집합에 넣고
그 집합의 인덱스를 증가시킴
· 만약 두 집합이 현재 원소가 같으면 하나만을 넣음, 인덱스를 모두 증가시킴
·· 한쪽 집합이 모두 처리되면 나머지 집합의 남은원소들을 순서대로
· 새 집합에 넣음
· 시간복잡도 O(n)
· 합집합과 교집합, 차집합 같은 방법또 구현가능

~~value 개의~~

7. 탐색과 맵 구조

탐색
· 테이블에서 원하는 탐색키를 가진 레코드를 찾는 작업
맵또는 딕셔너리
· 탐색을 위한 자료구조이며 맵은 키의 중복 X, 딕셔너리는 키중복허용
· 엔트리, 또는 키를 가진 레코드의 집합
엔트리
· 키와 쌍으로 이루어짐
키 : 영어단어와 같은 레코드를 구분할수있는 탐색키
값 : 단어의 의미와 같이 탐색키와 관련된 값

맵 ADT
데이터 : 키를 가진 레코드의 집합

search(key) : 탐색키 key를 가진 레코드를 찾아 반환
insert(entry) : 주어진 entry를 맵에 삽입
delete(key) : 탐색키 key를 가진 레코드를 찾아 삭제한다

9.5 간단한 탐색 알고리즘

순차탐색
- 데이터가 담겨있는 리스트를 앞에서부터 하나씩 비교해서 원하는 데이터를 찾는 방법
- 정렬되지 않은 배열에 적용가능
  - ✓ 가장간단하고 직접적인 탐색 방법
  - ✓ 평균 비교횟수: (n+1)/2번비교

이진탐색
- 정렬된 배열의 탐색에 적합

✓ 배열의 중앙에 있는 값을 조사하여 찾고자 하는 항목이 왼쪽 또는 오른쪽 부분 배열에 있는지를 알아내어 탐색의 범위를 반으로 줄여가며 탐색 진행
- ✓ 사전에서 단어 찾기
- ✓ 이진탐색 : 단지 30번의 비교필요
- ✓ 순차탐색 : 평균5억번의 비교필요

보간탐색
탐색키가 존재 할 위치를 예측하여 탐색
리스트값을 균등하게 분할하여 탐색

해싱
· 키값을 해시함수라는 수식에 대입시켜 계산한 후 나온 결과를 주소로 사용하여 바로 값에 접근하게 하는 방법

해시함수
키값을 값이 저장되는 주소값으로 바꾸기위한 수식

해시 테이블
해시함수에 의해 계산된 위치에 레코드를 저장한 테이블

버킷과 슬롯
해시 테이블은 M개의 버킷으로 구성
하나의 버킷은 S개의 슬롯을 가질
하나의 슬롯에는 하나의 레코드가 저장

특징

· 단 한번의 접근으로 찾을 수 있음
· 자료의 삽입과 탐색은 탐색키를 인덱스로 생각하고 그위치에 저장하면?
· 적절한 해싱함수 구현해야함
· 해싱 충돌의 문제(발생 (주소의 중복)
  ✓ 서로다른 키가 해시 함수에 의해 같은 주소로 계산되는 상황
  ✓ 이것을 충돌이라 함, 충돌을 없애는 → 들을 동의어라 함
파이썬에는 해쉬를 별도 구현하지 않고 딕셔너리타입을 사용하여 구현

해싱 충돌의 문제1

· 오버플로 : 버킷에 여러개의 슬롯을 더 이상 저장할수 없는, 넘쳐나는 현
• 이상적인 해싱 : 충돌 절대 일어나지 않는 해싱
V 해시 테이블의 크기를 충분히 크게 함
✓ 충돌과 오버플로 조정을 목표로 함

선형조사의 의한 오버플로 처리
· 선형조사법 : 해싱 함수로 계산된 버킷에 빈슬롯이 없으면 그다음
버킷에서 빈슬롯이 있는지를 찾는 방법
조사 : 비어 있는 공간을 찾는것
· 선형조사법에는 삽입, 탐색, 삭제 연산존재

군집화
충돌 발생위치에서 항목들이 집중
선형조사법은 간단하지만 오버플로 발생 구탐색효율저하

완화방법
이차조사법, 이중해싱법