강의 8: 해싱 I

강의 개요

- 파이썬에서의 딕셔너리
- 동기
- 프리해싱
- 해싱
- 체이닝
- 간단 균일 해싱
- 좋은 해시 함수들

딕셔너리 문제

추상 자료형 (ADT) — 다음과 같은 연산으로 각 키에 따른 항목 집합을 유지

- insert(item): 집합에 항목 추가
- delete(item): 집합에서 항목 제거
- search(key): 존재할 경우 키와 항목 반환

항목은 각자의 키를 가지고 있다고 가정 (또는 새 것을 삽입할 때 옛 것을 덮어 씌움)

균형 이진 탐색 트리는 연산 하나 당 O(lg n) 시간 소요 (그 다음 큰 수 탐색과 같은 정밀하지 않은 탐색도 가능)

목표: 연산 하나 당 $O(1)$ 시간

파이썬의 딕셔너리:

항목은 (키, 값) 쌍 예. d = {‘algorithms’: 5, ‘cool’: 42}

d.items()	→	[(‘algorithms’, 5),(‘cool’,5)]
d[‘cool’]	→	42
d[42]	→	KeyError
‘cool’ in d	→	True
42 in d	→	False

파이썬 집합은 항목이 곧 키가 되는 진짜 dict(사전)에 가까움(값 없음)

동기

딕셔너리는 컴퓨터 과학 분야에서 가장 많이 쓰이는 자료구조이다.

- 현대 프로그래밍 언어에 다 내장되어 있다. (Python, Perl, Ruby, JavaScript, Java, C++, C#, ...)
- 예) 문서 거리 문제 : 단어 세기 & 내적
- 데이터베이스 구현: (버클리DB의 DB_HASH)
  - 영어 단어 → 정의 (말 그대로의 사전)
  - 영어 단어: 철자 교정
  - 단어 → 그 단어를 포함한 모든 웹페이지
  - 사용자 이름 → 계정 객체
- 컴파일러 & 인터프리터: 이름 → 변수
- 네트워크 라우터: IP 주소 → 랜선
- 네트워크 서버: 포트 번호 → 소켓/앱
- 가상 메모리: 가상 주소 → 물리적 주소

그 외에도 간접적으로 해싱을 쓰는 경우:

- 부분 문자열 탐색 (grep, Google) [L9]
- 염기 공통성 (DNA) [PS4]
- 파일 또는 디렉토리 동기화 (rsync)
- 암호학: 파일 변환 & 식별 [L10]

딕셔너리 문제를 어떻게 풀까요?

간단한 접근법: 직접 접근 테이블

인덱스가 키가 되며, 항목은 배열에 저장되어야 함. (임의 접근)

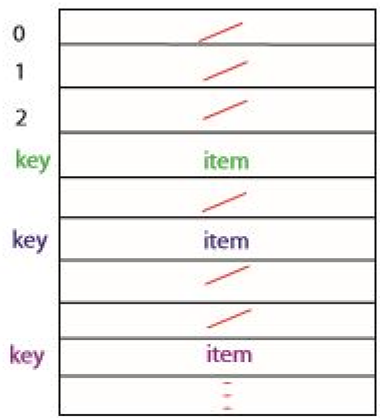

그림 1: 직접-접근 테이블

direct-access-table

문제:

1. 키는 반드시 음이 아닌 정수여야 한다. (또는 두 개의 배열, 정수를 사용하면 됨)
2. 넓은 키의 범위 ⇒ 큰 공간 필요 — 예. 2^{256} 인 키는 좋지 않다.

2 해결법:

1번에 대한 해결법: 키를 정수로 "프리해시".

- 키가 유한하기 때문에 이론적으로 가능 ⇒ 키의 집합을 셀 수 있다.
- 파이썬: 해시(객체) (사실 "해시"는 적절한 명칭이 아님. "프리 해시"가 정확한 표현) 객체는 숫자, 문자열, 튜플 등 또는 __hash__가 구현된 객체가 될 수 있다. (기본값 = id = 메모리 주소)
- 정리에 의해, x = y ⇔ hash(x) = hash(y)
- 파이썬은 실용성을 위해 경험적인 방법을 일부 적용했다. : 예로, hash('\0B ') = 64 = hash('\0\0C')
- 테이블에서 객체의 키는 바뀌어서는 안된다. (바뀌면 더 이상 찾을 수 없다.)
- 리스트와 같은 가변 객체는 안된다.

2번에 대한 해결법: 해싱 (프랑스어 'hache' 에서 유래 = hatchet, & 옛 독일어 'happja' = scythe)

- 모든 키(예를 들면, 정수)가 있는 전체집합 U를 합리적인 크기 m인 테이블로 줄이는 것

- <u>아이디어</u>: $m \approx n$ = 딕셔너리에 저장된 키의 개수
- <u>해시 함수</u> h: $U \rightarrow \{0,1,\ldots,m-1\}$
- <u>hash function</u> h: $U \rightarrow \{0,1,\ldots,m-1\}$

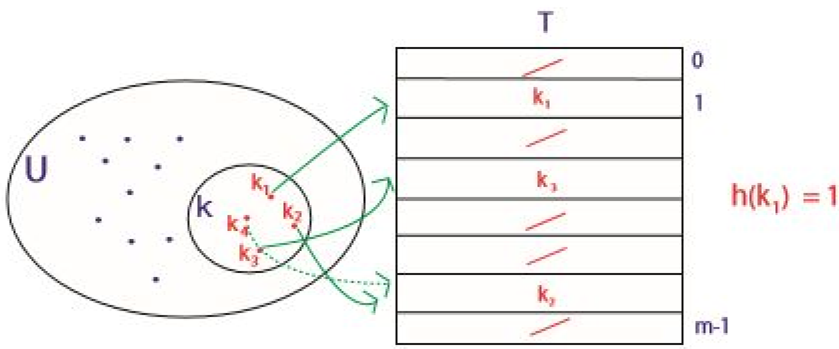

그림 2: 키를 테이블에 대응시키는 과정

- 키 두 개 $k_i, k_j \in K$ <u>충돌</u> if $h(k_i) = h(k_j)$

<u>충돌을</u> 어떻게 다뤄야 할까요?

두 가지 방법 존재

1. 체이닝: 오늘
2. 개방 주소법: 열 번째 강의

체이닝

테이블의 각 슬롯마다 충돌하는 원소들의 연결 리스트를 만듬

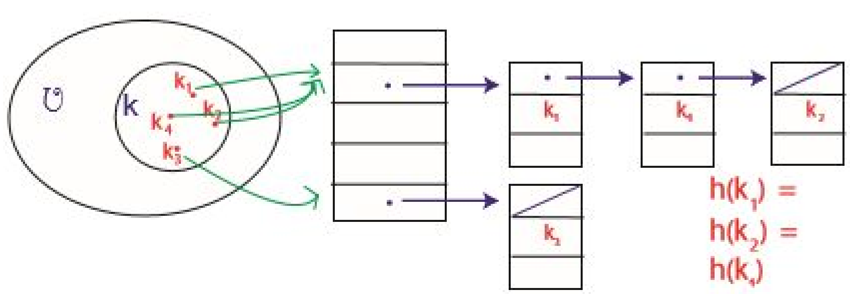

그림 3: 해시 테이블에서의 체이닝

- 탐색시 리스트 전체인 T[h(key)] 를 훑어야 한다.
- 최악의 경우: 모든 n개의 키가 같은 슬롯에 있는 경우 ⇒ 연산 당 $\Theta(n)$

간단 균일 해싱:

가정 (일종의 속임수): 모든 키는 테이블의 아무 슬롯에나 해시될 가능성을 동등하게 가지고 있다. 각 키의 해시는 독립적으로 다른 키가 해시되었는지와 상관없다.

let n = 테이블에 저장된 키의 개수

m = 테이블의 슬롯 개수

<u>적재율 α</u> = n/m = 슬롯 당 예상되는 키의 개수 = 체인의 예상되는 길이

load factor = $n \times \frac{1}{m}$ = 개수 × 확률

성능

탐색의 예상되는 실행 시간이 $\Theta(1+\alpha)$라는 의미한다. — 1은 해시 함수를 적용하고 슬롯에 임의 접근하는 것을 의미하는 반면 α는 리스트를 탐색하는 것을 의미한다. $\alpha = O(1)$이면 $O(1)$이 된다. 즉, $m = \Omega(n)$.

해시 함수

위와 같은 성능을 달성하는 방법으로 3가지가 있다. :

나머지 연산법:

$$h(k) = k \bmod m$$

m이 2의 제곱이나 10의 제곱에 가까이 있지 않은 소수일 경우 실용적이다. (이때 낮은 자릿수나 비트에 따라 다르다.)

소수를 찾는 것이 편리하지 않고, 나머지 연산법은 느리다.

곱셈 방법:

$$h(k) = [(a \cdot k) \bmod 2\ w] \gg (w-r)$$

a는 임의값, k는 w 비트, 그리고 $m = 2^r$ 이다.

a가 홀수 & $2^{w-1} < a < 2^w$ & a가 2^{w-1} 또는 2^w 에 너무 가깝지 않아야 이 방법이 실용적이다.

곱셈과 비트 추출은 나눗셈보다 빠르다.

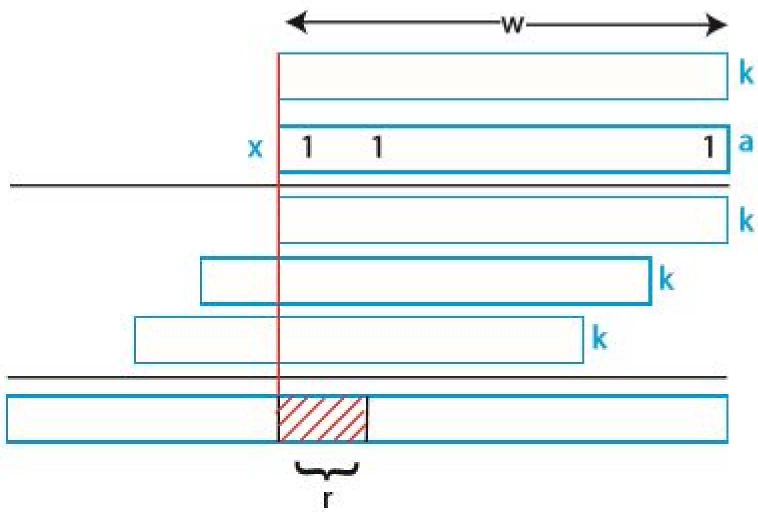

그림 4: 곱셈 방법

유니버설 해싱

[6.046; CLRS 11.3.3]

예: h(k) = [(ak+b) mod p] mod m, a 와 b는 임의값 $\in \{0,1,\ldots p-1\}$, 그리고 p는 큰 소수 (> |U|).

이 말인 즉, 최악의 경우에는 키가 $k1 \neq k2$ 이다. (h의 선택 a, b에 의해):

$$Pr_{a,b}\{\text{event } X_{k_1k_2}\} = Pr_{a,b}\{h(k_1) = h(k_2)\} = \frac{1}{m}$$

이 보조 정리는 여기에 증명되어 있지 않다.

이것은 다음을 의미한다. :

$$
\begin{aligned}
E_{a,b}[\#\text{ collisions with } k_1] &= E[\sum_{k_2} X_{k_1 k_2}] \\
&= \sum_{k_2} E[X_{k_1 k_2}] \\
&= \sum_{k_2} \underbrace{Pr\{X_{k_1 k_2} = 1\}}_{\frac{1}{m}} \\
&= \frac{n}{m} = \alpha
\end{aligned}
$$

위에 있는 것만큼 좋다!