

강의 9: 해싱II

강의 개요

- 테이블 더블링
- 분할상환
- Karp-Rabin 문자열 찾기
- 툴링 해시

복습:

체이닝을 이용한 해싱:

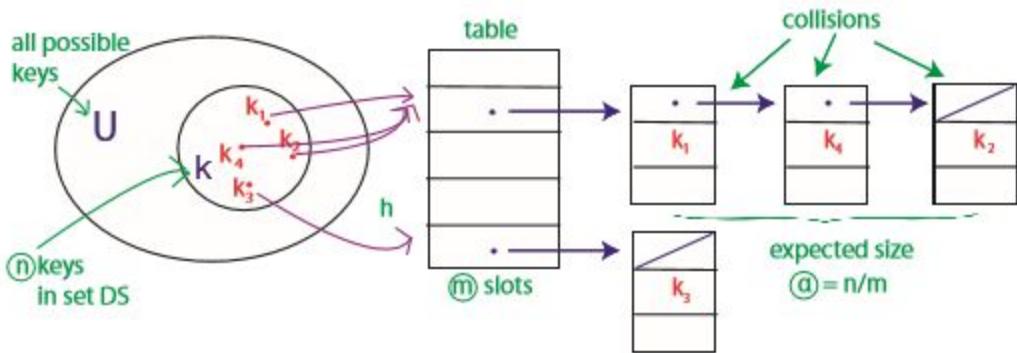


그림 1: 체이닝을 이용한 해싱

기대 비용 (데이터 삽입/삭제/찾기): $\Theta(1 + \alpha)$, 단순 균일 해싱이나 유니버설 해싱이고 해시 함수 h 의 실행시간이 $O(1)$ 라고 가정한다.

나누기 방법:

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

m 은 이상적으로는 소수이다.

곱하기 방법:

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

a 는 2^{w-1} 과 2^w 사이의 무작위 훈수이고, k 는 w 비트로 주어졌으며, $m =$ 해시 표의 크기 = 2^r 이다.

해시 표의 크기는 얼마나 커야 하나?

- ~~항상 $m = \Theta(n)$ 이길 원한다.~~
- 해시 표를 만들 때는 n 이 얼마나 큰지 모른다.
- m 이 너무 작으면 느리고, m 이 너무 크면 낭비가 된다.

발상:

작은 상수로 시작해 필요에 따라 늘리거나 줄인다.

다시 해싱하기:

해시 테이블을 늘리거나 줄이기 위해서는 해시 함수의 (m, r) 도 바꿔어야 한다.

=> 처음부터 다시 해시 테이블을 만들어야 함.

for 항목 in 기존 표: -> for 각 슬롯, for 항목 in 슬롯

새로운 해시 테이블로 삽입

=> $m = \Theta(n)$ 이면, $\Theta(n + m) = \Theta(n)$ 이다.

늘리는 데 소요되는 시간은?

n 이 m 이 되면,

- m을 1만 늘린다? ~~X~~
 - ⇒ 매번 새롭게 늘려야 한다.
 - ⇒ n 번의 삽입이 $\Theta(1 + 2 + \dots + n) = \Theta(n^2)$ 만큼 걸린다.
- m을 2배로 늘린다? $m = \Theta(n)$ still ($r + = 1$)
 - ⇒ 2^i 번마다 새롭게 해시 표를 만들어준다.
 - ⇒ n 번의 삽입이 $\Theta(1 + 2 + 4 + 8 + \dots + n) = \Theta(n)$ 만큼 걸린다. n 은 바로 다음 2의 제곱일 때
 - 몇몇의 삽입은 선형시간이 걸리지만, 평균으로는 $\Theta(1)$ 만큼 걸린다.

분할상환 분석

분할상환

데이터 구조에서 자주 쓰는 기법이다. \$1500 월세를 하루에 \$50씩 나눠내는 것과 비슷하다.

- k 번의 연산 시간 $\leq k \cdot T(n)$ 이면, 연산이 분할상환 실행시간 $T(n)$ 이 든다.
- “분할상환 $T(n)$ ”은 대략 “평균적으로” $T(n)$ 을 의미하고, 모든 연산에 대한 평균이다.

- 예시. 해시 표에 데이터 삽입은 O(1) 분할상환 실행시간이 든다.

~~데이터 구조를 쓸 때는 총 연산 시간이 중요하지, 각 연산의 실행 시간은 중요하지 않다.
그렇기 때문에 분할 상황 실행시간을 사용할 수 있다.~~

다시 해싱으로 돌아와서:

$m = \Theta(n) \Rightarrow \alpha = \Theta(1) \Rightarrow$ 찾기의 기대 실행시간은 $O(1)$ 이다.(단순 균일 해싱이나 유니버설 해싱을 전제로 함.)

데이터 삽제:

기대 실행시간은 $O(1)$ 이다.



- 공간이 n 과 같이 커진다. 예시. n 번의 삽입, n 번의 삭제
- 해결책: n 이 $m/4$ 로 떨어지면, 해시 표를 반으로 줄인다. \Rightarrow 삽입과 삭제 둘 다 $O(1)$ 분할상환 시간이 걸린다. - 분석은 더 복잡함. CLRS 17.4 참조

크기 조절이 가능한 배열:

- 같은 기법이 파이썬의 “리스트” (배열)에도 쓰인다.
- \Rightarrow `list.append` 와 `list.pop` 을 $O(1)$ 분할상환 시간으로 할 수 있다.

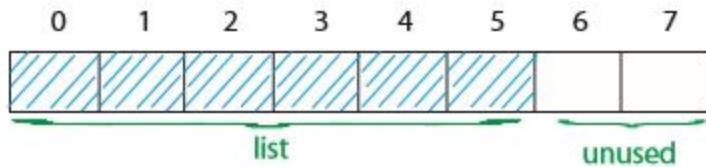


그림 2: 크기 조절이 가능한 배열

문자열 매칭

두 개의 문자열 s 와 t 가 주어졌을 때, s 가 t 의 부분 문자열인가? (그리고 그렇다면, 어디에 몇개나 있는가?)

예시. $s = '6.006'$ 이고 $t =$ 전체 받은 메일함 (UNIX에선 ‘grep’)

간단한 알고리즘:

어떠한 ($s == t[i : i + len(s)]$ for i in range($len(t) - len(s)$))

— $O(|s|)$ 시간이 각 부분 문자열을 비교할 때 걸린다.

- ⇒ 총 $O(|s| \cdot (|t| - |s|))$ 이 걸린다.
= $O(|s| \cdot |t|)$ 2차가 될 수 있다.



그림 3: 문자열 매칭 문제의 간단한 알고리즘 삽화

Karp-Rabin 알고리즘:

- $h(s) == h(t[i : i + len(s)])$ 를 비교한다.
- 해시 값이 같으면, 문자열도 같을 수 있으니
 - $s == t[i : i + len(s)]$ 를 확인한다. 비용 $O(|s|)$ 이 걸린다.
 - 확인해서 같다면, 짹을 찾은 것이다.
 - 확인해서 다른 경우는, $\frac{1}{|s|}$ 보다 작은 확률로 발생한다.
 \Rightarrow i당 기대 실행시간은 $O(1)$ 이다.
- 적당한 해시 함수가 필요하다.
- 기대 시간 = $O(|s| + |t| \cdot cost(h))$
 - 순진하게 말하면, $h(x)$ 는 $|x|$ 정도 걸린다.
 - $O(1)$ 안에 실행할 수 있다!
 - 발상: $t[i : i + len(s)] \approx t[i + 1 : i + 1 + len(s)]$.

롤링 해시 ADT

문자열 x 가 다음 연산자에 적용된다.

- $r()$: 문자열 x 의 합리적인 해시 함수 $h(x)$
- $r.append(c)$: 문자열 x 끝에 문자 c 를 넣는다.
- $r.skip(c)$: 문자열 x 의 첫 문자가 c 라는 가정 하에 첫 문자를 지운다.

Karp-Rabin 적용:

```
for c in s: rs.append(c)
for c in t[:len(s)]: rt.append(c)
```

```
if rs() == rt(): ...
```

이 코드의 첫 부분은 $O(|s|)$ 만큼 걸린다.

```
for i in range(len(s), len(t)):
    rt.skip(t[i-len(s)])
    rt.append(t[i])
    if rs() == rt(): ...
```

코드의 두 번째 부분은 $O(|t|) + O(\#matches - |s|)$ 만큼 걸린다.

자료구조:

문자열 x 를 a 진법으로 쓰여진 여러 자릿수의 수, u 로 본다. a 는 알파벳의 크기이다.

예시. 256

- $r() = u \bmod p$, (이상적으로는 무작위임) 소수 $p \approx |s|$ 또는 $|t|$ (나누기 방법)
- r 은 u 가 아닌, $u \bmod p$ 와 $|x|$ (사실 $a^{|x|}$) 를 가진다.

⇒ 작고 빠르게 실행할 수 있다. ($u \bmod p$ 는 하나의 기계 단어와 동일)

- $r.append(c)$: $(u \cdot a + \text{ord}(c)) \bmod p = [(u \bmod p) \cdot a + \text{ord}(c)] \bmod p$
- $r.skip(c)$: $[u - \text{ord}(c) \cdot (a^{|u|} - 1 \bmod p)] \bmod p$

$$= [(u \bmod p) - \text{ord}(c) \cdot (a^{|u|} - 1 \bmod p)] \bmod p$$

MIT OpenCourseWare

<http://ocw.mit.edu>

6.006 알고리즘의 기초

가을 2011

본 자료 이용 또는 이용 약관에 대한 정보를 확인하려면 다음의 사이트를 방문하십시오:

<http://ocw.mit.edu/terms>.