

## 강의 2: 계산 모델

### 강의 개요

- 알고리즘이란? 시간이란?
- 임의 접근 머신
- 포인터 머신
- 파이썬 모델
- 문서 거리: 문제 & 알고리즘

알고리즘이란?

### 역사

Al-Khwārizmī “al-kha-raz-mi” (c. 780-850)

- “대수학의 아버지”로, “완성과 균형 맞춤을 통한 계산에 대해 모든 것을 담은 책(The Compendious Book on Calculation by Completion & Balancing)” 저술

- 1차 & 2차 방정식 풀기: 최초의 알고리즘 중 일부

알고리즘이란?

① 컴퓨터 프로그램의 수학적 추상화

- 문제 해결을 위한 계산 과정

→ input → alg → output

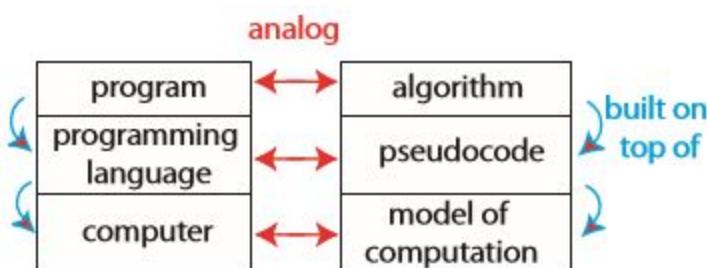


그림 1: 알고리즘

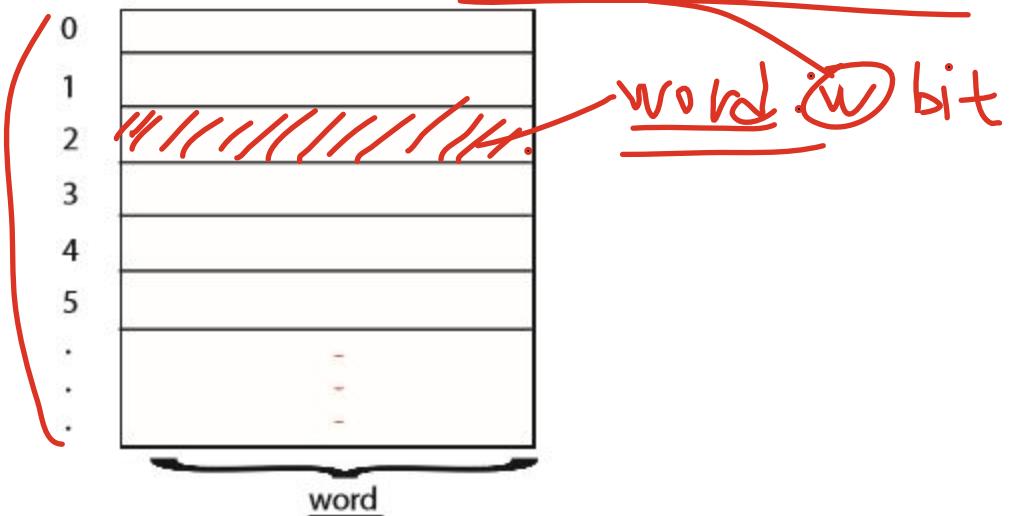
### 계산 모델이 규정하는 것

- 알고리즘이 할 수 있는 연산
- 각 연산의 비용 (시간, 공간, ...)
- 알고리즘의 비용 = 연산 비용의 합

워드가 배열의 인덱스를 지정할 수  
있어야 하므로  
 $w \geq \log_2(\text{size of memory})$

임의 접근 머신 (RAM)

RAM은  
기본적으로 어떤  
배열

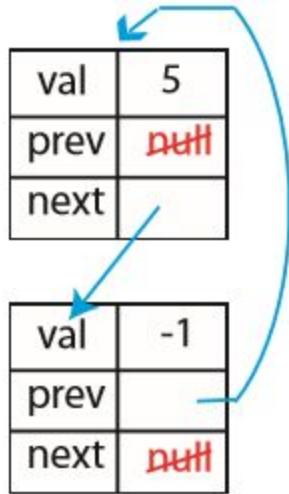


- 거대한 배열로 만들어진 임의 접근 머신 (RAM)
- $\Theta(1)$  레지스터 (각 1개의 워드)
- $\Theta(1)$  시간 안에 할 수 있는 일
  - 레지스터  $r_i$  에 있는 워드를 레지스터  $r_j$  으로 불러오기
  - 레지스터에서 ( $+, -, *, /, \&, |, ^$ ) 계산
  - 레지스터  $r_j$  를  $r_i$  에 있는 메모리에 저장
- 워드란?  $w \geq \log_2(\text{memory size})$  bit
  - 기본적인 객체(e.g., 정수)가 워드에 들어맞는다고 가정한다
  - 4단원에서 큰 수를 다룬다
- 현실적이고 강력함 → 추상적 개념 구현

포인터 머신

연결리스트 (linked list)

- 동적 할당된 객체 (네임드 튜플)
- 객체는  $O(1)$ 개의 필드를 갖는다.
- 필드 = 워드 (e.g., 정수) 또는 객체/널을 가리키는 포인터 (a.k.a. 참조)
- RAM보다 약하다 (RAM으로 구현 가능)



## 파이썬 모델

파이썬은 두 가지 사고를 모두 적용할 수 있다.

1. “리스트”는 사실 배열이다. → RAM

$L[i] = L[j]+5 \rightarrow \Theta(1)$  시간

2.  $O(1)$ 개의 속성(참조 포함)을 가진 객체 → 포인터 머신

$x = x.next \rightarrow \Theta(1)$  시간

파이썬에는 많은 연산이 있다. 각 연산의 비용을 알아보려면 (1)이나 (2)로 구현해보면 된다. :

### 1. 리스트

(a)  $L.append(x) \rightarrow \theta(1)$  time

obvious if you think of infinite array

but how would you have  $> 1$  on RAM?

via table doubling [Lecture 9]

(b)  $\underbrace{L = L1 + L2}_{(\theta(1+|L1|+|L2|) \text{ time})} \equiv L = [] \rightarrow \theta(1)$

for $x$ in $L1$ : $L.append(x) \rightarrow \theta(1)$	$\left. \right\} \theta( L_1 )$
for $x$ in $L2$ : $L.append(x) \rightarrow \theta(1)$	$\left. \right\} \theta( L_2 )$

}

(c) $L1.\text{extend}(L2) \equiv \text{for } x \text{ in } L2:$ $\quad \equiv L1 += L2 \quad L1.\text{append}(x) \rightarrow \theta(1)$	$\left. \begin{array}{l} \\ \end{array} \right\} \theta(1 +  L_2 ) \text{ time}$
(d) $L2 = L1[i:j] \equiv L2 = []$ $\quad \text{for } k \text{ in range}(i,j):$ $\quad \quad L2.\text{append}(L1[i]) \rightarrow \theta(1)$	$\left. \begin{array}{l} \\ \end{array} \right\} \theta(j - i + 1) = O( L )$
(e) $b = x \text{ in } L \equiv \text{for } y \text{ in } L:$ $\quad \& L.\text{index}(x) \quad \text{if } x == y:$ $\quad \& L.\text{find}(x) \quad \quad b = \text{True};$ $\quad \quad \quad \text{break}$ $\quad \quad \quad \text{else}$ $\quad \quad \quad b = \text{False}$	$\left. \begin{array}{l} \theta(1) \\ \end{array} \right\} \theta(\text{index of } x) = \theta( L )$

(f)  $\text{len}(L) \rightarrow \theta(1)$  시간 - 리스트는 자신의 길이를 필드에 저장한다

(g)  $L.\text{sort}() \rightarrow \theta(|L| \log |L|)$  - 비교 정렬을 사용 [강의 3, 4 & 7]

2. tuple, str: similar, (think of as immutable lists)

3. dict: via *hashing* [Unit 3 = Lectures 8-10]

D[key] = val key in D	$\left. \begin{array}{l} \\ \end{array} \right\} \theta(1) \text{ time w.h.p.}$
--------------------------	---

\* 키가 중복되는 경우

4. set: similar (think of as dict without vals)

5. heapq: heappush & heappop - via *heaps* [Lecture 4]  $\rightarrow \theta(\log(n))$  time

6. long: via *Karatsuba algorithm* [Lecture 11]

$x + y \rightarrow O( x  +  y )$ time	where $ y $ reflects # words
$x * y \rightarrow O(( x  +  y )^{\log(3)})$	$\approx O(( x  +  y )^{1.58})$ time

## 문서 거리 문제 — $d(D_1, D_2)$ 계산

문서 거리 문제는 유사한 문서 탐색이나 중복(위키피디아 미러 사이트와 구글)과 표절 발견, 그리고 웹 검색에 적용된다. ( $D_2$  = 질의어).

정의:

- 단어 = 영어와 숫자로 이루어진 문자열
- 문서 = 단어의 나열 (공백, 문장 부호 등 무시)

공통으로 갖는 단어를 통해 거리를 정의하고자 한다. 문서  $D$ 를 벡터로 생각한다. :  $D[w] =$  단어  $w$ 의 등장 횟수. 예를 들면:

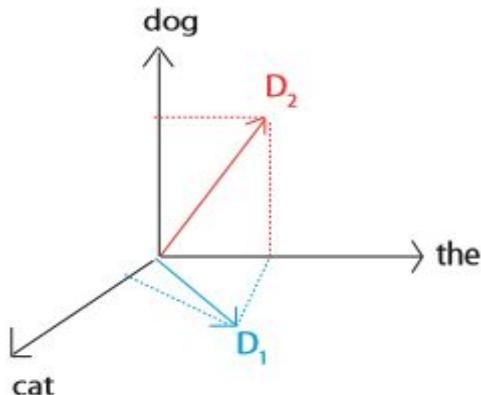


그림 2:  $D_1 = \text{"the cat"}$ ,  $D_2 = \text{"the dog"}$

첫 번째 시도로, 문서 거리를 다음과 같이 정의한다:

$$d'(D_1, D_2) = D_1 \cdot D_2 = \sum_W D_1[W] \cdot D_2[W]$$

문제는 규모에 따라 변한다는 것이다. 즉, 99%의 단어가 일치하는 긴 문서들이 10%만 일치하는 짧은 문서들보다 거리가 멀게 된다.

이 문제는 단어의 개수로 정규화함으로써 해결할 수 있다:

$$d''(D_1, D_2) = \frac{D_1 \cdot D_2}{|D_1| \cdot |D_2|}$$

여기서  $|D_i|$ 는 문서  $i$ 에 들어있는 단어의 개수이다. 이것을 기하학적으로 해석(재구성)하면 다음과 같다. :

$$d(D_1, D_2) = \arccos(d''(D_1, D_2))$$

즉, 문서 거리는 두 벡터 사이의 각도이다.  $0^\circ$ 는 두 문서가 같다는 것을 의미하고,  $90^\circ$ 는 공통 단어가 없다는 것을 의미한다. 이 방법은 1975년에 Salton, Wong, Yang에 의해 소개되었다.

## 문서 거리 알고리즘

- ① 각 문서를 단어들로 쪼갠다.
  - ② 단어 빈도를 센다. (문서 벡터)
  - ③ 내적을 계산한다. (& 나눈다.)
- 각 문서에 단어가 몇번씩 등장하는  
count[word] += 1

- (1) `re.findall(r“ w+”, doc)` → what cost?
- in general `re` can be exponential time  
 $\rightarrow$  for char in doc:  
 if not alphanumeric  
 add previous word  
 (if any) to list  
 start new word
- $\Theta(1)$
- $\Theta(|doc|)$
- (2) sort word list  $\leftarrow O(k \log k \cdot |word|)$  where  $k$  is #words
- for word in list:  
 if same as last word:  $\leftarrow O(|word|)$   
 increment counter  
 else:  
 add last word and count to list  
 reset counter to 0
- $\Theta(1)$
- $O(\sum |word|) = O(|doc|)$
- (3) for word, count1 in doc1:  
 if word, count2 in doc2:  $\leftarrow \Theta(k_2)$   
 total += count1 \* count2  $\Theta(1)$
- $O(k_1 \cdot k_2)$
- (3)' start at first word of each list  
 if words equal:  $\leftarrow O(|word|)$   
 total += count1 \* count2  
 if word1  $\leq$  word2:  $\leftarrow O(|word|)$   
 advance list1  
 else:  
 advance list2  
 repeat either until list done
- $O(\sum |word|) = O(|doc|)$

### Dictionary Approach

- (2)' count = {}
- for word in doc:  
 if word in count:  $\leftarrow \Theta(|word|) + \Theta(1)$  w.h.p  
 count[word] += 1  
 else  
 count[word] = 1
- $\Theta(1)$
- $O(|doc|)$  w.h.p.
- (3)' as above  $\rightarrow O(|doc_1|)$  w.h.p.

## 코드 (웹 사이트의 lecture\_code.zip & \_data.zip)

t2.bobsey.txt 268,778 chars/49,785 words/3354 uniq

t3.lewis.txt 1,031,470 chars/182,355 words/8534 uniq

seconds on Pentium 4, 2.8 GHz, C-Python 2.62, Linux 2.6.26

- docdist1: 228.1 — (1), (2), (3) (추가적인 정렬)  
words = words + words\_on\_line

- docdist2: 164.7 — words += words\_on\_line

- docdist3: 123.1 — (3)'... 삽입 정렬 이용

- docdist4: 71.7 — (2)' 이지만 (3)'의 정렬 이용

- docdist5: 18.3 — string.translate로 단어 분할

- docdist6: 11.5 — 병합 정렬 (vs. 삽입 정렬)

- docdist7: 1.8 — (3) (완전한 딕셔너리)

- docdist8: 0.2 — 한 줄 한 줄이 아닌 문서 전체

MIT OpenCourseWare

<http://ocw.mit.edu>

6.006 알고리즘 개론

가을 2011

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

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