VE281

Data Structures and Algorithms

Recitation Class

Oct. 14 2018

VE281 TA Group

Linear Time Selection

- Selection Problem:
  - Input: array A with n distinct numbers and a number i
  - Output: i-th smallest element in the array
- Solution:
  - Reduction to sorting $O(n \log n)$
  - Randomized selection algorithm $O(n)$
  - Deterministic selection algorithm $O(n)$

Linear Time Selection

- Randomized selection algorithm

```
Rselect(int A[], int n, int i) {
// find i-th smallest item of array A of
size n
  if(n == 1) return A[1];
  Choose pivot p from A uniformly at random;
  Partition A using pivot p;
  Let j be the index of p;
  if(j == i) return p;
  if(j > i) return Rselect(1st part of A,
j-1, i);
  else return Rselect(2nd part of A, n-j,
i-j);
}
```

Linear Time Selection

- Randomized selection algorithm
  - Worst case: $O(n^2)$
  - Best case: $O(1)$
  - Average case: $O(n)$

$$E[N] = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot (1 + E[N])$$

$$E[runtime] \leq E\left[\sum_j X_j \cdot c \cdot \left(\frac{3}{4}\right)^j n\right]$$

$$= cn \sum_j \left(\frac{3}{4}\right)^j E[X_j] \;\; \leq 2cn \sum_j \left(\frac{3}{4}\right)^j \leq 2cn \frac{1}{1 - \frac{3}{4}}$$

$$= 8cn = O(n)$$

Linear Time Selection

- Deterministic selection algorithm

`ChoosePivot(A, n)`

1. Break A into n/5 groups of size 5 each
2. Sort each group (e.g., use insertion sort)
3. Copy n/5 medians into new array C
4. Recursively compute median of C
   - By calling the deterministic selection algorithm!
5. Return the median of C as pivot

Time complexity: $O(n)$

Not as good as Rselect in practice

Linear Time Selection

- Deterministic selection algorithm
  - $T(n) \le cn + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) \le 10n$
  - ```
    Dselect(int A[], int n, int i) {
    // find i-th smallest item of array A of size
    n
    if(n == 1) return A[1];
    Break A into groups of 5, sort each group; Θ(n)
    C = n/5 medians;  Θ(n)
    p = Dselect(C, n/5, n/10);  T(n/5)
    Partition A using pivot p;  Θ(n)
    Let j be the index of p;
    if(j == i) return p;
    if(j > i) return Dselect(1st part of A, j-1,
    i);                                          T(0.7n)
    else return Dselect(2nd part of A, n-j, i-j);
    }
    ```
    ```

Hashing: Basics and Hash Function

- Dictionary
  - **(key, value)**
  - <u>Different pairs have different keys</u>
  - Sorted/Unsorted array
  - find()
  - insert()
  - remove()

Hashing: Basics and Hash Function

- Hashing Basics
  - Pick an array A (hash table) of n buckets.
    - $n = c|S|$: a small multiple of $|S|$.
  - Choose a hash function $h: U \to \{0,1,\ldots,n-1\}$
    - h is fast to compute.
    - The same key is always mapped to the **same** location.
      But different keys may maps to the same location
  - Store item k in A[h(k)]
  - Collision:
    - Separate chaining
    - Open addressing

Hashing: Basics and Hash Function

- Hashing Function
  - Must compute a bucket for every key in the universe.
  - Must compute the same bucket for the same key.
  - Should be easy and quick to compute.
  - Minimizes collision

Hashing by Modulo

The bias in the keys does not result in a bias toward home buckets.

Ideally, choose the hash table size n as a **large prime number**.

Hashing: Basics and Hash Function

- Collision:
  - Separate chaining
  - Open addressing
    - Linear probing:
      $h_i(x)$ = (h(x) + i) % n
    - Quadratic probing:
      $h_i(x)$ = (h(x) + i^2) % n
    - $L = \frac{m}{n} = \frac{\#\text{objects in hash table}}{\#\text{buckets in hash table}} \leq 0.5$ (find an empty slot)
    - Double hashing:
      $h_i(x)$ = (h(x) + i*g(x)) % n