Data Structure Programming Project #5

郭建志

Image

- You want to count elements
- You don’t need exact results

Problem

- Given:
- keys in many documents

- Goal:
- Count the key frequency
- Bounded error is allowed

- Constraint:
- Limited storage and limited computation

Simple Solution

- Construct a map from elements to counts
- Balanced binary tree: `map` in C++
- Hash table: `unordered_map` in C++
- You may want to use libraries Guava or FastUtil in Java for convenience and better performance

Problem

- The number of of distinct elements might be very large
- You have a limited memory space
- For real-time applications you need runtime guarantees

Solution: Count-Min Sketch

- The trick: don’t store the distinct elements, but just the counters
- Create an integer array of length x initially filled with 0s
- Each incoming element gets mapped to a number between 0 and x
- The corresponding counter in the array gets incremented
- To query an element’s count, simply return the integer value at it’s position

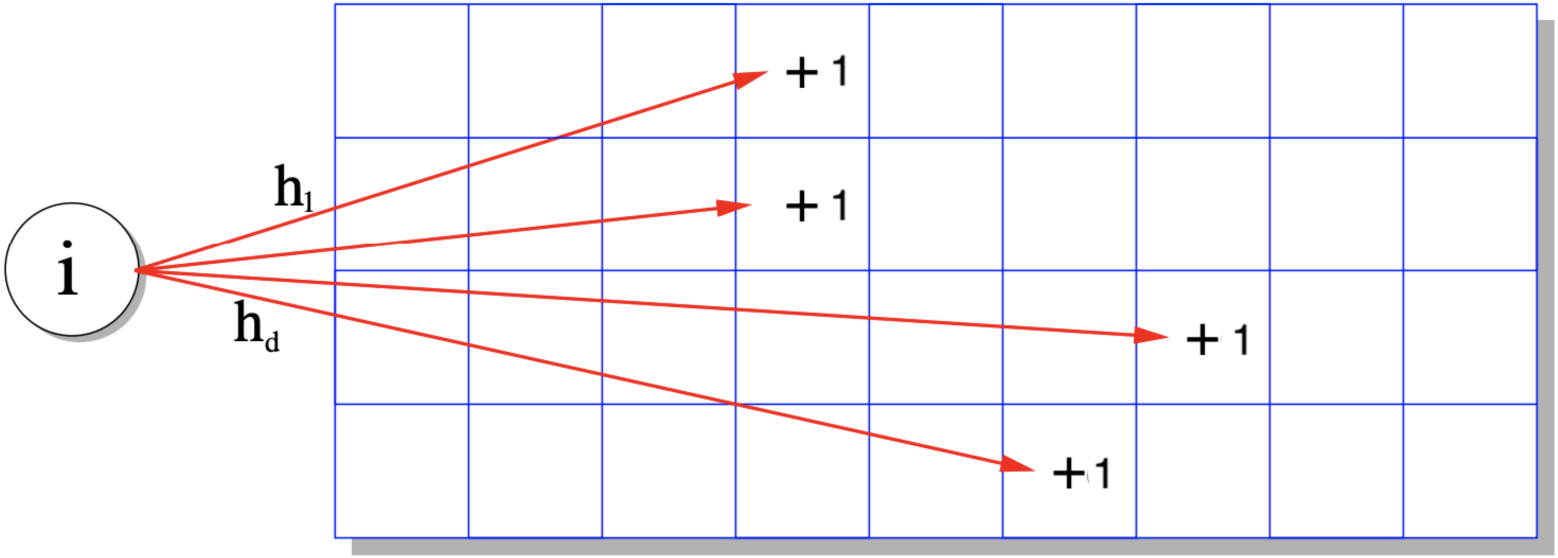
Solution: Count-Min Sketch

- The trick: don't store the distinct elements, but just the counters
- Create an integer array of length x initially filled with 0s
- Each incoming element gets mapped to a num
- The ets incre
- To query an element's count, simply return the integer value at it's position

You are completely right:
There will be collisions!

Solution: Count-Min Sketch

- Use multiple arrays with different hash functions to compute the index
- When queried, return the minimum of the numbers the array



Solution: Count-Min Sketch

- Use multiple arrays with different hash functions to compute the index
- When queried, return the minimum of the numbers the array

You are completely right:
There will still be collisions!

Solution: Count-Min Sketch

- Use multiple arrays with different hash functions to compute the index
- When queried, return the minimum of the numbers the array

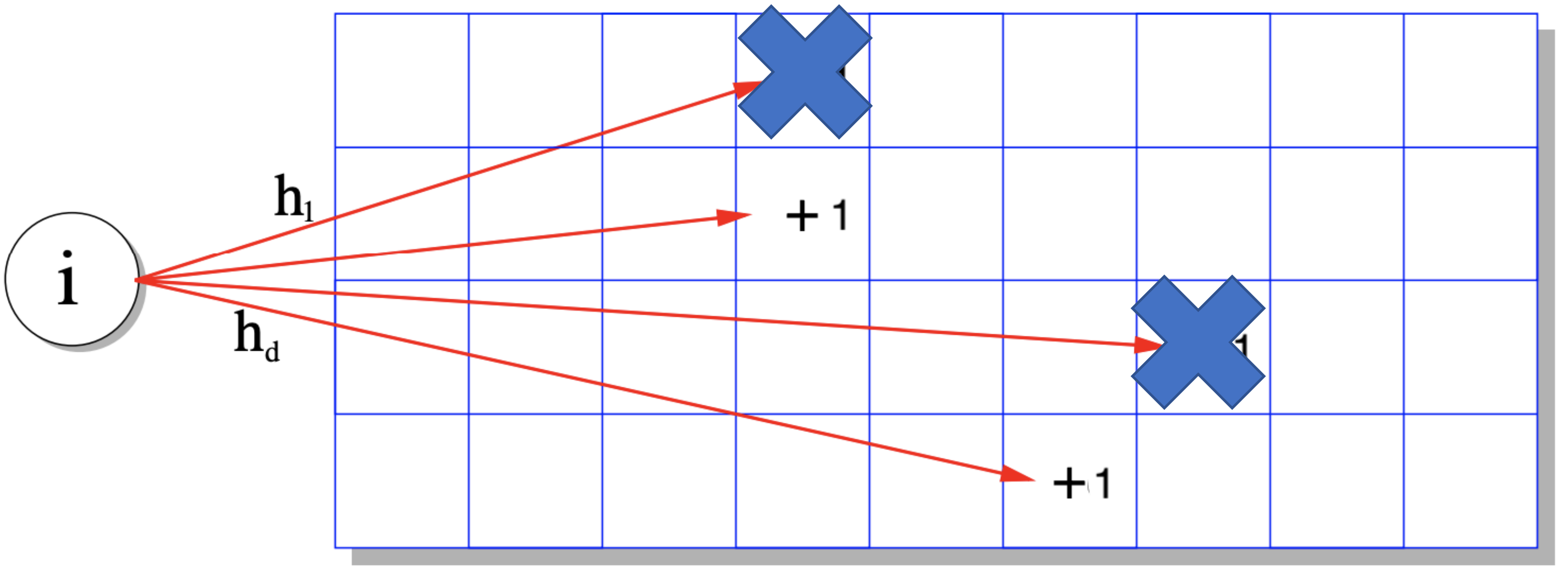
You are completely right:
There will still be collisions!
… but less

Some properties

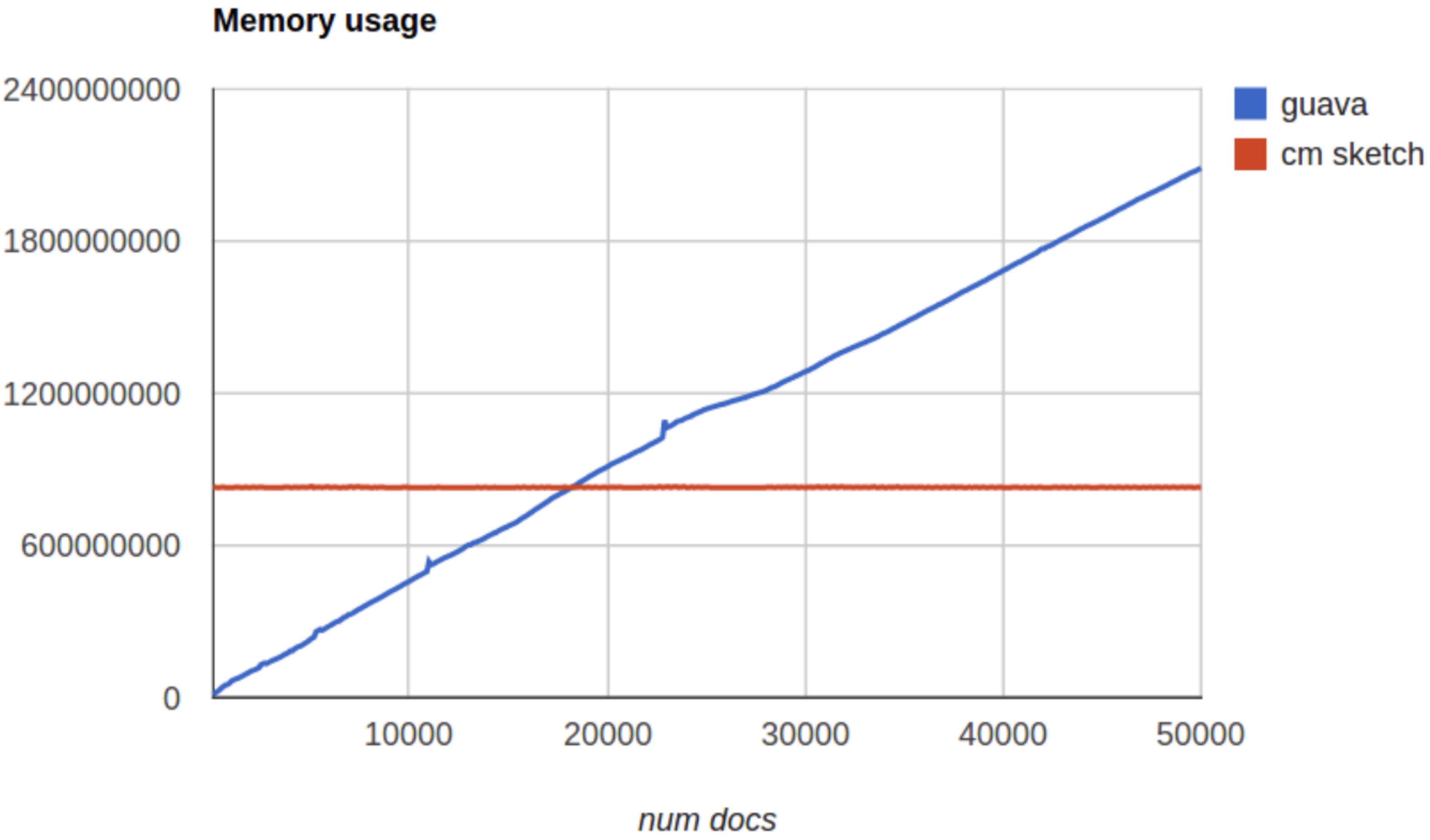
- Only over-estimates, never under-estimates the true count
- Has a constant memory and time consumption independent of the number of elements
- The relative error may be high for low-frequent elements

To be practical: Conservative Update

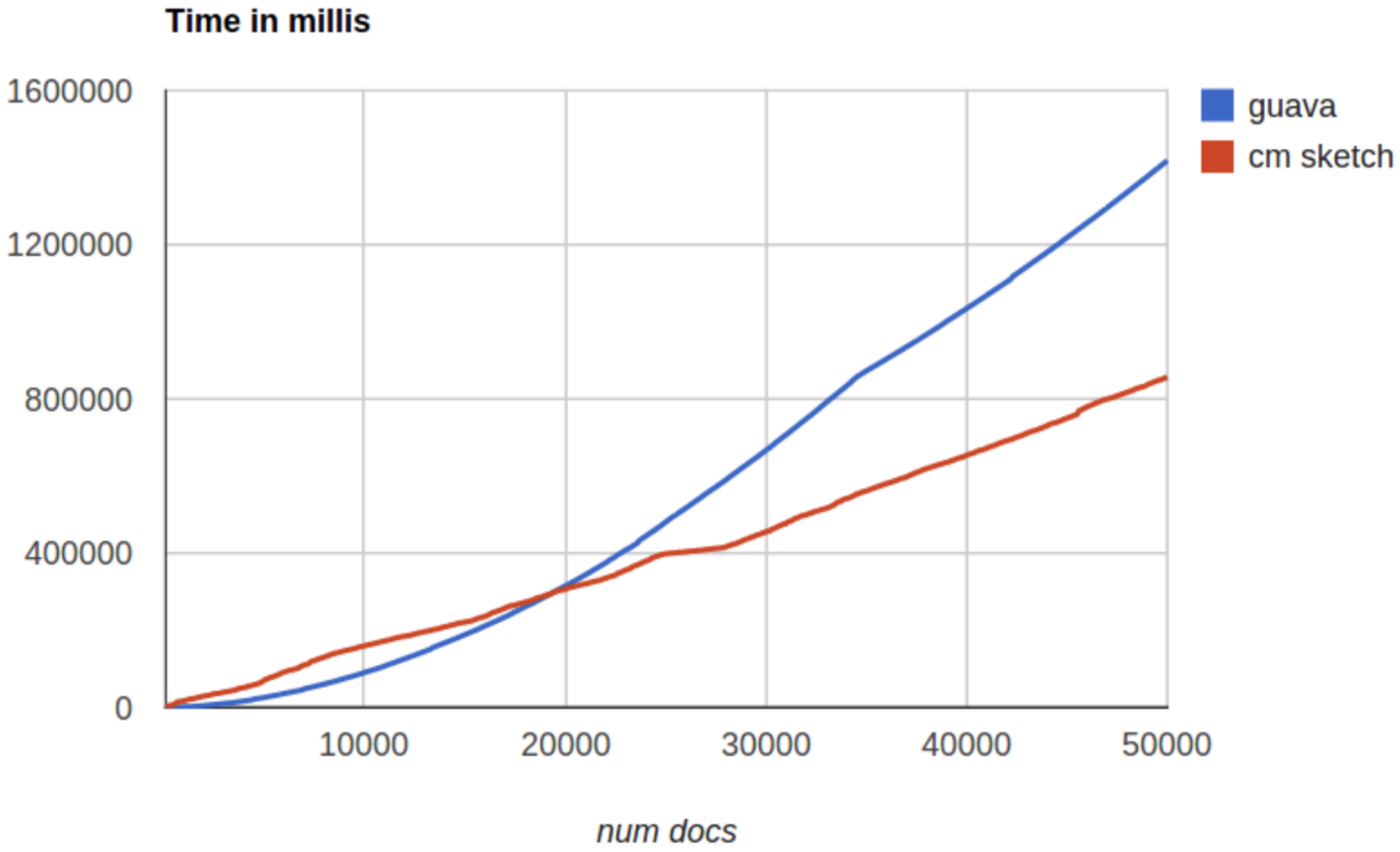
- Instead of incrementing every counter, only increment the ones which need an update
- Experiments show a strong error reduction using this heuristic



Comparison


Memory usage
2400000000
1800000000
1200000000
600000000
0
10000
20000
30000
40000
50000
num docs
guava
cm sketch

Comparison


Time in millis
1600000
1200000
800000
400000
0
10000
20000
30000
40000
50000
num docs
guava
cm sketch

Usage

- General:
  - Every counting problem on large data sets, where errors are acceptable
  - Counting elements in data streams (e.g., trend detection on Twitter, Facebook, and news)

Usage

- Mine:
  - Feature selection for large sets using pointwise mutual information (PMI)
  - PMI has been used for finding associations between words
  - All kinds of feature statistics like word co-occurrences

You need to implement:

```
void init(int **map, int r, int c, int *a, int *b, int p)
1. Create a 2D array with r rows and c columns for pointer map
2. Create an array with r elements uniformly chosen from [1, p-1] for pointer a
3. Create an array with r elements uniformly chosen from [1, p-1] for pointer b (note: a[i] and b[i] should be independent)
```

```
int myhash(char *str, int count, int r, int c, int p, int *a, int *b)
Use hash in <string> to covert str to an integer key
// note that 0 <= count <= r-1
return (a[count] * key + b[count]) % p  % c;
```

```
void insert(int **map, int r, int c, int p, char *str, int *a, int *b)
1. Find the smallest ones in the following positions,
map[count][myhash(str, count, r, c, a, b)] for 0 ≤ count ≤ r-1
2. Increment the smallest ones by 1
```

```
void query(int **map, int r, int c, int p, char *str, int *a, int *b)
1. Find the smallest one in the following positions,
map[count][myhash(str, count, r, c, a, b)] for 0 ≤ count ≤ r-1
2. Return the smallest one
```

Input Sample: input.txt

```
10 10 1019
Data structures
serve as the
basis
for abstract
data type. The
abstract data
type defines
the logical
form of the
data type. The
data structure
implements the
physical form
of the data
type.
```

#rows #cols prime

Text...

Note: your code should understand that ”Data” is equivalent to “data” and ignore punctuation marks (e.g., commas)

You don’t need to output anything, but..

Command:

```
# data
6
# serve
1
# type
4
```

Input a key
Return a value

Note: The value is **allowed** to have a small error to some extent, since you are using cm sketch instead of binary search tree

Note

- Deadline:
  1/3 Thu (有問題可以再調整)

- This project accounts for 15% of the total score
- You are not allowed to use ”class” in STL to count the words
- You must implement a 2D array with the given size (i.e., #rows and #cols in input.txt) to count the words

- **E-course**

- **C or C++ Source code**

- You can read this paper if you are interested in the research field:
- http://dimacs.rutgers.edu/~graham/pubs/papers/cmsoft.pdf