Δομές Δεδομένων και Αλγόριθμοι (Γ' εξάμηνο)

Τμήμα Μηχανικών Πληροφορικής ΤΕ, ΤΕΙ Ηπείρου

Γκόγκος Χρήστος

08/01/2015

Άσκηση εργαστηρίου #7 (Bloom Filters)

Στην εργασία αυτή εξετάζονται 5 εναλλακτικοί τρόποι με τους οποίους ελέγχεται για ένα μεγάλο πλήθος τιμών (5.000.000), πόσες από αυτές δεν περιέχονται σε ένα δεδομένο σύνολο τιμών 1000 τυχαίων ακεραίων τιμών που κυμαίνονται στο διάστημα [0, 100.000). Ο χρόνος που απαιτεί η κάθε προσέγγιση μετράται έτσι ώστε να καταδειχθεί η απόδοση της κάθε προσέγγισης.

Η πρώτη προσέγγιση χρησιμοποιεί ένα vector για να αποθηκεύσει το σύνολο των 1000 τυχαίων ακεραίων τιμών και με τη συνάρτηση **scenario1** αναζητά σειριακά κάθε στοιχείο στο vector.

Η δεύτερη προσέγγιση χρησιμοποιεί επίσης ένα vector για να αποθηκεύσει το σύνολο των 1000 τυχαίων ακεραίων τιμών και με τη συνάρτηση **scenario2** το ταξινομεί και αναζητά με τη δυαδική αναζήτηση κάθε στοιχείο στο vector.

Η τρίτη προσέγγιση χρησιμοποιεί ένα set για να αποθηκεύσει το σύνολο των 1000 τυχαίων ακεραίων τιμών και με τη συνάρτηση **scenario3** αναζητά κάθε στοιχείο στο set. Στη C++ το set υλοποιείται ως δυαδικό δένδρο αναζήτησης.

Η τέταρτη προσέγγιση χρησιμοποιεί ένα unordered_set για να αποθηκεύσει το σύνολο των 1000 τυχαίων ακεραίων τιμών και με τη συνάρτηση **scenario4** αναζητά κάθε στοιχείο στο unordered_set. Στη C++ το unordered_set υλοποιείται ως πίνακας κατεκερματισμού.

Στην πέμπτη και τελευταία προσέγγιση (**scenario5**) υλοποιείται ένα Bloom Filter που χρησιμοποιεί ως βασική δομή δεδομένων ένα σύνολο με 10001 δυαδικά ψηφία (bitset) καθώς και 3 απλές συναρτήσεις κατακερματισμού.

```
/*
 * ergasia07.cpp
 *
 *  Created on: 8 Ιαν 2015
 *      Author: chgogos
 */

#include <iostream>
#include <set>
#include <cstdlib>
#include <ctime>
#include <bitset>
#include <vector>
#include <algorithm>
#include <unordered_set>

using namespace std;

// πλήθος στοιχείων συνόλου
const int N = 1000;
// ανώτατο όριο αριθμητικής τιμής που μπορούν να λάβουν οι τυχαίες τιμές του συνόλου
const int UPPER_LIMIT_RANDOM_VALUE = 100000;
// αριθμός τιμών για τις οποίες θα εξεταστεί το εαν δεν υπάρχουν στο σύνολο
const int M = 5000000;
// πλήθος δυαδικών ψηφίων του bitset
const int BS_SIZE = 10001;
```

```
void scenario1(vector<uint32_t> &avector) {
	srand(1821);
	int c = 0;
	for (int i = 0; i < M; i++) {
		uint32_t x = rand() % UPPER_LIMIT_RANDOM_VALUE;
		if (find(avector.begin(), avector.end(), x) == avector.end())
			c++;
	}
	printf("Values not in the set (using unsorted vector) %d\n", c);
}

void scenario2(vector<uint32_t> &avector) {
	sort(avector.begin(), avector.end());
	srand(1821);
	int c = 0;
	for (int i = 0; i < M; i++) {
		int x = rand() % UPPER_LIMIT_RANDOM_VALUE;
		if (!binary_search(avector.begin(), avector.end(), x))
			c++;
	}
	printf("Values not in the set (using sorted vector + binary search) %d \n", c);
}

// http://www.cplusplus.com/reference/set/set/
// binary search trees
void scenario3(set<uint32_t> &aset) {
	srand(1821);
	int c = 0;
	for (int i = 0; i < M; i++) {
		int x = rand() % UPPER_LIMIT_RANDOM_VALUE;
		const bool is_in = aset.find(x) != aset.end();
		if (!is_in)
			c++;
	}
	printf("Values not in the set (using set) %d \n", c);
}

// http://www.cplusplus.com/reference/unordered_set/unordered_set/
// Hash tables
void scenario4(unordered_set<uint32_t> &aset) {
	srand(1821);
	int c = 0;
	for (int i = 0; i < M; i++) {
		int x = rand() % UPPER_LIMIT_RANDOM_VALUE;
		const bool is_in = aset.find(x) != aset.end();
		if (!is_in)
			c++;
	}
	printf("Values not in the set (using unordered set) %d \n", c);
}

inline uint32_t hash1(uint32_t x) {
	return x;
}

inline uint32_t hash2(uint32_t x) {
	return 3 * x;
}

inline uint32_t hash3(uint32_t x) {
	return 5 * x + 7;
}

void scenario5(bitset<BS_SIZE> &bs, unordered_set<uint32_t> &aset) {
	srand(1821);
```

```
	int c = 0;
	for (int i = 0; i < M; i++) {
		uint32_t x = rand() % UPPER_LIMIT_RANDOM_VALUE;
		uint32_t h1 = hash1(x);
		uint32_t h2 = hash2(x);
		uint32_t h3 = hash3(x);
		if (!bs.test(h1 % BS_SIZE) || !bs.test(h2 % BS_SIZE)
				|| !bs.test(h3 % BS_SIZE))
			c++;
		else {
			const bool is_in = aset.find(x) != aset.end();
			if (!is_in)
				c++;
		}
	}
	printf("Values not in the set (using bloom filter + unordered set) %d\n", c);
}

int main(int argc, char **argv) {
	vector<uint32_t> avector;
	set<uint32_t> aset;
	unordered_set<uint32_t> uset;
	bitset<BS_SIZE> bs;
	srand(time(0));
	for (int i = 0; i < N; i++) {
		uint32_t x = rand() % UPPER_LIMIT_RANDOM_VALUE;
		aset.insert(x);
		uset.insert(x);
		avector.push_back(x);
		bs.set(hash1(x) % BS_SIZE, true);
		bs.set(hash2(x) % BS_SIZE, true);
		bs.set(hash3(x) % BS_SIZE, true);
	}
	cout << "Number of bits set = " << bs.count() << endl;
	clock_t t1, t2, t3, t4, t5, t6;
	t1 = clock();
	scenario1(avector);
	t2 = clock();
	double elapsed_time1 = (double) (t2 - t1) / CLOCKS_PER_SEC;
	scenario2(avector);
	t3 = clock();
	double elapsed_time2 = (double) (t3 - t2) / CLOCKS_PER_SEC;
	scenario3(aset);
	t4 = clock();
	double elapsed_time3 = (double) (t4 - t3) / CLOCKS_PER_SEC;
	scenario4(uset);
	t5 = clock();
	double elapsed_time4 = (double) (t5 - t4) / CLOCKS_PER_SEC;
	scenario5(bs, uset);
	t6 = clock();
	double elapsed_time5 = (double) (t6 - t5) / CLOCKS_PER_SEC;
	printf(
			"Elapsed times: Vector(unsorted)=%.2f Vector(sorted)=%.2f Set=%.2f
Unordered Set=%.2f Bloom Filter=%.2f\n",
			elapsed_time1, elapsed_time2, elapsed_time3, elapsed_time4,
			elapsed_time5);

}
```

Παράδειγμα εξόδου που παράγεται κατά την εκτέλεση του ανωτέρω κώδικα:

```
Values not in the set (using unsorted vector) 4513058
Values not in the set (using sorted vector + binary search) 4513058
Values not in the set (using set) 4513058
Values not in the set (using unordered_set) 4513058
Values not in the set (using bloom filter + unordered_set) 4513058
Elapsed times:
Vector(unsorted)=53.62 Vector(sorted)=2.64 Set=2.55 Unordered Set=0.93 Bloom Filter=0.58
```

Ερώτημα 1

Να συμπληρώσετε τον κώδικα έτσι ώστε να εμφανίζει στην περίπτωση του Bloom Filter (scenario5) πόσες φορές δεν ήταν δυνατόν να διαπιστωθεί ότι το στοιχείο δεν υπήρχε στο σύνολο εξετάζοντας μόνο το Bloom Filter και χρειάστηκε να χρησιμοποιηθεί το unordered_set για τον έλεγχο.

Ερώτημα 2

Τροποιήστε και τις 3 συναρτήσεις κατακερματισμού που χρησιμοποιεί το Bloom Filter έτσι ώστε η τιμή που παράγει η κάθε μια συναρτηση κατακερματσιμού να αποτελεί παράμετρο της συνάρτησης `fnv32` και να επιστρέφει το αποτέλεσμα που επιστρέφει αυτή. Εκτελέστε ξανά το πρόγραμμα.

```
inline uint32_t fnv32(uint32_t key) {
	uint8_t* bytes = (uint8_t*) (&key);
	uint32_t hash = 2166136261U;
	hash = (16777619U * hash) ^ bytes[0];
	hash = (16777619U * hash) ^ bytes[1];
	hash = (16777619U * hash) ^ bytes[2];
	hash = (16777619U * hash) ^ bytes[3];
	return hash;
}
```