Δομές Δεδομένων και Αλγόριθμοι (Γ' εξάμηνο)

Τμήμα Μηχανικών Πληροφορικής ΤΕ, ΤΕΙ Ηπείρου

Γκόγκος Χρήστος

6/11/2014

Άσκηση εργαστηρίου #3 (Αλγόριθμοι αναζήτησης)

Στην εργασία αυτή θα χρησιμοποιηθούν οι αλγόριθμοι αναζήτησης: σειριακή αναζήτηση και δυαδική αναζήτηση. Θα χρειαστεί να δηλωθούν οι ακόλουθες βιβλιοθήκες στην αρχή του κώδικα:

```
#include <algorithm>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <iostream>
#include <string>
```

Ερώτημα 1 (δημιουργία δεδομένων)

Ο ακόλουθος κώδικας ορίζει μια δομή δεδομένων με όνομα customer (πελάτης) που περιέχει δύο πεδία: το name (όνομα πελάτη) και το balance (υπόλοιπο λογαριασμού). Επιπλέον ορίζει και την απαραίτητη συνάρτηση σύγκρισης έτσι ώστε να είναι δυνατόν να διαταχθούν δεδομένα τύπου customer με βάση τα ονόματα των πελατών.

```
struct Customer {
	string name;
	int balance;
	bool operator<(Customer other) {
		return name < other.name;
	}
};
```

Δημιουργήστε έναν πίνακα με στοιχεία 10000 υποθετικών πελατών. Το όνομα κάθε πελάτη να αποτελείται από 3 γράμματα που θα επιλέγονται με τυχαίο τρόπο από τα τα γράμματα της Αγγλικής αλφαβήτου. Το δε υπόλοιπο κάθε πελάτη θα είναι ένας τυχαίος αριθμός από το 1 μέχρι το 5000. Για την δημιουργία των πελατών μπορείτε να χρησιμοποιήσετε τον ακόλουθο κώδικα.

```
string generate_random_name(int k) {
	string name { };
	char letters_en[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
	int L = strlen(letters_en);
	for (int j = 0; j < k; j++) {
		char c { letters_en[rand() % L] };
		name += c;
	}
	return name;
}

void generate_data(struct Customer* customers, int N) {
	srand(time(0));
	for (int i = 0; i < N; i++) {
		customers[i].name = generate_random_name(3);
		customers[i].balance = rand() % 5000 + 1;
	}
}
```

Ερώτημα 2 (υπολογισμοί)

Υπολογίστε το μέσο όρο των υπολοίπων των λογαριασμών και εμφανίστε τα ονόματα των πελατών με υπόλοιπο λογαριασμού που να απέχει το πολύ ±2% από το μέσο όρο υπολοίπων. Η συνάρτηση που θα κατασκευαστεί θα πρέπει να έχει την ακόλουθη μορφή.

```
void compute_avg_balance_print_customer_names(struct Customer* customers, int N,
            double percentage) {
    // to be completed
}
```

Ερώτημα 3 (σειριακή αναζήτηση)

Γράψτε κώδικα που να υλοποιεί την σειριακή αναζήτηση ενός ονόματος πελάτη και να επιστρέφει τη θέση του πελάτη στον πίνακα. Αν ο πελάτης δεν υπάρχει να επιστρέφει την τιμή -1. Η συνάρτηση που θα κατασκευαστεί θα πρέπει να έχει την ακόλουθη μορφή.

```
int sequential_search(struct Customer* customers, int N,  string s) {
    // to be completed
}
```

Να κληθεί η συνάρτηση `sequential_search` για όνομα πελάτη "ΑΑΑ" και να εμφανιστεί το υπόλοιπο του λογαριασμού του αλλιώς να εμφανιστεί ότι δεν υπάρχει πελάτης με αυτό το όνομα.

```
string a_customer = "AAA";
int pos = sequential_search(customers, N, a_customer);
if (pos == -1)
      cout << "Ο πελάτης " << a_customer << " δεν υπάρχει" << endl;
else
      cout << "Το υπόλοιπο του πελάτη " << a_customer << " είναι "
           << customers[pos].balance << endl;
```

Η σειριακή αναζήτηση να επαναληφθεί 100 φορές για ονόματα που θα δημιουργηθούν με τον ίδιο τυχαίο τρόπο όπως στο ερώτημα 1 και να εμφανίζει το μέσο όρο αριθμού συγκρίσεων που απαιτήθηκαν από όλες τις αναζητήσεις.

Ερώτημα 4 (δυαδική αναζήτηση)

Γράψτε κώδικα που να υλοποιεί τη δυαδική αναζήτηση ενός ονόματος πελάτη και να επιστρέφει τη θέση του πελάτη στον πίνακα. Αν ο πελάτης δεν υπάρχει να επιστρέφει την τιμή -1. Η συνάρτηση που θα κατασκευαστεί θα πρέπει να έχει την ακόλουθη μορφή.

```
int binary_search(struct Customer* customers, int l, int r, string s) {
    // to be completed
}
```

Οι πελάτες να ταξινομηθούν αλφαβητικά με τη χρήση της συνάρτησης sort της STL. Να κληθεί η συνάρτηση `binary_search` για όνομα πελάτη "ΑΑΑ" και να εμφανιστεί το υπόλοιπο του λογαριασμού του αλλιώς να εμφανιστεί ότι δεν υπάρχει πελάτης με αυτό το όνομα.

```
sort(customers, customers + N);
pos = binary_search(customers, 0, N - 1, a_customer);
if (pos == -1)
      cout << "Ο πελάτης " << a_customer << " δεν υπάρχει" << endl;
else
      cout << "Το υπόλοιπο του πελάτη " << a_customer << " είναι "
```

```
<< customers[pos].balance << endl;
```

Όπως καις τη σειριακή αναζήτηση, η δυαδική αναζήτηση να επαναληφθεί 100 φορές για ονόματα που θα δημιουργηθούν με τον ίδιο τυχαίο τρόπο όπως στο ερώτημα 1 και να εμφανιστεί ο μέσος όρος αριθμού συγκρίσεων που απαιτήθηκαν από όλες τις αναζητήσεις.

Ερώτημα 5 (δυαδική αναζήτηση STL)

Με τη χρήση της συνάρτησης `std::lower_bound` της STL να πραγματοποιηθεί αναζήτηση για την εύρεση του πελάτη με όνομα “AAA”. Αν ο πελάτης δεν υπάρχει να εμφανιστεί κατάλληλο μήνυμα. Ο κώδικας για το ερώτημα αυτό δίνεται στη συνέχεια

```
struct Customer b_customer = { "AAA", 0 };
auto it = std::lower_bound(customers, customers + N, b_customer);
if (*it == b_customer) {
	cout << "Το υπόλοιπο του πελάτη " << it->name << " είναι " << it->balance <<endl;
} else {
	cout << "Ο πελάτης " << b_customer.name << " δεν υπάρχει" << endl;
}
```

Θα χρειαστεί να προστεθεί στον ορισμό της δομής Customer ο ακόλουθος κώδικας ο οποίος ορίζει την συμπεριφορά της δομής Customer για τον τελεστή ισότητας ==.

```
bool operator==(Customer other) {
	return name == other.name;
}
```

Ερώτηση: Μπορεί οι τρεις αλγόριθμοι αναζήτησης που χρησιμοποιήθηκαν για τον εντοπισμό του πελάτη με όνομα AAA να επιστρέψουν διαφορετικό αποτέλεσμα;