Εργασία 3η - Πίνακας Κατακερματισμού (HashTable)

Αναρτήθηκαν τα αρχεία ελέγχου και τα αρχεία εισόδου στο wiki

Σε αυτή την εργασία θα υλοποιήσετε την **αποθήκευση λέξεων** (*δηλ. αλφαριθμητικών που δεν περιέχουν κενά*) σε ένα πίνακα κατακερματισμού (*HashTable*). Οι πίνακες κατακερματισμού χρησιμοποιούνται για την γρήγορη ένθεση, αναζήτηση και διαγραφή στοιχείων μέσα σε αυτούς. Το μειονέκτημα των πινάκων κατακερματισμού είναι ότι κατά κανόνα χρησιμοποιούν περισσότερο χώρο από όσο πραγματικά χρειάζονται, πράγμα που τους καθιστά ακατάλληλους για μεγάλο όγκο δεδομένων.

Ένθεση στον πίνακα κατακερματισμού

Η θέση ένθεσης ενός στοιχείου στον πίνακα κατακερματισμού δίνεται από την συνάρτηση κατακερματισμού η οποία για την εργασία συνοψίζεται στα εξής:

$$h_n(x) = (x+n) \text{ modulo } TABLE\_SIZE$$

δηλαδή $h_0(x) = h(x)$, $h_1(x) = h(x+1)$, $h_2(x) = h(x+2)$ κ.ο.κ., όπου **x** είναι το άθροισμα των ASCII κωδικών των χαρακτήρων κάθε περιεχόμενης λέξης. Για παράδειγμα το **x** για την λέξη **apple** είναι

Χαρακτήρας	Κωδικός ASCII
'a'	97
'p'	112
'p'	112
'l'	108
'e'	101
Άθροισμα	**530**

Αρχικά η συνάρτηση κατακερματισμού επιχειρεί να εισάγει το νέο στοιχείο στην θέση που προκύπτει από την παραπάνω σχέση για **n=0**. Εάν η θέση είναι διαθέσιμη τότε το στοιχείο εισάγεται στην θέση αυτή. Εάν η θέση δεν είναι διαθέσιμη τότε προχωρά στην εξέταση της επόμενης θέσης προς ένθεση που δίνει η συνάρτηση κατακερματισμού για **n=1**. Εάν η νέα θέση είναι διαθέσιμη τότε το στοιχείο εισάγεται στην θέση αυτή, ενώ εάν δεν είναι διαθέσιμη προχωρά στην επόμενη θέση. Η διαδικασία συνεχίζεται μέχρι να βρούμε διαθέσιμη θέση ή μέχρι να επιστρέψουμε στην αρχική θέση που μας επέστρεψε η συνάρτηση κατακερματισμού για **n=0**.

Με βάση την συνάρτηση κατακερματισμού αναζητούμε την 1η διαθέσιμη προς ένθεση θέση. Μία διαθέσιμη θέση ορίζεται ως μία θέση που είτε δεν έχει αποθηκευτεί κανένα στοιχείο κατά το παρελθόν, είτε έχει αποθηκευτεί κάποιο στοιχείο, αλλά πλέον έχει διαγραφεί.

1. Εάν δεν έχει αποθηκευτεί κανένα στοιχείο κατά το παρελθόν τότε αντικείμενο τύπου ***string*** στην αντίστοιχη θέση είναι κενό.

2. Εάν έχει αποθηκευτεί κάποιο στοιχείο κατά το παρελθόν και στη συνέχεια έχει διαγραφεί, τότε το αντικείμενο ***string*** στην αντίστοιχη θέση έχει το περιεχόμενο **"##tomb##"** (*από τα αρχικά της λέξης tombstone*).

Σημείωση: Κατά την διαγραφή είναι απαραίτητο να βάλουμε ένα χαρακτηριστικό που να δηλώνει ότι στη συγκεκριμένη θέση υπήρχε στοιχείο που διαγράφηκε. Η προσθήκη αυτή είναι σημαντική κατά την αναζήτηση διότι ενημερώνει, ότι στη συγκεκριμένη θέση υπήρχε στοιχείο το οποίο πλέον έχει διαγραφεί

Αναζήτηση στον πίνακα κατακερματισμού

Η αναζήτηση στον πίνακα κατακερματισμού χρησιμοποιεί τη συνάρτηση κατακερματισμού $\mathbf{h_n(x)}$ για να εντοπίσει εάν υπάρχει το στοιχείο προς αναζήτηση στον πίνακα ή όχι. Η αναζήτηση ξεκινά για **n=0** και επαναλαμβάνεται για **n=1,2,3** κλπ εφόσον δεν βρεθεί η ζητούμενη λέξη. Η αναζήτηση επαναλαμβάνεται μόνο όταν εντοπίζονται θέσεις που έχουν περιεχόμενο διαφορετικό του επιθυμητού ή θέσεις έχουν διαγραφεί κατά το παρελθόν και περιέχουν τη δεσμευμένη λέξη **"##tomb##"**. Η αναζήτηση σταματά επιτυχώς εάν βρεθεί το επιθυμητό περιεχόμενο και ανεπιτυχώς εάν βρεθεί κενή θέση ή εάν επιστρέψουμε στην θέση του πίνακα από την οποία εκκινήσαμε την αναζήτηση για **n=0**.

Διαγραφή από τον πίνακα κατακερματισμού

Η επιτυχής διαγραφή προϋποθέτει ότι έχει γίνει επιτυχής αναζήτηση του στοιχείου προς διαγραφή. Κατά την διαγραφή είναι απαραίτητο να βάλουμε ένα χαρακτηριστικό που να δηλώνει ότι στη συγκεκριμένη θέση υπήρχε στοιχείο που διαγράφηκε. Κατά σύμβαση στην παρούσα εργασία τοποθετούμε την λέξη **"##tomb##"** .

Υλοποίηση εργασίας

Μέρος 1ο - Πίνακας κατακερματισμού

Προκειμένου να υλοποιήσετε την λειτουργία του πίνακα κατακερματισμού φτιάξτε την κλάση ***HashTable*** η οποία περιγράφεται παρακάτω:

Η κλάση *HashTable*

```
class HashTable {
protected:
  unsigned int size;      // ο αριθμός των αποθηκευμένων στοιχείων
  unsigned int capacity;  // η χωρητικότητα του πίνακα
  string *table;
  static int getHashCode(const char *str);

public:
  HashTable(int capacity=8);
  HashTable(const HashTable &ht);
  int getSize();
  int getCapacity();
  bool isEmpty(int pos);
  bool isTomb(int pos);
  bool isAvailable(int pos);
  bool contains(const string &s);
  bool contains(const char *s);
  virtual bool add(const string &s);
  virtual bool add(const char *s);
  virtual bool remove(const string &s);
  virtual bool remove(const char *s);
  void print();

  virtual bool operator << (string str);
  virtual bool operator >> (string str);
  HashTable operator+(HashTable &t);
  HashTable &operator+=(HashTable &t);
  HashTable &operator=(const HashTable &t);

};
```

Οι μέθοδοι της κλάσης *HashTable* είναι οι εξής:

Μέθοδος	Περιγραφή
`HashTable(int capacity=12);`	Κατασκευαστής που λαμβάνει το μέγεθος του HashTable.
`HashTable(const HashTable &ht);`	Copy constructor (κατασκευαστής αντιγραφέας)
`int getSize();`	Επιστρέφει τον αριθμό των αποθηκευμένων στοιχείων στον πίνακα
`int getCapacity();`	Επιστρέφει την μέγιστη χωρητικότητα του πίνακα.
`bool isEmpty(int pos);`	Επιστρέφει ***true*** εάν η συγκεκριμένη θέση του πίνακα είναι άδεια, διαφορετικά ***false***. Εάν **`pos >= capacity`** επιστρέφει ***false***.
`bool isTomb(int pos);`	Επιστρέφει ***true*** εάν η συγκεκριμένη θέση του πίνακα περιέχει τη λέξη "**`##tomb##`**" *(σηματοδοτώντας ότι έχει διαγραφεί κάποια λέξη κατά το παρελθόν)* διαφορετικά ***false***. Εάν **`pos >= capacity`** επιστρέφει ***false***.
`bool isAvailable(int pos);`	Επιστρέφει ***true*** εάν επιστρέφει ***true*** μία από τις συναρτήσεις **`isTomb`** και **`isEmpty`**.
`bool contains(const string &s);`	Επιστρέφει ***true*** εάν το αλφαριθμητικό περιέχεται στον πίνακα.
`bool contains(const char *s);`	Επιστρέφει ***true*** εάν το αλφαριθμητικό περιέχεται στον πίνακα.
`virtual bool add(const string &s);`	Ενθέτει το αλφαριθμητικό στον πίνακα, αφού προηγουμένως βεβαιωθεί ότι δεν υπάρχει. Επιστρέφει ***true*** εάν η ένθεση είναι επιτυχής, διαφορετικά επιστρέφει ***false***. Εάν το αλφαριθμητικό υπάρχει ήδη αποτυγχάνει.
`virtual bool add(const char * s);`	Ενθέτει το αλφαριθμητικό στον πίνακα, αφού προηγουμένως βεβαιωθεί ότι δεν υπάρχει. Επιστρέφει ***true*** εάν η ένθεση είναι επιτυχής, διαφορετικά επιστρέφει ***false***. Εάν το αλφαριθμητικό υπάρχει ήδη αποτυγχάνει.
`virtual bool remove(const string &s);`	Διαγράφει το αλφαριθμητικό από τον πίνακα εφόσον αυτό υπάρχει. Επιστρέφει ***true*** εάν η διαγραφή είναι επιτυχής, διαφορετικά επιστρέφει ***false***.
`virtual bool remove(const char *s);`	Διαγράφει το αλφαριθμητικό από τον πίνακα εφόσον αυτό υπάρχει. Επιστρέφει ***true*** εάν η διαγραφή είναι επιτυχής, διαφορετικά επιστρέφει ***false***.
`void print();`	Εκτυπώνει στο STDOUT το περιεχόμενο του πίνακα ως εξής: • Εκτυπώνει μία λέξη ανά γραμμή στη μορφή **`position -> word`** (*το **`position`** ξεκινά από 0*) • Στο τέλος εκτυπώνει τις τιμές **capacity** και **size**

	ως εξής: `capacity: CCCC, size: SSSS`
`virtual bool operator << (string str);`	Λειτουργικότητα όμοια με τη συνάρτηση `virtual bool add(const string &s);`
`virtual bool operator >> (string str);`	Λειτουργικότητα όμοια με τη συνάρτηση `virtual bool remove(const string &s);`
`HashTable operator+(HashTable &t);`	Προσθήκη δύο πινάκων και αποθήκευσης του αποτελέσματος σε ένα νέο τρίτο πίνακα. Οι χωρητικότητες τους αθροίζονται. Ο νέος πίνακας περιέχει τα στοιχεία και των δύο πινάκων (προφανώς σε διαφορετικές θέσεις από τις αρχικές).
`HashTable &operator+=(HashTable &t);`	Προσθήκη δύο πινάκων και αποθήκευση του αποτελέσματος στον αριστερό τελεστέο. Ο ανανεωμένος πίνακας περιέχει τα στοιχεία και των δύο πινάκων.
`HashTable &operator=(const HashTable &t);`	Ανάθεση του περιεχομένου του δεξιού τελεστέου στον αριστερό. Επιστρέφει μια αναφορά στον αριστερού τελεστέο.

Μέρος 2ο - Ανακατακερματισμός (Προαιρετικό Μέρος, bonus +5)

Σας ζητείται να κατασκευάσετε τη κλάση ***ExtensibleHashTable*** η οποία αποτελεί επέκταση της κλάσης ***HashTable*** και έχει τα εξής χαρακτηριστικά:

1. Κατά την ένθεση ενός στοιχείου εξετάζει εάν ο λόγος ***size/capacity*** είναι μεγαλύτερο από ένα προκαθορισμένο ποσοστό (***upper_bound_ratio***). Εάν ναι, διπλασιάζει τη χωρητικότητα του πίνακα, ενθέτοντας ξανά τα στοιχεία στον νέο πίνακα.
2. Κατά τη διαγραφή ενός στοιχείου εξετάζει εάν ο λόγος ***size/capacity*** είναι μικρότερος από ένα προκαθορισμένο ποσοστό (***lower_bound_ratio***). Εάν ναι, υπο-διπλασιάζει τη χωρητικότητα του πίνακα, ενθέτοντας ξανά τα στοιχεία στον νέο πίνακα.

Η κλάση ExtensibleHashTable περιγράφεται ως εξής:

Η κλάση *ExtensibleHashTable*

```
class ExtensibleHashTable: public HashTable {
private:
  double upper_bound_ratio, lower_bound_ratio;
  void rehash();
  void rehash(int newcapacity); Δεν χρειάζεται να την υλοποιήσετε.

public:
  ExtensibleHashTable(double upper_bound_ratio=0.5,
                      double lower_bound_ratio=0.125,
                      int capacity=8);
  ExtensibleHashTable(const ExtensibleHashTable &t);
  bool add(const string &str);
  bool remove(const string &str);
  bool operator << (string str);
  bool operator >> (string str);
  ExtensibleHashTable operator+(const ExtensibleHashTable &t) const;
  ExtensibleHashTable &operator+=(const ExtensibleHashTable &t);
  ExtensibleHashTable &operator=(const ExtensibleHashTable &t);
};
```

Οι μέθοδοι της κλάσης περιγράφονται ως εξής:

Μέθοδος	Περιγραφή

`ExtensibleHashTable(`
`  double  upper_bound_ratio=0.5,`
`  double lower_bound_ratio=0.125,`
`  int capacity=8);`
	Κατασκευαστής.

`ExtensibleHashTable(ExtensibleHashTable &t);`	Copy constructor
`bool add(const string &str);`	Όμοια με την συνάρτηση της γονικής κλάσης. Καλεί εσωτερικά τη συνάρτηση **`rehash(void)`**.
`bool remove(const string &str);`	Όμοια με την συνάρτηση της γονικής κλάσης. Καλεί εσωτερικά τη συνάρτηση **`rehash(void)`**.
`bool operator << (string str);`	Όμοια με την συνάρτηση της γονικής κλάσης. Καλεί εσωτερικά τη συνάρτηση **`rehash(void)`**. *Πιθανόν να σας αρκεί η υλοποίηση της γονικής κλάσης. Εάν δεν την υλοποιήσετε βάλτε τη σε σχόλια.*
`bool operator << (string str);`	Όμοια με την συνάρτηση της γονικής κλάσης. Καλεί

	εσωτερικά τη συνάρτηση **`rehash(void)`**. *Πιθανόν να σας αρκεί η υλοποίηση της γονικής κλάσης. Εάν δεν την υλοποιήσετε βάλτε τη σε σχόλια.*
`ExtensibleHashTable operator+(ExtensibleHashTable &t);`	Προσθήκη δύο πινάκων και αποθήκευσης του αποτελέσματος σε ένα νέο τρίτο πίνακα. Ο νέος πίνακας περιέχει τα στοιχεία και των δύο πινάκων (προφανώς σε διαφορετικές θέσεις από τις αρχικές).
`ExtensibleHashTable &operator+=(ExtensibleHashTable &t);`	Προσθήκη δύο πινάκων και αποθήκευση του αποτελέσματος στον αριστερό τελεστέο. Ο ανανεωμένος πίνακας περιέχει τα στοιχεία και των δύο πινάκων.
`ExtensibleHashTable &operator=(const ExtensibleHashTable &t);`	Ανάθεση του περιεχομένου του δεξιού τελεστέου στον αριστερό. Επιστρέφει μια αναφορά στον αριστερού τελεστέο.
`void rehash();`	Ελέγχει εάν θα πρέπει να γίνει rehash και υλοποιεί τη λειτουργία rehashing εφόσον απαιτείται.

Γενικές Οδηγίες

1. Μπορείτε να ορίσετε και δικές σας συμπληρωματικές συναρτήσεις μέλη της κλάσης ή φιλικές συναρτήσεις εφόσον το επιθυμείτε. Κατά κανόνα ορίζετε τις συναρτήσεις αυτές ως ***private***.

Τρόπος Αποστολής

Σε ένα φάκελο με όνομα το όνομα και το ΑΕΜ κάθε μέλους της ομάδας σας, αποθηκεύστε ΜΟΝΟ τα αρχεία πηγαίου κώδικα C++. Παράδειγμα φακέλου είναι το εξής: *GiorgosThanos_1234_PeterGordon_1235.*

Αφού συμπιέσετε το φάκελο σε ένα αρχείο ZIP με το ίδιο όνομα και κατάληξη ZIP ή TGZ ή TAR.GZ (**όχι RAR**), αποστείλετε την δουλειά σας με e-mail στην διεύθυνση **ce325.course@gmail.com** ως εξής:

Τίτλος (subject): CE325 hw03

Συνημμένο: Το παραπάνω συμπιεσμένο αρχείο.

Σώμα μηνύματος: Τα ονόματα και τα ΑΕΜ της ομάδας σας.

Εργασίες που δεν είναι συνεπείς με τους παραπάνω περιορισμούς δεν αξιολογούνται.