# Άσκηση 1
# Προγραμματισμός Συστήματος

Προθεσμία: 15 Μαρτίου 2015

Σ' αυτή την άσκηση καλείστε να αποσπάσετε την «κρυφή λέξη» από έναν «μάντη». Η επικοινωνία σας με τον μάντη περιορίζεται στην ανταλλαγή λέξεων (οι οποίες αναπαρίστανται ως συμβολοσειρές). Δοκιμάζοντας μία λέξη τη φορά, ο μάντης θα σας απαντήσει αν η λέξη που επιλέξατε είναι η σωστή, διαφορετικά θα σας επιστρέψει μία λίστα από λέξεις για να δοκιμάσετε στη συνέχεια. Το παιχνίδι τελειώνει όταν βρείτε την σωστή λέξη, οπότε και το πρόγραμμά σας θα πρέπει να τερματίζει.

Η υλοποίηση του μάντη θα σας δοθεί σε binary μορφή, ενώ θα έχει το εξής interface:

```
const char **oracle(const char *word);
```

Συγκεκριμένα, το όρισμα `word` είναι η λέξη που επιλέγετε να δοκιμάσετε, και το αποτέλεσμα είναι είτε `NULL` (που σημαίνει ότι το word ήταν η κρυφή λέξη) είτε ένας πίνακας από λέξεις/συμβολοσειρές, αγνώστου μεγέθους, ο οποίος θα είναι πάντα τερματισμένος με `NULL`. Η μνήμη που έχει δεσμευτεί για τον πίνακα των αποτελεσμάτων θα επαναχρησιμοποιηθεί σε επόμενες κλήσεις της `oracle()`, οπότε φροντίζετε να αντιγράφετε τα αποτελέσματά της (αν τα χρειάζεστε) πριν την καλέσετε ξανά. Μπορείτε να ξεκινήσετε την πρώτη κλήση δίνοντας ως όρισμα την κενή συμβολοσειρά.

Μπορεί κανείς να σκεφτεί την έρευνα που κάνετε σαν ένα (νοητό) δέντρο. Κάθε κόμβος είναι μία λέξη και τα παιδιά του είναι οι επόμενες λέξεις για έρευνα, όπως τις επιστρέφει η oracle, *εκτός* από όσες από αυτές τις λέξεις έχουν ερευνηθεί στο παρελθόν. Μια και η oracle επιστρέφει πίνακα άγνωστου μεγέθους, κάθε κόμβος έχει μεταβλητό αριθμό από παιδιά.

Το πρόβλημα είναι ότι ένα τέτοιο δέντρο γίνεται τεράστιο: θα έχει έναν κόμβο για κάθε λέξη που εξετάζετε. Για να αντιμετωπιστεί αυτό το πρόβλημα εφαρμόζουμε δύο ιδέες: α) κρατάτε το δέντρο μόνο μέχρι ένα συγκεκριμένο βάθος $d$ το οποίο είναι παράμετρος (command-line parameter) στο πρόγραμμά σας, β) για να μην πέσετε σε κύκλο και εξετάζετε συνεχώς τις ίδιες λέξεις, χρησιμοποιείτε μια τεχνική ανίχνευσης στοιχείων σε σύνολο, που ονομάζεται bloom filter. Το bloom filter κρατάει μια υπερ-εκτίμηση του τι υπάρχει μέσα στο (νοητό) δέντρο.

**Bloom Filter – Περιγραφή Δομής.** Ένα bloom filter (βλ. `http://en.wikipedia.org/wiki/Bloom_filter`) είναι μία συμπαγής δομή η οποία χρησιμοποιείται για τον έλεγχο της ύπαρξης ενός στοιχείου σε ένα σύνολο. Αναλυτικότερα, χρησιμοποιώντας το bloom filter για ένα δεδομένο στοιχείο, μπορούμε να εξάγουμε ένα από τα παρακάτω συμπεράσματα. Είτε ότι (i) το στοιχείο ***σίγουρα δεν ανήκει*** στο σύνολο, ή ότι (ii) το στοιχείο είναι ***πιθανόν να ανήκει*** στο σύνολο.

Δηλαδή το bloom filter μπορεί να απαντήσει θετικά ότι ένα στοιχείο ανήκει στο σύνολο, ακόμα κι όταν δεν ανήκει (false positive). Αντίθετα, η αρνητική απάντηση είναι πάντα έγκυρη. Με αυτόν το τρόπο, το bloom filter κάνει μία υπερεκτίμηση των στοιχείων του συνόλου. Η πιθανότητα λάθους (σε περίπτωση θετικής απάντησης) αυξάνεται όσο προστίθενται στοιχεία στο σύνολο.

Ένα bloom filter αναπαρίσταται ως ένας πίνακας από bits $M$ θέσεων (αρχικοποιημένος με 0), και συνοδεύεται από $K$ hash functions. Ακολουθούν οι βασικές πράξεις που υποστηρίζει και το πως υλοποιούνται:

1. Για να εισάγουμε ένα στοιχείο $e$ στο σύνολο, εφαρμόζουμε κάθε μία από αυτές τις $K$ hash functions στο εν λόγω στοιχείο. Αυτές θα επιστρέψουν $K$ θέσεις στον πίνακα (που κάποιες πιθανόν να συμπίπτουν). Έπειτα, σε κάθε μία από αυτές τις θέσεις θέτουμε το αντίστοιχο bit σε 1.

2. Για να ελέγξουμε αν ένα στοιχείο $e$ ανήκει στο σύνολο, εφαρμόζουμε κάθε μία από αυτές τις $K$ hash functions στο εν λόγω στοιχείο. Αν σε κάθε μία από τις $K$ θέσεις που μας επιστρέφουν υπάρχουν 1, τότε το στοιχείο $e$ (πιθανόν) να ανήκει στο σύνολο. Διαφορετικά (αν υπάρχει έστω κι ένα 0), σίγουρα το στοιχείο δεν ανήκει στο σύνολο.

**Παράδειγμα** Έστω $M = 16, K = 3$, και $hash_1(), hash_2(), hash_3()$ οι τρεις hash functions που σας έχουν δοθεί. Ο πίνακας των bits αρχικά θα είναι: $\underbrace{00\,00\,00\,00\;\;00\,00\,00\,00}_{16}$.

Έστω ότι εισάγουμε διαδοχικά τα στοιχεία $s_1, s_2, s_3$, για τα οποία έχουμε:

| | | |
|---|---|---|
| $hash_1(s_1) = 3 \bmod 16$ | $hash_1(s_2) = 10 \bmod 16$ | $hash_1(s_3) = 2 \bmod 16$ |
| $hash_2(s_1) = 7 \bmod 16$ | $hash_2(s_2) = 1 \bmod 16$ | $hash_2(s_3) = 4 \bmod 16$ |
| $hash_3(s_1) = 11 \bmod 16$ | $hash_3(s_2) = 7 \bmod 16$ | $hash_3(s_3) = 2 \bmod 16$ |
| Bit Array $\Rightarrow \underbrace{00\,01\,00\,01\;\;00\,01\,00\,00}_{16}$ | $\Rightarrow \underbrace{01\,01\,00\,01\;\;00\,11\,00\,00}_{16}$ | $\Rightarrow \underbrace{01\,11\,10\,01\;\;00\,11\,00\,00}_{16}$ |

Έπειτα, ελέγχουμε την ύπαρξη των στοιχείων $s_4, s_5, s_3$, για τα οποία έχουμε:

| | | |
|---|---|---|
| $hash_1(s_4) = 5 \bmod 16$ | $hash_1(s_5) = 2 \bmod 16$ | $hash_1(s_3) = 2 \bmod 16$ |
| $hash_2(s_4) = 6 \bmod 16$ | $hash_2(s_5) = 10 \bmod 16$ | $hash_2(s_3) = 4 \bmod 16$ |
| $hash_3(s_4) = 1 \bmod 16$ | $hash_3(s_5) = 11 \bmod 16$ | $hash_3(s_3) = 2 \bmod 16$ |
| Bit Array $\Rightarrow \underbrace{\underline{01}\,11\,\underline{10}\,\underline{01}\;\;00\,11\,00\,00}_{16}$ | $\Rightarrow \underbrace{01\,\underline{11}\,10\,01\;\;00\,\underline{11}\,00\,00}_{16}$ | $\Rightarrow \underbrace{01\,\underline{11}\,\underline{10}\,01\;\;00\,11\,00\,00}_{16}$ |
| Answer : *no* | *yes* | *yes* |

Στη περίπτωση του $s_5$ έχουμε ένα false positive αφού το στοιχείο φαίνεται πως ανήκει στο σύνολο, ενώ στην πραγματικότητα δεν ανήκει. Οι υπόλοιπες δύο απαντήσεις είναι έγκυρες.

**Δενδρική Δομή – Περιγραφή Δομής.** Η δομή που πρέπει να υλοποιήσετε είναι ένα δένδρο που κάθε κόμβος μπορεί να έχει ένα οποιοδήποτε αριθμό από παιδιά (k-ary tree). Το δένδρο κατασκευάζεται δυναμικά μετά από κάθε ανάκτηση ενός αποτελέσματος της oracle. Για κάθε μία συμβολοσειρά, ελέγχετε το bloom filter και σε περίπτωση αρνητικής απάντησης [1] δεσμεύεται ένας καινούριος κόμβος ο οποίος συνδέεται με τον αντίστοιχο πατέρα. Η διαδικασία επαναλαμβάνεται για κάθε νέο αποτέλεσμα της oracle.

**Εκτέλεση Προγράμματος** Το πρόγραμμα θα καλείται ως εξής, από τη γραμμή εντολών:

```
./invoke-oracle [-k NUM] SIZE DEPTH
```

1. Η προαιρετική επιλογή '`-k NUM`' καθορίζει το πλήθος των hash functions που θα χρησιμοποιηθούν για το bloom filter. Θεωρείστε ότι η default τιμή είναι 3, αν παραλειφθεί.
2. Η παράμετρος `SIZE` καθορίζει το μέγεθος του bloom filter σε *bytes*. (Χρησιμοποιείστε τη σταθερά `CHAR_BIT` του `limits.h` για να μετατρέψετε σε bits.) Ενδεικτικό μέγεθος του bloom filter, για τα δεδομένα της άσκησης, θα είναι της τάξης των $2^{20}$ bits.
3. Η παράμετρος `DEPTH` καθορίζει το μέγιστο βάθος του δένδρου.

### Λεπτομέρειες Υλοποίησης

– Πριν την εκτέλεση του αλγορίθμου, πρέπει να αρχικοποιήσετε την `oracle()` καλώντας την συνάρτηση `void initSeed(int seed)`.

– Θα πρέπει να αξιοποιήσετε το bloom filter για να ελέγχετε αν έχετε εξετάσει ξανά τις συμβολοσειρές που επιστρέφει η `oracle()`. Ανάλογα με το αποτέλεσμα του ελέγχου (και με το τρέχον βάθος του δένδρου) αποφασίζετε αν θα εισαχθεί το αποτέλεσμα στο δένδρο.

– Το πρόγραμμά σας θα τερματίζει είτε:

**με επιτυχία** όταν βρεθεί η κρυφή λέξη, οπότε και θα πρέπει να εκτυπώνετε αυτή τη λέξη καθώς και το δένδρο αναζήτησης που έχετε κατασκευάσει μέχρι αυτό το σημείο, ή

**με αποτυχία** όταν εξαντληθούν όλες οι λέξεις που κληθήκατε να δοκιμάσετε από την `oracle()` δίχως να βρείτε την ζητούμενη. Σε αυτή τη περίπτωση, ο χρήστης θα πρέπει να ενημερώνεται αναλόγως με κάποιο μήνυμα.

[1] Θυμίζουμε ότι αρνητική απάντηση σημαίνει ότι το στοιχείο *δεν* ανήκει στο δένδρο.

**Hash Functions** Εκτός της `oracle()` θα σας παρέχεται επίσης και ένα σύνολο από hash functions (για την υλοποίηση του bloom filter), οι οποίες θα είναι προσβάσιμες μέσω της ακόλουθης συνάρτησης:

```
uint64_t hash_by(int i, const char *word);
```

Αυτή, δεδομένη μίας συμβολοσειράς `word` και ενός δείκτη `i`, θα καλεί την $i$-οστή hash function και θα σας επιστρέφει το αποτέλεσμα (τύπου `uint64_t`, δηλαδή 64-bit μη-προσημασμένου ακεραίου).

**Διαδικαστικά:**
- Το πρόγραμμά σας θα πρέπει να τρέχει στα μηχανήματα Linux/Unix της σχολής.
- Για επιπρόσθετες ανακοινώσεις, παρακολουθείτε το forum του μαθήματος στο piazza.com για να δείτε ερωτήσεις/απαντήσεις/διευκρινήσεις που δίνονται σχετικά με την άσκηση. Η παρακολούθηση του φόρουμ στο piazza.com είναι *υποχρεωτική*.

**Τι πρέπει να παραδοθεί:**
1. Όλη η δουλειά σας σε ένα tar-file που να περιέχει όλα τα source files, header files, Makefile. **Προσοχη:** φροντίστε να τροποποιήσετε τα δικαιώματα αυτού του αρχείου πριν την υποβολή, π.χ. με την εντολή `chmod 755 OnomaEponymoProject1.tar` (περισσότερα στη σελίδα του μαθήματος).
2. Ένα README (απλό text αρχείο) με κάποια παραδείγματα μεταγλώττισης και εκτέλεσης του προγράμματός σας. Επίσης, μπορείτε να συμπεριλάβετε άλλες διευκρινήσεις που κρίνετε απαραίτητες (για τυχόν παραδοχές που έχετε κάνει, κτλ).
3. Οποιαδήποτε πηγή πληροφορίας, συμπεριλαμβανομένου και κώδικα που μπορεί να βρήκατε στο Διαδίκτυο θα πρέπει να αναφερθεί και στον πηγαίο κώδικά σας αλλά και στο παραπάνω README.

**Τι θα βαθμολογηθεί:**
1. Η συμμόρφωση του κώδικά σας με τις προδιαγραφές της άσκησης.
2. Η οργάνωση και η αναγνωσιμότητα (μαζί με την ύπαρξη σχολίων) του κώδικα.
3. Η χρήση Makefile και η κομματιαστή σύμβολο-μετάφραση (separate compilation).

**Άλλες σημαντικές παρατηρήσεις:**
1. Οι εργασίες είναι **ατομικές**.
2. Όποιος υποβάλλει / δείχνει κώδικα που δεν έχει γραφτεί από την ίδια/τον ίδιο **μηδενίζεται** στο μάθημα.
3. Αν και αναμένεται να συζητήσετε με φίλους και συνεργάτες το πως θα επιχειρήσετε να δώσετε λύση στο πρόβλημα, *αντιγραφή κώδικα* (οποιασδήποτε μορφής) είναι κάτι που **δεν επιτρέπεται** και δεν πρέπει να γίνει. Οποιοσδήποτε βρεθεί αναμεμειγμένος σε αντιγραφή κώδικα *απλά παίρνει μηδέν* στο μάθημα. Αυτό ισχύει για *όλους όσους εμπλέκονται*, ανεξάρτητα από το ποιος έδωσε/πήρε κλπ.
4. Το πρόγραμμά σας θα πρέπει να γραφτεί σε C (ή C++ χωρίς όμως STL extensions).
5. Η υλοποίηση της `oracle()` ενδέχεται να αλλάξει. Το πρόγραμμά σας θα πρέπει να λειτουργεί σωστά για οποιαδήποτε έκδοσή της.