Οικονομικό Πανεπιστήμιο Αθηνών, Τμήμα Πληροφορικής
Μάθημα: Προγραμματισμός Υπολογιστών με C++
Ακαδημαϊκό έτος: 2011–12
Διδάσκων: Ι. Ανδρουτσόπουλος

4η Εργασία

Η εργασία αυτή ολοκληρώνει την κατασκευή του φίλτρου ανεπιθύμητων διαφημιστικών μηνυμάτων (ΑΔΜ), με τη συγγραφή του κώδικα που παράγει το αρχείο των λέξεων-κλειδιών.

Ζητείται να γράψετε ένα πρόγραμμα C++ το οποίο θα παράγει ένα αρχείο της μορφής του *keywords.txt* της 1ης εργασίας. Το πρόγραμμα θα πρέπει να δέχεται κατά σειρά τις εξής παραμέτρους από τη γραμμή εντολών:

- *hamKeywords*: Επιθυμητός αριθμός λέξεων-κλειδιών που είναι ενδεικτικές επιθυμητών μηνυμάτων.
- *spamKeywords*: Επιθυμητός αριθμός λέξεων-κλειδιών που είναι ενδεικτικές ΑΔΜ.
- *hamFilenames*: Το αρχείο που περιέχει τα μονοπάτια των επιθυμητών μηνυμάτων της συλλογής εκπαίδευσης. Το αρχείο αυτό πρέπει να έχει την ίδια μορφή με το *ling_filenames.txt* της 1ης εργασίας.
- *spamFilenames*: Το αρχείο με τα μονοπάτια των ΑΔΜ της συλλογής εκπαίδευσης. Το αρχείο αυτό πρέπει να έχει την ίδια μορφή με το *spam_filenames.txt* της 1ης εργασίας.
- *threshold*: Κατώφλι αριθμού εμφανίσεων μιας λέξης (βλ. παρακάτω). Η προεπιλεγμένη τιμή να είναι 10.

Κατά τις δοκιμές του προγράμματός σας, μπορείτε να χρησιμοποιήσετε ως συλλογή εκπαίδευσης τα μηνύματα των *ling.tar.gz* και *spam.tar.gz* της 1ης εργασίας.

Για κάθε μία λέξη *w* που εμφανίζεται σε τουλάχιστον *threshold* μηνύματα της συλλογής εκπαίδευσης (στα μηνύματα του *hamFilenames* και *spamFilenames* μαζί), το πρόγραμμά σας θα πρέπει να υπολογίζει τις εξής ποσότητες:

$$spamPrecision(w) = S(w) / N(w),$$

$$hamPrecision(w) = H(w) / N(w),$$

$$spamRecall(w) = S(w) / Nspam$$

$$hamRecall(w) = H(w) / Nham$$

$$spamFmeasure(w) = \frac{2 \cdot spamPrecision(w) \cdot spamRecall(w)}{spamPrecision(w) + spamRecall(w)}$$

$$hamFmeasure(w) = \frac{2 \cdot hamPrecision(w) \cdot hamRecall(w)}{hamPrecision(w) + hamRecall(w)}$$

όπου *S(w)* είναι ο αριθμός των ΑΔΜ εκπαίδευσης στα οποία εμφανίζεται η *w*, *H(w)* είναι ο αριθμός των επιθυμητών μηνυμάτων εκπαίδευσης στα οποία εμφανίζεται η *w*, *N(w)* είναι ο αριθμός των μηνυμάτων εκπαίδευσης στα οποία εμφανίζεται η *w*, *Nspam* είναι ο συνολικός αριθμός ΑΔΤ εκπαίδευσης και *Nham* ο συνολικός αριθμός επιθυμητών μηνυμάτων εκπαίδευσης.

Το πρόγραμμά σας θα πρέπει να τυπώνει στο *cout* τις *hamKeywords* (ή λιγότερες, αν δεν υπάρχουν τόσες) λέξεις *w* με το υψηλότερο *hamFmeasure(w)* κατά φθίνουσα σειρά *hamFmeasure(w)* και κατόπιν τις *spamKeywords* (ή λιγότερες, αν δεν υπάρχουν τόσες) λέξεις *w* με το υψηλότερο *spamFmeasure(w)* κατά φθίνουσα σειρά *spamFmeasure(w)*. Οι λέξεις θα πρέπει να τυπώνονται μία ανά γραμμή. Το πρόγραμμά σας δεν θα πρέπει να τυπώνει τίποτα άλλο στο *cout*, εκτός αν έχει ενεργοποιηθεί η επιλογή *-scores* (βλ. παρακάτω).

Παράδειγμα κλήσης του προγράμματός σας (έστω ότι το εκτελέσιμο αρχείο λέγεται *myprogram*) από τη γραμμή εντολών:

```
c:\> myprogram 100 100 ling_filenames.txt
spam_filenames.txt 3 > keywords.txt
```

Το πρόγραμμά σας θα πρέπει να υποστηρίζει, επίσης, την προαιρετική επιλογή *-scores* στη γραμμή εντολών, όπως φαίνεται στο παρακάτω παράδειγμα κλήσης του:

```
c:\> myprogram -scores 100 100 ling_filenames.txt
spam_filenames.txt 3 > keywords.txt
```

Στην περίπτωση που έχει ενεργοποιηθεί η επιλογή *-scores*, το πρόγραμμά σας θα πρέπει να τυπώνει σε κάθε γραμμή στο *cout*, μετά από την κάθε λέξη *w*, κατά σειρά το *spamFmeasure(w)* και το *hamFmeasure(w),* χωρισμένα μεταξύ τους με κενά (ένα κενό μεταξύ *w* και *spamFmeasure(w)* και άλλο ένα μεταξύ *spamFmeasure(w)* και *hamFmeasure(w)*).

Το πρόγραμμά σας θα πρέπει να χρησιμοποιεί εσωτερικά μια τάξη *FeatureSelector* που να παρέχει επαναλήπτες και να μπορεί να χρησιμοποιηθεί ως ακολούθως (τα *hamFilenames*, *spamFilenames*, *threshold* είναι μεταβλητές που αντιστοιχούν στα ομώνυμα ορίσματα της γραμμής εντολών, ενώ το *withScores* είναι Boolean μεταβλητή που δείχνει αν η γραμμή εντολών περιείχε ή όχι την επιλογή *-scores*).

```
FeatureSelector selector(hamFilenames,
                         spamFilenames, threshold);
FeatureSelector::const_iterator iter =
   selector.hamBegin();
unsigned featuresPrinted = 0;
while(featuresPrinted < hamKeywords &&
      iter != selector.hamEnd()) {
```

```
      iter.print(cout, withScores);
      iter++; featuresPrinted++;
   }
   iter = selector.spamBegin();
   featuresPrinted = 0;
   while(featuresPrinted < spamKeywords &&
         iter != selector.spamEnd()) {
      iter.print(cout, withScores);
      iter++; featuresPrinted++;
   }
```

Υπόδειξη: Μπορείτε να ταξινομήσετε ένα *vector<Class>* αντικειμένων μιας τάξης *Class* χρησιμοποιώντας τη συνάρτηση *sort* (της βιβλιοθήκης *algorithm*), αρκεί να έχετε ορίσει τον τελεστή *operator<* της τάξης *Class*. Για παράδειγμα:

```
#include <algorithm>
…
class WordScore {
   string word;
   double score;
public:
   WordScore(const string& w, const double sc) : word(w),
score(sc) { }
   string getWord() const { return word; }
   double getScore() const { return score; }
   bool operator<(const WordScore& right) const {
      return (score < right.score); }
};
…
vector<WordScore> wordScoreVector;
…
sort(wordScoreVector.begin(), wordScoreVector.end());
```

Συμπληρωματικές οδηγίες:

- Θα πρέπει να περιλάβετε στο πρόγραμμά σας πλήρεις ελέγχους λαθών και δυνατότητες ανάνηψης από σφάλματα (π.χ. αν ένα αρχείο εισόδου δεν είναι δυνατόν να διαβαστεί, να ζητείται από το χρήστη να δώσει άλλο όνομα αρχείου κλπ).
- Μπορείτε να χρησιμοποιήσετε οποιεσδήποτε βιβλιοθήκες της C++.
- Μη συμπεριλάβετε μηνύματα εκπαίδευσης στα αρχεία που θα παραδώσετε.
- Το αρχείο *keywords.txt* που είχατε χρησιμοποιήσει στην πρώτη εργασία είχε δημιουργηθεί χρησιμοποιώντας διαφορετικές μεθόδους επιλογής λέξεων-κλειδιών. Η λέξεις-κλειδιά που θα τυπώνει το πρόγραμμά αυτής της εργασίας ενδέχεται να είναι διαφορετικές, έστω κι αν χρησιμοποιείτε τα ίδια μηνύματα εκπαίδευσης που είχατε χρησιμοποιήσει στην 1^{η} εργασία.