

Καθηγητής Π. Λουρίδας

Τμήμα Διοικητικής Επιστήμης και Τεχνολογίας

Οικονομικό Πανεπιστήμιο Αθηνών

## Λεκτικά Παραλληλόγραμμα

Ένα λεκτικό παραλληλόγραμμα  $m \times n$  είναι ένας πίνακας που οι γραμμές του είναι λέξεις με  $n$  γράμματα, ενώ οι στήλες του είναι λέξεις με  $m$  γράμματα. Για παράδειγμα, τα παρακάτω είναι λεκτικά παραλληλόγραμμα  $5 \times 6$ :

|   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|
| C | U | R | S | E | D | A | B | A | S | E | D |
| O | P | E | N | L | Y | B | A | N | T | E | R |
| U | P | S | I | D | E | O | S | I | E | R | Y |
| L | E | A | P | E | R | U | T | O | P | I | A |
| D | R | Y | E | R | S | T | E | N | S | E | D |

Πώς μπορούμε να βρούμε τέτοια λεκτικά παραλληλόγραμμα; Αν έχουμε στη διάθεσή μας λίστες λέξεων με  $m$  και  $n$  χαρακτήρες, τότε μπορούμε να προσπαθήσουμε να χτίσουμε ένα τέτοιο παραλληλόγραμμα γεμίζοντας τις στήλες τη μία μετά την άλλη με λέξεις  $m$  χαρακτήρων, εξασφαλίζοντας ταυτόχρονα ότι στις γραμμές προκύπτουν λέξεις  $n$  χαρακτήρων.

Έστω για παράδειγμα ότι ξεκινάμε με μια λέξη πέντε γραμμάτων και την τοποθετούμε στην πρώτη στήλη:

C  
O  
U  
L  
D

Αυτό σημαίνει ότι το παραλληλόγραμμό μας θα έχει οριζόντια μια λέξη που ξεκινάει με C, μια λέξη που ξεκινάει με O, μια λέξη που ξεκινάει με U, μια λέξη που ξεκινάει με U, μια λέξη που ξεκινάει με L, και μια λέξη που ξεκινάει με D.

Έστω τώρα ότι στη δεύτερη στήλη επιλέγουμε τη λέξη UPPER. Τότε το παραλληλόγραμμό μας μεγαλώνει:

C U  
O P  
U P  
L E  
D R

Βλέπουμε ότι το παραλληλόγραμμό μας θα πρέπει να έχει οριζόντια πέντε λέξεις που ξεκινούν με CU, OP, UP, LE, και DR.

Αυτό μας δείχνει ότι όχι μόνο δεν μπορούμε να επιλέγουμε λέξεις στην τύχη, αλλά θα πρέπει οι λέξεις που θα επιλέγουμε για τις στήλες να είναι τέτοιες ώστε να υπάρχουν λέξεις με τα αντίστοιχα προθέματα που προκύπτουν οριζόντια στις γραμμές. Για να το πετύχουμε αυτό, θα πρέπει να έχουμε έναν μηχανισμό ώστε να μπορούμε να ελέγχουμε αποτελεσματικά ότι τέτοιες λέξεις πράγματι υπάρχουν. Για να επιλέξουμε τη λέξη UPPER, είναι προαπαιτούμενο να υπάρχουν λέξεις με πρόθεμα CU, OP, UP, LE, και DR, και θέλουμε να μπορούμε να το διαπιστώσουμε αυτό εύκολα.

Μια δομή δεδομένων η οποία ακριβώς εξυπηρετεί αυτόν τον σκοπό, δηλαδή την αποθήκευση λέξεων σύμφωνα με το πρόθεμά τους, είναι η δομή *trie* (από το *retrieval*, αλλά προφέρεται *try*). Ένα *trie* είναι ένα δένδρο, οι κόμβοι του οποίου έχουν  $M$  στοιχεία, όπου  $M$  είναι τα στοιχεία του αλφαβήτου μας. Κάθε κόμβος στο επίπεδο  $l$  του δένδρου αντιπροσωπεύει όλες τις λέξεις, ή όπως τις αποκαλούμε *κλειδιά*, που ξεκινούν με το ίδιο πρόθεμα μήκους  $l$ . Ο κόμβος έχει  $M$  παιδιά, ένα για κάθε δυνατό  $l + 1$  χαρακτήρα των κλειδιών που καταχωρούμε στο *trie*.

Ως αλφάβητο θα θεωρήσουμε τους κεφαλαίους χαρακτήρες του αγγλικού αλφαβήτου, συν έναν επιπλέον ειδικό χαρακτήρα ( $\_$ ) που θα χρησιμοποιούμε για να δείξουμε το τέλος μιας λέξης. Μπορούμε να αποθηκεύσουμε ένα *trie* σαν αυτό που περιγράψαμε σε έναν πίνακα με  $M$  γραμμές. Η κάθε στήλη του πίνακα θα αντιστοιχεί σε έναν κόμβο του *trie*. Στον παρακάτω πίνακα βλέπουμε ένα τέτοιο *trie* στο οποίο έχουμε αποθηκεύσει τις 31 πιο κοινές λέξεις στα αγγλικά. Η στήλη 1 είναι η ρίζα του δένδρου και εκεί αναζητούμε τον πρώτο χαρακτήρα της κάθε λέξης. Αν ο πρώτος χαρακτήρας είναι N, βρίσκουμε αμέσως τη λέξη NOT στη θέση N, δηλαδή τη θέση 15, του κόμβου. Αν ο πρώτος χαρακτήρας είναι ο A, πηγαίνουμε στη θέση A του κόμβου, η οποία μας υποδεικνύει να αναζητήσουμε τη λέξη στη στήλη 2. Αν η λέξη είναι η A, τότε τη βρίσκουμε στη θέση  $\_$ , αφού δεν έχει επιπλέον γράμματα και είμαστε στο τέλος της λέξης. Διαφορετικά, οι υπόλοιπες από τις λέξεις οι οποίες ξεκινούν από A βρίσκονται αποθηκευμένες στην αντίστοιχη θέση του κόμβου: η AND στη θέση N, η R στη θέση ARE, κ.λπ. Η στήλη 2, ως κόμβος παιδί της στήλης 1, περιέχει τις λέξεις εκείνες που έχουν κοινό πρόθεμα έναν χαρακτήρα, στην περίπτωση μας το A.

Με την ίδια λογική, ας δούμε τι συμβαίνει με τις λέξεις που ξεκινούν από H. Η στήλη 1 του πίνακα μας λέει να προχωρήσουμε την αναζήτηση στη στήλη 5. Η στήλη 5 λοιπόν θα περιέχει όλες τις λέξεις που έχουν κοινό πρόθεμα το γράμμα H. Βλέπουμε ότι η στήλη 5 δείχνει σε δύο άλλες στήλες, την 6 και την 7. Η στήλη 6, ως κόμβος στο δεύτερο επίπεδο του *trie*, περιέχει λέξεις που έχουν κοινό πρόθεμα δύο γραμμάτων: τις λέξεις HAD και HAVE με το πρόθεμα HA. Η στήλη 7, πάλι ως κόμβος στο δεύτερο επίπεδο του *trie*, περιέχει επίσης λέξεις που έχουν κοινό πρόθεμα δύο γραμμάτων: τις λέξεις HE και HER με το πρόθεμα HE.

|   | (1)  | (2) | (3) | (4)  | (5) | (6)  | (7) | (8) | (9) | (10) | (11) | (12)  |
|---|------|-----|-----|------|-----|------|-----|-----|-----|------|------|-------|
| ␣ |      | A   |     |      |     |      | HE  | I   |     |      |      |       |
| A | (2)  |     |     |      | (6) |      |     |     |     |      | THAT | WAS   |
| B | (3)  |     |     |      |     |      |     |     |     |      |      |       |
| C |      |     |     |      |     |      |     |     |     |      |      |       |
| D |      |     |     |      |     | HAD  |     |     |     |      |      |       |
| E |      |     | BE  |      | (7) |      |     |     |     |      | THE  |       |
| F | (4)  |     |     |      |     |      |     |     | OF  |      |      |       |
| G |      |     |     |      |     |      |     |     |     |      |      |       |
| H | (5)  |     |     |      |     |      |     |     |     | (11) |      | WHICH |
| I | (8)  |     |     |      | HIS |      |     |     |     |      | THIS | WITH  |
| J |      |     |     |      |     |      |     |     |     |      |      |       |
| K |      |     |     |      |     |      |     |     |     |      |      |       |
| L |      |     |     |      |     |      |     |     |     |      |      |       |
| M |      |     |     |      |     |      |     |     |     |      |      |       |
| N | NOT  | AND |     |      |     |      |     | IN  | ON  |      |      |       |
| O | (9)  |     |     | FOR  |     |      |     |     |     | TO   |      |       |
| P |      |     |     |      |     |      |     |     |     |      |      |       |
| Q |      |     |     |      |     |      |     |     |     |      |      |       |
| R |      | ARE |     | FROM |     |      | HER |     | OR  |      |      |       |
| S |      | AS  |     |      |     |      |     | IS  |     |      |      |       |
| T | (10) | AT  |     |      |     |      |     | IT  |     |      |      |       |
| U |      |     | BUT |      |     |      |     |     |     |      |      |       |
| V |      |     |     |      |     | HAVE |     |     |     |      |      |       |
| W | (12) |     |     |      |     |      |     |     |     |      |      |       |
| X |      |     |     |      |     |      |     |     |     |      |      |       |
| Y | YOU  |     | BY  |      |     |      |     |     |     |      |      |       |
| Z |      |     |     |      |     |      |     |     |     |      |      |       |

Η διαδικασία της αναζήτησης σε ένα τέτοιο trie μπορεί να περιγραφεί ως εξής.

1. Ξεκινάμε από τη ρίζα του trie. Η ρίζα είναι ο τρέχων κόμβος που επισκεπτόμαστε.
2. Για κάθε χαρακτήρα της λέξης (θεωρούμε ότι η κάθε λέξη τελειώνει με τον ειδικό χαρακτήρα που συμβολίσαμε με ␣):
  - 2.1 Πηγαίνουμε στην αντίστοιχη θέση μέσα στον τρέχοντα κόμβο. Η θέση στον κόμβο είναι αυτή που αντιστοιχεί στον χαρακτήρα που εξετάζουμε.
  - 2.2 Αν η θέση δείχνει σε μια άλλη στήλη, δηλαδή κόμβο, του trie, τότε πηγαίνουμε σε εκείνη την στήλη, η οποία είναι πλέον ο τρέχων κόμβος που επισκεπτόμαστε και συνεχίζουμε στο βήμα 2. Αν δεν δείχνει σε μια άλλη στήλη του κόμβου, τότε ή δεν θα δείχνει πουθενά, ή θα πρέπει να περιέχει μια λέξη. Αν δεν δείχνει πουθενά, σταματάμε τη διαδικασία και επιστρέφουμε ότι δεν βρήκαμε τη λέξη, και τον τρέχοντα κόμβο όπου σταμάτησε η ανεπιτυχής αναζήτηση. Αν περιέχει μια λέξη, ελέγχουμε αν η λέξη αυτή είναι η λέξη που αναζητάμε. Αν είναι, σταματάμε τη διαδικασία και επιστρέφουμε ότι βρήκαμε τη λέξη και τον τρέχοντα κόμβο, όπου σταμάτησε η επιτυχής αναζήτηση. Αν δεν είναι, σταματάμε τη διαδικασία και επιστρέφουμε ότι δεν βρήκαμε τη λέξη και τον τρέχοντα κόμβο, όπου σταμάτησε η ανεπιτυχής αναζήτηση.

Για να δημιουργήσουμε ένα τέτοιο trie, εργαζόμαστε παρόμοια. Ξεκινάμε δημιουργώντας έναν κενό κόμβο, δηλαδή μία στήλη του πίνακα με  $M$  θέσεις που δεν περιέχουν κάποια λέξη ούτε δείχνουν σε κάποια άλλη στήλη. Αυτός ο κόμβος είναι η ρίζα του trie.

Για κάθε λέξη που θέλουμε να εισάγουμε:

1. Αναζητούμε τη λέξη στο trie.
2. Αν βρούμε τη λέξη, σταματάμε.
3. Αν δεν βρούμε τη λέξη, τότε:
  - 3.1 Αν στον κόμβο που σταμάτησε ανεπιτυχώς η αναζήτηση αυτό συνέβη διότι στη θέση του κόμβου που την αναζητήσαμε δεν υπάρχει τίποτε, εισάγουμε τη λέξη εκεί και σταματάμε. Αν στον κόμβο που σταμάτησε ανεπιτυχώς η αναζήτηση αυτό συνέβη διότι στη θέση που σταματήσαμε υπάρχει άλλη λέξη, τότε πρέπει να επεκτείνουμε το trie.
  - 3.2 Συνεχίζουμε εξετάζοντας τον κάθε χαρακτήρα της λέξης που βρήκαμε αποθηκευμένη στο trie, από τον χαρακτήρα που έχουμε μέχρι τώρα ταιριάζει και έπειτα:
    - i. Δημιουργούμε έναν νέο κόμβο στο trie.
    - ii. Στη θέση του τρέχοντα κόμβου που βρισκόμαστε δείχνουμε τον νέο κόμβο που φτιάξαμε.
    - iii. Ελέγχουμε αν επόμενος χαρακτήρας της λέξης που εισάγουμε είναι ο ίδιος με τον επόμενο χαρακτήρα της λέξης που βρήκαμε στο trie. Αν είναι, προχωράμε έναν χαρακτήρα παρακάτω και στις δύο λέξεις, θέτουμε ως τρέχοντα κόμβο το νέο κόμβο που μόλις φτιάξαμε, και επιστρέφουμε στο βήμα 3.2.i. Αν δεν είναι, τότε αποθηκεύουμε στον νέο κόμβο τη λέξη που βρήκαμε στο trie και τη λέξη που εισάγουμε στο trie στις αντίστοιχες θέσεις του κόμβου (που διαφέρουν).

Η ουσία της διαδικασίας είναι η επανάληψη στο 3.2: μεγαλώνουμε το trie προσθέτοντας κόμβους όσο η λέξη που θέλουμε να εισάγουμε έχει κοινό πρόθεμα με τη λέξη που βρήκαμε στο trie.

Έχοντας δει πώς δουλεύει ένα trie, μπορούμε να κατασκευάσουμε παραλληλόγραμμα λέξεων ως εξής. Έστω ότι θέλουμε να κατασκευάσουμε παραλληλόγραμμα  $5 \times 6$ . Χτίζουμε το παραλληλόγραμμα στήλη-στήλη, παίρνοντας με τη σειρά έξι λέξεις πέντε γραμμάτων:  $w_1 w_2 w_3 w_4 w_5 w_6$ . Έστω ότι έχουμε επιλέξει δύο λέξεις πέντε γραμμάτων και αναζητούμε την τρίτη λέξη για να την τοποθετήσουμε στην τρίτη στήλη. Θα συμβολίζουμε με  $w_i^j$  το  $j$  γράμμα της  $i$  λέξης. Η κατάσταση θα είναι όπως φαίνεται παρακάτω:

$$\begin{array}{ccc} w_1^1 & w_2^1 & w_3^1 \\ w_1^2 & w_2^2 & w_3^2 \\ w_1^3 & w_2^3 & w_3^3 \\ w_1^4 & w_2^4 & w_3^4 \\ w_1^5 & w_2^5 & w_3^5 \end{array}$$

Αναζητούμε στο trie κάθε ένα από τα  $w_1^1 w_2^1 w_3^1, w_1^2 w_2^2 w_3^2, \dots, w_1^5 w_2^5 w_3^5$ . Αν η αναζήτηση ενός από αυτά καταλήξει σε κόμβο του trie και σε θέση που δεν δείχνει πουθενά, τότε δεν υπάρχει λέξη έξι γραμμάτων με αυτό το πρόθεμα. Επομένως ξέρουμε ότι δεν μπορούμε να προχωρήσουμε χρησιμοποιώντας την λέξη  $w_3$  που δοκιμάζουμε, και θα πρέπει να δοκιμάσουμε μια άλλη λέξη  $w'_3$  εναλλακτικά. Αν εξαντλήσουμε όλες τις λέξεις πέντε γραμμάτων στην τρίτη στήλη χωρίς επιτυχία, τότε σημαίνει ότι πρέπει να πισωγυρίσουμε και να δοκιμάσουμε άλλη λέξη πέντε γραμμάτων  $w'_2$  από την  $w_2$  που είχαμε τοποθετήσει στη δεύτερη στήλη, κ.ο.κ. Αν χρησιμοποιήσουμε με επιτυχία έξι λέξεις  $w_1 w_2 w_3 w_4 w_5 w_6$ , τότε έχουμε βρει ένα λεκτικό παραλληλόγραμμο. Βεβαίως δεν σημαίνει ότι υπάρχει μόνο ένα. Στην πραγματικότητα εξερευνούμε συστηματικά τις εξάδες  $w_1 w_2 w_3 w_4 w_5 w_6$ , προσέχοντας όμως ώστε αν η αναζήτηση στο trie αποτύχει για ένα υποσύνολο της εξάδας να μην προχωρήσουμε μάταια δοκιμάζοντας τις υπόλοιπες λέξεις της εξάδας.

Σκοπός της εργασίας είναι η συγγραφή προγράμματος που βρίσκει λεκτικά τετράγωνα παίρνοντας στην είσοδο λέξεις μήκους  $m$  και  $n$  χαρακτήρων.

## Απαιτήσεις Προγράμματος

Κάθε φοιτητής θα εργαστεί σε αποθετήριο στο GitHub. Για να αξιολογηθεί μια εργασία θα πρέπει να πληροί τις παρακάτω προϋποθέσεις:

- Για την υποβολή της εργασίας θα χρησιμοποιηθεί το ιδιωτικό αποθετήριο του φοιτητή που δημιουργήθηκε για τις ανάγκες του μαθήματος και του έχει αποδοθεί. Το αποθετήριο αυτό έχει όνομα του τύπου `username-algo-assignments`, όπου `username` είναι το όνομα του φοιτητή στο GitHub. Για παράδειγμα, το σχετικό αποθετήριο του διδάσκοντα θα ονομαζόταν `louridas-algo-assignments` και θα ήταν προσβάσιμο στο <https://github.com/dmst-algorithms-course/louridas-algo-assignments>. *Τυχόν άλλα αποθετήρια απλώς θα αγνοηθούν.*
- Μέσα στο αποθετήριο αυτό θα πρέπει να δημιουργηθεί ένας κατάλογος `assignment-2024-4`.
- Μέσα στον παραπάνω κατάλογο το πρόγραμμα θα πρέπει να αποθηκευτεί με το όνομα `word_rectangles.py`.
- Δεν επιτρέπεται η χρήση έτοιμων βιβλιοθηκών γράφων ή τυχόν έτοιμων υλοποιήσεων των αλγορίθμων, ή τμημάτων αυτών, εκτός αν αναφέρεται ρητά ότι

επιτρέπεται.

- Επιτρέπεται η χρήση δομών δεδομένων της Python όπως στοίβες, λεξικά, σύνολα, κ.λπ.
- Επιτρέπεται η χρήση των παρακάτω βιβλιοθηκών ή τμημάτων τους όπως ορίζεται:
  - `sys.argv`
  - `argparse`
- Το πρόγραμμα θα πρέπει να είναι γραμμένο σε Python 3.
- Η εργασία είναι αποκλειστικά ατομική. Δεν επιτρέπεται συνεργασία μεταξύ φοιτητών στην εκπόνησή της, με ποινή το μηδενισμό. Επιπλέον η εργασία δεν μπορεί να είναι αποτέλεσμα συστημάτων Τεχνητής Νοημοσύνης (όπως ChatGPT). Ειδικότερα όσον αφορά το τελευταίο σημείο προσέξτε ότι τα συστήματα αυτά χωλαίνουν στην αλγοριθμική σχέση, άρα τυχόν προτάσεις που κάνουν σε σχετικά θέματα μπορεί να είναι λανθασμένες. Επιπλέον, αν θέλετε να χρησιμοποιήσετε τέτοια συστήματα, τότε αν και κάποιος άλλος το κάνει αυτό, μπορεί οι προτάσεις που θα λάβετε να είναι παρόμοιες, οπότε οι εργασίες θα παρουσιάσουν ομοιότητες και άρα θα μηδενιστούν.
- Η έξοδος του προγράμματος θα πρέπει να περιλαμβάνει μόνο ό,τι φαίνεται στα παραδείγματα που παρατίθενται. *Η φλυαρία δεν επιβραβεύεται.*

### Χρήση του GitHub

Όπως αναφέρθηκε το πρόγραμμά σας για να αξιολογηθεί θα πρέπει να αποθηκευθεί στο GitHub. Επιπλέον, θα πρέπει το GitHub να χρησιμοποιηθεί καθόλη τη διάρκεια της ανάπτυξης του.

Αυτό σημαίνει ότι *δεν θα ανεβάσετε στο GitHub απλώς την τελική λύση του προβλήματος μέσω της λειτουργίας "Upload files"*. Στο GitHub θα πρέπει να φαίνεται *το ιστορικό της συγγραφής του προγράμματος*. Αυτό συνάδει και με τη φιλοσοφία του εργαλείου: λέμε "commit early, commit often". Εργαζόμαστε σε ένα πρόγραμμα και κάθε μέρα, ή όποια στιγμή έχουμε κάνει κάποιο σημαντικό βήμα, αποθηκεύουμε την αλλαγή στο GitHub. Αυτό έχει σειρά ευεργετικών αποτελεσμάτων:

- Έχουμε πάντα ένα αξιόπιστο εφεδρικό μέσο στο οποίο μπορούμε να ανατρέξουμε αν κάτι πάει στραβά στον υπολογιστή μας (μας γλιτώνει από πανικούς του τύπου: ένα βράδυ πριν από την παράδοση ο υπολογιστής μας ή ο δίσκος του πνέει τα λοίσθια, και εμείς τι θα υποβάλουμε στον Λουρίδα;).
- Καθώς έχουμε πλήρες ιστορικό των σημαντικών αλλαγών και εκδόσεων του προγράμματός μας, μπορούμε να επιστρέψουμε σε μία προηγούμενη αν συνειδητοποιήσουμε κάποια στιγμή ότι πήραμε λάθος δρόμο (μας γλιτώνει από πανικούς του τύπου: μα το πρωί δούλευε σωστά, τι στο καλό έκανα και τώρα δεν παίζει τίποτα;).

- Καθώς φαίνεται η πρόοδος μας στο GitHub, επιβεβαιώνουμε ότι το πρόγραμμα δεν είναι αποτέλεσμα της όποιας επιφοίτησης (μας γλιτώνει από πανικούς του τύπου: καλά, πώς το έλυσες το πρόβλημα αυτό με τη μία, μόνος σου;).
- Αν δουλεύετε σε μία ομάδα που βεβαίως δεν είναι καθόλου η περίπτωση μας εδώ, μπορούν όλοι να εργάζονται στα ίδια αρχεία στο GitHub εξασφαλίζοντας ότι ο ένας δεν γράφει πάνω στις αλλαγές του άλλου. Παρά το ότι η εργασία αυτή είναι ατομική, καλό είναι να αποκτάτε τριβή με το εργαλείο git και την υπηρεσία GitHub μιας και χρησιμοποιούνται ευρέως και όχι μόνο στη συγγραφή κώδικα.

Άρα επενδύστε λίγο χρόνο στην εκμάθηση των git / GitHub, των οποίων ο σωστός τρόπος χρήσης είναι μέσω γραμμής εντολών (command line), ή ειδικών προγραμμάτων (clients) ή μέσω ενοποίησης στο περιβάλλον ανάπτυξης (editor, IDE).

### Τελικό Πρόγραμμα

Το πρόγραμμα θα καλείται ως εξής (όπου python η κατάλληλη εντολή στο εκάστοτε σύστημα):

```
python word_rectangles.py print_trie horizontal_words
```

ή

```
python word_rectangles.py find_rectangles horizontal_words vertical_words n_rows
```

Η σημασία των παραμέτρων του προγράμματος είναι:

- `print_trie` αν δίνεται αυτή η παράμετρος, το πρόγραμμα θα τυπώνει στην οθόνη το trie που κατασκευάζει και θα σταματάει.
- `find_rectangles` αν δίνεται αυτή η παράμετρος, το πρόγραμμα θα αναζητά λεκτικά παραλληλόγραμμα.
- Η παράμετρος `horizontal_words` δείχνει το αρχείο που περιέχει τις λέξεις  $n$  γραμμών (τις οριζόντιες λέξεις στο παραλληλόγραμμα). Προσέξτε, το αρχείο δεν θα ονομάζεται απαραίτητως `horizontal_words`, απλώς έτσι το παραθέτουμε εδώ, μπορεί να έχει οποιοδήποτε όνομα.
- Η παράμετρος `vertical_words` δείχνει το αρχείο που περιέχει τις λέξεις  $m$  γραμμών (τις κάθετες λέξεις στο παραλληλόγραμμα). Προσέξτε, το αρχείο δεν θα ονομάζεται απαραίτητως `vertical_words`, απλώς έτσι το παραθέτουμε εδώ, μπορεί να έχει οποιοδήποτε όνομα.
- `n_rows` δείχνει τον αριθμό των γραμμών του παραλληλογράμμου

Σκοπός της εργασίας είναι η συγγραφή προγράμματος που θα εντοπίζει τις περιόδους δραστηριότητας ενός συστήματος με τον τρόπο που περιγράψαμε. Υλοποιήσεις άλλων τρόπων δεν είναι αποδεκτές.

## Παραδείγματα

### Παράδειγμα 1

Αν ο χρήστης του προγράμματος δώσει:

```
python word_rectangles.py print_trie top_31.txt
```

το πρόγραμμά σας θα διαβάσει το αρχείο [top\\_31.txt](#). Το αρχείο αυτό περιέχει τις 31 πιο κοινές λέξεις στα αγγλικά. Στην έξοδο το πρόγραμμά σας θα πρέπει να εμφανίσει ακριβώς τα παρακάτω:

```
[0, 'A', 0, 0, 0, 0, 'HE', 'I', 0, 0, 0, 0]
A [2, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 'THAT', 'WAS']
B [3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
C [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
D [0, 0, 0, 0, 0, 'HAD', 0, 0, 0, 0, 0, 0, 0]
E [0, 0, 'BE', 0, 7, 0, 0, 0, 0, 0, 0, 'THE', 0]
F [4, 0, 0, 0, 0, 0, 0, 0, 0, 'OF', 0, 0, 0]
G [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
H [5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 0, 'WHICH']
I [8, 0, 0, 0, 'HIS', 0, 0, 0, 0, 0, 0, 'THIS', 'WITH']
J [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
K [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
L [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
M [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
N ['NOT', 'AND', 0, 0, 0, 0, 0, 0, 'IN', 'ON', 0, 0, 0]
O [9, 0, 0, 'FOR', 0, 0, 0, 0, 0, 0, 'TO', 0, 0]
P [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Q [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
R [0, 'ARE', 0, 'FROM', 0, 0, 'HER', 0, 'OR', 0, 0, 0]
S [0, 'AS', 0, 0, 0, 0, 0, 0, 'IS', 0, 0, 0, 0]
T [10, 'AT', 0, 0, 0, 0, 0, 0, 'IT', 0, 0, 0, 0]
U [0, 0, 'BUT', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
V [0, 0, 0, 0, 0, 'HAVE', 0, 0, 0, 0, 0, 0, 0]
W [12, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
X [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Y ['YOU', 0, 'BY', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Z [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Όπως βλέπετε, πρόκειται για το trie που δείξαμε παραπάνω. Στις θέσεις των κόμβων που δεν δείχνουν πουθενά, εμφανίζουμε το 0.

### Παράδειγμα 2

Αν ο χρήστης του προγράμματος δώσει:

```
python word_rectangles.py find_rectangles sgb-words_uc.txt sgb-words_uc.txt 5
```



τότε το πρόγραμμά σας θα αρχίσει να κατασκευάζει λεκτικά παραλληλόγραμμα (για την ακρίβεια στην περίπτωση μας τετράγωνα)  $5 \times 5$  με οριζόντιες λέξεις και κάθετες λέξεις από το αρχείο [sgb-words\\_uc.txt](#). Το αρχείο `sgb-words_uc.txt` προέρχεται από το συνοδευτικό υλικό του βιβλίου του Donald Knuth [The Stanford GraphBase: A Platform for Combinatorial Computing](#). Το πρόγραμμά σας θα τυπώνει τα τετράγωνα που θα βρίσκει το ένα κάτω από το άλλο, όπως:

```
W H I P S
H E L L O
I L I A C
C L A N K
H O C K S
```

```
W H I S H
H E L L O
I L I A C
C L A C K
H O C K S
```

Προσοχή, τα τετράγωνα είναι πολλά, το πρόγραμμά σας θα τρέχει μέχρι να το διακόψει ο χρήστης. Επίσης, δεν είναι απαραίτητο ότι τα δύο πρώτα που θα βρείτε είναι τα παραπάνω.

### Παράδειγμα 3

Αν ο χρήστης του προγράμματος δώσει:

```
python word_rectangles.py find_rectangles words_6.txt sgb-words_uc.txt 6
```

τότε το πρόγραμμά σας θα αρχίσει να κατασκευάζει λεκτικά παραλληλόγραμμα  $5 \times 6$  με οριζόντιες λέξεις από το αρχείο [words\\_6.txt](#) και κάθετες λέξεις από το αρχείο [sgb-words\\_uc.txt](#). Το αρχείο `words_6.txt` περιέχει τις επιτρεπόμενες λέξεις έξι γραμμμάτων στο παιχνίδι Scrabble. Το πρόγραμμά σας θα τυπώνει τα παραλληλόγραμμα που θα βρίσκει το ένα κάτω από το άλλο, όπως:

```
W R E S T S
H A P T I C
I D E A T A
C I E L E D
H O S E R S
```

```
W R A S S E
H A P P I S
I D E A L S
C I R C L E
H O S E Y S
```

Προσοχή, τα παραλληλόγραμμα είναι πολλά, το πρόγραμμά σας θα τρέχει μέχρι να το διακόψει ο χρήστης. Επίσης, δεν είναι απαραίτητο ότι τα δύο πρώτα που θα βρείτε

είναι τα παραπάνω.

#### Παράδειγμα 4

Αν ο χρήστης του προγράμματος δώσει:

```
python word_rectangles.py find_rectangles words_4.txt words_3.txt 4
```

τότε το πρόγραμμά σας θα αρχίσει να κατασκευάζει λεκτικά παραλληλόγραμμα 4×3 με οριζόντιες λέξεις από το αρχείο `words_4.txt` και κάθετες λέξεις από το αρχείο `words_3.txt`. Το αρχείο `words_4.txt` περιέχει τις επιτρεπόμενες λέξεις τεσσάρων γραμμάτων στο παιχνίδι Scrabble και το αρχείο `words_3.txt` περιέχει τις επιτρεπόμενες λέξεις τριών γραμμάτων στο παιχνίδι Scrabble. Το πρόγραμμά σας θα τυπώνει τα παραλληλόγραμμα που θα βρίσκει το ένα κάτω από το άλλο, όπως:

```
A A L S  
A B U T  
H A Z Y
```

```
A A L S  
A B Y E  
H A E N
```

Προσοχή, τα παραλληλόγραμμα είναι πολλά, το πρόγραμμά σας θα τρέχει μέχρι να το διακόψει ο χρήστης. Επίσης, δεν είναι απαραίτητο ότι τα δύο πρώτα που θα βρείτε είναι τα παραπάνω.

Καλή Επιτυχία!