Καθηγητής Π. Λουρίδας

Τμήμα Διοικητικής Επιστήμης και Τεχνολογίας

Οικονομικό Πανεπιστήμιο Αθηνών

Λεκτικά Παραλληλόγραμμα

Ένα λεκτικό παραλληλόγραμμο $m \times n$ 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 5×6:

```
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 αυτό το παραλληλόγραμμο επιλέγοντας τη μία μετά την άλλη 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

```
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 ως δεύτερη λέξη επιλέγουμε τη λέξη `UPPER`. Τότε το παραλληλόγραμμό μας μεγαλώνει:

```
C U
O P
U P
L E
D R
```

Από αυτό 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 κάθετα να είναι τέτοιες ώστε να υπάρχουν λέξεις με τα αντίστοιχα προθέματα που προκύπτουν οριζόντια. 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`.

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 re*trie*val, 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, η καλύτερα *κλειδιά*, που ξεκινούν με το ίδιο πρόθεμα μήκους l. Ο κόμβος έχει M παιδιά, ένα για κάθε δυνατό $l + 1$ χαρακτήρα.

Μπορούμε να αποθηκεύσουμε ένα τέτοιο trie σε έναν πίνακα με M γραμμές, όπου M είναι ο αριθμός των χαρακτήρων του αλφαβήτου συν ένα που θα χρησιμοποιούμε για να υποδείξουμε το τέλος μιας λέξης. 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 trie, 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. Διαφορετικά, οι υπόλοιπες από τις λέξεις οι οποίες ξεκινούν από `A` βρίσκονται αποθηκευμένες στην αντίστοιχη θέση του πίνακα: 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 δέυτερο 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 κενό χαρακτήρα που συμβολίσαμε με ␣):
   2.1 Πηγαίνουμε στην αντίστοιχη θέση του κόμβου. Η θέση του κόμβου είναι αυτή που αντιστοιχεί στον χαρακτήρα που εξετάζουμε.
   2.2 Αν η θέση δείχνει σε μια άλλη στήλη του κόμβου, 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 θα πρέπει να περιέχει μια λέξη. Ελέγχουμε λοιπόν 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 26 + 1 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 ταιριάξει 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 (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. Χτίζουμε το παραλληλόγραμμο 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 επόμενη στήλη. Θα συμβολίζουμε με w_i^j το j γράμμα της i λέξης.

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

Τότε, αναζητούμε στο trie κάθε ένα από τα $w_1^1 w_2^1 w_3^1$, $w_1^2 w_2^2 w_3^2$, ..., $w_1^5 w_2^5 w_3^5$. 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 δύο γραμμάτων από αυτή 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 τίποτε;).
- 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 [-p] horizontal_words vertical_words n_rows
```

Η σημασία των παραμέτρων του προγράμματος είναι:

- Η παράμετρος `horizontal_words` δείχνει το αρχείο που περιέχει τις λέξεις *n* γραμμάτων (τις οριζόντιες λέξεις στο παραλληλόγραμμο). Προσέξτε, το αρχείο δεν θα ονομάζεται απαραιτήτως `horizontal_words`, απλώς έτσι το παραθέτουμε εδώ, μπορεί να έχει οποιοδήποτε όνομα.
- Η παράμετρος `vertical_words` δείχνει το αρχείο που περιέχει τις λέξεις *m* γραμμάτων (τις κάθετες λέξεις στο παραλληλόγραμμο). Προσέξτε, το αρχείο δεν θα ονομάζεται απαραιτήτως `vertivcal_words`, απλώς έτσι το παραθέτουμε εδώ, μπορεί να έχει οποιοδήποτε όνομα.
- `n_rows` δείχνει τον αριθμό των γραμμών του παραλληλογράμμου
- `-p` αν δίνεται το πρόγραμμα θα εκτυπώνει στην έξοδο το trie που θα κατασκευάσει και θα σταματά.

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

Παραδείγματα

Παράδειγμα 1

Αν ο χρήστης του προγράμματος δώσει:

```
python bursts.py viterbi two_states.txt
```

το πρόγραμμά σας θα διαβάσει το αρχείο `two_states.txt`. Το αρχείο αυτό περιέχει τις χρονικές στιγμές στις οποίες το σύστημα εξέπεμψε μηνύματα. Στην έξοδο το πρόγραμμά σας θα πρέπει να εμφανίσει ακριβώς τα παρακάτω:

```
0 [0.0 30.0)
1 [30.0 35.0)
0 [35.0 40.0)
```

Αυτό σημαίνει ότι το σύστημα βρίσκεται στην κατάσταση 0 από τη χρονική στιγμή 0 μέχρι τη χρονική στιγμή 30, τη χρονική στιγμή 30 μεταπίπτει στην κατάσταση 1, και τη χρονική στιγμή 35 επιστρέφει στην κατάσαση 0. Επομένως, εντοπίσαμε μία

περίοδο αυξημένης δραστηριότητας στο διάστημα από τη χρονική στιγμή 30 έως τη χρονική στιγμή 35.

Παράδειγμα 2

Αν ο χρήστης του προγράμματος δώσει:

```
python bursts.py viterbi two_states.txt -d
```

Όπως και πριν, το πρόγραμμά σας θα διαβάσει το αρχείο `two_states.txt`. Στην έξοδο το πρόγραμμά σας θα πρέπει να εμφανίσει ακριβώς τα παρακάτω (στρογγυλοποιούμε στα δύο δεκαδικά ψηφία):

```
[0, inf, inf, inf, inf, inf, inf]
[3.74, 7.5, 13.5, 24.0, 43.51, 81.01, 154.52]
[7.48, 11.24, 17.24, 27.75, 47.25, 84.75, 158.26]
[11.22, 14.98, 20.98, 31.49, 50.99, 88.5, 162.0]
[12.94, 14.67, 16.62, 19.03, 22.33, 27.44, 36.14]
[14.66, 15.92, 17.63, 20.03, 23.34, 28.44, 37.15]
[16.37, 17.17, 18.64, 21.04, 24.34, 29.45, 38.15]
[18.09, 18.42, 19.64, 22.04, 25.35, 30.45, 39.16]
[19.81, 19.66, 20.65, 23.05, 26.35, 31.46, 40.16]
[22.28, 22.71, 25.25, 31.26, 41.76, 61.26, 98.77]
10 [0, 0, 0, 0, 1, 1, 1, 1, 1, 0]
0 [0.0 30.0)
1 [30.0 35.0)
0 [35.0 40.0)
```

Οι πρώτες δέκα γραμμές δίνουν, με ακρίβεια δύο δεκαδικών ψηφίων, την αρχική τιμή της γραμμής μηδέν του πίνακα C και τη γραμμή t του πίνακα C στο τέλος κάθε επανάληψης των γραμμών 1–12 του αλγορίθμου.

Στη συνέχεια εμφανίζεται μια γραμμή με τον αριθμό και τις καταστάσεις στις οποίες βρίσκεται σε κάθε χρονική στιγμή το σύστημα, και ακολουθεί η υπόλοιπη έξοδος του προγράμματος όπως προηγουμένως.

Παράδειγμα 3

Αν ο χρήστης του προγράμματος δώσει:

```
python bursts.py trellis two_states.txt
```

το πρόγραμμά σας θα διαβάσει το αρχείο το ίδιο αρχείο όπως και πριν και θα εμφανίσει την ίδια έξοδο, πλην όμως χρησιμοποιώντας τον αλγόριθμο Bellman-Ford:

```
0 [0.0 30.0)
1 [30.0 35.0)
0 [35.0 40.0)
```

Παράδειγμα 4

Αν ο χρήστης του προγράμματος δώσει:

```
python bursts.py trellis two_states.txt -d
```

το πρόγραμμά σας θα λειτουργήσει όπως στο προηγούμενο παράδειγμα, αλλά θα εμφανίσει επιπλέον στην έξοδο την εξέλιξη των χαλαρώσεων (relaxations) κατά την εκτέλεση του αλγορίθμου Bellman Ford. Έτσι, η γραμμή:

```
(9, 5) 62.48 -> 61.26 from (8, 2) 20.65 + 6.59 + 34.03
```

δείχνει ότι το κόστος του μονοπατιού για τον κόμβο `(9, 5)`, που αντιστοιχεί στην κατάσταση q_5 τη χρονική στιγμή t_9, χαλάρωσε από `62.48` σε `62.21` προερχόμενοι από τον κόμβο `(8, 2)`, που αντιστοιχεί στην κατάσταση q_2 την προηγούμενη χρονική στιγμή t_8. Το μονοπάτι μέχρι τον `(8, 2)` έχει κόστος `20.65`, ενώ `6.59` είναι το κόστος της μετάβασης μεταξύ των κόμβων και `34.03` το κόστος εκπομπής. Οι αριθμοί θα εμφανίζονται με ακρίβεια δύο δεκαδικών ψηφίων. Για λόγους οικονομίας χώρου παρακάτω εμφανίζονται μόνο οι δύο πρώτες και οι δύο τελευταίες γραμμές των χαλαρώσεων—το πρόγραμμά σας θα τις εμφανίζει όλες.

Στη συνέχεια εμφανίζεται μια γραμμή με τον αριθμό και τις καταστάσεις στις οποίες βρίσκεται σε κάθε χρονική στιγμή το σύστημα, και ακολουθεί η υπόλοιπη έξοδος του προγράμματος όπως προηγουμένως.

```
(1, 0) inf -> 3.74 from (0, 0) 0.00 + 0.00 + 3.74
(1, 1) inf -> 7.50 from (0, 0) 0.00 + 2.20 + 5.30
...
(9, 5) 62.48 -> 61.26 from (8, 2) 20.65 + 6.59 + 34.03
(9, 6) 99.98 -> 98.77 from (8, 2) 20.65 + 8.79 + 69.33
10 [0, 0, 0, 0, 1, 1, 1, 1, 1, 0]
0 [0.0 30.0)
1 [30.0 35.0)
0 [35.0 40.0)
```

Παράδειγμα 5

Αν ο χρήστης του προγράμματος δώσει:

```
python bursts.py viterbi three_states_1.txt
```

το πρόγραμμά σας θα διαβάσει το αρχείο three_states_1.txt και θα εμφανίσει στην έξοδο ακριβώς τα παρακάτω:

```
0 [0.0 410.0)
1 [410.0 450.0)
2 [450.0 469.0)
1 [469.0 600.0)
0 [600.0 710.0)
1 [710.0 800.0)
0 [800.0 1000.0)
```

Παράδειγμα 6

Αν ο χρήστης του προγράμματος δώσει:

```
python bursts.py trellis three_states_1.txt
```

το πρόγραμμά σας θα διαβάσει πάλι το αρχείο three_states_1.txt και θα εμφανίσει στην έξοδο τα ίδια όπως στο προηγούμενο παράδειγμα, χρησιμοποιώντας τον αλγόριθμο Bellman-Ford:

```
0 [0.0 410.0)
1 [410.0 450.0)
2 [450.0 469.0)
1 [469.0 600.0)
0 [600.0 710.0)
1 [710.0 800.0)
0 [800.0 1000.0)
```

Παράδειγμα 7

Αν ο χρήστης του προγράμματος δώσει:

```
python bursts.py viterbi three_states_2.txt -s 3 -g 0.5
```

το πρόγραμμά σας θα διαβάσει το αρχείο three_states_2.txt και θα εμφανίσει στην έξοδο τα παρακάτω:

```
0 [0.0 400.0)
1 [400.0 450.0)
2 [450.0 470.0)
1 [470.0 601.0)
0 [601.0 700.0)
1 [700.0 800.0)
0 [800.0 1000.0)
```

Παράδειγμα 8

Αν ο χρήστης του προγράμματος δώσει:

```
python bursts.py trellis three_states_2.txt -s 3 -g 0.5
```

το πρόγραμμά σας θα διαβάσει το αρχείο του προηγούμενου παραδείγματος και θα βγάλει την ίδια έξοδο, χρησιμοποιώντας τον αλγόριθμο Bellman-Ford:

```
0 [0.0 400.0)
1 [400.0 450.0)
2 [450.0 470.0)
1 [470.0 601.0)
0 [601.0 700.0)
1 [700.0 800.0)
0 [800.0 1000.0)
```

Παράδειγμα 9

Αν ο χρήστης του προγράμματος δώσει:

```
python bursts.py viterbi four_states.txt -s 1.1 -g 0.025
```

το πρόγραμμά σας θα διαβάσει το αρχείο `four_states.txt` και θα εμφανίσει στην έξοδο τα παρακάτω:

```
0 [2.4 14.02)
2 [14.02 22.99)
0 [22.99 33.81)
6 [33.81 46.29)
0 [46.29 59.87)
1 [59.87 66.21)
```

Προσέξτε ότι, αφού οι καταστάσεις του συστήματος αντιστοιχούν σε συγκεκριμένες εκθετικές κατανομές, δεν είναι απαραίτητο οι καταστάσεις που προκύπτουν να έχουν μεταξύ τους διαφορά ένα.

Παράδειγμα 10

Αν ο χρήστης του προγράμματος δώσει:

```
python bursts.py trellis four_states.txt -s 1.1 -g 0.025
```

το πρόγραμμά σας θα διαβάσει το αρχείο του προηγούμενου παραδείγματος και θα εμφανίσει την ίδια έξοδο, χρησιμοποιώντας όμως τον αλγόριθμο Bellman-Ford:

```
0 [2.4 14.02)
2 [14.02 22.99)
0 [22.99 33.81)
6 [33.81 46.29)
0 [46.29 59.87)
1 [59.87 66.21)
```

Περισσότερες Πληροφορίες

Η έρευνα για τα πρότυπα αλληλεπίδρασης στα κοινωνικά μέσα, και η οποία για τον εντοπισμό των εξάρσεων χρησιμοποιεί την προσέγγιση Viterbi που παρουσιάσαμε εδώ, είναι η [**avalle:2024**]. Η προσέγγιση αυτή για την εντοπισμό εξάρσεων προτάθηκε από τον Jon Kleinberg [**kleinberg:2003**], ο οποίος τον δοκίμασε στην προσωπική ηλεκτρονική του αλληλογραφία. Ο αλγόριθμος Viterbi πήρε το όνομά του από τον Andrew Viterbi που τον παρουσίασε το 1967 [**viterbi:1967**], αν και έχει εφευρεθεί ανεξάρτητα από διάφορους ερευνητές· στοιχεία για τον αλγόριθμο μπορείτε να δείτε στη σχετική σελίδα της Wikipedia.

Καλή Επιτυχία!