Assignment 3

r e a d m e

ΣΩΤΗΡΙΑ ΚΑΣΤΑΝΑ, 2995

ΜΕΡΟΣ Ι

>py algorithmA.py K

Αρχικά ανοίγω τα δύο ταξινομημένα αρχεία με βάση το πεδίο ‘instance weight’ το καθένα από τα οποία περιέχει τις εγγραφές για τους άντρες και τις γυναίκες αντίστοιχα (‘males_sorted.txt’ & ‘females_sorted.txt’). Χρησιμοποιώ μια μεταβλητή ‘gender’ η οποία μεταβάλλεται από Male σε Female κάθε φορά και έτσι με βοηθάει στην σωστή εναλλαγή μεταξύ τους. Επίσης αρχικοποιώ τον μετρητή k, ο οποίος είναι αυτός που θα κρατάει το πλήθος των join results που έχουν τυπωθεί και όταν φτάσει ίσο με το K θα σταματήσει ο αλγόριθμος. Ξεκινάω βρίσκοντας τον πρώτο έγκυρο άντρα με την κλήση της συνάρτησης *GetValidLine(gender)*. Ποίο συγκεκριμένα αυτή η συνάρτηση καλείται με όρισμα το αρχείο είτε των γυναικών είτε των αντρών (εναλλάξ) και διαβάζει γραμμές του αρχείου μέχρις ότου βρει την επόμενη έγκυρη γραμμή (ηλικία>= 18 και όχι παντρεμένος/η), όπου σταματάει και την επιστρέφει. Γενικά, όταν η *GetValidLine* επιστρέψει 0, σημαίνει πως έφτασε στο τέλος του αρχείου, συνεπώς σταματάει την διαδικασία της ανάγνωσης του αρχείου και συνεπώς αρχίζει η διαδικασία της ανάγνωσης του άλλου αρχείου που έμεινε, μέχρις ότου φτάσει και αυτό στο τέλος του και αρχίσει η διαδικασία αναφοράς και αφαίρεσης top K ζευγαριών και των αποτελεσμάτων τους από την Q που έχει σχηματιστεί μέχρι εκείνη την στιγμή (*Σημείωση 1*). Σημειώνεται ότι σε όλο το πρόγραμμα μετά την κλήση της *GetValidLine*, με την προυπόθεση ότι δεν έχει επιστρέψει 0, αυξάνονται και οι δύο μετρητές των έγκυρων γραμμών ‘valid_males’ & ‘valid_females’ για τους άντρες και τις γυναίκες αντίστοιχα. Στη συνέχεια ενημερώνω τις μεταβλητές ‘mal_cur’ & ‘mal_max’ που περιέχουν το ‘instance weight’ και στην συνέχεια εισάγω αυτή την πρώτη έγκυρη εγγραφή στο λεξικό που αναφέρεται στους άντρες (‘mal_dict’) . Στο λεξικό γενικά, (ίδια δομή για γυναίκες και άντρες) το κλειδί είναι η ηλικία τους. Έτσι καθώς η κάθε ηλικία πρέπει να εμφανίζεται μια φορά στο κάθε λεξικό, όμως μπορούν να είναι πολλοί αυτοί που έχουν αυτή την ηλικία, φτιάχνω μια λίστα ‘VAL’ για κάθε ηλικία, όπου σε κάθε θέση της κρατάω όλους τους ανθρώπους με αυτήν την ηλικία και τα χαρακτηριστικά που με ενδιαφέρουν(id,age,instance weight). Κάνω την ίδια ακριβώς διαδικασία αφού βρω την πρώτη έγκυρη γυναίκα με την διαφορά ότι τώρα υπολογίζω και το κατώφλι T, καθώς για τον πρώτο άντρα δεν είχε νόημα. Να αναφερθεί πως οι μεταβλητές ‘mal_max’ & ’fem_max’ ενημερώνονται μόνο για τον πρώτο έγκυρο άντρα και την πρώτη έγκυρη γυναίκα και για όλους τους υπόλοιπους διατηρούν αυτή την τιμή . Στην συνέχεια ελέγχω αν η πρώτη γυναίκα έχει ίδια ηλικία με τον πρώτο άντρα, ψάχνοντας δηλαδή στο λεξικό των αντρών(που θα έχει βέβαια αυτή τη στιγμή μόνο έναν). Αν ναι, τότε καλείται η συνάρτηση *probe*, η οποία ουσιαστικά υπολογίζει τα join results για κάθε άνθρωπο με όλους τους ανθρώπους του αντίθετου φύλλου που έχουν ίδια ηλικία με αυτόν (διατρέχοντας τη κατάλληλη θέση-λίστα στο λεξικό τους) και τα αποτελέσματα τα προσθέτει κάθε φορά στην heap Q, την οποία επιστρέφει ενημερωμένη κάθε φορά. Η *probe* για να υπολογίσει το score μεταξύ κάθε άντρα και κάθε γυναίκας καλεί με την σειρά της την συνάρτηση *f*, η οποία απλά προσθέτει

τα αντίστοιχα instance weights και επιστρέφει το αποτέλεσμα αυτό. Αν βρεθεί πως η γυναίκα έχει ίδια ηλικία με τον άντρα, δηλαδή η Q δεν είναι άδεια, ενημερώνεται η μεταβλητή 'F' , όπου περιέχει το max score της Q, και αν η 'F' είναι μεγαλύτερη από το κατώφλι T τότε τυπώνεται το join result του ζευγαριού, αυξάνεται συνεπώς ο μετρητής k και εξάγεται από την heap αυτό το top ζευγάρι . Αυτή η διαδικασία που ακολουθείται εδώ μέσα στην if, αφού ξέρω πως σε αυτό το σημείο έχω μόνο δύο ανθρώπους, άρα αν η Q είναι γεμάτη θα έχει μόνο μια θέση, ακολουθείται και παρακάτω κάθε φορά μέσα σε μια while, δηλαδή για όσο η Q είναι γεμάτη και η 'F' > = από το κατώφλι (επίσης βγαίνει από αυτή την wile εάν k = K). Τέλος να σημειωθεί πως για να πάρω κάθε φορά το ζευγάρι με το μέγιστο score από την heap Q, στην συνάρτηση *probe* , όπου εισάγει τα join results, το κάθε score μπαίνει με αρνητικό πρόσημο. Αυτό συμβαίνει γιατί γνωρίζουμε πως η priority heap Q θα έχει στην πρώτη θέση της πάντα το μικρότερο στοιχείο. Όμως εγώ θέλω κάθε φορά να παίρνω το μεγαλύτερο και να ξέρω την θέση του για να το αφαιρώ. Έτσι για να αποφευχθεί η χρονοβόρα ταξινόμηση της Q, αλλάζοντας τα πρόσημα, το min στοιχείο, ουσιαστικά θα είναι το max και συνεπώς θα γνωρίζουμε ότι πάντα θα είναι στην πρώτη θέση της Q και έτσι θα έχουμε γρήγορη και άμεση πρόσβαση σε αυτό. Για αυτό λοιπόν όταν παίρνω το top score του βάζω πάλι το αρνητικό πρόσημο, για να γίνει θετικό, δηλαδή όπως κανονικά είναι και να γίνει η σωστή σύγκριση.

Τα παραπάνω βήματα, που αναφέρονται στην πρώτη γυναίκα, επαναλαμβάνονται αντίστοιχα για κάθε επόμενο έγκυρο άντρα και για κάθε επόμενη έγκυρη γυναίκα εναλλάξ, με τις μόνες διαφορές αυτής που σημειώνεται λίγο πριν το τέλος της παραπάνω παραγράφου ('while Q' αντί για if) και της εισαγωγής κάθε ανθρώπου στο αντίστοιχο λεξικό του, όπου γίνεται ο έλεγχος αν έχει ξανά βρεθεί κάποιος άλλος άνθρωπος του ίδιου φύλλου με αυτόν που έχει την ίδια ηλικία για να προστεθεί στην ήδη υπάρχουσα λίστα αυτής τη θέσης, ή να είναι αυτός ο πρώτος. Επίσης σαν συνέχεια της *Σημείωσης1* παρακάτω συμπληρώνονται οι εξής αναφορές, στην περίπτωση που φτάσουν στο τέλος και τα δύο αρχεία (δλδ για K=275 το αρχείο των αντρών και για K = 503 αυτό των γυναικών). Η περίπτωση αυτή, γίνεται αντιληπτή όταν και οι δύο μεταβλητές 'eof_males '& 'eof_females' γίνουν ίσες με 1. Από την στιγμή που τελειώσουν και τα δύο αρχεία για κάποια αποτελέσματα μένουν μέσα στην while καθώς το score τους είναι μεγαλύτερο από το κατώφλι. Όταν βγούνε όμως, καθώς το κατώφλι πλέον έχει μεγαλύτερη τιμή, πρέπει να συνεχίσουν να επιστρέφουν join results και να αφαιρούν το καθένα από αυτά κάθε φορά από την Q. Έτσι μπαίνουν μέσα σε μια while όπου κάνουν αυτή την δουλεία για όσο είναι γεμάτη η Q και για όσο ο μετρητής k δεν έχει φτάσει ακόμα το K.

Εν κατακλείδι, με τον παραπάνω αλγόριθμο: διαβάζονται επιτυχώς εναλλάξ('gender'='Male'/'Female') οι επόμενες έγκυρες γραμμές των αντρών και των γυναικών('line_mal','line_fem'), εισάγεται η καθεμία από αυτές στα αντίστοιχα λεξικά ('mal_dict', 'fem_dict') με κλειδί το age, ενημερώνεται το T, υπολογίζονται τα join results και εισάγονται στην Q, επιστρέφοντας κάθε φορά το επόμενο join result (σαν generator function) . Στο τέλος τυπώνεται ο χρόνος εκτέλεσης και οι έγκυρες γραμμές που διάβασε για τους άντρες και τις γυναίκες αντίστοιχα.

ΜΕΡΟΣ ΙΙ

>py algorithmB.py K

Σ'αυτό το μέρος με το που ξεκινάει ο αλγόριθμος, διαβάζονται εξολοκλήρου όλες οι γραμμές του αρχείου 'males_sorted' και μπαίνουν σε ένα λεξικό 'mal_dict' μόνο οι έγκυρες, με κλειδί την ηλικία. Και εδώ πέρα η μορφή αποθήκευσης των έγκυρων αντρών στο λεξικό είναι ίδια, δηλαδή για κάθε ηλικία κρατάμε μια λίστα με όλους τους άντρες που την έχουν και τα χαρακτηριστικά τους, κάθε θέση αυτής της λίστας αναφέρεται σε έναν άντρα. Στην συνέχεια ανοίγω το αρχείο 'females_sorted' και διαβάζω κάθε γραμμή του μια προς μια ακολουθώντας την παρακάτω διαδικασία. Με το που βρίσκω μια γυναίκα που είναι έγκυρη, τότε ελέγχω αν η ηλικία της υπάρχει στο λεξικό των αντρών. Αν υπάρχει, τότε θέλω να βρω όλα τα αποτελέσματα join αυτής με τους άντρες της ίδιας ηλικίας και αν και όσα ανήκουν στα κορυφαία να τα κρατήσω στο min heap 'joins' , μήκους K, κάθε στιγμή. Έτσι για κάθε join ελέγχω το μήκος αυτής της heap, αν είναι < K σημαίνει ότι όσα αποτελέσματα βρίσκω τα εισάγω, για να γεμίσει η 'joins' μου με το επιθυμητό μήκος. Διαφορετικά, σημαίνει πως πλέον πρέπει

να διατηρήσω αυτό το μήκος με τα top K αποτελέσματα. Έτσι με το που βρίσκω ένα αποτέλεσμα, αν είναι μεγαλύτερο από το μικρότερο αποτέλεσμα('min_joins') που είναι αποθηκευμένο στην 'joins' βγάζω το 'min_joins' και εισάγω αυτό στην θέση του, διαφορετικά σταματάω να κάνω join με τους άντρες, γιατί ξέρω ότι από εδώ και πέρα όλα τα join results που θα βρω, θα είναι σίγουρα μικρότερα από το μικρότερο των top K, αφού οι άντρες που βρίσκονται αποθηκευμένοι στη λίστα για κάθε ηλικία, είναι σε φθίνουσα σειρά με βάση το 'instance weight', λόγω της φθίνουσας ταξινόμησης του αρχείου 'males_sorted'. Έτσι αφού διαβαστούν όλες οι γραμμές του 'females_sorted' στην heap 'joins' θα υπάρχουν τα K κορυφαία join results. Στη συνέχεια απλά ταξινομώ τα στοιχεία της heap σε μια list 'minheap' ίδιου μήκους, για να είναι σωστές οι print, και μαζί με τα αποτελέσματα, τα οποία βγαίνουν ίδια με αυτά του μέρους Ι, τυπώνω και τον χρόνο εκτέλεσης του αλγορίθμου.

ΜΕΡΟΣ ΙΙΙ

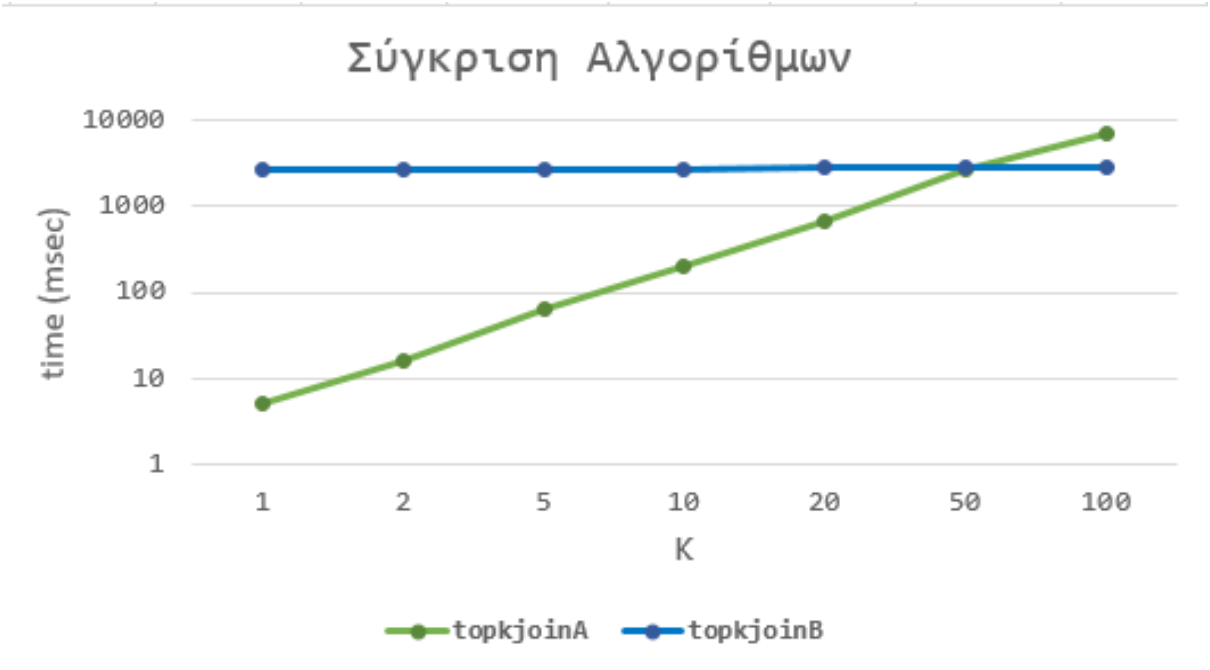
K	1	2	5	10	20	50	100	200
time(secs)	0.00499987602234	0.0160000324249	0.0659999847412	0.200999975204	0.680999994278	2.72799992561	7.11999988556	19.5429999828
mal_lines	44	134	437	1145	2656	6030	1039	18306
fem_lines	44	134	437	1145	2656	6030	1039	18306
all_valid_lines	88	268	874	2290	5312	12060	20278	36612

***Πίνακας 1:** Algorithm A top-K join*

K	1	2	5	10	20	50	100	200
time(secs)	2.75999999046	2.74799990654	2.72600007057	2.73600006104	2.90500020981	2.86199998856	2.90100002289	2.87400007248

***Πίνακας 2:** Algorithm B top-K join*

Διάγραμμα:



Συμπεριφορά Αλγορίθμων:

Παρατηρώ τόσο από το διάγραμμα, όσο και από το πινακάκι (που έχω βάλει extra τον χρόνο και για K=200) πως ο χρόνος του αλγορίθμου A είναι αυξανόμενος όσο μεγαλώνει το K, ενώ ο αλγόριθμος B για όλες τις τιμές του K έχει την ίδια συμπεριφορά και ο χρόνος είναι σταθερός για όλες τις τιμές. Έτσι συμπεραίνεται πως ο αλγόριθμος B είναι πιο αποδοτικός και πιο γρήγορος για τιμές του K μέχρι και το 50, όπου είναι το σημείου που οι 2 αλγόριθμοι εκτελούνται στον ίδιο χρόνο και από εκείνο το σημείο και μετά , δηλαδή για K > 50 , προτιμάται ο αλγόριθμος B καθώς είναι αποδοτικότερος και τρέχει σε πολύ λιγότερο χρόνο συγκριτικά με τον αλγόριθμο B, καθώς όπως αναφέρθηκε και παραπάνω ο B εκτελείται σε σταθερό χρόνο για όλες τις τιμές, ενώ ο A αυξάνεται διαρκώς ο χρόνος εκτέλεσης του αναλογικά με την αύξηση του K.

Πλεονεκτήματα/Μειονεκτήματα A έναντι B:

(+) ο A για μικρές τιμές του K είναι γρηγορότερος σε σχέση με B
(+) ο A υπολογίζει τα join results για κάθε άνθρωπο κάθε φορά, και αν φτάσει στο επιθυμητό πλήθος αυτών σταματάει, ενώ ο B υπολογίζει όλα τα join results για όλους τους ανθρώπους που είναι έγκυρους ακόμα και αν θέλει να επιστρέψει μόνο ένα αποτέλεσμα
(+) ο A διαθέτει δύο λεξικά και για τους άντρες και τις γυναίκες που οι εγγραφές τους είναι έγκυρες, συνεπώς αυτό συνεπάγεται την γρήγορη αναζήτηση και πρόσβαση τους σε αυτά με βάση το κλειδί (ηλικία), ενώ ο B διαθέτει μόνο για τους άντρες και δεν κρατάει κάπου τα δεδομένα των γυναικών
(+) ο A λειτουργεί επιστρέφοντας κάθε φορά το ζευγάρι με το top join αποτέλεσμα ανεξάρτητα με το K, ενώ ο B υπολογίζει πρώτα όλα τα join result και μετά τυπώνει τα K κορυφαία αυτών. Έτσι σε περίπτωση που κάτι πάει στραβά κατά τη διάρκεια εκτέλεσης του αλγορίθμου, ακόμα και στο τελευταίο αποτέλεσμα, στην περίπτωση του B θα χαθούν όλα τα αποτελέσματα και δεν θα τυπωθεί τίποτα, αντίθετα με την περίπτωση του A στην οποία θα έχουν τυπωθεί όλα τα αποτελέσματα που έχουν υπολογιστεί μέχρι εκείνη την στιγμή.
(-) ο A για μεγάλες τιμές του K είναι πολύ πιο αργός σε σχέση με B
(-) ο A όσο αυξάνεται το K, αυξάνεται εκθετικά και ο χρόνος εκτέλεσης του, σε αντίθεση με τον B που έχει σταθερό χρόνο εκτέλεσης για όλες τις τιμές
(-) ο A ήταν πιο πολύπλοκος στην υλοποίηση του σε σχέση με τον B