Διαχείριση Σύνθετων Δεδομένων
Εργασία 2

Όθωνας Γκαβαρδίνας
ΑΜ: 2620

Μέρος 1
Πρόβλημα: Δημιουργία ευρετηρίου στις δύο διαστάσεις.
Αρχείο εισόδου: Beijing_restaurants.txt
Γλώσσα: Python 3.6.7

Για την επίλυση του προβλήματος διαχώρισα το πρόβλημα στα παρακάτω βήματα, που αποτελούν ξεχωριστές συναρτήσεις:

(1) find_grid_boarders():

Στη συνάρτηση αυτή διάβασα το αρχείο εισόδου, και με απλό αλγόριθμο εύρεσης μεγίστου και ελαχίστου για κάθε συντεταγμένη (συνολικά τέσσερις περιπτώσεις) βρήκα τα σημεία κάτω-αριστερά και πάνω-δεξιά που οριοθετούν όλα τα σημεία με ένα ορθογώνιο.

<<**Παρένθεση**>>

Για τον καλύτερο διαχωρισμό του προβλήματος σε συναρτήσεις, έφτιαξα μία κλάση Structure, η οποία έχει το ρόλο μιας δομής που περιλαμβάνει όλα τα στοιχεία που βρίσκω σταδιακά και χρησιμοποιώ στο πρόγραμμά μου. Τη δομή αυτή την χρησιμοποιώ ως όρισμα στις συναρτήσεις μου για να αποφευχθεί η χρήση πολλών ορισμάτων.

<<>>

(2) find_block_difference():

Το ορθογώνιο που υπολόγισα, πρέπει να αποτελείται από 10x10 μικρότερα ορθογώνια. Με τη συνάρτηση αυτή υπολογίζω τα όρια αυτών των μικρότερων ορθογωνίων.

(3) init_dict():

Δημιούργησα ένα λεξικό, το block_dict, το οποίο θα περιλαμβάνει πλειάδες του τύπου (άνω_όριο_block_x, άνω_όριο_block_y). Το λεξικό αυτό αρχικοποιείται με αυτή τη συνάρτηση, με τιμές για τα κλειδιά του, κενές λίστες. Στη συνέχεια στις λίστες αυτές θα τοποθετηθούν τα διάφορα σημεία που έχουν μικρότερα x, και y από το κάθε κλειδί του λεξικού.

(4) make_lists():

Η συνάρτηση αυτή φτιάχνει για κάθε συντεταγμένη μία λίστα που περιέχει τα διάφορα άνω όρια για τα blocks. Είναι μία βοηθητική συνάρτηση που χρησιμοποιείται πιο κάτω.

(5) fill_dict():

Η συνάρτηση αυτή διαβάζει από το αρχείο εισόδου, μία μία τις γραμμές, και για κάθε γραμμή (που περιέχει ένα σημείο), εκτελεί δυαδική αναζήτηση για κάθε συντεταγμένη, έτσι ώστε να βρει το σωστά άνω όρια, και στη συνέχεια, τη σωστή θέση στο λεξικό block_dict, που έχει ως κλειδιά τα διάφορα άνω όρια. Επίσης εδώ, τοποθετώ ένα id, για κάθε σημείο στο αρχείο, το οποίο και τοποθετώ ως τρίτη τιμή της πλειάδας για ευκολία.

(+) binary_search():

Βοηθητική συνάρτηση για το βήμα (5). Λίγο τροποποιημένη από την κλασική υλοποίηση δυαδικής αναζήτησης. Ξεκινά από μία αρχική λίστα, διαχωρίζει τη λίστα με βάση το μέγιστο στοιχείο, και αναδρομικά επιλέγει μια μικρότερη λίστα. Η διαφορά από τον κλασσικό αλγόριθμο είναι ότι δεν αναζητώ κάποιο στοιχείο, αλλά αποκλείω κάθε φορά περιπτώσεις έτσι ώστε να βρω το κατάλληλο άνω όριο για κάποιο στοιχείο.

(6) create_files():

Η συνάρτηση αυτή δημιουργεί την έξοδο του προγράμματος, η οποία είναι τα αρχεία grid.grd, και grid.dir.

grid.grd:

Με τη χρήση των βοηθητικών λιστών που υπολόγισα, λαμβάνω τα στοιχεία της κάθε θέσης του λεξικού μου με αύξουσα σειρά και τα αποθηκεύω σ’ αυτό το αρχείο.

grid.dir:

Στο αρχείο αυτό αποθηκεύω τα μέγιστα και ελάχιστα σημεία για το αρχικό ορθογώνιο. Υπολογίζω ένα όνομα της μορφής (0, 0) (0, 1) ... για το κάθε όνομα και το τυπώνω. Μαζί με το όνομα του, για κάθε block, την εμφάνιση του πρώτου στοιχείου με βάση το offset του file descriptor, το οποίο λαμβάνεται με την εντολή tell(). Επίσης μαζί τοποθετείται και ο αριθμός των σημείων που βρίσκονται στο κάθε block, ο οποίος είναι απλά το μέγεθος της λίστας για τη θέση του block στο λεξικό.

(!) Η κύρια δομή του προγράμματός μου βρίσκεται στη συνάρτηση main().

<u>Έλεγχος ορθότητας:</u>
Για το πρόγραμμά μου, έτρεξα τις εξής δοκιμές, οι οποίες περιλαμβάνονται στη συνάρτηση test():

Test 1:
Έλεγξα τις γραμμές του αρχείου grid.grd, εάν είναι 51970, δηλαδή όσες και οι εγγραφές του αρχείου εισόδου.

Test 2:
Αποθήκευσα σε μια λίστα με πλειάδες κάθε εγγραφή του αρχείου εισόδου, και έλεγξα αν όλες περιλαμβάνονται στο αρχείο grid.grd. Για κάθε γραμμή στο αρχείο grid.grd σβήνω την πλειάδα που την αποτελεί. Έτσι τελικά μένει μια άδεια λίστα.

Test 3:
Έλεγξα αν το άθροισμα των σημείων που περιέχει κάθε block, είναι ίσο με 51970, δηλαδή όσες και οι εγγραφές του αρχείου εισόδου.

Μέρος 2

Πρόβλημα: Αξιοποίηση ευρετηρίου με χρήση ερωτημάτων επιλογής.
Αρχεία εισόδου: grid.dir, grid.grd
Γλώσσα: Python 3.6.7

Για την επίλυση του προβλήματος διαχώρισα το πρόβλημα στα παρακάτω βήματα, που αποτελούν ξεχωριστές συναρτήσεις:

(1) get_user_window():

Η συνάρτηση αυτή λαμβάνει τα ορίσματα του χρήστη και με βάση αυτά, ορίζει τα σημεία κάτω-αριστερά και πάνω-δεξιά, που μαζί χαρακτηρίζουν το παράθυρο του ερωτήματος του χρήστη.

(2) read_grid_dir():

Η συνάρτηση αυτή λαμβάνει τα δεδομένα που είναι αποθηκευμένα στο αρχείο ‘grid.dir’, και τα αποθηκεύει σε κατάλληλες δομές. Συγκεκριμένα, βρίσκει στην πρώτη γραμμή το μέγιστο και ελάχιστο σημείο ολόκληρου του πλέγματος, που αποτελούν και τα όρια του πλέγματος.

<<**Παρένθεση**>>

Για τον καλύτερο διαχωρισμό του προβλήματος σε συναρτήσεις, έφτιαξα μία κλάση Structure, η οποία έχει το ρόλο μιας δομής που περιλαμβάνει όλα τα στοιχεία που βρίσκω σταδιακά και χρησιμοποιώ στο πρόγραμμά μου. Τη δομή αυτή την χρησιμοποιώ ως όρισμα στις συναρτήσεις μου για να αποφευχθεί η χρήση πολλών ορισμάτων.

<<>>

Στην άσκηση αυτή, κρατώ τα δεδομένα σε δύο λεξικά με κλειδί το id για κάθε block (που ορίστηκε στην πρώτη άσκηση) και τιμές την αρχική θέση κάθε μπλοκ και το πλήθος των στοιχείων μπλοκ αντίστοιχα.

(3) collect_points():

Η συνάρτηση αυτή παράγει το αποτέλεσμα της αναζήτησης σε μία λίστα. Αρχικά υπολογίζεται η διαφορά ανάμεσα σε δύο block ως προς τους δύο άξονες. Χρησιμοποιούνται δύο δείκτες, για τα δύο όρια του κάθε block. Στη συνέχεια με τη χρήση βοηθητικών συναρτήσεων ελέγχεται η σχέση μεταξύ του παραθύρου εισόδου και των block του πλέγματος. Αρχικά ελέγχεται αν έχουν κοινά σημεία. Στη συνέχεια, εάν έχουν, ελέγχεται εάν ολόκληρο το block βρίσκεται μέσα στο παράθυρο ή μέρος αυτού και ανάλογα με την περίπτωση γίνονται κάποιες ενέργειες.

(4) check_overlap():

Βοηθητική στην (3). Ελέγχει εάν δύο ορθογώνια έχουν κοινό σημείο.

(5) check_inside():

Βοηθητική στην (3). Ελέγχει εάν κάποιο block βρίσκεται ολόκληρο στο παράθυρο εισόδου.

(6) put_block_points():

Αναζητεί τη θέση του πρώτου στοιχείου ενός block στο αρχείο ‘grid.grd’, και από εκεί λαμβάνει όλα τα σημεία που βρίσκονται μέσα στο παράθυρο και τα προσθέτει στο αποτέλεσμα.

(7) write_results():

Αποθηκεύει το αποτέλεσμα σε αρχείο.

Μέρος 3
Πρόβλημα: Αξιοποίηση ευρετηρίου με χρήση ερωτημάτων εύρεσης πλησιέστερου γείτονα.
Αρχεία εισόδου: grid.dir, grid.grd
Γλώσσα: Python 3.6.7

Για την επίλυση του προβλήματος διαχώρισα το πρόβλημα στα παρακάτω βήματα, που αποτελούν ξεχωριστές συναρτήσεις:

(1) get_args():
Η συνάρτηση αυτή λαμβάνει τα ορίσματα του χρήστη και με βάση αυτά, ορίζει το πλήθος των πλησιέστερων γειτόνων k, και το σημείο από το οποίο θα βρεθούν οι γείτονες q.

(2) read_grid_dir():
Ομοίως με τη συνάρτηση (2) του μέρους 2.
Η συνάρτηση αυτή λαμβάνει τα δεδομένα που είναι αποθηκευμένα στο αρχείο 'grid.dir', και τα αποθηκεύει σε κατάλληλες δομές. Συγκεκριμένα, βρίσκει στην πρώτη γραμμή το μέγιστο και ελάχιστο σημείο ολόκληρου του πλέγματος, που αποτελούν και τα όρια του πλέγματος.

<<**Παρένθεση**>>
Ομοίως με την παρένθεση του μέρους 2.
Για τον καλύτερο διαχωρισμό του προβλήματος σε συναρτήσεις, έφτιαξα μία κλάση Structure, η οποία έχει το ρόλο μιας δομής που περιλαμβάνει όλα τα στοιχεία που βρίσκω σταδιακά και χρησιμοποιώ στο πρόγραμμά μου. Τη δομή αυτή την χρησιμοποιώ ως όρισμα στις συναρτήσεις μου για να αποφευχθεί η χρήση πολλών ορισμάτων.
<<>>
Στην άσκηση αυτή, κρατώ τα δεδομένα σε δύο λεξικά με κλειδί το id για κάθε block (που ορίστηκε στην πρώτη άσκηση) και τιμές την αρχική θέση κάθε μπλοκ και το πλήθος των στοιχείων μπλοκ αντίστοιχα.

(3) init_queue():
Στη συνάρτηση αυτή, ελέγχω εάν το σημείο q βρίσκεται εντός ή εκτός του πλέγματος, και το χειρίζομαι με μια βοηθητική συνάρτηση για κάθε περίπτωση. Βρίσκω με τη βοηθητική συνάρτηση το κοντινότερο block στο σημείο q, το εισάγω στην ουρά και το προσθέτω στη λίστα με τα block που έχω διανύσει ως τώρα.

(4) find_q_block_inside():
Βοηθητική στην (3). Με βάση το άνω φράγμα κάθε block, εντοπίζω σε ποιο block βρίσκεται το q.

(5) find_q_block_ outside():
Βοηθητική στην (3). Με βάση μία συνάρτηση που θα περιγραφεί στη συνέχεια, βρίσκω την απόσταση του q, από κάθε ένα από τα block του πλέγματος που βρίσκονται στα άκρα του. Από τις αποστάσεις αυτές, κρατώ τη μικρότερη, η οποία μου αποκαλύπτει το κοντινότερο block στο σημείο q.

(6) find_nearest_k():
Δοθέντος ενός k, επιστρέφει τους k πλησιέστερους γείτονες.

(7) find_nearest():
Βοηθητική στην (5). Πρόκειται για iterator που επιστρέφει κάθε φορά τον πλησιέστερο γείτονα.

(8) save_results():

Αποθηκεύει το αποτέλεσμα στο αρχείο ‘results3’.
Επίσης τυπώνονται στο τερματικό τα blocks που χρησιμοποιήθηκαν.

Στη συνέχεια περιγράφεται ο εσωτερικός μηχανισμός του προγράμματος:
Χρησιμοποιείται δομή ουράς προτεραιότητας:
-Η δομή διαθέτει τις βοηθητικές μεθόδους **fixUp()**, **fixDown()**, **swap()**, και **greater()**, οι οποίες υλοποιούν την μετακίνηση των κόμβων στη στοίβα.

(9) PriorityQueue . insert():

Υπάρχουν δύο κατηγορίες στοιχείων για εισαγωγή, τα block που δίδονται ως πλειάδα (x, y) και μια δομή σημείου δύο διαστάσεων που έχω δημιουργήσει. Ανάλογα με την περίπτωση υπολογίζεται μία απόσταση σημείου από ορθογώνιο ή σημείου από σημείο. Η απόσταση αυτή στη συνέχεια θα αποτελέσει το κλειδί στην ουρά προτεραιότητας. Η απόσταση εισάγεται σε λεξικό ως κλειδί, και τιμή τα δεδομένα, και επίσης εισάγεται στο τέλος της ουράς προτεραιότητας, όπου ανεβαίνει ένα δυαδικό δέντρο σε μορφή πίνακα μέχρι να βρει κατάλληλη θέση.
Σημείωση: Ως τιμή στο λεξικό χρησιμοποιείται πίνακας για να καλυφθεί η περίπτωση να έχω δύο σημεία με ίδια απόσταση.

(10) calc_eucl_dist():

Βοηθητική στην (8). Τύπος υπολογισμού ευκλείδειας απόστασης δύο σημείων χωρίς χρήση τετραγωνικής ρίζας.

(11) calc_block_dist():

Βοηθητική στην (8). Η συνάρτηση αυτή υπολογίζει ευκλείδεια απόσταση σημείου από ορθογώνιο.
Εκμεταλλεύομαι το γεγονός ότι το πλέγμα έχει κάποια συμμετρία, και έτσι με βάση τις συντεταγμένες του σημείου βρίσκω την κοντινότερη απόσταση από ένα block. Αρχικά ελέγχω αν το ζητούμενο block βρίσκεται σε κάποια από τις κατευθύνσεις πάνω, κάτω, δεξιά και αριστερά. Στη συνέχεια με βάση κάποια flags διαπιστώνω εάν το block βρίσκεται σε κάποια από τις τέσσερεις πλάγιες θέσεις που αποτελούν συνδυασμό δύο κατευθύνσεων.

(12) find_b_points():

Βοηθητική στην (10). Με βάση συντεταγμένες (x, y) με x, y να ανήκουν στο [0, 9], εντοπίζει τα πάνω δεξιά και κάτω αριστερά σημεία, ενός block.

(13) PriorityQueue . popMin():

Η συνάρτηση αυτή επιστρέφει και διαγράφει την κεφαλή της ουράς προτεραιότητας, η οποία αποτελεί και το σημείο με την ελάχιστη απόσταση από το q. Συγκεκριμένα, κρατείται η τιμή προς εξαγωγή, αλλάζει θέση με το τελευταίο στοιχείο της ουράς, βγαίνει από την ουρά και από το λεξικό, και γίνονται οι απαραίτητες αλλαγές για να διατηρηθεί η δομή στη σωστή της μορφή. Εάν το στοιχείο που εξήχθη είναι block (έχει τη μορφή πλειάδας), εισάγονται στην ουρά τα γειτονικά blocks, και τα σημεία που αποτελούν το ίδιο το block.

(14) PriorityQueue . insert_neighbour_blocks():

Βοηθητική στην (12). Εδώ ελέγχονται όλες οι περιπτώσεις πριν τον έλεγχο και την εισαγωγή των γειτονικών block.

(15) PriorityQueue . check_insert():

Βοηθητική στην (13). Η συνάρτηση ελέγχει εάν κάποιο block έχει ήδη τοποθετηθεί στην ουρά, και αν δεν έχει, το τοποθετεί και το προσθέτει στη λίστα με τα τοποθετημένα.

(16) PriorityQueue . inesert_block_points():

Βοηθητική στην (12). Η συνάρτηση αυτή λαμβάνει συντεταγμένες ενός block, διαβάζει το αρχείο ‘grid.grd’, και εξάγει από αυτό τα κατάλληλα σημεία τα οποία και τοποθετεί στην ουρά προτεραιότητας.