Ημερομηνία Ανάρτησης: 10/1/2017
Ημερομηνία Παράδοσης: -
Αρχές Γλωσσών Προγραμματισμού

Περιγραφή Προβλήματος

Το θέμα της παρούσας εργασίας είναι η κατασκευή και επίλυση λαβυρίνθων. Ένας *λαβύρινθος* (maze) είναι κατά κανόνα ένα μονοπάτι ή ένα σύνολο μονοπατιών από μία είσοδο σε ένα σημείο στόχο. Καθότι υπάρχουν διάφορα είδη λαβυρίνθων, επικεντρωνόμαστε σε εκείνο των *τέλειων λαβυρίνθων* (perfect mazes) σε επιφάνειες δύο διαστάσεων. Στόχος είναι το πρόγραμμα σας να μπορεί να κατασκευάζει έναν τυχαίο (δηλαδή όχι σταθερό ανά τις εκτελέσεις) τέλειο λαβύρινθο, χρησιμοποιώντας μια παραλλαγμένη έκδοση του αλγορίθμου του Kruskal. Επίσης, θα πρέπει να είναι σε θέση να επιλύει έναν τέλειο λαβύρινθο και να τον τυπώνει στην οθόνη μαζί με το μονοπάτι της λύσης.

Τέλειος Λαβύρινθος

Ένας τέλειος λαβύρινθος είναι ένας λαβύρινθος ο οποίος δεν έχει κύκλους. Δηλαδή για κάθε ζεύγος δύο τυχαίων κελιών - σημείων του λαβυρίνθου, υπάρχει ακριβώς ένα μονοπάτι που τα ενώνει. Αναλογιζόμενοι την ιδιότητα αυτή των τέλειων λαβυρίνθων μπορούμε να αντιληφθούμε ότι το σημείο εισόδου και το σημείο στόχος δεν επηρεάζουν την κατασκευή του συγκεκριμένου τύπου λαβυρίνθων. Συνεπώς, θα μπορούσαμε να ορίσουμε ως σημείο εισόδου και σημείο στόχο - σημείο εξόδου οποιοδήποτε ζεύγος σημείων (π.χ. πάνω αριστερή γωνία και κάτω δεξιά γωνία του λαβυρίνθου), αφού σίγουρα θα υπάρχει ένα μοναδικό μονοπάτι που θα τα ενώνει.

Αναπαράσταση Λαβυρίνθου

Στην ουσία ένας λαβύρινθος θα πρέπει να αναπαρίσταται στο πρόγραμμα σας με το data type *Maze* (Σχήμα 1). Το data type Maze έχει τρία fields, εκ των οποίων τα δύο, *width* και *height*, αποθηκεύουν τις τιμές των διαστάσεων του λαβυρίνθου στο διδιάστατο επίπεδο. Το field *cells* είναι μία λίστα από tuples τύπου (Bool, Bool) τα οποία αντιπροσωπεύουν για κάθε κελί - σημείο του λαβυρίνθου την ύπαρξη (True) ή τη μη ύπαρξη (False) τοίχου δεξιά και αντίστοιχα κάτω του κελιού - σημείου.

```
data Maze = Maze { cells :: [(Bool, Bool)]  -- [(rightWall, downWall)]
                 , width :: Int
                 , height :: Int
                 }
```

Σχήμα 1: Maze data type

Στο σχήμα 2 παρουσιάζεται ο κώδικας κατασκευής ενός Maze και μια ενδεικτική του εκτύπωση.

```
Maze [(False, True), (False, False),
      (False, False), (True, False),
      (False, False), (True, False),
      (True, True), (True, False),
      (True, True), (False, True),
      (True, True), (True, True)]
      4 3
```

```
+---+---+---+---+---+
|                   |
+---+   +   +   +   +
|       |   |   |   |
+   +   +---+   +   +
|   |       |   |   |
+---+---+---+---+---+
```

Σχήμα 2: Παράδειγμα Maze

Ζητούμενο: Συνάρτηση makeMaze :: Int → Int → Maze, η οποία δέχεται δύο ακεραίους, το width και height του λαβυρίνθου που θα επιστρέφει και ο οποίος θα περιέχει όλους τους τοίχους.

Αλγόριθμος Κατασκευής

Ο αλγόριθμος κατασκευής που θα υλοποιήσετε θα είναι μια παραλλαγή του αλγόριθμου του Kruskal ο οποίος υπολογίζει το δέντρο επικάλυψης ελαχίστου κόστος (minimum-cost spanning tree) ενός γράφου. Στην ουσία δεν μας ενδιαφέρει η ιδιότητα του ελάχιστου κόστους αλλά ο υπολογισμός ενός δέντρου επικάλυψης που περιέχει κάθε κόμβο του γράφου, όπου ως γράφο φανταζόμαστε έναν τέλειο λαβύρινθο με τα κελιά - σημεία να είναι κόμβοι και τα μονοπάτια που ενώνουν γειτονικά κελιά - σημεία (μονοπάτια μήκους 1) να είναι ακμές μεταξύ των κόμβων.

Το γεγονός ότι ο λαβύρινθος σας αναπαρίσταται στην ουσία ως ένα πλέγμα από κελιά (Σχήμα 1) δεν σας εμποδίζει να τον φαντάζεστε και να τον διαχειρίζεστε ως μια δενδρική δομή. Εξάλλου, ο τρόπος που περιγράφεται ένα κελί - (ύπαρξη ή μη ύπαρξη τοίχου δεξιά, ύπαρξη ή μη ύπαρξη τοίχου κάτω) - παραπέμπει σε δενδρική δομή.

Ο αλγόριθμος κατασκευής έχει ως εξής: Αρχικά, φτιάχνουμε ένα σετ για κάθε κελί c_i του λαβυρίνθου και προσθέτουμε το κελί c_i στο αντίστοιχο σετ. Επίσης, φτιάχνουμε μία λίστα όλων των τοίχων που θα μπορούσαν να υπάρχουν στο λαβύρινθο, η οποία έχει τη μορφή [... , (c_i, c_j), ...], όπου το ζεύγος (c_i, c_j) δηλώνει την ύπαρξη τοίχου μεταξύ των κελιών c_i και c_j. Εν συνεχεία, ανακατεύουμε τυχαία τη λίστα και τη διατρέχουμε. Για κάθε τοίχο ελέγχουμε αν τα κελιά c_i και c_j τα οποία διαχωρίζει ανήκουν σε διαφορετικά σετς και αν κάτι τέτοιο ισχύει, αφαιρούμε τον τοίχο και ενώνουμε τα σετς των κελιών c_i και c_j. Παρακάτω παρουσιάζεται ο αλγόριθμος σε ψευδοκώδικα.

Algorithm 1 Modified Kruskal's Algorithm

```
1: sets[i] ← Set(c_i) for all cells c_i in maze
2: walls ← (c_i, c_j) for all neighboring cells c_i, c_j in maze
3: shuffled_walls ← shuffle(walls)
4:
5: for (c_i, c_j) in shuffled_walls do
6:     if c_i not in sets[j] and c_j not in sets[i] then
7:         connect(c_i, c_j).
8:         joined_set ← union(sets[i], sets[j])
9:         for c_k in joined_set do
10:            sets[k] ← joined_set
```

Ζητούμενο: Συνάρτηση kruskal :: Maze → Maze, η οποία δέχεται ένα λαβύρινθο χωρίς μονοπάτια (όπως δηλαδή αυτός επιστρέφεται από μια κλήση της `makeMaze`) και εκτελώντας την παραλλαγή του αλγορίθμου του Kruskal επιστρέφει έναν τυχαίο τέλειο λαβύρινθο.

Αλγόριθμος Επίλυσης

Ο αλγόριθμος επίλυσης που θα υλοποιήσετε είναι ο Αναζήτησης Κατά Βάθος (Depth-First Search). Ως γνωστόν ο αλγόριθμος αυτός εξερευνά όσο το δυνατόν περισσότερο κατά μήκος ένα μονοπάτι μέχρι να βρει το στόχο ή ένα αδιέξοδο, οπότε και οπισθοδρομεί. Ο αλγόριθμος σας θα πρέπει να είναι σε θέση να δέχεται τις συντεταγμένες ενός κελιού αρχής (s_x, s_y) και ενός κελιού στόχου (g_x, g_y) και να υπολογίζει το μονοπάτι μεταξύ αυτών (το οποίο σίγουρα υπάρχει σε έναν τέλειο λαβύρινθο). Επίσης, *δε θα πρέπει σε καμία περίπτωση* η υλοποίηση του DFS να χρησιμοποιεί δομή στην οποία θα αποθηκεύει τα κελιά που έχουν ήδη επισκεφθεί.

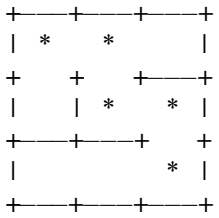
Ζητούμενο: Συνάρτηση solvePerfect :: Maze → (Int, Int) → (Int, Int) → [(Int, Int)], η οποία δέχεται ένα τέλειο λαβύρινθο και τις συντεταγμένες ενός κελιού αρχής (s_x, s_y) και ενός κελιού στόχου (g_x, g_y) και επιστρέφει μια λίστα με τα κελιά που ανήκουν στο μονοπάτι της λύσης.

Εκτύπωση Λαβυρίνθου και Λύσης

Ζητούμενο: Συνάρτηση showMaze :: Maze → [(Int, Int)] → String, η οποία δέχεται ένα λαβύρινθο και μία λίστα με τα κελιά που ανήκουν στο μονοπάτι της λύσης και επιστρέφει σε ένα String την αναπαράσταση του λαβυρίνθου και της λύσης όπως περιγράφεται παρακάτω.

Κάθε κελί θα αναπαρίσταται σαν ένα κουτάκι (σχήμα 3, αριστερά) από το οποίο όμως μπορεί να λείπει οποιοσδήποτε τοίχος. Επειδή κάθε κελί έχει πληροφορία μόνο για τους τοίχους που βρίσκονται δεξιά και κάτω του, μία πιο ακριβής αναπαράσταση του φαίνεται στο σχήμα 3, δεξιά.

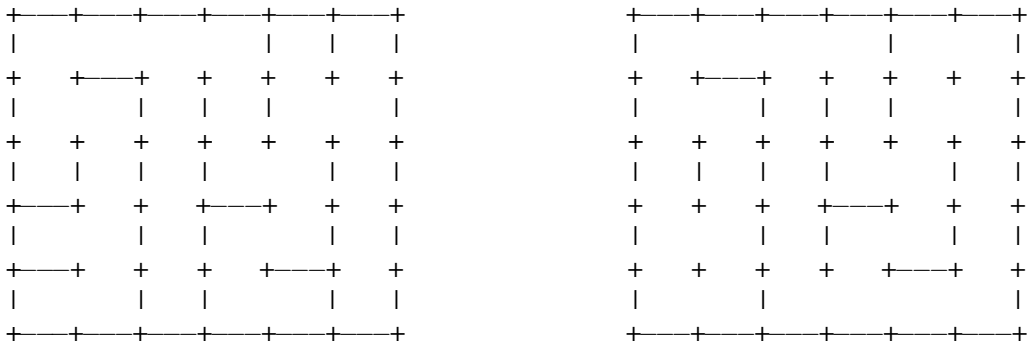
Σχήμα 3: Αναπαράσταση κελιών



Σχήμα 4: Παράδειγμα εκτέλεσης showMaze m [(0,0),(1,0),(1,1),(2,1),(2,2)], όπου m ένας 3x3 maze.

Bonus Ερώτημα +20%

Ένας λαβύρινθος braid είναι ένας λαβύρινθος στον οποίο δεν υπάρχουν αδιέξοδα. Αντί αυτού, τα διάφορα μονοπάτια μπορεί να χωρίζονται και να ξαναενώνονται. Αυτό σημαίνει πως μπορεί να υπάρχουν πολλά μονοπάτια τα οποία οδηγούν από το ένα σημείο στο άλλο, αλλά και πως μπορεί κάποιος να πέσει σε κύκλο προσπαθώντας να βρει τη λύση, σε αντίθεση με έναν τέλειο λαβύρινθο.



Σχήμα 5: Παράδειγμα Perfect Maze (αριστερά) και Braid Maze (δεξιά)

Ένας απλός τρόπος να σχηματίσετε έναν braid λαβύρινθο είναι να πάρετε έναν τέλειο λαβύρινθο, να βρείτε όλα τα αδιέξοδα (δηλαδή τα κελιά τα οποία συνδέονται μόνο με ένα άλλο κελί) και να ενώσετε το κελιά στα οποία υπάρχει αδιέξοδο με ένα ακόμα από τα γειτονικά τους κελιά. Λόγω του τύπου λαβυρίνθων που σχηματίζει η παραλλαγή του αλγορίθμου του Kruskal, συνήθως θα υπάρχουν

αρκετά αδιέξοδα, άρα το αποτέλεσμα θα είναι ένας έυκολος braid λαβύρινθος. Όμως, εξαιτίας των κύκλων που σχηματίζονται ένας απλός DFS δε θα μπορεί να βρει πάντοτε τη λύση, αφού είναι πολύ πιθανό να πέσει σε κύκλο. Επομένως θα πρέπει να τροποποιηθεί ο DFS ή να χρησιμοποιηθεί κάποιος άλλος αλγόριθμος, ο οποίος θα βρίσκει εγγυημένα μια λύση.

Ζητούμενα:

- Συνάρτηση braid :: Maze → Maze, η οποία δέχεται ένα τέλειο λαβύρινθο και τον μετατρέπει σε braid, αφαιρώντας τους απαραίτητους τοίχους.
- Συνάρτηση solveBraid :: Maze → (Int, Int) → (Int, Int) → [(Int, Int)], η οποία δέχεται ένα braid λαβύρινθο και τις συντεταγμένες ενός κελιού αρχής (s_x, s_y) και ενός κελιού στόχου (g_x, g_y) και επιστρέφει μια λίστα με τα κελιά που ανήκουν στο μονοπάτι της λύσης. Μπορεί να υπάρχουν πολλά τέτοια μονοπάτια, αλλά δεν χρειάζεται να επιστρέφεται το συντομότερο.

Βοηθητικό Αρχείο

Σας δίνεται ένα αρχείο το οποίο περιέχει το `data type Maze` το οποίο θα πρέπει να χρησιμοποιήσετε και δύο συναρτήσεις, `rand` και `shuffle`, τις οποίες **προτείνεται** να χρησιμοποιήσετε *(αν τις χρειαστείτε)* στις λύσεις σας και να **μην** προσπαθήσετε να τις υλοποιήσετε ή να τις τροποποιήσετε.

- Συνάρτηση rand :: Int → Int, η οποία δέχεται έναν ακέραιο, έστω `m`, και επιστρέφει ένα τυχαίο ακέραιο στο διάστημα `0` έως `m-1`.
- Συνάρτηση shuffle :: [a] → [a], η οποία δέχεται μια λίστα και επιστρέφει ένα τυχαίο ανακάτεμμα της.

Παράδοση Άσκησης

Η άσκηση μπορεί να γίνει από ένα άτομο ή μια όμαδα δύο ατόμων (προτείνεται). Στη δεύτερη περίπτωση θα παραδοθεί **μόνο** από το ένα άτομο της ομάδας.

Τα ονόματα των συναρτήσεων που θα δημιουργήσετε πρέπει να είναι **ακριβώς** τα ίδια με αυτά που καθορίζονται από την παραπάνω εκφώνηση.

Θα πρέπει να παραδώσετε ένα αρχείο της μορφής sdiYY00NNN_sdiYY00NNN.zip με όνομα τους αριθμούς μητρώου σας. Το zip θα περιέχει **ένα** αρχείο maze.hs με τις λύσεις σας για τα παραπάνω ζητούμενα και ένα σύντομο README.pdf που θα περιγράφει την υλοποίηση σας και θα αναφέρει τυχόν παραδοχές που κάνατε.

Η ημερομηνία παράδοσης θα ανακοινωθεί σύντομα. Το παραδοτέο αρχείο .zip θα το στείλετε με *email* στις παρακάτω διευθύνσεις: `sdi1000103@di.uoa.gr`, `sdi1100157@di.uoa.gr` και `prondo@di.uoa.gr`. Ερωτήσεις σχετικά με την άσκηση θα πρέπει να απευθύνονται στα πρώτα δύο emails (Νίκος Φιλιππάκης και Κωνσταντίνος Τριανταφύλλου).