12/20/2016

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

2η εργασία

Αχιλλέας Παλάσκος –
Μάριος Παλάσκος

ΤΑ ΣΤΟΙΧΕΙΑ ΜΑΣ...

ΟΝΟΜΑΤΕΠΩΝΥΜΑ: Παλάσκος Αχιλλέας , Παλάσκος Μάριος

ΑΕΜ: 8493 , 8492

ΤΗΛ: 6946949687 , 6986012613

ΗΛ. ΔΙΕΥΘΥΝΣΕΙΣ: palach2@yahoo.gr mariospala@gmail.com

ΠΕΡΙΕΧΟΜΕΝΑ

int [] consecutiveVerTiles (int x, int y, int color, Board board)

- **ΓΕΝΙΚΑ:**
  - Η συνάρτηση αυτή είναι γενική. Δέχεται ως ορίσματα τις συντεταγμένες x,y του ζαχαρωτού (ζαχαρωτό αναφοράς), το χρώμα του και το αντικείμενο board Board.
  - Επιστρέφει ένα πίνακα 1x 2 (myarray) όπου:
    - myarray[0]: περιέχει τον αριθμό των ζαχαρωτών που φεύγουν στην στήλη x (sumCol).
    - myarray[1]: περιέχει τη θέση στην κατακόρυφη στήλη του χαμηλότερου στοιχείου που φεύγει (minj).

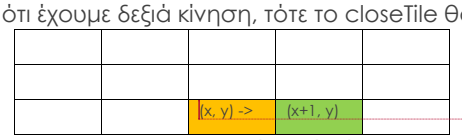
- **ΔΙΑΔΙΚΑΣΙΑ:**
  - Στη συνάρτηση αυτή ορίζονται κατ' αρχάς οι μεταβλητές sumCol και minj όπως ορίστηκαν παραπάνω. Επίσης ορίζεται και η j η οποία είναι ένας δείκτης στα ζαχαρωτά που βρίσκονται πάνω και κάτω από το ζαχαρωτό αναφοράς.
  - Αυτή αρχικοποιείται στην τιμή y+1 ώστε να δείχνει στο ζαχαρωτό ακριβώς πάνω από το ζαχαρωτό αναφοράς και όσο είμαστε εντός του ταμπλό (j<CrushUtilities.NUMBER_OF_PLAYABLE_ROWS) ελέγχουμε πόσα συνεχόμενα ζαχαρωτά ίδιου χρώματος με αυτό του ζαχ. αναφοράς υπάρχουν και κάθε φορά που βρίσκεται ένα αυξάνουμε τις sumCol και j. Αν βρεθεί ζαχαρωτό διαφορετικού χρώματος κάνουμε break.
  - Η minj αρχικοποιείται στην τιμή y, διότι είτε σχηματιστεί 3,4,5άδα πάνω από το ζαχ. αναφοράς το ελάχιστο j θα είναι το ίδιο το y.
  - Στη συνέχεια ακολουθούμε τη ίδια διαδικασία για τα ζαχαρωτά που βρίσκονται από κάτω. Η j αρχικοποιείται στην y-1 και ο έλεγχος γίνεται όσο είμαστε εντός του ταμπλό, δηλαδή j>=0. Για κάθε ζαχαρωτό ίδιου χρώματος αυξάνεται η sumcol, η minj παίρνει την τιμή της j και μετά μειώνεται η j.
  - Για την επιστροφή του πίνακα ελέγχουμε 2 περιπτώσεις: Αν δεν φεύγουν ζαχαρωτά (sumCol < 3) και αν φεύγουν, οπότε ως πρώτο στοιχείο επιστρέφεται ο αντίστοιχος αριθμός ζαχαρωτών που φεύγουν και ως δεύτερο η τιμή της minj σε περίπτωση που φεύγουν αλλιώς ο αριθμός των γραμμώς του ταμπλό, διότι είναι εκτός των ορίων του ταμπλό και είναι χαρακτηριστική για να μπορούμε να ελέγχουμε αν στη στήλη έχουν φύγει ή όχι ζαχαρωτά.

int [] consecutiveHorTiles (int x, int y, int color, Board board)

- **ΓΕΝΙΚΑ:**
  - Η συνάρτηση αυτή κάνει την ίδια δουλειά (με μικρές αλλαγές) με την consecutiveVerTiles με τη διαφορά ότι ελέγχει στη γραμμή του ζαχαρωτού για τυχόν 3,4,5άδες. Δέχεται τα ίδια ορίσματα με την consecutiveVerTiles.
  - Επιστρέφει ένα πίνακα 1x 2 (myarray) όπου:
    - myarray[0]: περιέχει τον αριθμό των ζαχαρωτών που φεύγουν στη γραμμή y (sumRows). Αν δεν φεύγουν (προφανώς) επιστρέφεται 0.
    - myarray[1]: περιέχει τη γραμμή y αν φεύγουν ζαχαρωτά αλλιώς τον αριθμό των γραμμών του ταμπλό για τον ίδιο λόγο.
- **ΔΙΑΔΙΚΑΣΙΑ:**
  - Στη συνάρτηση αυτή ορίζονται οι μεταβλητές sumRows και η i η οποία είναι ένας δείκτης στα ζαχαρωτά που βρίσκονται δεξιά και αριστερά από το ζαχαρωτό αναφοράς.
  - Η i αρχικοποιείται στην τιμή x+1 ώστε να δείχνει στο αμέσως δεξιά ζαχαρωτό από το ζαχαρωτό αναφοράς και όσο είμαστε εντός του ταμπλό (j<CrushUtilities.NUMBER_OF_COLUMNS) ελέγχουμε πόσα συνεχόμενα ζαχαρωτά ίδιου χρώματος με αυτό του ζαχ. αναφοράς υπάρχουν και κάθε φορά που βρίσκεται ένα αυξάνουμε τις sumRows και i. Αν βρεθεί ζαχαρωτό διαφορετικού χρώματος κάνουμε break.
  - Στη συνέχεια ακολουθούμε τη ίδια διαδικασία για τα ζαχαρωτά που βρίσκονται από αριστερά. Η i αρχικοποιείται στην x-1 και ο έλεγχος γίνεται όσο είμαστε εντός του ταμπλό, δηλαδή i>=0. Για κάθε ζαχαρωτό ίδιου χρώματος αυξάνεται η sumRows και μειώνεται η i.
  - Για την επιστροφή του πίνακα ελέγχουμε 2 περιπτώσεις: Αν δεν φεύγουν ζαχαρωτά (sumRows < 3) και αν φεύγουν, οπότε επιστρέφονται οι τιμές όπως περιγράφεται στα γενικά της συνάρτησης.

int[] deletedCandies(int[] move, Board board)

- **ΓΕΝΙΚΑ:**
  - Δέχεται μία απο τις διαθέσιμες κινήσεις και το ταμπλό τη δεδομένη στιγμή
  - Επιστρέφει ενα vector(1x3) με όνομα results που έχει ως στοιχεία:
    - results[0]: συνολικα tiles που διαγράφονται σε γραμμή και στήλη
    - results[1]: το ύψος απο το οποιο διαγραφεται το χαμηλοτερο tile της κατακορυφης 3/4/5αδας(αν εχουμε δυο στηλες παιρνουμε το ελαχιστο απο τα δυο)
    - results[2]: το υψος της γραμμης που διαγραφεται(αν στη κινηση διαγραφονται tiles απο δυο γραμμες, παιρνουμε το υψος της χαμηλοτερης(απο το y=0) γραμμης)
- **ΔΙΑΔΙΚΑΣΙΑ:**
  - Οι τέσσερις πίνακες που ορίζουμε στην αρχή r1, r2, r3, r4
    Είναι δυσδιάστατοι και θα δεχθούν το αποτέλεσμα των συναρτήσεων consecutiveHorTiles & consecutiveVerTiles. Τα διανύσματα αυτά είναι διαφορετικά για κάθε μια από τις 4 διαθέσιμες κινήσεις(πάνω, κάτω, αριστερά, δεξιά), αλλά ο πίνακας result για να υπολογιστεί, χρειάζεται πρώτα να δώσουμε τιμές στα r1, r2, r3, r4, γι'αυτό και ο υπολογισμός του γίνεται έξω από το switch ελέγχου της κίνησης.
  - Πώς καλούνται οι δύο συναρτήσεις:
    Αρχικά, δημιουργούμε ένα αντικείμενο moveTile που αναφέρεται στο tile που θα μετακινηθεί (εδώ εκείνου που βρίσκεται στη θέση (x, y), όπου x=move[0], y=move[1]). Στη συνέχεια παίρνουμε περιπτώσεις για την κίνηση και δημιουργούμε το αντίστοιχο αντικείμενο closeTile που αναφέρεται στο διπλανό αντικείμενο.
    Έστω ότι έχουμε δεξιά κίνηση, τότε το closeTile θα έχει συν/νες (x+1, y).



Εφόσον το moveTile θα μετακινηθεί δεξιά τότε οι r1, r2 «αφορούν το moveTile» και παίρνουν ορισματα:
(x+1, y, color(x, y)), δλδ παίρνουν το χρώμα του Tile που θα μετακινηθεί και τις συν/νες του σημείου **μετά** τη μετακίνηση του, διότι μας ενδιαφέρουν τα Tiles ίδιου χρώματος που βρίσκονται ΔΙΠΛΑ ΣΤΟ ΣΗΜΕΙΟ ΚΑΤΑΛΗΞΗΣ! Η r1 επιστρέφει (σε πίνακα) πόσα tiles φεύγουν στη στήλη x+1 και και το ύψος του χαμηλότερου tile της ν-αδας(σε περίπτωση που διαγράφονται, αλλιως επιστρεφει το ύψος του board), ενώ η r2 πόσα tiles φεύγουν στη γραμμή y δεξια του (x, y) και το ύψος της γραμμής(που είναι y σε περιπτωση που φευγουν αλλιώς επιστρέφει το μήκος του board).
Παρομοίως εργαζόμαστε και για το διπλανό tile: δίνουμε τα ορίσματα (x, y, color(x+1,y)) στις r3, r4 και επιστρέφουν tiles που φεύγουν στη στήλη y (r3) και

Commented [AP1]: Το κίτρινο κελί αναφέρεται στο **movetile** ενώ το πράσινο στο **closetile.**

αριστερά του (x, y) στη γραμμή y (r4) και το ύψος του χαμηλότερου tile της ν-αδας και το ύψος της γραμμής (ισχύουν τα ιδια με πριν στην περίπτωση που δεν διαγράφονται tiles). Στη συνέχεια το result[1] περιέχει το min από τα ελάχιστα τών στηλών x, x+1 και το result[2] το min από τα ελάχιστα της γραμμής(που θα είναι είτε y είτε το μέγεθος του πίνακα).

Σε περίπτωση αριστερής κίνησης έχουμε τις στήλες x, x-1 και γραμμή y και η διαδικασία είναι ακριβώς η ίδια.

Αν έχουμε κατακόρυφη κίνηση, τότε τα δύο σημεία που μας ενδιαφέρουν είναι τα (x, y)->(x, y+1) για κίνηση προς τα πάνω και (x, y) ->(x, y-1) για κίνηση προς τα κάτω. Σε αυτή την περίπτωση ελέγχουμε σε μία στήλη και σε δύο γραμμές. Συνεπώς οι r1 και r3 θα αναφέρονται στην ίδια στήλη x και οι r2, r4 στις γραμμες y+1, y αντίστοιχα για προς τα πάνω κίνηση και στις γραμμές y-1 ,y αντίστοιχα.

double moveEvaluation (int [] move, Board board)

- **ΓΕΝΙΚΑ:**
  Η συνάρτηση αυτή δέχεται ως ορίσματα τον πίνακα της κίνησης και το ταμπλό και επιστρέφει την αξιολόγηση/βαθμολογία της.
- **ΔΙΑΔΙΚΑΣΙΑ:**
  Ορίζεται η μεταβλητή totalPoints που περιέχει τους συνολικούς πόντους/βαθμολογία της εκάστοτε κίνησης. Καλείται deletedCandies() που επιστρέφει τον πίνακα moveResults και ορίζεται η totalCandies (συνολικά ζαχαρωτά που φεύγουν) η οποία είναι ίση με την moveResults[0]. Στην συνάρτηση αυτή, για την αξιολόγηση της κάθε κίνησης λαμβάνονται υπόψη 3 κριτήρια με την εξής προτεταιότητα (μεγαλύτερη προτεραιότητα σημαίνει μεγαλύτερο συντελεστή βαρύτητας στους συνολικούς πόντους):
  1. Σύνολο ζαχαρωτών που απομακρύνονται.
  2. Ελάχιστο ύψος γραμμής απο την οποία φεύγουν ζαχαρωτά.
  3. Χαμηλότερο στοιχείο απο στήλη.

  - Κατ' αρχάς αν δεν φεύγουν στοιχεία (totalCandies=0) τότε η κίνηση έχει μηδενική βαθμολογία.
  - Σε αντίθετη περίπτωση:

- κάθε ζαχαρωτό που φεύγει δίνει 150 πόντους , άρα ελάχιστοι πόντοι:
- 3x150 = 450.
- Αν υπάρχει γραμμή από τη οποία φεύγουν στοιχεία, τότε θεωρούμε εκθετική αύξηση ανάλογα με το πόσο κάτω (κοντά στο y=0) είναι, δηλαδή αν είναι στην 3η γραμμή τότε έχω exp(10-3) = exp(7). Επίσης, ο εκθετικός παράγοντας πολλαπλασιάζεται με έναν συντελεστή 0.01, ώστε να έχει μεγαλύτερο συντελεστή από το επόμενο κριτήριο. Έτσι, η μέγιστη δυνατή συγκομιδή πόντων από το κριτήριο αυτό να είναι: 0.01*exp(10-0)≅ 220
- Αν υπάρχει στήλη από τη οποία φεύγουν στοιχεία, τότε θεωρούμε και πάλι εκθετική αύξηση ανάλογα με το πόσο κάτω (κοντά στο y=0) είναι. Επίσης, ο εκθετικός παράγοντας πολλαπλασιάζεται με έναν συντελεστή 0.005, απ'όπου

Commented [AP2]: (totalPoints=150.0 * totalCandies)

Commented [AP3]: moveResults[2]!=CrushUtilities.NUMBER_OF_PLAYABLE_ROWS

Commented [AP4]: Math.exp(CrushUtilities.NUMBER_OF_PLAYABLE_ROWS-moveResults[2])

Commented [AP5]: moveResults[1]!=CrushUtilities.NUMBER_OF_PLAYABLE_ROWS

Commented [AP6]: Math.exp(CrushUtilities.NUMBER_OF_PLAYABLE_ROWS-moveResults[1])

φείνεται αυτό που προαναφέρθηκε πριν, οτι αυτό το κριτήριο έχει μικρότερο συντελεστή. Έτσι, η μέγιστη δυνατή συγκομιδή πόντων από το κριτήριο αυτό να είναι: 0.005*exp(10-0)≅ 110

ΣΗΜΕΙΩΣΕΙΣ - ΠΑΡΑΤΗΡΗΣΕΙΣ

1. Η παραπάνω προτεραιότητα λήφθηκε διαισθητικά και προφανώς δεν είναι απολύτως ορθή αφού τα κριτήρια αυτά δεν είναι ανεξάρτητα μεταξύ τους.
2. Το κριτήριο «ελάχιστο ύψος γραμμής» θεωρήθηκε πιο σημαντικό από το κριτήριο «χαμηλότερο στοιχείο στήλης», γιατί όταν θα φύγουν στοιχεία σε μια γραμμή, τότε περισσότερα ζαχαρωτά θα «κατέβουν» και άρα περισσότεροι συνδυασμοί ίδιων χρωμάτων είναι πιθανό να σχηματιστούν.
3. Στα κριτήρια 2 και 3 έγινε χρήση της εκθετικής συνάρτησης , γιατί προφανώς οι μετέπειτα (αφού φύγουν τα candies της κίνησης) πιθανοί συνδυασμοί (όμοιων χρωμάτων) δεν είναι γραμμική εξάρτηση του ύψους της γραμμής αλλά όσο πιο κάτω φεύγουν στοιχεία όλο και καλυτερο είναι.
4. Στα κριτήρια 2 και 3, οι συντελεστές των εκθετικών επιλέχθηκαν έτσι ώστε το μέγιστο των πόντων που μπορεί να ληφθεί από αυτά τα μαζί (220+110=330 πόντοι περίπου) να είναι μικρότερο από το ελάχιστο των πόντων που μπορούμε να έχουμε από το πρώτο κριτήριο (450 πόντοι), ώστε μία κίνηση κατά την οποία δεν φεύγουν ζαχαρωτά να είναι πάντα χειρότερη από μία στην οποία φεύγουν.
5. Επίσης, όσο πιο κεντρική είναι η κίνηση, τόσο καλύτερη είναι, γιατί μπορούν να σχηματιστούν συνδυασμοί προς περισσότερες κατευθύνσεις. Ωστόσο, το κριτήριο αυτο δεν λήφθηκε υπόψιν, γιατί η επίδρασή του δεν θα επηρέαζε και πολύ το αποτέλεσμα στη συγκεκριμένη άσκηση.

int findBestMoveIndex (Arraylist<int[]> availableMoves, Board board)

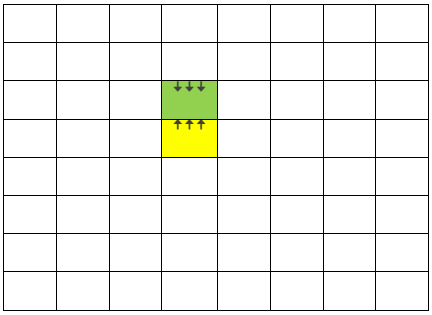
Στη συνάρτηση αυτή ορίζεται ο μονοδιάστατος πίνακας evals μεγέθους availableMoves.size() που περιέχει για κάθε μια από τις διαθέσιμες κινήσεις την αξιολόγησή της με βάση την συνάρτηση moveEvaluation. Η τοποθέτηση των τιμών στον evals γίνεται μέσω του loop for, όπου σε κάθε επανάληψη καλείται η συνάρτηση αξιολόγησης με ορίσματα το αντικείμενο board Board και την κίνηση νούμερο i (η αρίθμηση ξεκινά από το 0) που λαμβάνεται με την εντολή availableMoves.get(i). Μετά την τοποθέτηση των τιμών γίνεται μια «κλασική» αναζήτηση στον πίνακα αυτόν για την εύρεση της μέγιστης τιμής. Παράλληλα κρατάμε τον δείκτη j που δείχνει τον αριθμό της κίνησης με τη μέγιστη αξιολόγηση στην i-στή επανάληψη. Μετά το τέλος του loop αυτού επιστρέφεται η τιμή του j, αφού δείχνει στην κίνηση με τη μέγιστη βαθμολογία.

boolean nearTiles (int x, int y, int color, Board board) &&

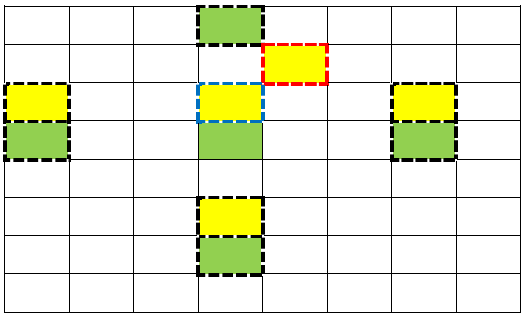
int noTakenCandies (int [] move, Board board)

Οι συναρτήσεις nearTiles και noTakenCandies δεν χρησιμεύουν στη συγκεκριμένη άσκηση, διότι καλούνται στην περίπτωση που από τις διαθέσιμες κινήσεις δεν υπάρχει καμία με την οποία να διαγράφονται candies. Ωστόσο δεν υπάρχει καμία τέτοια κίνηση, διότι σε αυτή την περίπτωση γίνεται reset του board και παίζει ο επόμενος παίκτης. Αυτό δεν διευκρινίζεται μέσα στην εργασία και είχαμε την εντύπωση ότι σε περίπτωση που δεν φεύγουν candies οι παίχτες απλώς μετακινούν tiles μέχρι να προκύψει κάποια ν-άδα. Για να μη πάει χαμένος ο κόπος μας θα ανεβάσουμε το σκεπτικό και τον κώδικα υλοποίησής του στο πρόγραμμα και αν θέλετε το ρίχνετε μια ματιά. Η λογική είναι η εξής:

Αν πρέπει να παίξουμε και δεν μπορούμε να διαγράψουμε candies, τότε απλώς επιλέγουμε εκείνη την κίνηση που δημιουργεί μια «ασπίδα» κύκλου ακτίνας 2 και κέντρου τη ΝΕΑ ΘΕΣΗ του κάθε tile. Αυτό σημαίνει ότι για το κάθε tile, από αυτά που ανταλλάχθηκαν, πρέπει να υπάρχει tile ίδιου χρώματος σε θέση τέτοια που μεταξύ τους να υπάρχουν τουλάχοστον 2 tiles διαφορετικού χρώματος, προκειμένου να μη μπορεί ο αντίπαλος να σχηματίσει κάποια ν-άδα με τα tiles που ανταλλάξαμε.



Δλδ για να επιλέξουμε την παραπάνω κίνηση, τότε η κοντινότερη θέση που μπορούν να έχουν άλλα tiles ίδιου χρώματος θα πρέπει είναι αυτές με τις ΜΑΥΡΕΣ διακεκομμένες που φαίνονται παρακάτω:



Παρατηρούμε ότι μας ενδιαφέρει η απόσταση «2 tiles» μόνο σε οριζόντια και κάθετη διεύθυνση από τη θέση που θα καταλήξει το tile που θα μετακινηθεί. Με αυτό τον τρόπο,

στο παραπάνω παράδειγμα αν υπάρχει ένα κίτρινο tiles(με τις κόκκινες διακεκομμένες) δίπλα και διαγώνια από το σημείο που θα μετακινηθεί το δικό μας tile(μπλε διακεκομμένες), τότε δε θα μπορεί να σχηματίσει τριάδα με το «δικό μας tile»μετακινούμενο αριστερά! Αν το πάνω-πάνω πράσινο κελί ήταν κίτρινο τότε θα μπορούσε. Αλλά τότε θα είχε απορριφθεί η κίνηση, καθώς το κίτρινο tile θα απείχε μόνο ένα κελί από το tile μας, πράγμα που έρχεται σε αντίθεση με την υπόθεσή μας.

Αυτό τον αλγόριθμο υλοποιεί η nearTiles(int x, int y, int color, Board board) που δέχεται σαν ορίσματα τις συν/νες (x,y) στις οποίες θα καταλήξει ένα tile και το χρώμα του και από εκείνο το σημείο ελέγχει προς τις 4 κατευθύνσεις την απόσταση του από άλλο όμοιου χρώματος tile. Στον αλγόριθμο απαιτείται προσοχή στην περίπτωση όπως εδώ του κίτρινου κελιού με μπλέ διακεκομμενες που δεν μπορεί να υπάρξει απόσταση «2 tiles» προς τα επάνω, διότι τότε έχουμε βγει έξω από τον πίνακα. Σε αυτή την περίπτωση, στην έξοδο της while, αν δεν έχουμε βρει όμοιο χρώμα(κίτρινο εδώ) και ταυτόχρονα ο δείκτης επανάληψης έχει γίνει ίσος με NUMBER_OF_PLAYABLE_ROWS, τότε f1=true. f1, f2, f3, f4 είναι 4 boolean μεταβλητές οι οποίες είναι true όταν προς την αντίστοιχη κατεύθυνση τηρείται η προϋπόθεσή μας. Όταν ταυτόχρονα προς τις 4 κατευθύνσεις η «απόσταση ασφαλείας» τηρείται τότε η nearTiles επιστρέφει true.

Η συνάρτηση int noTakenCandies(int[] move, Board board) παίρνει σαν ορίσματα την κίνηση και το ταμπλό και ουσιαστικά καλείται όταν δεν διαγράφονται candies (έχουμε βάλει ενα if-else στην moveEvaluation) και αξιολογεί ποιά κίνηση είναι καλύτερη τότε. Ορίζουμε δύο boolean μεταβλητές f1, f2 και ανάλογα με την κίνηση καλείται η nearTiles για καθένα από τα candies που ανταλλάσσονται. Αν τα δύο αυτά candies είναι ίδιου χρώματος, τότε αξιολογούμε την κίνηση με 1(αν ανταλλαξουμε δυο όμοιου χρώματος, τότε σε εκείνο το σημείο εφόσον δεν μπορέσαμε να σχηματίσουμε εμείς τριάδα, τότε ούτε και αντίπαλος προφανώς θα μπορέσει, οπότε είναι καλη κίνηση). Αχ όχι, τότε ελέγχουμε αν f1&&f2==1, καθώς θέλουμε η προϋπόθεση με τις αποστάσεις να τηρείται και από τα δύο tiles που θα ανταλλαχθούν. Σε αυτή την περίπτωση επιστρέφουμε 2(τη θεωρούμε καλύτερη κίνηση από το να αλλάξω δύο ίδια χρώματα). Αν τίποτα από τα δύο δεν τηρείται η συνάρτηση επιστρέφει 0.