Πρώτη Εργασία Τεχνητής Νοημοσύνης

Νικολαΐδης Σταύρος
ΑΕΜ: 3975

1. Μοντελοποίηση του προβλήματος

Κλάση Tube

Η κλάση αυτή χρησιμοποιείτε για την αναπαράσταση των δοχείων. Αποτελείται από 2 private fields και 10 public.

Private:

- Ένας πίνακας **st** τύπου char μεγέθους 4 που κρατάει το περιεχόμενο του δοχείου, τα χρώματα δηλαδή, με χαρακτήρες για κάθε χρώμα.
- Έναν δείκτη (index) **size** τύπου int για την περιεκτικότητα του δοχείου.

Public:

- Έναν **default constructor** που γεμίζει όλα τα δοχεία με τον κενό χαρακτήρα.
- Μια μέθοδο **push** που δέχεται ως όρισμα των χαρακτήρα που πρέπει να προστεθεί στην κορυφή του δοχείου και επιστρέφει true άμα ολοκληρωθεί με επιτυχία η προσθήκη, αλλιώς false.
- Μια μέθοδο **pop** που αφαιρεί το στοιχείο της κορυφής από τον πίνακα st. Επιστρέφει true άμα ολοκληρωθεί με επιτυχία η αφαίρεση, αλλιώς false.
- Μια μέθοδο **isEmpty** για έλεγχο του άμα το δοχείο είναι άδειο ή όχι.
- Μια μέθοδο **isFull** για έλεγχο του άμα το δοχείο είναι γεμάτο ή όχι.
- Μια μέθοδο **getTop** που επιστρέφει τον χαρακτήρα (χρώμα) που υπάρχει στην κορυφή του δοχείου.
- Μια **υπερφόρτωση του τελεστή =** (εκχώρηση) για να γίνεται εύκολη μεταφορά των στοιχείων ενός δοχείου σε ένα νέο αντιγραφό του, χρησιμοποιείται στον copy constructor της game παρακάτω.
- Μια **υπερφόρτωση του ==** (ισότητα) για έλεγχο αν 2 δοχεία έχουν το ίδιο περιεχόμενο. Χρησιμοποιείται για τον έλεχγο ισότητας των καταστάσεων αργότερα.
- Μια μέθοδο **getElement** τύπου char που δέχεται ως όρισμα έναν ακέραιο αριθμό i από το 0 έως και το 3 και επιστρέφει το στοιχείο που βρίσκεται στην θέση i του st.

Κλάση Game

Η κλάση αυτή χρησιμοποιείται για την αναπαράσταση μιας κατάστασης του παιχνιδιού. Αποτελείται από 3 private και 14 public πεδία.

Private:

- Μια μεταβλητή **n** τύπου int στην οποία αποθηκεύεται ο αριθμός των δοχείων για την κατάσταση.
- Έναν pointer **prev** τύπου Game που δείχνει στην διεύθυνση της προηγούμενης κατάστασης (γονέας της τρέχουσας κατάστασης).
- Έναν pointer tubes τύπου Tube όπου θα αποθηκευτούν τα n δοχεία της κατάστασης.

Public:

- Έναν **default constructor** που αρχικοποιεί την n με 7. Δεν χρησιμοποιείται κάπου απλώς επέλεξα να τον υλοποιήσω για ορισμένα tests.
- Έναν **constructor** που δέχεται ως όρισμα έναν ακέραιο αριθμό και αρχικοποιεί το n με αυτό το όρισμα και στην συνέχεια κατασκευάζει τον πίνακα tubes που περίεχει τα n δοχεία του παιχνιδιού. Έπειτα γεμίζει τα δοχεία με χρώματα στην τύχη (αξιοποιώντας την srand και την rand συνάρτηση) και θέτει τον pointer prev ίσο με nullptr γιατί ο constructor παράγει την αρχική κατάσταση του προβλήματος επομένως δεν έχει κατάσταση γονέα.
- Έναν **copy constructor** που δέχεται ως όρισμα μια κατάσταση και την αντιγράφει στην τρέχουσα, χρησιμοποιείται αργότερα στην expand μέθοδο.
- Έναν setter **setPrevious** που αλλάζει τον pointer previous από την κλάση game.
- Έναν getter **getPrevious** που επιστρέφει το prev της κατάστασης.
- Μια **move** μέθοδο τύπου bool που χρησιμοποιείται για την μετάγγιση υγρών από το ένα δοχείο στο άλλο. Δέχεται τρια ορίσματα, έναν pointer που δείχνει στην κατάσταση που θέλουμε να κάνουμε μετάγγιση και δύο int, a και b που είναι οι δείκτες (indexes) των δοχείων, a: από, b: προς. Η συνάρτηση ελέγχει άμα μπορεί να συμβεί η επιθυμητή μετάγγιση και άμα γίνει με επιτυχία επιστρέφει true αλλιώς false.
- Μια **print** μέθοδο τύπου void που εκτυπώνει την κατάσταση στην κονσόλα.
- Μια **υπερφόρτωση του τελεστή ==** που χρησιμοποιείται για να ελέγξουμε αργότερα, στην BFS συνάρτηση, αν δύο καταστάσεις είναι ίσες. Αξοιποιεί 2 for loops και ελέγχει αν κάθε δοχείο της μιας κατάστασης υπάρχει τουλάχιστον μία φορά στην άλλη κατάσταση.
- Μια μέθοδο **expand,** η οποία χρησιμοποιεί 2 for loops όπου υπολογίζει όλες τις πιθανές μεταγγίσεις και όσες είναι επιτρεπτές (καταστάσεις παιδιά) τις βάζει σε ένα vector το οποίο επιστρέφει.
- Μια μέθοδο **getKey** τύπου string η οποία επιστρέφει μια αναπαράσταση της κατάστασης σε string, χρησιμοποιείται αργότερα από την BFS συνάρτηση. Το string που επιστρέφεται αγνοεί την διάταξη με έξυπνο τρόπο. Αρχικά δημιουργείται ένας πίνακας τύπου string με n θέσεις και μέσα σε αυτόν περνάμε τα περιεχόμενα του πίνακα και μια παύλα ‘-’ στο τέλος του string για διαχωρισμό των δοχείων. Σε περίπτωση κενού δοχείου θα έχουμε απλά μια συμβολοσειρά που θα αποτελείται μόνο από τον χαρακτήρα της παύλας. Έπειτα, ταξινομεί τον πίνακα αυτόν και δημιουργεί την συμβολοσειρά κάνοντας append κάθε στοιχείο του πίνακα στην συμβολοσειρά που επιστρέφεται. Η ταξινόμηση ουσιαστικά αγνοεί την διάταξη των δοχείων αφού θα ταξινομούμε τις συμβολοσειρές των δοχείων κάθε κατάστασης.
- Μια μέθοδο **getDepth**, η οποία επιστρέφει το βάθος της τρέχουσας κατάστασης, μετρώντας όλους τους προγόνους της τρέχουσας κατάστασης.

- Μια μέθοδο **getGoal** η οποία επιστρέφει έναν pointer της goal κατάστασης. Υπολογίζει το goal μιας κατάστασης (συνήθως της αρχικής) βάζοντας σε κάθε ένα από τα n-2 δοχεία από 4ml του ίδιου χρώματος.
- Μια αναδρομική μέθοδο **printPath** τύπου void, η οποία δέχεται μια κατάσταση (Game) με αναφορά και εμφανίζει με την σειρά που έγιναν οι κινήσεις ξεκινώντας από την αρχική κατάσταση (τον πιο μακρινό γονέα της κατάστασης που περάσαμε ως όρισμα στην συνάρτηση). Η αναδρομή μας επιτρέπει να εμφανίσουμε τις κινήσεις από την αρχή προς το τέλος και όχι ανάποδα.

Κλάση Timer

Χρησιμοποιείται και μια κλάση Timer για μέτρηση και εμφάνιση του χρόνου που θα χρειαστεί το πρόγραμμα να εκτελεστεί. Θεωρώ ότι δεν χρειάζεται να την αναλύσω πολύ, ουσιαστικά χρησιμοποιεί την βιβλιοθήκη chrono. Επίσης την είχα χρησιμοποιήσει στην εργασία των Δομών Δεδομένων στο 2ο εξάμηνο.

2. Breadth First Search αλγόριθμος

Για την υλοποιήση του BFS αλγορίθμου έχω φτιάξει μια συνάρτηση στο αρχείο “main.cpp” η οποία δέχεται τρία ορίσματα: την αρχική κατάσταση του προβλήματος και 2 long μεταβλητές με αναφορά για μέτρηση των καταστάσεων που έγιναν expand (**mem**) και εξετάστηκαν (**examined**).

Χρησιμοποιείται μια δομή ουράς (queue) όπου αποθηκεύεται το μέτωπο αναζήτησης (**agenda**) δηλαδή οι καταστάσεις που έχουν γίνει expand αλλά δεν τις έχουμε επισκεφτεί ακόμα. Ακόμα, χρησιμοποιείται ένα unordered map με key τύπου string και value τύπου Game*, όπου κρατάμε τις καταστάσεις όπου έχουμε ήδη επισκεφθεί (**closed**), το κλειστό σύνολο δηλαδή.

Έπειτα τοποθετούμε την αρχική κατάσταση στο μέτωπο αναζήτησης και έπειτα ξεκινάμε να κάνουμε expand τα παιδιά της αρχικής κατάστασης και αρχίζουμε να τα επισκεφτόμαστε όλα ένα-ένα. Άμα βρούμε τελική κατάσταση τότε την επιστρέφουμε και η συνάρτηση σταματά, αλλιώς κάνουμε expand την κατάσταση, ελέγχοντας αν υπάρχουν στο unordered map (κλειστό σύνολο) ήδη τα παιδιά της ή ισοδύναμες καταστάσεις με το ίδιο κλειδί ώστε να μην τα προσθέσουμε στο μέτωπο αναζήτησης, και τοποθετούμε τα παιδιά της στο μέτωπο και επισκεπτόμαστε την επόμενη κατάσταση, που είναι η επόμενη κατάσταση του ίδιου επιπέδου στο δένδρο (εκτός αν είναι η τελευταία του επιπέδου) αφού αξιοποιούμε την δομή της ουράς για το μέτωπο.

Ο αλγόριθμος άμα δεν βρει λύση τότε απλά θα επιστρέψει nullptr.

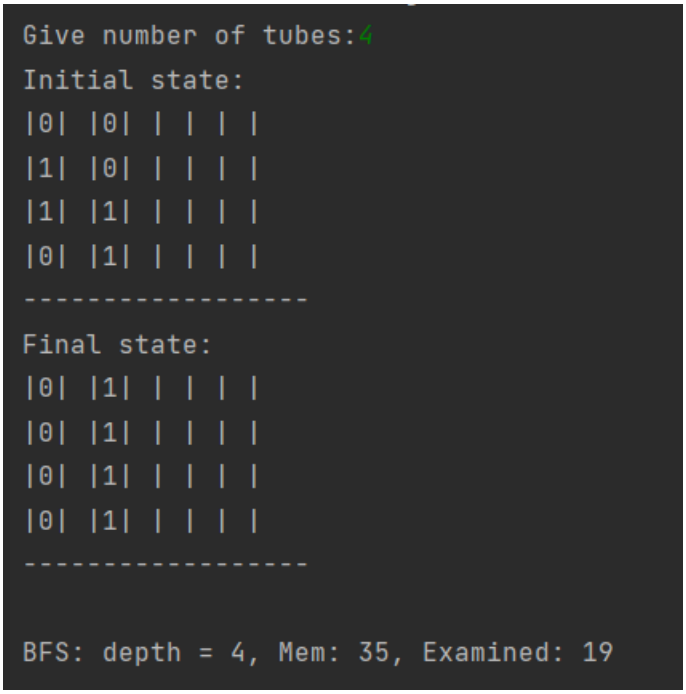
Να σημειωθεί ότι η χρήση unordered map εγγυάται πιο γρήγορη προσπέλαση των στοιχείων (καταστάσεων) χάρη στο μοναδικό key τους σε αντίθεση με μια υλοποίηση η οποία χρησιμοποιεί vector.

Ο αλγόριθμος επιστρέφει πάντα την καλύτερη λύση.

3. Αποτελέσματα για σενάρια

- Για n = 4:

Απλή περίπτωση για να δείξω στο word ότι εμφανίζεται η λύση βήμα προς βήμα, γιατί αν το έκανα με μεγαλύτερο n τότε θα έπιανε πολύ χώρο στο word.



Βήμα προς βήμα λύση στην επόμενη σελίδα

```
Step by step:
|0| |0| | | | |
|1| |0| | | | |
|1| |1| | | | |
|0| |1| | | | |
------------------
| | |0| | | | |
|1| |0| | | | |
|1| |1| | | | |
|0| |1| |0| | |
------------------
| | | | | | | |
|1| | | |0| | |
|1| |1| |0| | |
|0| |1| |0| | |
------------------
| | |1| | | | |
| | |1| |0| | |
| | |1| |0| | |
|0| |1| |0| | |
------------------
| | |1| |0| | |
| | |1| |0| | |
| | |1| |0| | |
| | |1| |0| | |
------------------
```

Χρόνος εκτέλεσης:

```
Seconds: 0.030019
```

- Για n = 12:

Μέση περίπτωση

```
Give number of tubes:12
Initial state:
|7| |0| |2| |2| |4| |3| |7| |5| |8| |6| | | | |
|9| |9| |2| |4| |3| |1| |7| |1| |5| |1| | | | |
|0| |6| |4| |7| |8| |0| |6| |5| |9| |5| | | | |
|6| |9| |2| |3| |4| |0| |1| |3| |8| |8| | | | |
------------------
Final state:
|0| |1| |2| |3| |4| |5| |6| |7| |8| |9| | | | |
|0| |1| |2| |3| |4| |5| |6| |7| |8| |9| | | | |
|0| |1| |2| |3| |4| |5| |6| |7| |8| |9| | | | |
|0| |1| |2| |3| |4| |5| |6| |7| |8| |9| | | | |
------------------

BFS: depth = 30, Mem: 19965, Examined: 19459
```

Χρόνος εκτέλεσης:

```
Seconds: 2.36538
```

- Για n = 19:

  Χειρότερη περίπτωση αφού στην εκφώνηση ζητάει για n < 20.

```
Give number of tubes:19
Initial state:
|b| |4| |b| |f| |e| |f| |6| |0| |8| |9| |7| |2| |g| |5| |d| |1| |1| | | | |
|2| |7| |g| |8| |9| |6| |2| |c| |0| |a| |d| |c| |g| |9| |a| |4| |1| | | | |
|0| |b| |3| |b| |5| |7| |3| |f| |e| |a| |7| |g| |c| |8| |a| |d| |4| | | | |
|6| |3| |c| |e| |9| |f| |3| |5| |6| |0| |2| |e| |8| |4| |5| |1| |d| | | | |
------------------
Final state:
|0| |1| |2| |3| |4| |5| |6| |7| |8| |9| |a| |b| |c| |d| |e| |f| |g| | | | |
|0| |1| |2| |3| |4| |5| |6| |7| |8| |9| |a| |b| |c| |d| |e| |f| |g| | | | |
|0| |1| |2| |3| |4| |5| |6| |7| |8| |9| |a| |b| |c| |d| |e| |f| |g| | | | |
|0| |1| |2| |3| |4| |5| |6| |7| |8| |9| |a| |b| |c| |d| |e| |f| |g| | | | |
------------------

BFS: depth = 52, Mem: 93478, Examined: 92772
```

Χρόνος εκτέλεσης:

```
------------------
Seconds: 37.0187
```