HY255 Εργαστήριο Λογισμικού – L13

Assignment 4 - sudoku

HY255 Εργαστήριο Λογισμικού
Άνοιξη 2021
Άγγελος Μπίλας

Άσκηση 4 – Λύση sudoku puzzles

- Συναρτήσεις input/output: Να γράψουμε συναρτήσεις read(), print(), check() που διαβάζουν, τυπώνουν και ελέγχουν ένα puzzle
  - Για τον έλεγχο, μπορούμε να χρησιμοποιήσουμε τους τρεις απλούς κανόνες που ορίζουν ένα σωστό sudoku puzzle: κάθε γραμμή, στήλη και υπο-τετράγωνο (tile) να περιέχει μια φορά όλους τους αριθμούς από το 1-9.
  - Οι συναρτήσεις read/print να διαβάζουν/τυπώνουν ακριβώς το format που ορίζει η άσκηση ώστε αργότερα να μπορείτε να παράγετε και να λύνετε τα δικά σας puzzles αυτόματα, π.χ. με: $ sudoku –g N | sudoku
- solve()
  - Για τους σκοπούς μας, ας πούμε ότι κάθε puzzle αναπαρίσταται ως ένα grid g
  - Για να βρούμε τις επιλογές που έχουμε σε κάποιο βήμα της διαδικασίας, μπορούμε σε κάθε τετράγωνο να σημειώσουμε όλες τις δυνατές επιλογές για το τετράγωνο αυτό. Στη συνέχεια κάθε φορά που βρίσκουμε μια τιμή για ένα τετράγωνο, αφαιρούμε αυτή την επιλογή από τα υπόλοιπα τετράγωνα της ίδιας γραμμής, στήλης, και tile.
  - Πως μπορούμε να αναπαραστήσουμε το grid g και τις επιλογές του κάθε τετραγώνου στο g;
- Απόφαση για το ποια τιμή να βάλουμε που
  - Καθώς εξετάζουμε ένα τετράγωνο του grid για τις δυνατές τιμές, μπορεί να έχουμε δύο περιπτώσεις
  - (α) Κάποιο τετράγωνο, σε κάθε βήμα της επίλυσης, να έχει μοναδική επιλογή ➔ Σε αυτή την περίπτωση λέμε ότι το puzzle έχει "λύση μοναδικής επιλογής"
  - (β) Κανένα τετράγωνο, σε κάποιο βήμα της επίλυσης, να μην έχει μοναδική επιλογή
- Ας εξετάσουμε πρώτα την περίπτωση (α)
  - Σε αυτή την περίπτωση, είναι εύκολη η απόφαση για το επόμενο βήμα, αρκεί να εφαρμόσουμε αυτή την μοναδική επιλογή σε κάποιο από τα τετράγωνα. Αυτό είτε θα μας οδηγήσει σε λύση, είτε σε puzzle χωρίς λύση, αλλά σε αυτό το σημείο δεν έχουμε επιλογή.
  - Άρα ο κώδικας θα είναι κάπως σαν

```
solve(grid g){
  while(!done){
    for (;;) { /* Βρες ένα στοιχείο i,j με μοναδική επιλογή n και εφάρμοσέ της στο στοιχείο αυτό */ }
    eliminate_choice(I,j,n); /* Αφαίρεσε την μοναδική επιλογή από τα στοιχεία της ίδιας γραμμής, στήλης, tile */
  }
}
```

  - Αυτή η επαναληπτική συνάρτηση θα τελειώσει διότι κάνουμε την υπόθεση ότι σε κάθε βήμα υπάρχει στοιχείο με μοναδική επιλογή. Αν το τελικό puzzle δεν είναι σωστό, σημαίνει ότι δεν υπάρχει λύση
  - Σε αυτή την περίπτωση επομένως, δεν υπάρχει αβεβαιότητα στις επιλογές που κάνουμε
- Υλοποιήστε πρώτα αυτή την περίπτωση (a) ώστε μετά να ασχοληθούμε με την γενική περίπτωση

Γενική περίπτωση

- Περίπτωση (β) → Λύση ”μη μοναδικής επιλογής”: Τι γίνεται αν σε κάποιο βήμα της επίλυσης όλα τα τετράγωνα που απομένουν έχουν πάνω από μια επιλογές
  - Άραγε σημαίνει αυτό ότι το puzzle έχει πάνω από μια λύσεις; Όχι απαραίτητα… Απλά στο σημείο αυτό δεν έχουμε απλό τρόπο να διαλέξουμε το επόμενο βήμα
- Για το τετράγωνο ? οι κανόνες μας για την απάλειψη επιλογών μας οδηγούς στις επιλογές: 1,3,5
  - Μπορούμε να βρούμε εμείς την τιμή του τετραγώνου αυτού κοιτάζοντας το puzzle; (→ 5, γιατί;)
  - Όταν λύνουμε εμείς ένα puzzle, εφαρμόζουμε συνήθως με το μυαλό μας heuristics που μας επιτρέπουν να κάνουμε σωστές επιλογές και σε αυτή την περίπτωση
  - Τα heuristics αυτά έχουν την μορφή: “Αν το τετράγωνο αυτό έχει την τιμή Χ τότε αυτό δεν θα μας οδηγήσει σε λύση γιατί ”κάποιο σκεπτικό”
- Το αντίστοιχο στο πρόγραμμά μας είναι να κάνουμε “δοκιμές” με τιμές που είναι ενδεχόμενες επιλογές και αν δεν μας οδηγούν πουθενά να μην επηρεάζουν την κατάσταση του προγράμματός μας
- Η αναδρομή είναι ένας εύκολος τρόπος να εκφράσουμε αυτές τις δοκιμές
- Η στοίβα είναι ο μηχανισμός που μας επιτρέπει (με προσεκτική χρήση) να ΜΗΝ αλλάζουμε την κατάσταση του προγράμματος με μόνιμο τρόπο, παρά μόνο προσωρινά

5								6
		3		1		7		
	6		2		5		3	
		5	6		8	3		
	9		?				7	
		8	4		7	2		
	5		7		9		2	
		1		5		9		
2								7

solve()

```
grid sudoku_solve(grid g){                          # To g είναι ένα αντίγραφο του grid στη στοίβα
                                                    # και επομένως οποιεσδήποτε αλλαγές δεν θα
                                                    # επηρεάσουν το υπόλοιπο πρόγραμμα
  grid tmp_g;                                       # αντίστοιχα με το g
  while( (i,j,n) = try_next(g) ) {                  # δοκίμασε όλες τις επιλογές που έχει το g,
                                                    # μια μια, με την όποια σειρά
    tmp_g ← update & eliminate g+(i,j,n)            # αλλάζει το tmp_g προσωρινά
    tmp_g ← sudoku_solve (tmp_g)                    # η sudoku δεν έχει καμιά άλλη επίδραση στη μνήμη,
                                                    # πέρα από την τιμή που επιστρέφει
     if (check(tmp_g)) return tmp_g;                # αν βρήκαμε την λύση ας την επιστρέψουμε
  }
  return tmp_g;                                     # δεν βρήκαμε λύση, ας επιστρέψουμε "κάτι"
}
```

- Κλειδί: Η sudoku_solve() είναι μια συνάρτηση που δεν αλλάζει τίποτε άλλο στη μνήμη του προγράμματος, πέρα από τις τοπικές μεταβλητές και την τιμή που επιστρέφει ➔ Λέμε ότι συνάρτησεις όπως η sudoku_solve() δεν έχουν side-effects
- Άρα όταν επιστρέφει η sudoku_solve(), όσες αναδρομικές κλήσεις και αν έχουν γίνει, δεν έχει αλλάξει η μνήμη του προγράμματος καθόλου, εκτός από την τιμή επιστροφής. Άρα, όσες δοκιμές και να κάνει η sudoku_solve(), το μόνο που θα μείνει τελικά είναι η τιμή που θα επιστρέψει.
- Αυτό είναι η μεγάλη βοήθεια της στοίβας και της αναδρομής, στο να εκφράζουμε εύκολα προβλήματα όπου δεν θέλουμε να υπάρχουν side-effects.
- Για να το πετύχουμε αυτό, οι τύποι των παραμέτρων και τοπικών μεταβλητών πρέπει να είναι προσεκτικά ορισμένοι, ώστε σε κάθε κλήση να δημιουργούνται νέα αντίγραφα.
- Optimizations: Πολλά – σκεφτείτε/δοκιμάστε όσο χρόνο έχετε

Απόδοση του προγράμματός μας

- Τελικά σε κάθε κλήση της συνάρτησης αντιγράφουμε στις input/return values της sudoku_solve() μεγάλα structs
  - Προσοχή: δεν περνάμε pointers γιατί διαφορετικά υπάρχουν side-effects
- Αυτές οι αντιγραφές/περάσματα μεγάλων μεταβλητών έχουν (σημαντικό) κόστος
- Για να το διαπυστώσετε, δοκιμάστε ένα puzzle που έχει λύση μοναδικής επιλογής να το λύσετε με τις δύο εκδόσεις του προγράμματός σας
  - ”Επαναληπτική” έκδοση που μπορεί να λύσει μόνο την περίπτωση (α)
  - ”Αναδρομική” έκδοση που μπορεί να λύσει όλες τις περιπτώσεις
- Δείτε πόσο διαφορά έχουν στον χρόνο εκτέλεσης…

Αναδρομή και global μεταβλητές

- Συνταγή για προβλήματα…
- Αν μια αναδρομική κλήση αλλάζει μια global μεταβλητή, στην περίπτωση των ”δοκιμών” μας θα πρέπει να επαναφέρουμε τις προηγούμενες τιμές των global μεταβλητών
- Αντί να το κάνουμε αυτό εμείς στο πρόγραμμά μας, ουσιαστικά το πετυχαίνει η στοίβα, χωρίς δική μας προσπάθεια, αρκεί οι αντίστοιχες συναρτήσεις να μην έχουν side-effects
- Μερικές γλώσσες υποχρεώνουν όλες τις συναρτήσεις να ΜΗΝ έχουν side-effects και να επιστρέφουμε τα αποτελέσματά τους μόνο μέσω της τιμής επιστροφής (και όχι άλλων αλλαγών)
- Αυτές οι γλώσσες ονομάζονται functional programming languages (π.χ. Lisp, Haskel) και έχουν πλεονεκτήματα σε μερικές περιπτώσεις (as opposed to procedural programming languages, όπως η C, Java, Scala)
  - Για εμάς το κυριότερο πλεονέκτημα θα ήταν ότι πολύ λίγα πράγματα μπορούν να πάνε στραβά στη μνήμη του προγράμματος, σε αντίθεση με προγράμματα στη C που σχεδόν ΌΛΑ μπορούν (και θα) πάνε στραβά με την μνήμη το προγράμματος
  - Όπως και σε πολλά άλλα πράγματα, τελικά η απόδοση του παραγόμενου προγράμματος, είναι μια άλλη σημαντική διάσταση
  - Αυτή η ισορροπία ανάμεσα σε ”ευκολία” (με την ευρεία έννοια) και απόδοση είναι ακόμη και σήμερα κάτι που δεν έχει τέλεια λύση. Για αυτό και τέτοια θέματα εξακολουθούν να είναι προβλήματα που μας κεντρίζουν την προσοχή και το ενδιαφέρον.

Δημιουργία puzzles

- Πως δημιουργούμε ένα puzzle με περίπου n elements συμπληρωμένα;
  - Όσο μεγαλύτερο είναι το n τόσο μικρότερο το search space για την αναδρομή μας (και κατά μια έννοια πιο γρήγορη η όποια λύση)
- Δημιουργούμε ένα πλήρες και σωστό grid
- Αφαιρούμε τυχαία στοιχεία έως ότου μείνουν n elements μόνο συμπληρωμένα
  - Αν θέλουμε το puzzle να έχει λύση μοναδικής επιλογής, τότε δοκιμάζουμε να το λύσουμε και βλέπουμε αν η λύση είναι μοναδικής επιλογής
  - Λόγω τυχαιότητας, στην περίπτωση που θέλουμε puzzle με λύση μοναδικής επιλογής, βάζουμε ένα όριο στο πόσες φορές θα δοκιμάσουμε να παράγουμε το puzzle των n elements
- Πως διαλέγουμε ένα τυχαίο στοιχείο ή μια τυχαία τιμή σε ένα εύρος, π.χ. 1…N; Η stdlib της C μας δίνει δύο συναρτήσεις:
- rand(): επέστρεψε έναν random integer. Οπότε εμείς μπορούμε να πάρουμε τον αριθμός μας στο εύρος 1..N με την χρήση του modulo, ως rand()%N.
- srand(seed): όρισε την αρχική παράμετρο του random number generator της libc σε seed. Για το ίδιο seed, η libc θα παράγει την ίδια ακολουθία αριθμών σε κάθε εκτέλεση. Αυτό είναι χρήσιμο όταν κάνουμε debug το πρόγραμμά μας.
- Έχοντας τη δυνατότητα να παράγετε puzzles, μπορείτε πλέον να βάλετε το πρόγραμμά σας να τα παράγει και να τα λύνει διαρκώς, π.χ.

```
sudoku –g 70 –u | sudoku     # παρήγαγε ένα puzzle με λύση μοναδικής επιλογής που έχει
                             # περίπου 70 συμπληρωμένα στοιχεία  | λύσ'το
sudoku –g 50 | sudoku        # παρήγαγε ένα puzzle που έχει περίπου 50 συμπληρωμένα στοιχεία  | λύσ'το
```

Εργασία

- Διαβάστε την εκφώνηση για την Άσκηση 4
- Υλοποιήστε την Άσκηση 4