Κ08 Δομές Δεδομένων και Τεχνικές Προγραμματισμού (Τμήμα Φοιτητών/τριών με Άρτιο ΑΜ)

Διδάσκων: Μανόλης Κουμπαράκης

Εαρινό Εξάμηνο 2020-2021

Εργασία 1 (ανακοινώθηκε στις 6 Μαρτίου 2021, προθεσμία παράδοσης: 10 Απριλίου 2021, ώρα 23:59)

20% του συνολικού βαθμού στο μάθημα. Άριστα=180 μονάδες (υπάρχουν 150 μονάδες bonus, σύνολο 330 μονάδες)

Προσοχή: Πριν διαβάσετε παρακάτω, διαβάστε παρακαλώ προσεκτικά τις οδηγίες υποβολής των ασκήσεων που βρίσκονται στην ιστοσελίδα http://cgi.di.uoa.gr/~k08/homework.html, ειδικά ότι αναφέρεται στο github και τα σχετικά αρχεία.

Απορίες: Αν έχετε απορίες σχετικά με την εργασία, ρωτήστε στο piazza και όχι στέλνοντας e-mail στον διδάσκοντα. Τέτοια e-mails ΔΕΝ θα λαμβάνουν απάντηση.

Κώδικας: Ο κώδικας που παρουσιάσαμε στις διαλέξεις του μαθήματος (Ενότητες 1-5) και θα χρειαστείτε για την εργασία αυτή βρίσκεται στο παρακάτω private repository του classroom του μαθήματος: https://github.com/artioi-k08/2021-ergasia-1. Για να συνδεθείτε στο repository αυτό, θα πρέπει να χρησιμοποιήσετε το λινκ https://classroom.github.com/a/X-ER_4mK .

Όταν συνδεθείτε θα μπορείτε να δείτε το προσωπικό σας repository https://github.com/artioi-k08/2021-ergasia-1-<github-your-username> στο οποίο θα δουλέψετε για την Εργασία 1.

Μετά τη σύνδεση σας, στο προσωπικό σας repository, θα βρείτε τον κώδικα που καλύπτει τις Ενότητες 1-5 του μαθήματος οργανωμένο σε κατάλληλους φακέλους. Θα βρείτε επίσης και ένα φάκελο solutions-ergasia1 με υπο-φακέλους question1, question2, …, question9 στους οποίους θα πρέπει να γράψετε τον κώδικα σας για τα 9 ερωτήματα της εργασίας που θα βρείτε παρακάτω.

Παρακαλώ τηρείστε ευλαβικά αυτήν την οργάνωση αλλιώς θα χάσετε 20% του συνολικού βαθμού κατά την βαθμολόγηση.

Κύριο πρόγραμμα: Σε όλες τις παρακάτω προγραμματιστικές ασκήσεις θα πρέπει να υλοποιήσετε και ένα κύριο πρόγραμμα (συνάρτηση `main`) το οποίο θα διαβάζει τα δεδομένα εισόδου, θα επιδεικνύει τη λειτουργικότητα της συνάρτησης σας για κατάλληλα επιλεγμένες εισόδους, και θα πείθει τον βαθμολογητή ώστε να σας βαθμολογήσει με τον υψηλότερο δυνατό βαθμό.

1. Εξηγήστε πως μπορεί να χρησιμοποιηθεί μια στοίβα και μια ουρά για να δημιουργηθούν με μη αναδρομικό τρόπο όλα τα πιθανά υποσύνολα ενός πεπερασμένου συνόλου (αυτά τα σύνολα είναι μέλη του **δυναμοσύνολου** ενός συνόλου). Η απάντηση σας για το ερώτημα αυτό θα πρέπει να είναι σε ένα pdf αρχείο το οποίο θα βάλετε στον φάκελο solutions-ergasia1/question1 του repository σας.

   (5 μονάδες)

2. Στην άσκηση αυτή θα υλοποιήσετε τον αφαιρετικό τύπο δεδομένων **Διπλά Συνδεδεμένη Λίστα** χρησιμοποιώντας ενότητες (modules) της C και κάνοντας καλή απόκρυψη πληροφορίας.

   Κάθε κόμβος σε μια διπλά συνδεδεμένη λίστα περιέχει ένα δείκτη στον επόμενο κόμβο και ένα δείκτη στο προηγούμενο κόμβο. Είναι βολικό επίσης να έχουμε ειδικούς κόμβους στα άκρα μιας διπλά συνδεδεμένης λίστας: ένα **κόμβο κεφαλής** ακριβώς πριν από τον πρώτο κόμβο της λίστας, και ένα **κόμβο ουράς** ακριβώς μετά από τον τελευταίο κόμβο της λίστας. Θα πρέπει να ορίσετε μια κατάλληλη δομή της C για την αναπαράσταση μιας διπλά συνδεδεμένης λίστας, και να υλοποιήσετε τις παρακάτω πράξεις:

   - `Create`: Δημιουργεί μια κενή διπλά συνδεδεμένη λίστα.
   - `Size`: Επιστρέφει ένα ακέραιο που είναι ο αριθμός των στοιχείων της λίστας.
   - `IsEmpty`: Επιστρέφει 1 ή 0 ανάλογα αν η λίστα είναι κενή ή όχι.
   - `GetFirst`: Επιστρέφει τον πρώτο κόμβο της λίστας.
   - `GetLast`: Επιστρέφει τον τελευταίο κόμβο της λίστας.

- `GetPrev`: Επιστρέφει τον προηγούμενο ενός δοσμένου κόμβου.
- `GetNext`: Επιστρέφει τον επόμενο ενός δοσμένου κόμβου.
- `AddBefore`: Εισάγει ένα δοσμένο νέο κόμβο `z` πριν τον δοσμένο κόμβο `v`.
- `AddAfter`: Εισάγει ένα δοσμένο νέο κόμβο `z` μετά τον δοσμένο κόμβο `v`.
- `AddFirst`: Εισάγει ένα δοσμένο νέο κόμβο στην αρχή της λίστας.
- `AddLast`: Εισάγει ένα δοσμένο νέο κόμβο στο τέλος της λίστας.
- `Remove`: Αφαιρεί ένα δοσμένο κόμβο από τη λίστα.
- `Print`: Τυπώνει τα στοιχεία που περιέχονται στους κόμβους της λίστας, από τον πρώτο μέχρι τον τελευταίο.

Προσέξτε ότι δεν σας δίνουμε τους τύπους που θα χρειαστείτε για τον αφαιρετικό τύπο δεδομένων, ούτε τους τύπους των ορισμάτων για τις παραπάνω συναρτήσεις, αλλά περιμένουμε από σας να τους ορίσετε με τον καλύτερο δυνατό τρόπο!

Να δώσετε την υπολογιστική πολυπλοκότητα των παραπάνω πράξεων για διπλά συνδεδεμένες λίστες. Η απάντηση σας για την υπολογιστική πολυπλοκότητα θα πρέπει να είναι σε ένα pdf αρχείο το οποίο θα βάλετε στον φάκελο solutions-ergasia1/question1 του repository σας.

(30+5=35 μονάδες)

3. Να υλοποιήσετε μια αναδρομική συνάρτηση η οποία να βρίσκει το μέγιστο στοιχείο μιας συνδεδεμένης λίστας της οποίας οι κόμβοι περιέχουν ακέραιους αριθμούς.

(10 μονάδες)

4. Ένα πρόβλημα **κρυπταριθμητικής** είναι ένα πάζλ το οποίο αφορά αριθμητικές πράξεις μεταξύ γραμμάτων που παριστάνουν ψηφία φυσικών αριθμών. Για παράδειγμα, αν οι μεταβλητές *A, B, E, G, L, M* και *S* παριστάνουν ψηφία, δηλαδή παίρνουν τιμές από το σύνολο $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, ποια είναι η λύση του προβλήματος κρυπταριθμητικής

BASE + BALL = GAMES;

Δηλαδή, ποιες τιμές πρέπει να πάρουν οι μεταβλητές ώστε η παραπάνω πρόσθεση να ισχύει; Όπως μπορείτε να επιβεβαιώσετε, η λύση του πάζλ είναι η

A=4, B=7, E=3, G=1, L=5, S=8, M=9

επειδή 7483 +7455= 14938.

Οι τιμές που παίρνουν οι μεταβλητές *A, B* κλπ. πρέπει να είναι διαφορετικές μεταξύ τους.

Να γράψετε μία αναδρομική συνάρτηση η οποία λύνει ένα δοσμένο πρόβλημα πρόσθεσης όπως το παραπάνω.

(30 μονάδες)

5. Θεωρήστε το πρόγραμμα που δώσαμε στην Ενότητα 3 (Στοίβες) το οποίο μετατρέπει μια ενθεματική (infix) παράσταση σε μεταθεματική (postfix) χρησιμοποιώντας μια στοίβα. Μετατρέψτε το πρόγραμμα αυτό ώστε να αποτιμά την δοσμένη ενθεματική παράσταση. Η λύση σας θα πρέπει να παραδοθεί σαν ένα module της C.

(20 μονάδες)

6. Ο αλγόριθμος *ταξινόμησης* ενός πίνακα ακεραίων `A` *με εισαγωγή* λειτουργεί ως εξής. Διασχίζουμε τον πίνακα `A` από το πρώτο έως το τελευταίο στοιχείο του. Κάθε φορά που επισκεπτόμαστε ένα στοιχείο `A[i]`, το βάζουμε στη σωστή σειρά στον υποπίνακα `A[0..i-1]` μετακινώντας προς τα δεξιά όσα στοιχεία χρειάζεται να μετακινηθούν για να κάνουν χώρο για το εισαγόμενο στοιχείο. Όταν εκτελέσουμε αυτή τη διαδικασία για το τελευταίο στοιχείο του πίνακα, ο πίνακας είναι ταξινομημένος.

   Υλοποιήστε τον παραπάνω αλγόριθμο για να ταξινομήσετε μια απλά συνδεδεμένη λίστα. Μπορείτε να χρησιμοποιήσετε τους ορισμούς τύπων από το πρόγραμμα που παρουσιάστηκε στην πρώτη ενότητα διαλέξεων (Συνδεδεμένες Αναπαραστάσεις Δεδομένων) αφού αλλάξετε των τύπο των στοιχείων της λίστας ώστε να είναι ακέραιοι.

   (25 μονάδες)

7. Να υλοποιήσετε μια συνάρτηση η οποία παίρνει σαν όρισμα δύο λίστες, τις ταξινομεί χρησιμοποιώντας τον αλγόριθμο της προηγούμενης άσκησης, μετά τις ενώνει σε μια νέα ταξινομημένη λίστα, και επιστρέφει ένα δείκτη στην τρίτη λίστα.

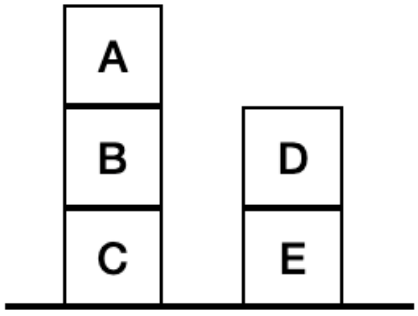
(25 μονάδες)

8. Υλοποιήστε τον αφαιρετικό τύπο δεδομένων Ουρά Προτεραιότητας (Priority Queue) που συζητήσαμε στην Ενότητα 3 των διαλέξεων ώστε να γίνεται καλή απόκρυψη πληροφορίας (information hiding) με χρήση modules της C και handles όπως δείξαμε στα εξής παραδείγματα αφαιρετικών τύπων δεδομένων: Μιγαδικοί Αριθμοί, Στοίβα και Ουρά Αναμονής (χρησιμοποιώντας κώδικα από το βιβλίο του Sedgewick). Γράψτε επίσης και ένα κύριο πρόγραμμα που επιδεικνύει τη λειτουργικότητα της ουράς προτεραιότητας χρησιμοποιώντας την για να ταξινομήσει ένα πίνακα ακεραίων όπως κάναμε στη σχετική διάλεξη.

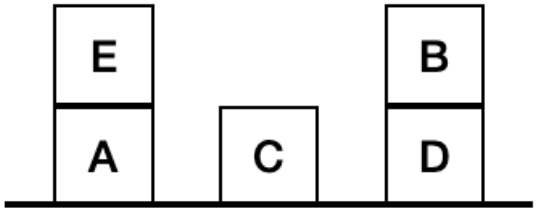
(30 μονάδες)

9. Στην άσκηση αυτή θα μελετήσουμε το **πρόβλημα των κύβων (blocks world problem)**, ένα δημοφιλές πρόβλημα από την περιοχή της Τεχνητής Νοημοσύνης. Το πρόβλημα των κύβων ορίζεται ως εξής. Έχουμε στη διάθεση μας ένα πεπερασμένο αριθμό ομοιόμορφων κύβων και ένα τραπέζι αρκετά μεγάλο για να τους χωρέσει όλους (αυτά είναι τα μόνα αντικείμενα του προβλήματος). Κάθε αντικείμενο βρίσκεται πάνω σε κάποιο άλλο αντικείμενο (τραπέζι ή κύβο). Για κάθε κύβο ισχύει το εξής: είτε είναι ελεύθερος, είτε κάποιος άλλος κύβος βρίσκεται πάνω του. Υπάρχει μια μόνο **ενέργεια** στη διάθεση ενός **ρομπότ** που λύνει το πρόβλημα: η μετακίνηση ενός μόνο ελεύθερου κύβου από ένα άλλο κύβο στο τραπέζι ή από ένα αντικείμενο (κύβο ή τραπέζι) σε ένα ελεύθερο κύβο. Η **αρχική κατάσταση** και η κατάσταση στην οποία θέλουμε να φτάσουμε (που τη λέμε **κατάσταση στόχου**) περιγράφονται δίνοντας με ακρίβεια την θέση κάθε κύβου (αν βρίσκεται στο τραπέζι ή αν βρίσκεται πάνω σε άλλο κύβο) και αν κάποις κύβος και ποιος βρίσκεται πάνω του. Το **κόστος κάθε ενέργειας** που κάνει το ρομπότ είναι 1. Μια **λύση** στο πρόβλημα είναι μια ακολουθία ενεργειών (δηλαδή μετακινήσεων κύβων) που μας επιτρέπει να φτάσουμε από μια δοσμένη αρχική κατάσταση σε μια κατάσταση στόχου. Η **βέλτιστη λύση** του προβλήματος είναι αυτή που έχει το ελάχιστο κόστος (δηλαδή, αυτή που απαιτεί τις λιγότερες μετακινήσεις).

Παράδειγμα:



Αρχική Κατάσταση



Κατάσταση Στόχου

Στο παραπάνω παράδειγμα μια βέλτιστη λύση αποτελούμενη από 4 μετακινήσεις είναι:

1. Μετακίνησε τον D στο τραπέζι
2. Μετακίνησε τον A στο τραπέζι
3. Μετακίνηση τον B πάνω στον D
4. Μετακίνησε τον E πάνω στον A

Μπορείτε να βρείτε πολλές μη βέλτιστες λύσεις όπως για παράδειγμα:

1. Μετακίνηση τον D στο τραπέζι
2. Μετακίνησε τον A στο τραπέζι
4. Μετακίνησε τον B στο τραπέζι
4. Μετακίνηση τον B πάνω στον D
5. Μετακίνησε τον E πάνω στον A

Έχετε να γράψετε μια συνάρτηση η οποία να χρησιμοποιεί επανάληψη (loop) και μια ουρά προτεραιότητας για να λύσει ένα δοσμένο πρόβλημα κύβων και να βρει την βέλτιστη λύση κάνοντας **αναζήτηση**. Η υλοποίηση της ουράς προτεραιότητας που θα χρησιμοποιήσετε θα πρέπει να είναι αυτή του ερωτήματος 8.

Η ουρά προτεραιότητας θα χρησιμοποιείται για να αποθηκεύουμε τις **καταστάσεις** στις οποίες βρίσκεται ο κόσμος των κύβων ενώ το ρομπότ αλληλεπιδρά με αυτόν και προσπαθεί να λύσει το πρόβλημα. Κάθε κατάσταση στην ουρά προτεραιότητας μας περιγράφει με ακρίβεια τη θέση όλων των κύβων και συνοδεύεται από μια **αξιολόγηση** που μας λέει πόσο καλή είναι ώστε, μέσω αυτής, να οδηγηθούμε στην κατάσταση στόχου. Το ρομπότ αξιολογεί κάθε κατάσταση k στην οποία μπορεί να βρεθεί ο κόσμος με τη χρήση μιας συνάρτησης αξιολόγησης $f(k) = g(k) + h(k)$. Η συνάρτηση $g(k)$ μας δίνει το κόστος που έχει το ρομπότ για να φτάσει στην κατάσταση k ξεκινώντας από την αρχική κατάσταση. Η $h(k)$ είναι μια **ευρετική συνάρτηση** η οποία, για το πρόβλημα μας, δίνει τον αριθμό των κύβων που δεν είναι στη σωστή θέση στην κατάσταση k. Αρχικά στην ουρά προτεραιότητας βρίσκεται η αρχική κατάσταση (με την αξιολόγηση της). Ο επαναληπτικός αλγόριθμος λειτουργεί ως εξής. Εξάγουμε το στοιχείο της ουράς με την μεγαλύτερη προτεραιότητα. Ορίζουμε να έχει μεγαλύτερη προτεραιότητα το στοιχείο (κατάσταση) k με το **μικρότερο** $f(k)$. Αν το στοιχείο αυτό αντιστοιχεί σε κατάσταση στόχου, τότε έχουμε τελειώσει και εκτυπώνουμε την ακολουθία κινήσεων που μας οδήγησε από την αρχική κατάσταση στην κατάσταση στόχου. Αν το στοιχείο αυτό δεν αντιστοιχεί σε κατάσταση στόχου, τότε εισάγουμε στην ουρά όλες τις καταστάσεις (μαζί με τις αξιολογήσεις τους) στις οποίες μπορεί να βρεθεί το ρομπότ από την παρούσα κατάσταση, εκτελώντας μία μόνο κίνηση. Μετά ο αλγόριθμος συνεχίζει με τον ίδιο τρόπο: εξάγει το στοιχείο της ουράς με τη μεγαλύτερη προτεραιότητα κλπ.

Προτείνουμε να εκτελέσετε αυτό τον αλγόριθμο στο χαρτί για να επιβεβαιώσετε ότι λύνει ένα δοσμένο πρόβλημα κύβων και βρίσκει την βέλτιστη λύση, πριν δοκιμάσετε να τον υλοποιήσετε.

Το πρόγραμμα σας θα πρέπει να δουλεύει με όσους κύβους Ν θέλει ο χρήστης. Θα παρατηρήσετε όμως ότι, όσο καλή κι αν είναι η υλοποίηση σας, ο παραπάνω αλγόριθμος θα καθυστερεί να βρει λύση από ένα (μικρό) αριθμό κύβων Ν και μετά. Ο βασικός λόγος που συμβαίνει αυτό είναι γιατί το πρόβλημα των κύβων είναι **NP-hard**[1] δηλαδή δεν γνωρίζουμε γι αυτό αλγόριθμους που τρέχουν σε πολυωνυμικό χρόνο, και πιστεύουμε ότι τέτοιοι αλγόριθμοι δεν υπάρχουν (αν και δεν το έχουμε αποδείξει μαθηματικά – η ερώτηση αυτή είναι μια σπουδαία ανοικτή ερώτηση στην Πληροφορική!). Για να δείτε αυτή τη συμπεριφορά καθαρά, θέλουμε να υπολογίσετε το **χρόνο εκτέλεσης** του αλγόριθμου σας για N=1, 2, 3, … και να παραδώσετε ένα αρχείο pdf που θα περιέχει ένα πίνακα όπως τον παρακάτω:

Αριθμός Κύβων	**Χρόνος Εκτέλεσης σε …**
1	…
2	…
3	…
…	…
…	(η εκτέλεση του προγράμματος διακόπηκε μετά από … λεπτά)

Σε ποιο Ν μπορέσατε να φτάσετε χωρίς να βαρεθείτε και να τερματίσετε το πρόγραμμα σας; 😊

(50 μονάδες μπόνους)

Αν βρείτε την άσκηση αυτή ενδιαφέρουσα, μπορείτε να διαβάσετε για **προβλήματα αναζήτησης** και ευρετικές συναρτήσεις από τις διαφάνειες του μαθήματος της Τεχνητής Νοημοσύνης

[1] Μια απόδειξη βρίσκεται στο άρθρο https://www.aaai.org/Papers/AAAI/1991/AAAI91-097.pdf . Δεν χρειάζεται να το διαβάσετε για την εργασία αυτή! Για NP-hard προβλήματα γενικά, θα μάθετε στο μάθημα Αλγόριθμοι και Πολυπλοκότητα.

(http://cgi.di.uoa.gr/~ys02/siteAI2020/lectures.html) ή από όποιο άλλο υλικό βρείτε στον παγκόσμιο ιστό, με σκοπό να υλοποιήσετε μια ευρετική συνάρτηση **καλύτερη από την *h*** που δώσαμε παραπάνω. Έτσι θα μπορέσετε να λύσετε σε μικρότερο χρόνο ένα δοσμένο πρόβλημα κύβων. Για να παρουσιάσετε τα αποτελέσματα σας, θα πρέπει να φτιάξετε ένα πίνακα όπως τον παραπάνω που να έχει διαφορετικές στήλες για κάθε μια από τις ευρετικές συναρτήσεις που θα δοκιμάσετε:

Αριθμός Κύβων	**Χρόνος Εκτέλεσης σε … με χρήση της ευρετικής *h***	**Χρόνος Εκτέλεσης σε … με χρήση της ευρετικής *h1***
1	…	…
2	…	…
3	…	…
…	…	…
…	(η εκτέλεση του προγράμματος διακόπηκε μετά από … λεπτά)	(η εκτέλεση του προγράμματος διακόπηκε μετά από … λεπτά)

(100 μονάδες bonus, παράδοση μέχρι το την ημερομηνία παράδοσης της 2^{ης} εργασίας του μαθήματος)

Καλή Επιτυχία!