K08 Δομές Δεδομένων και Τεχνικές Προγραμματισμού
Περιττοί ΑΜ

- Εαρινό Εξάμηνο 2018-2019
- Διδάσκων: Κώστας Χατζηκοκολάκης
- Εργασία 2
  - Ανακοινώθηκε στις 2 Απριλίου 2019
  - Προθεσμία παράδοσης: 1 Μαΐου 2019, 12 τα μεσάνυχτα
  - 15% του συνολικού βαθμού στο μάθημα

Πιθανές αντιγραφές

Τα προγράμματα σας θα ελεγχθούν αυτόματα χρησιμοποιώντας κατάλληλο «έξυπνο» λογισμικό για ομοιότητες. Αν υπάρχουν ομοιότητες, όλες οι σχετικές ασκήσεις ή τμήματα ασκήσεων μηδενίζονται. Δεν δίνονται λεπτομέρειες για το πως και το γιατί, απλά ένα στρογγυλό μηδέν (0). Δεν θα έχετε τη δυνατότητα να μιλήσετε με τον καθηγητή ή τους βοηθούς του μαθήματος για περιπτώσεις αντιγραφής.

Κύριο πρόγραμμα

Σε όλες τις παρακάτω ασκήσεις ζητείται η υλοποίηση συνάρτησεων που έχουν κάποια λειτουργικότητα. Θα πρέπει μαζί με τη συνάρτηση αυτή να υλοποιήσετε και ένα κύριο πρόγραμμα (συνάρτηση main) το οποίο θα επιδεικνύει τη λειτουργικότητα **όλων των συναρτήσεων** για κατάλληλα επιλεγμένες εισόδους, και θα πείθει τον βαθμολογητή ότι οι συναρτήσεις λειτουργούν σωστά σε όλες τις ενδιαφέρουσες περιπτώσεις, ώστε να σας βαθμολογήσει με τον υψηλότερο δυνατό βαθμό. Ειναι προτιμότερο η main **να ΜΗΝ δέχεται είσοδο από το χρήστη**, αλλά να χρησιμοποιεί κατάλληλες προκαθορισμένες εισόδους (ώστε ο διορθωτής να μπορεί να την εκτελέσει εύκολα). Επίσης είναι χρήσιμο η main να τυπώνει μηνύματα που να εξηγούν το τί κάνει.

Πως να παραδώσετε τις λύσεις σας

Οι λύσεις θα πρέπει να σταλούν στο e-mail `ddproj@di.uoa.gr` μέχρι την προθεσμία παράδοσης και να είναι οργανωμένες ως εξής. Θα στείλετε **ακριβώς ένα συμπιεσμένο αρχείο**. Η λύση κάθε προβλήματος θα είναι σε **χωριστό directory** το οποίο θα περιλαμβάνει τα

source files που χρειάζονται, και ένα **Makefile** το οποίο θα μπορεί να χρησιμοποιηθεί για να μεταγλωττιστούν τα αρχεία σας και να παραχθεί το αντίστοιχο executable. Θα πρέπει να υπάρχει και ένα **αρχείο pdf** το οποίο θα περιέχει **όσο documentation χρειάζεται** ώστε οι βαθμολογητές να κατανοήσουν πλήρως τις λύσεις σας και να τις βαθμολογήσουν ανάλογα. Αυτό θα πρέπει να γίνει ανεξάρτητα με το αν είναι τα προγράμματα σας καλά σχολιασμένα, πράγμα που συνιστάται. Τέλος, το αρχείο pdf θα πρέπει να ξεκινά με το όνομα σας γραμμένο με Ελληνικούς χαρακτήρες και τον αριθμό μητρώου σας.

Παρατηρήσεις

- Οι λύσεις σας για όλες τις ασκήσεις πρέπει να είναι οργανωμένες σε modules της C όπως έχουμε συζητήσει στο μάθημα. Για κάθε module απαιτούνται τουλάχιστον 2 αρχεία, `module.h, module.c` και όπου χρειάζεται και ένα `moduleTypes.h`, όπως έχουμε δει στο μάθημα.
- **Δεν χρειάζεται** να ελέγχετε παντού ότι τα ορίσματα μιας συνάρτησης είναι σύμφωνα με τις προδιαγραφές. Ενας τέτοιος έλεγχος μπορεί να είναι χρονοβόρος και άσκοπος (αν πχ η συνάρτηση καλείται μόνο από άλλες συναρτήσεις, όχι απ'ευθείας από το χρήστη, και πάντα με σωστό τρόπο). Σε τέτοιου είδους συναρτήσεις η σύμβαση συνήθως είναι ότι αν η είσοδος είναι λάθος το αποτέλεσμα είναι απρόβλεπτο.
- Συχνά πάλι είναι χρήσιμο να υπάρχουν τέτοιοι έλεγχοι, πχ όταν: τα δεδομένα έρχονται απ'ευθείας από το χρήστη, ο έλεγχος είναι εύκολος, γίνεται μια φορά μόνο στην αρχή, κλπ. Προσθέστε ελεύθερα ελέγχους όπου νομίζετε ότι είναι κατάλληλοι.
- Τα προγράμματα σας θα πρέπει να είναι όσο πιο καλά οργανωμένα γίνεται, σύμφωνα με όσα έχετε μάθει στο μάθημα «Εισαγωγή στον Προγραμματισμό».
- Χρησιμοποιήστε κατανοητά ονόματα μεταβλητών και προσθέστε σχόλια όπου χρειάζεται. Θυμηθείτε: όταν προγραμματίζουμε δεν μας ενδιαφέρει μόνο να τρέχει σωστά το πρόγραμμα, αλλά και να μπορεί να γίνει κατανοητό από κάποιον που το διαβάζει (το διορθωτή, στην προκειμένη περίπτωση)
- Τα προγράμματα σας πρέπει να «τρέχουν» στους υπολογιστές του Τμήματος με το λειτουργικό linux αφού μεταφραστούν με τον μεταγλωττιστή gcc.

Καλή Επιτυχία!

Ασκηση 1 (25 μονάδες)

Υλοποιήστε τον **αφαιρετικό τύπο δεδομένων BinaryTree** με τις ακόλουθες μεθόδους:

- BTCreate(maxelem) create an empty binary tree (maxelem is ignored)
- BTHeight(tree): return the height of the tree.
- BTSize(tree): return the number of elements in the tree
- BTGetRoot(tree): returns the root node (can be Nil)

- BTGetParent(tree, node): returns the parent of a node (prints error if node is root)
- BTGetChildLeft(tree, node): returns the left child of a node (can be Nil)
- BTGetChildRigth(tree, node): returns the right child of a node (can be Nil)
- BTIsNil(node): returns true if the node is Nil (see below), otherwise return false.
- BTGetItem(tree, node): returns the value of the given node (error if node is Nil)
- BTInsertRoot(tree, item): insert item as root of empty tree (prints error if not empty)
- BTInsertLeft(tree, node, item): insert item as left child of node (error if left child exists)
- BTInsertRight(tree, node, item): insert item as right child of node (error if exists)
- BTRemove(tree, node): removes node (error if it has children)
- BTChange(tree, node, item): sets item as the new value of the given node
- BTPreOrder(tree, visit), BTInOrder(tree, visit), BTPostOrder(tree, visit), BTLevelOrder(tree, visit): traverse the tree and visit its nodes in the respective order.
- BTDestroy(tree): destroys the tree by freeing all its nodes.

Η υλοποίηση πρέπει να χρησιμοποιεί **δείκτες** (αν θέλετε μπορείτε να προσθέσετε και δείκτες από τα παιδιά προς τον πατέρα). Ολόκληρο το δέντρο θα πρέπει να αντιπροσωπεύεται από τον τύπο **BTTree**, ενώ οι κόμβοι από τον τύπο **BTNode** (ορίστε όλους τους τύπους κατάλληλα, με βάση την αρχή του information hiding). Ο τύπος του item θα πρέπει να είναι ένα γενικό **BTItem**.

Αναφορικά με το node/BTNode: ο χρήστης του module δεν γνωρίζει πώς το δέντρο έχει υλοποιηθεί, και ο μόνος τρόπος να προσπελάσει τους κόμβους είναι μέσω των αντίστοιχων συναρτήσεων του module (BTGetRoot, BTGetChildLeft, BTGetChildRigth). Οι συναρτήσεις αυτές επιστρέφουν τύπο BTNode, ο οποίος αντιπροσωπεύει **ένα κόμβο του δέντρου**. Αν πχ ο χρήστης θέλει να διαβάσει την τιμή του δεξιού παιδιού της ρίζας ενός δέντρου ακεραίων:

```
BTNode root = BTGetRoot(tree);
BTNode child = BTGetChildRight(tree, root);
int value = BTGetItem(tree, child);
```

Ενας κόμβος μπορεί να είναι "Nil": αν πχ το δέντρο είναι άδειο η BTGetRoot θα επιστρέφει Nil, ή αν το node δεν έχει αριστερό παιδί η GetChildLeft θα επιστρέφει Nil. Η BTIsNil(node) ελέγχει αν ο κόμβος είναι Nil. Το πώς θα αναπαριστάτε τους κόμβους (κανονικούς ή Nil) είναι δικό σας θέμα. Πχ μια λογική λύση θα ήταν το BTNode να είναι δείκτης, ενώ οι "Nil" κόμβοι να είναι NULL (ή δείκτες σε dummy κόμβο).

Στις BT*Order συναρτήσεις, η παράμετρος visit είναι μια συνάρτηση που πρέπει να καλείται για κάθε κόμβο, παίρνοντας σαν παράμετρο είτε μόνο τον κόμβο (visit(node)) είτε το δέντρο και τον κόμβο (visit(tree, node)), ό,τι από τα δύο προτιμάτε.

Ασκηση 2 (25 μονάδες)

Δώστε μια εναλακτική υλοποίηση του ADT BinaryTree της Άσκησης 1 χρησιμοποιώντας **array**. Το δέντρο δεν είναι απαραίτητα πλήρες, οπότε ο πίνακας πιθανόν να περιέχει κενά.

Οι συναρτήσεις θα πρέπει να έχουν ακριβώς τους ίδιους ορισμούς με την άσκηση 1, ώστε ο χρήστης να μπορεί να αλλάξει υλοποίηση χωρίς αλλαγές στον κώδικα.

Οι τύποι βέβαια μπορεί κάλλιστα να είναι διαφορετικοί απ'ότι στην υλοποίηση με δείκτες. Πχ μπορεί να επιλέξετε (αν θέλετε βέβαια) το BTNode να μην είναι δείκτης, αλλά ένα index στον πίνακα (και οι Nil κόμβοι να αναπαριστώνται με -1).

Το maxelem είναι ο μέγιστος αριθμός στοιχείων του δέντρου. Η main πρέπει να είναι ολόιδια με αυτή της άσκησης 1 (ώστε να δείξουμε ότι η αλλαγή υλοποίησης δεν απαιτεί αλλαγή στον κώδικα.

Ασκηση 3 (15 μονάδες)

Τροποποιήστε την υλοποίηση του BinaryTree (header files, Makefile, ...) ώστε ο χρήστης χωρίς καμία αλλαγή στον κώδικα, να μπορεί να επιλέξει την υλοποίηση BinaryTree που επιθυμεί χρησιμοποιώντας **conditional compilation**:

```
make BT_IMPL=pointers
# or
make BT_IMPL=array
```

Ασκηση 4 (25 μονάδες)

Υλοποιήστε τον **αφαιρετικό τύπο δεδομένων CompleteBT**, ο οποίος μοντελοποιεί ένα **πλήρες** δέντρο. Υποστηρίζει **ακριβώς τις ίδιες μεθόδους με το Binary Tree** με τις εξής διαφορές:

- Το "BT" prefix αλλάζει σε "CBT": πχ CBTCreate (ώστε να μην υπάρχει conflict ανάμεσα στις 2 μεθόδους). Το ίδιο ισχύει και για τους τύπους: CBTTree, CBTNode.
- Επειδή το δέντρο είναι πλήρες, οι προσθήκες και διαγραφές γίνονται μόνο στο "τέλος" του δέντρου. Δηλαδή οι BTDelete, BTInsert* αφαιρούνται, και προστίθενται οι ακόλουθες:
  - CBTGetLast(tree): returns the "last" node (rightmost leaf of the last level)
  - CBTInsertLast(tree, item): insert item as a new "last" node
  - CBTRemoveLast(tree): removes the last node (error if the tree is empty)

Το CompleteBT δεν θα πρέπει να υλοποιεί από την αρχή ένα δέντρο, αλλά **θα πρέπει να χρησιμοποιεί τον ADT BinaryTree** από την Ασκηση 3. Η επιλογή της υλοποίησης θα πρέπει να γίνεται από το make μέσω conditional compilation (`make BT_IMPL=...`).

Ασκηση 5 (30 μονάδες)

Υλοποιήστε τον **αφαιρετικό τύπο δεδομένων BinaryHeap**: έναν **max-σωρό** που αποθηκεύει items με αυθαίρετο τύπο (BHItem), καθένα από τα οποία έχει μια προτεραιότητα (int). Λειτουργίες:

- BHCreate(maxelem): creates an empty heap
- BHIsEmpty(heap): return true if the heap is empty, otherwise return false.
- BHGetMaxPriority(heap): returns the highest priority of all heap elements
- BHGetMaxItem(heap): returns the item with the highest priority in the heap
- BHRemove(heap): removes the item with the highest priority in the heap
- BHInsert(heap, priority, item): inserts an item in the heap with the given priorty
- BHHeapify(n, priorities, items): given arrays (size n) of items and their priorities, creates a new heap containing exactly these items (by first inserting the elements in a complete tree, and then running the heapify algorithm)
- BHDestroy(heap): destroys the heap by freeing all its nodes.

Το BinaryHeap δεν θα πρέπει να υλοποιεί από την αρχή έναν σωρό, αλλά **θα πρέπει να χρησιμοποιεί τον ADT CompleteBT** από την Ασκηση 4. Εφόσον το CompleteBT χρησιμοποιεί εσωτερικά το BinaryTree, και πάλι η επιλογή της υλοποίησης θα μπορεί να γίνει από το make μέσω conditional compilation (`make BT_IMPL=...`).

Ας σημειωθεί ότι ο αλγόριθμος heapify στις διαφάνειες επισκέπτεται τους κόμβους σε reverse level-order σειρά (γιατί αυτό είναι το πιο εύκολο όταν ο σωρός υλοποιείται με πίνακα). Ο αλγόριθμος όμως δουλεύει εξ'ίσου καλά και σε post-order σειρά (το μόνο που χρειάζεται είναι τα παιδιά να επισκεφτούν πριν από τον κόμβο), το οποίο είναι πιο εύκολο στη δική μας γενική υλοποίηση (γιατί το post-order το έχουμε έτοιμο).

Ασκηση 6 (30 μονάδες)

Δημιουργήστε τον ADT BinarySearchTree: ένα δυαδικό δέντρο αναζήτησης στον οποίο αποθηκεύονται items (με αυθαίρετο τύπο BSTItem) καθένα από τα οποία έχει ένα key (τύπου BSTKeyType) με βάση το όποιο τοποθετείται στο δέντρο. Σχεδιάστε τη διεπαφή όπως κρίνετε σκόπιμο (είναι μέρος της άσκησης), συμπεριλαμβάνοντας τις "βασικές" λειτουργίες διαχείρισης, προσθήκης και διαγραφής (**χωρίς rotations**). Ολες οι συναρτήσεις πρέπει να έχουν prefix "BST".

Εφόσον το key είναι αυθαίρετου τύπου, ο χρήστης του module θα χρειαστεί να παρέχει τη δική του μέθοδο compare (όπως στην Ασκηση 2 της Εργασίας 1), κατάλληλα ορισμένη. Η compare μπορεί είτε να δίνεται ως όρισμα σε κάθε συνάρτηση που τη χρειάζεται, είτε να δίνεται μόνο στην Create και να αποθηκεύεται μαζί με το δέντρο.

Η υλοποίηση πρέπει να χρησιμοποιεί τον ADT BinaryTree (και όχι να υλοποιεί το δέντρο από την αρχή).

Ασκηση 7 (30 μονάδες)

Δημιουργήστε τον ADT PriorityQueue: μια ουρά προτεραιότητας στην οποία αποθηκεύονται items (με αυθαίρετο τύπο PQItem), καθένα από τα οποία έχει μια προτεραιότητα (int). Θα πρέπει να παρέχετε **2 διαφορετικές υλοποιήσεις** του ίδιου ADT:

1. Χρησιμοποιώντας τον ADP BinaryHeap (άσκηση 5)

2. Χρησιμοποιώντας τον ADP BinarySearchTree (άσκηση 6)

Οι υλοποιήσεις είναι πανεύκολες γιατί χρησιμοποιούν απ'ευθείας άλλους ADTs!

Σχεδιάστε τη διεπαφή όπως κρίνετε σκόπιμο (είναι μέρος της άσκησης), με prefix "PQ". Η επιλογή της υλοποίησης (τόσο για το priority queue, αλλά και για το binary tree που ίσως χρησιμοποιείται εσωτερικά) θα πρέπει να γίνεται με conditional compilation:

```
make PQ_IMPL=[bh|bst] BT_IMPL=[pointers|array]
```

Bonus (30 μονάδες): χρησιμοποιήστε το PriorityQueue στην άσκηση 7 της εργασίας 1, και συγκρίνετε τους χρόνους εκτέλεσης για τις διαφορετικές επιλογές της υλοποίησης. Παρουσιάστε τα αποτελέσματά σας (με όποιον "πειστικό" τρόπο νομίζετε) στο pdf.

Ασκηση 8 (30 μονάδες)

1. Στην υλοποίηση του BinaryHeap της άσκησης 5 προσθέστε μια συνάρτηση `BHNaiveHeapify(n, priorities, items)` η οποία θα δημιουργεί ένα νέο σωρό από ένα array από items, χρησιμοποιώντας όμως την απλή μέθοδο: προσθήκη στοιχείων ένα-ένα σε κενό σωρό. Τα priorities και items είναι arrays μεγέθους n.

   Στη main, δείξτε πειραματικά ότι επιλέγοντας κατάλληλα το array, ο αριθμός βημάτων της BHNaiveHeapify αυξάνεται γρηγορότερα από αυτόν της BHHeapify.

2. Στο μάθημα είδαμε ότι ενώ η BHNaiveHeapify έχει πολυπλοκότητα $O(n \log n)$ στην **χειρότερη** περίπτωση, στη **μέση** περίπτωση (για τυχάια δηλαδή επιλεγμένα δεδομένα) η πολυπλοκότητα είναι γραμμική! Τι μπορούμε να κάνουμε όμως αν τα δεδομένα **δεν είναι επιλεγμένα τυχαία**;

   Μια λύση είναι να **"δημιουργήσουμε εμείς τυχαιότητα"**, προσθέτωντας τα στοιχεία με **τυχαία σειρά**. Δημιουργήστε μια συνάρτηση `BHNaiveHeapifyRnd(n, priorities, items)` που να υλοποιεί αυτή τη μέθοδο.

   Στη main συγκρίνετε πειραματικά τη συμπεριφορά των τριών BHHeapify μεθόδων, και παρουσιάστε τα αποτελέσματα στο pdf (με όποιο τρόπο νομίζετε, αρκεί τα συμπεράσματα να είναι δικαιολογημένα) χρησιμοποιώντας τον "προβληματικό" τύπο εισόδου για τον οποίο η BHNaiveHeapify αργεί.

Για τις ανάγκες της main μπορείτε να προσθέσετε μια συνάρτηση BHGetLastHeapifySteps() που να επιστρέφει τον αριθμό βημάτων που χρειάστηκε η τελευταία κλήση της BHHeapify*.

Ασκηση 9 (50 μονάδες)

- Δημιουργήστε τον ADT Tree: ένα δέντρο στο οποίο κάθε κόμβος έχει ένα item (με αυθαίρετο τύπο TreeItem), και έναν **οποιοδήποτε αριθμό παιδιών.** Χρησιμοποιήστε έναν ADT LinkedList (με οποιαδήποτε υλοποίηση, πχ αυτή που φτιάξατε στην Εργασία 1) για την αποθήκευση των παιδιών. Σχεδιάστε τη διεπαφή όπως κρίνετε κατάλληλο, με όσες λειτουργίες χρειάζεστε για το 2ο μέρος της άσκησης.

- Χρησιμοποιήστε τον ADT Tree για την υλοποίηση ενός **Pairing Heap**. Το pairing heap είναι ένα είδος σωρού στον οποίο κάθε κόμβος μπορεί να έχει έναν οποιοδήποτε αριθμό υπο-σωρών, και όλοι οι κόμβοι έχουν την κλασσική max-heap ιδιότητα. Οι λειτουργίες που παρέχει είναι οι εξής:
  - PHGetMaxItem(heap): (εύκολο) επιστρέφουμε τη ρίζα
  - PHMerge(heap1, heap2): (εύκολο) για να ενώσουμε 2 σωρούς σε έναν μεγάλο, παίρνουμε αυτόν με τη μεγαλύτερη ρίζα, και βάζουμε τον άλλον ως υπο-σωρό. (αν ένας από τους δύο είναι empty το πρόβλημα είναι trivial)
  - PHInsert(heap, item): (εύκολο) δημιουργούμε ένα νέο σωρό που περιέχει μόνο το item, και τον κάνουμε merge με το heap.
  - PHDeleteMax(heap): (η μόνη non-trivial) Αρχικά αφαιρούμε τη ρίζα, οπότε μένουμε με μια λίστα υπο-σωρών, τους οποίους πρέπει να κάνουμε όλους merge. Η λειτουργία αυτή (`merge_many`) μπορεί να οριστεί αναδρομικά ως εξής (σε ξευδοκώδικα):

    ```
    merge_many(heaps)
        heap1 = remove(heaps)
        heap2 = remove(heaps)
        return merge( merge(heap1, heap2), merge-many(heaps) )
    ```

    (χρειάζεται προφανώς να προστεθεί και base-case, όταν το heaps περιέχει 0 ή 1 σωρό).

    Bonus (20 μονάδες): Δημιουργήστε έναν ADT Heap, με 2 υλοποιήσεις (BinaryHeap και PairingHeap), επιλογή μέσω conditional compilation, και χρησιμοποιήστε τον στην άσκηση 7.