Κ08 Δομές Δεδομένων και Τεχνικές Προγραμματισμού

Διδάσκων: Μανόλης Κουμπαράκης

Εαρινό Εξάμηνο 2024-2025

Εργασία 2

Ανακοινώθηκε την 30 Μαρτίου 2025

Προθεσμία: 27 Απριλίου 2025 στις 23:59

1.6 μονάδες στις 10 του βαθμού στο μάθημα (Άριστα=350 μονάδες, Υπάρχουν 50 μονάδες bonus)

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 6-12) και θα χρειαστείτε για την εργασία αυτή βρίσκεται στο παρακάτω private repository του classroom του μαθήματος: https://github.com/artioi-k08/2025-ergasia2. Για να συνδεθείτε στο repository αυτό, θα πρέπει να χρησιμοποιήσετε το λινκ https://classroom.github.com/a/7Lm4Nvhe.

Όταν συνδεθείτε θα μπορείτε να δείτε το προσωπικό σας repository https://github.com/artioi-k08/2025-ergasia2-<github-your-username> στο οποίο θα δουλέψετε για την Εργασία 2.

Μετά τη σύνδεση σας, στο προσωπικό σας repository, θα βρείτε ένα φάκελο solutions-ergasia2 με υπο-φακέλους question1, question2, …, question6 στους οποίους θα πρέπει να γράψετε τον κώδικα ή τις απαντήσεις σας για τις 6 ασκήσεις αυτής της εργασίας που θα βρείτε παρακάτω. Παρακαλώ τηρείστε ευλαβικά αυτήν την οργάνωση αλλιώς θα χάσετε 20% του συνολικού βαθμού κατά την

βαθμολόγηση. Για τα θεωρητικά ερωτήματα, οι απαντήσεις πρέπει να είναι σε ένα αρχείο τύπου pdf.

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. Επίσης, τα προγράμματα σας δεν θα πρέπει να έχουν memory errors ή leaks (θα ελεγχθούν με τη χρήση `valgrind`).

1. Θεωρήστε τον C κώδικα που βρίσκεται στο αρχείο `code.c` του φακέλου question1. Εξηγήστε με λεπτομέρειες τι κάνει ο κώδικας αυτός. Ποια είναι η χρονική υπολογιστική πολυπλοκότητα χειρότερης περίπτωσης της συνάρτησης `bubble`; Ποια είναι η αντίστοιχη πολυπλοκότητα της `main`; Υποθέστε ότι η κλήση `rand()` εκτελείται σε σταθερό χρόνο. Να εξηγήσετε με λεπτομέρεια τις απαντήσεις σας.

   (10 μονάδες)

2. Υποθέστε ότι έχουμε 10 αλγόριθμους Α, Β, Γ, Δ, Ε, Ζ, Η, Θ, Ι και Κ με τις παρακάτω υπολογιστικές πολυπλοκότητες χρόνου (παραλείπουμε το $O(\ )$).

   Α. $1000n$ Β. $500n + logn$ Γ. $2^{3000nlogn}$ Δ. $2^{300logn}$ Ε. n^8

   Ζ. $6n^8 + n$ Η. $nlogn2^{n+5}$ Θ. 2^{2^n} Ι. $2^{2^{logn}}$ Κ. $2^{2^{n+logn}}$

   Να ταξινομήσετε τους αλγόριθμους από τον καλύτερο στον χειρότερο με βάση την υπολογιστική πολυπλοκότητα τους. Να δώσετε λεπτομερώς όσους μαθηματικούς υπολογισμούς χρειάζονται για να τεκμηριώσετε την απάντηση σας.

   (10 μονάδες)

3. Δώστε μια ακολουθία 10 κλειδιών (χρησιμοποιήστε τα γράμματα Α έως Κ) η οποία, όταν τα κλειδιά εισάγονται με τη σειρά αυτή σε ένα αρχικά άδειο δυαδικό δένδρο αναζήτησης με τη μέθοδο της εισαγωγής στη ρίζα (σελίδες 81-90 των διαφανειών της Ενότητας 9), απαιτείται μέγιστος αριθμός συγκρίσεων για να χτιστεί το δένδρο. Να δώσετε αυτό τον αριθμό συγκρίσεων.

   (10 μονάδες)

4. Να υλοποιήσετε τον αφαιρετικό τύπο δεδομένων Ουρά Προτεραιότητας χρησιμοποιώντας σωρούς μεγίστων όπως δείξαμε στην Ενότητα 8 των διαφανειών. Με βάση τον κώδικα των διαλέξεων, να υλοποιήσετε την παρακάτω διεπαφή που σας επιτρέπει να έχετε παραπάνω από μια ουρές προτεραιότητας:

```
typedef struct priority_queue{
    int Count;
    Item *ItemArray;
}PriorityQueue;
typedef PriorityQueue *PQPointer;
PQPointer QUEUEinit(int maxN);
int QUEUEempty(PQPointer);
void QUEUEput(PQPointer, Item);
Item QUEUEget(PQPointer);
```

Ο τύπος `Item` μπορεί να είναι `int`. Να γράψετε και ένα κύριο πρόγραμμα το οποίο επιδεικνύει τη συμπεριφορά της ουράς προτεραιότητας. Μαζί με το κύριο πρόγραμμα, καλείστε να υλοποιήσετε unit tests για τις συναρτήσεις της διεπαφής. Μπορείτε να χρησιμοποιήσετε κώδικα από την Ενότητα 8 των διαφανειών.

(50 μονάδες)

5. Να υλοποιήσετε τον αφαιρετικό τύπο δεδομένων Δένδρο (2,4) που παρουσιάσαμε στην Ενότητα 11 των διαφανειών. Να χρησιμοποιήσετε την παρακάτω διεπαφή:

```
typedef struct t24 Tree24;
struct t24{
  int Count;
  Tree24 *parent;
  Item items[3];
  Tree24 *children[4];
  int N[4];
```

```
};
void init();
int count();
void insert(Item);
Item search(Key);
void delete(Item);
Item select(int);
void sort(void (*visit)(Item));
```

Να γράψετε και ένα κύριο πρόγραμμα το οποίο επιδεικνύει τη συμπεριφορά των παραπάνω συναρτήσεων. **ΔΕΝ** απαιτείται η υλοποίηση unit tests σε αυτή την άσκηση.

(120 μονάδες)

6. Στην άσκηση αυτή θα υλοποιήσουμε την δομή δεδομένων **kd-tree** που είναι χρήσιμη για την αποθήκευση και προσπέλαση πολυδιάστατων σημειακών δεδομένων. Η δομή αυτή περιγράφεται στο κεφάλαιο 5.2 του βιβλίου

   Mark de Berg, Otfried Cheong, Marc van Kreveld and Mark Overmars Computational Geometry: Algorithms and Applications. Springer. Available at https://link.springer.com/book/10.1007/978-3-540-77974-2 (μπορείτε να το κατεβάσετε με το λογαριασμό σας στο Τμήμα).

Έχετε να κάνετε τα εξής:

- Να κατανοήσετε τη λειτουργία της δομής kd-tree και των αλγορίθμων `BuilKdTree` και `SearchKdTree`. Δεν χρειάζεται να μελετήσετε τα θεωρητικά αποτελέσματα του παραπάνω κεφαλαίου π.χ., τις πολυπλοκότητες των αλγορίθμων κλπ.
- Να υλοποιήσετε τον αφαιρετικό τύπο δεδομένων `KdTree` βασισμένοι στην παρακάτω διεπαφή:

```
typedef enum node_type{
```

```
   VERTICAL_LINE,HORIZONTAL_LINE,LEAF_NODE
}NodeType;

typedef struct kdnode {
 // The point stored in this node
 // only for leaf nodes
 Point *point;
 // Value for line - NULL on leaf nodes
 int *line;
// VERTICAL_LINE if Vertical Line (x-axis)
//HORIZONTAL_LINE if Horizontal Line(y-axis)
//LEAF_NODE if Leaf Node
 NodeType type;
 struct kdnode *left;  // Left subtree
 struct kdnode *right; // Right subtree
} KDNode;


// Function to create a KDNode
KDNode* kdnode_init(Point *p, int *line,NodeType type);
// Function to build a kd tree
// Point **points : An array of points
// int n: The size of array points
```

```
// int depth: Depth of the current level. Should
ALWAYS be 0 when calling this function

KDNode* buildKDTree(Point **points, int n, int depth);

// Function search for points inside a given range.

// You are free to use any code for the List

// KDNode *root : The tree's root

// Range *range: The query range

// Range *region: the region of the current node i.e

// region(v). When calling this function initialize a
// Range object with xmin = INT_MIN, xmax = INT_MAX,

// ymin = INT_MIN, ymax = INT_MAX

// List *l: List to store the results

List *searchKDTree(KDNode *root, Range *range, Range
*region, List *l);

// Function to destroy a KDTree

void destroyKDTree(KDNode *root);
```

- Μαζί με την υλοποίηση της παραπάνω διεπαφής προτείνεται να υλοποιήσετε και τις παρακάτω διεπαφές για την αναπαράσταση των Point και Range. Για την ολοκλήρωση αυτής της άσκησης επιτρέπεται να χρησιμοποιήσετε την built-in συνάρτηση qsort() της c :
- Για την υλοποίηση του Point:

```
typedef struct point{
   double x;
   double y;
```

```
}Point;


//Point functions - Helper functions you will find
useful on your implementation

Point *point_init(double x, double y);

// Function type declaration - no need to implement
// anything for this - simply include it in your
code

typedef int (*PointComparator)(const void *a,const
void *b);

//Comparator functions to be used with qsort()

//Function to compare a point on the x/y-coordinate

int point_compare_x(const void *a,const  void *b);

int point_compare_y(const void *a,const  void *b);
```

- Για την υλοποίηση του Range:

```
// Using only four variables we can easily

// represent a square of the form:

// [(xmin,ymin),(xmax,ymin),(xmin,ymax),(xmax,ymax)

typedef struct range{

   double xmin, xmax;
```

```
   double ymin, ymax;
} Range;


// Function to initialize a new Range
Range *range_init(double xmin,double xmax,double
ymin,double ymax);
// Function to check if a point p is
// inside a square R
int point_in_range(Point *p, Range *R);
// Function to check if two squares intersect
int range_intersect(Range *r1, Range *r2);


// Four symmetrical functions. Very useful for
// calculating the region of a child of a node
// i.e region(lc(v)) or region(rc(v))
Range *intersect_square_on_y_up(Range *square,
double x);
Range *intersect_square_on_y_down(Range *square,
double x);
Range *intersect_square_on_x_left(Range *square,
double y);
Range *intersect_square_on_x_right(Range *square,
double y);
```

```
// Function to check if square 'outer' completely
// contains square 'inner'
int range_contains(Range *outer, Range *inner);
```

ΔΕΝ απαιτείται η υλοποίηση unit tests σε αυτή την άσκηση.

(100+100=200 μονάδες)

Καλή Επιτυχία!