Carnegie Mellon University
Department of Computer Science
15-826 Multimedia Databases and Data Mining
C. Faloutsos, Spring 2017

Homework 2 (by Yuang Liu) - Solutions

Due: **hard copy, in class, at 3:00pm, on 2/8/2017**
Due: **tarball, on Blackboard, at 3:00pm, on 2/8/2017**

VERY IMPORTANT:

- For each question, we expect *both* a **hard copy**, and a **tar file** with your code - see details next, on how to package your code.
- Deposit **hard copy** of your answers, in class.
  1. **Separate** your answers, on different page(s) for each question
  2. **Type** the full info on **each** page: your **name**, **Andrew ID**, **course#**, **Homework#**, **Question#** on each of the pages.

Reminders:

- *Plagiarism*: Homework is to be completed *individually*.
- *Typeset* your answers. Illegible handwriting may get zero points.
- *Late homeworks*: please email it
  - to all TAs
  - with the subject line exactly `15-826 Homework Submission (HW 2)`
  - and the count of slip-days you are using.

For your information:

- Graded out of **100** points; **3** questions total
- Rough time estimate: *18-24 hours ($\approx$ 6-8 hours per question)*

Revision : 2017/02/14 15:30

Question	Points	Score
KD-Trees	30	
Z-ordering and Hilbert Curve	40	
R-Tree	30	
Total:	100	

Code packaging info:
Submit your code to blackboard, in a single `tar` file, called ***andrewId***`-hw2.tar.gz` (where *andrewId* is your andrew id.) For your convenience, we provide a *tar-file package*, at `http://www.cs.cmu.edu/~christos/courses/826.S17/HOMEWORKS/HW2/hw2.tar.gz`. We will refer to it as the *tar-file package* from now on. It has 3 directories `/Q1, /Q2, /Q3`, including the k-d-tree and R-tree source code, and place-holder code.

- `tar xvf hw2.tar.gz`; `cd Q1; make` # *to work on kdtrees*
- Replace the placeholder code with your solutions, `tar` everything into ***andrewId***`-hw2.tar.gz` and submit to blackboard. We expect to do `tar xvfz; make` and see your answers.

Hints:
We strongly recommend that you explore the `makefiles` we have created, for your convenience. For example, from the top directory:

- `make hw2` will run the code for all 3 questions
- `make package` will create the tar file, for submission
- `make spotless` will clean up all the derived files (*.o, etc)

Make sure that you **exclude** redundant/derived files, in your tar file. Also, it is *your* responsibility to make sure that all necessary files are included in your tarball.

Homework 2 continues…

Question 1: KD-Trees .... [30 points]

On separate page, with '[course-id] [hw#] [question#] [andrew-id] [your-name]'

Grading info: graded by Tanay Varma

Problem Description: Consider the k-d-tree package written in C in the directory `./Q1` of the given *tar-file package*. Some functions, such as nn query and range query, have already been implemented. You can explore them by using `make demo` or command lines (`./main`).

- Your task is to add a new function called *count*, and indicated by `c`. Like the range query, it should read a rectangle, and return the count of points inside it.

Input Format: As shown in the example below, the input format is the same as the range query. For your convenience, we have already implemented the loading of input data in `main.c`. However, you may not assume that all the query inputs are valid. For example, a lower bound might be larger than the corresponding upper bound, in which case return '0'.

```
$ ./main -d 2
num. dimensions = 2
kdtree> c
counting ...
0-th attr. low value= 0
1-th attr. low value= 0
0-th attr. high value= 0
1-th attr. high value= 0
   searching - low values: 0 0
   searching - high values: 0 0
result:0
```

In the example above, the user asks for a count query (`c`), on an empty, 2-dimensional k-dtree, for the obvious rectangle - the answer in the last line, indicates that there are 0 points.

(a) [**18 points**] Modify the k-d-tree code to support the *count* queries. The output format is as the example above: a single line with the string "result:", the actual count, and a new-line at the end, i.e. `"result:%d\n"`. Remember to remove all other debugging messages (if any), before you submit the code.

IMPORTANT: Test your code, for corner cases (empty tree, illegal query input, duplicate points, etc). We will also test your code against other 'secret' data which will disclose after the due date.

Question 1 continues. . .

Solution: Please see `main.c`, `kdtree.h`, and `kdtree.c` for modified codes.

Grading info: -6 for every wrong output on hidden data

(b) [**12 points**] A hard copy of the output of your code, on the three included input scripts (`test1.txt`, `test2.txt`, and `test3.txt`; they are in the `./Q1` directory). Ignore irrelevant lines and keep only the result of count queries, to save paper (eg., `grep result`)

Solution: `test1.txt`: 81, `test2.txt`: 11, `test3.txt`: 301.

Grading info: -4 for every wrong output

What to turn in:

- **Code:** Submit your code to blackboard, in the `./Q1` directory of your `tar.gz` file.
- **Answers:** Submit hard copy for
  1. your code (only relevant parts),
  2. and the output of your code on three input files.

Homework 2 continues. . .

Question 2: Z-ordering and Hilbert Curve . . . . [40 points]

On separate page, with '[course-id] [hw#] [question#] [andrew-id] [your-name]'

Grading info: graded by Sanjay Chandrasekaran

Problem Description: The provided *tar-file package* contains complete implementations of the Z-ordering and the Hilbert curve. Next, you are supposed to fix some bugs in the Z-ordering code; plot some Z- and Hilbert-curves, and see why the Hilbert-curve is better.

For the latter, we need to define $D()$, the *pair-distance-sum-2* of a space-filling curve, and the Manhattan distance between two pixels.

Definitions: For every pixel p_i, consider its next two pixels p_{i+1} and p_{i+2} on the curve of interest. For example, for the 'snake' curve of Figure 1, for pixel p_3 (labeled as '3', with coordinates (3,3)), the next two pixels are obviously '4' and '5' (with coordinates (3,2), and (2,2) respectively - pixel p_{15} has coordinates (0,0)).

The Manhattan (= city-block) distance $d()$ of two pixels $p_i = (x_i, y_i)$, $p_j = (x_j, y_j)$, is

$$d(p_i, p_j) = |x_i - x_j| \ + \ |y_i - y_j|$$

The *pair-distance-sum-2* D of a curve with n pixels, is the sum of the Manhattan distances of all pixels $p_0, p_1, \ldots$, from their next 2 neighbors, that is

$$D = \sum_{i=0}^{n-3} \left(d(p_i, p_{i+1}) + d(p_i, p_{i+2}) \right)$$

Notice that the summation ignores the last, and second-to-last, pixels.

(a) [**10 points**] Fix the bug(s) in the provided code for Z-ordering in `zorder.c`. (We injected one or more bugs, on purpose). The function `zorder()` is supposed to return the Z-order value for the given point. You may use the inverse version of z-order, `izorder()` function, for reference.

Solution:

```
mask = 1 << j;
// Check whether the value in the position is 1
if (coor[i] & mask)
    // Do bit shuffling
    value |= 1 << ((j + 1) * dim - i - 1);
```

Grading info: -3 for unnecessary changes

Question 2 continues. . .

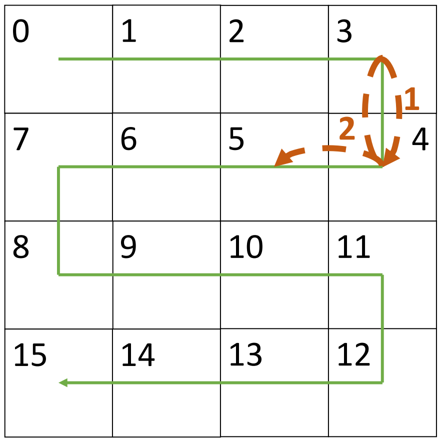
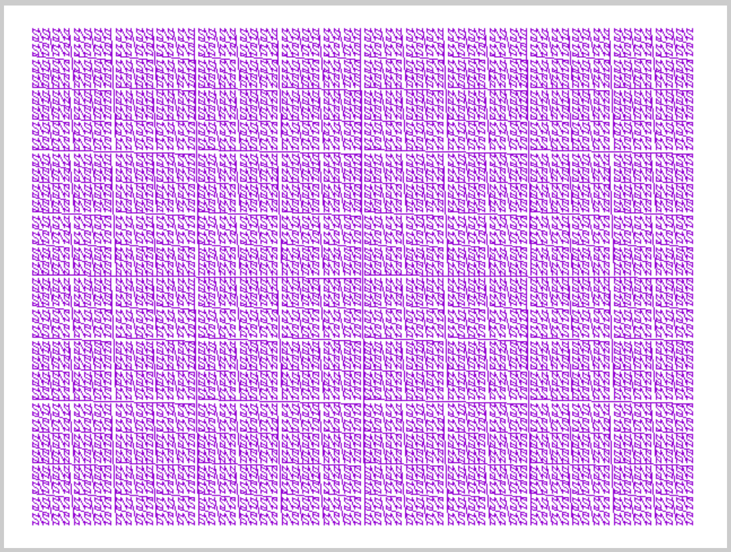


Figure 1: An example of pair-distance-sum-2 (on a snake curve): Manhattan distance between pixel '3' and pixel '4' is 1, and that between pixel '3' and pixel '5' is 2.

(b) [**5 points**] Use `zplot.c` to plot a 2-d Z-curve of 7-order (128 ∗ 128 grid); The output should be produced in `zcurve.png`. We provide some template code, the `makefile`, and some helper functions. We recommend `gnuplot` for plotting.



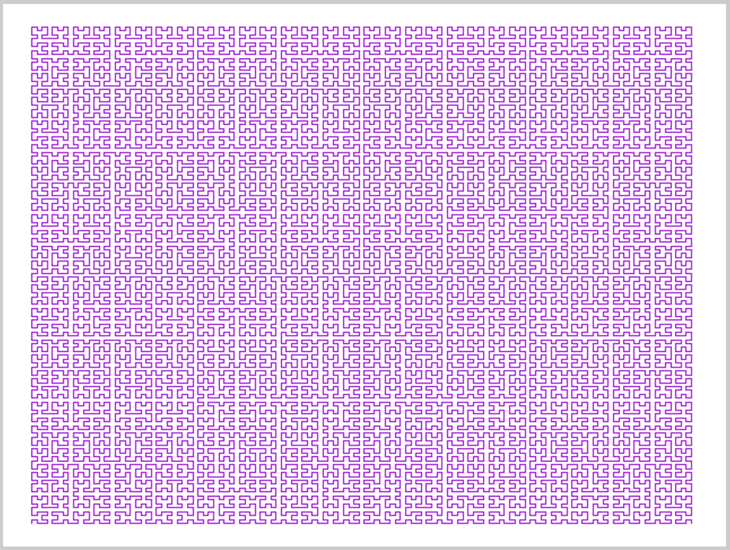
Solution:

Grading info: -1 for x and y interchanged

(c) [**5 points**] Repeat, for the Hilbert curve: Modify `hplot.c` to plot a 2-d Hilbert curve of 7-order (128 ∗ 128 grid). The output should be produced in `hcurve.png`.

Question 2 continues. . .

Solution:



Grading info: -1 for x and y interchanged

(d) [**10 points**] Calculate the *pair-distance-sum-2* (defined above) of a 2-d Z-curve, of order k=2; also, of order k=4; and of k=6. *Hint:* Modify the placeholder code in `zdist.c` to finish the task.

Solution: 2-order: 49, 4-order: 1097, 6-order: 18729.

Grading info: -5 for every wrong answer

(e) [**10 points**] Repeat, for the Hilbet curve: Calculate the *pair-distance-sum-2* of a 2-d Hilbert-curve of order k=2; also, of order k=4; and of k=6. Modify the placeholder code in `hdist.c` to finish the task.

Solution: 2-order: 42, 4-order: 762, 6-order: 12282.

Grading info: -5 for every wrong answer

What to turn in:

- **Code:** Submit your code to blackboard, in the `./Q2` directory of your `tar.gz` file. Please do not change the files that are not mentioned above.
- **Answers:** Hard copy of
  1. your code in `zorder.c`,
  2. your code in `zdist.c`,
  3. your code in `hdist.c`,
  4. your plot for (b),
  5. your plot for (c),
  6. your output for (d),
  7. and your output for (e).

Homework 2 continues…

Question 3: R-Tree . . . . [30 points]

On separate page, with '[course-id] [hw#] [question#] [andrew-id] [your-name]'

Grading info: graded by Joey Fernau

Problem Description: In this question, you will be asked to add a new functionality to the provided R-Tree package, called *DRTREE*[1]. The package is in the `./Q3` directory of the provided *tar-file package*. The R-tree package supports '`s`' for range search, 'n' for knn search and so on; use the `-h` flag, to list all the options.

- **Your task**: implement the command `c`, for Containment query, using the input dataset `test.txt` (which is invoked by default with `make hw2`).

The 'Containment query' should return all the objects that fully contain the query object. For example, in Fig. 2, the containment query for 'Q' will return only 'A' and 'C'. Your code should work for *any* dimensionality n.

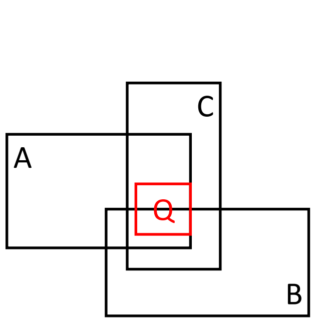


Figure 2: An example of a "containment query". For query 'Q', only 'A' and 'C' qualify.

Input Format: The *containment* query is indicated by '`c`' in the command line. Then, your system should ask the user for the lower and upper bounds of all the dimensions $(l_1, h_1, l_2, h_2, \cdots, l_n, h_n)$, where n is the number of dimensions. For your convenience, we have already implemented the loading of input data in `./binsrc/DRmain.c`.

(a) [**20 points**] Modify the DRTREE code to support the containment queries. The output format is the same with the range search, i.e. one record in tsv format per line and a single line of the count.

Important: The output format: `make test`, in `./Q3`, should give empty result. That is, running your code on input `test0.txt`, it should give exactly the file `output0.txt`.

Hint1: Remember to remove all other debugging messages before you submit the code.

[1]FYI, 'D' stands for 'deferred split' R-tree - but you do not need to worry about that.

Question 3 continues. . .

Hint2: **Test** your code, for corner cases. We will also test your code against other 'secret' data which will be disclosed after the due date.

Hint3: Check the code for the range query.

Solution: Please see `binsrc/DRmain.c`, `libsrc/DR.c`, `libsrc/DRtree.c`, and `libsrc/DRrect.c`.

Grading info:

- *-5 for +1/-1 in count*
- *-10 for every wrong output on hidden data*

(b) [**10 points**] A hard copy of the output of your code, on the input data (`./Q3/test.txt`. Delete irrelevant lines and keep only the complete result of containment queries.

Solution:

```
0 0 7 3 9
2 4 9 1 10
containment: 2 records found
```

Grading info:

- *-5 for +1/-1 in count*
- *-10 for wrong output*
- *-25 no electronic submission*

What to turn in:

- **Code:** Submit your code to blackboard, in the `./Q3` directory of your `tar.gz` file.
- **Answers:** Submit hard copy for
  1. your code (only relevant parts),
  2. and the output of your code on `test.txt`.