Heinrich Heine
University Düsseldorf

Institut für Informatik
Datenbanken und Informationssysteme
Prof. Dr. Stefan Conrad
Thomas Germer

WiSe 24/25
Assignment 11
Due January 14, 8:00 AM

Relational Databases and Data Analysis

- As usual, use `zipme.py` to create the archive to upload in ILIAS.
- The upcoming assignment 12 will have an additional exercise for 5 bonus points. The threshold for participating in the oral exam remains at 80 points.

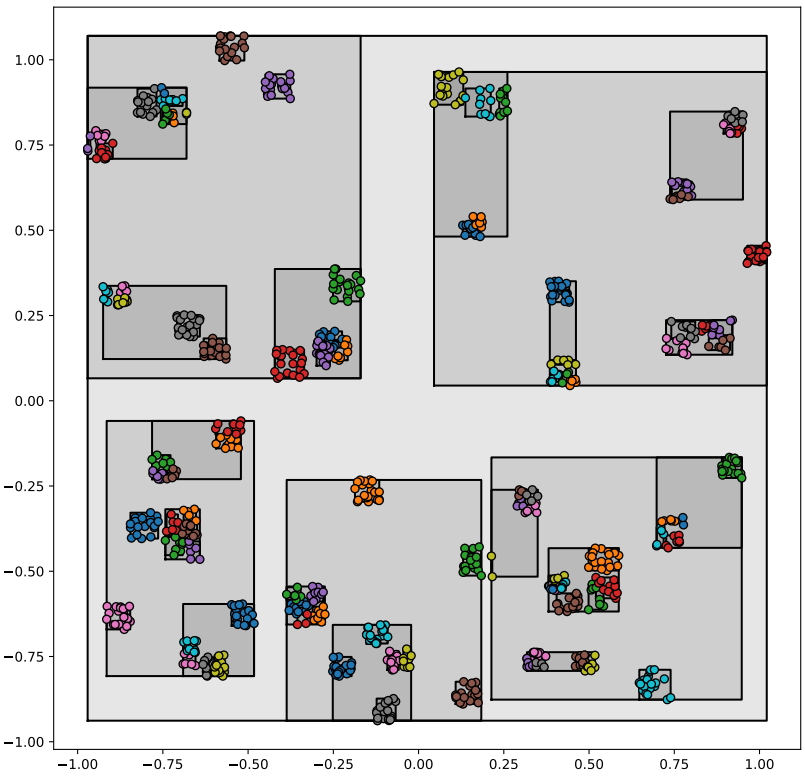
Exercise 1 *R-trees* (2 + 3 Points)

R-trees are useful data structures to accelerate spatial queries, for example to find all points within a circle. In this exercise, you may assume that data points are two-dimensional.

(a) You are given the root node of an R-tree (for details, see `test_exercise_1.py`).

- A tree node may either be an internal node or a leaf node. Internal nodes have child nodes, while leaf nodes have a list of data points.
- Each node has an axis-aligned bounding rectangle defined by the minimum and maximum coordinates of points contained in the descendents' leaf nodes.

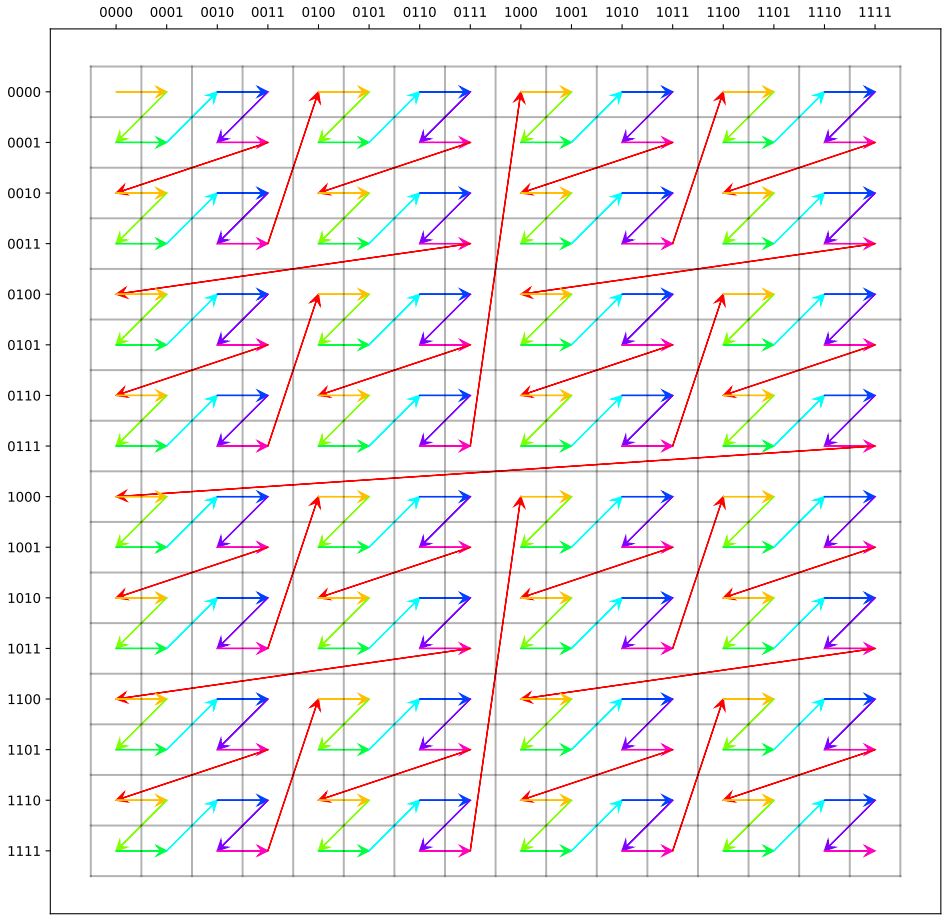
Plot the bounding rectangles of nodes and the the points stored in leaf nodes using Matplotlib.



(b) Implement a function to build a "somewhat good" R-tree from points. It will be judged by how much faster it is to search points in this tree as opposed to a very bad tree. For simplicity, the tree does not have to be strictly balanced.

Exercise 2 *Z-Curves* (1 + 1 Points)

Z-curves are useful to transform discrete values from a higher dimensional space to a one-dimensional space, while still preserving some locality.



(a) Implement a function to transform a one-dimensional index to an n-dimensional point on a z-curve. No additional imports allowed for this exercise.

(b) Do the same as in the previous exercise, but the other way round, i.e. implement a function to transform an n-dimensional point on a z-curve to a one-dimensional index.

Exercise 3 *Prelude to Bitmap Indexes* (3 Points)

In the past, many students had trouble with the algorithmic complexity of various operations, especially in Python. Since we will need a good understanding of algorithmic complexity to successfully solve the next assignment, we will explore this topic a bit more in this exercise.

(a) Devise experiments to measure the algorithmic complexity of the following operations with respect to n:

1. Appending a value to a `list` with n elements (using the `append` function).
2. Prepending a value to a `list` with n elements.
3. Appending a character to a string with n characters.
4. Retrieving the value for a random key from a `dict` (exclude the time to choose the key) with n key-value pairs.
5. Appending a (least-significant) bit to an integer with n bits.
6. Changing a bit in an integer with n bits.

Plot the measurements as subplots and save them as `exercise_3_a.png` using Matplotlib. Finally, write the algorithmic complexity of each operation in `exercise_3_a.txt`. Make sure to only measure the time of the specified operations, not other things.