Homework 3: Hash Tables and AVL Trees

L5-Group-3

Habib University
Spring 2023

Part I
Game of Life

> Conway's Game of Life is a simple simulation with surprisingly complex behavior. We start with a simple 2D grid of cells. Each cell can be either alive or dead. The rules are:
>
> - A live cell stays alive if it has two or three live neighbors.
> - A dead cell becomes alive if it has exactly three live neighbors.
>
> We start with a certain set of cell states and then iterate, applying the rules to every cell at each iteration. Despite their simplicity, these rules can produce amazingly complicated behavior.
>
> — A flexible implementation of Conway's Game of Life [3]

The rules above are slightly modified from their original formulation so as to be more amenable to our eventual implementation.

Initial Configurations

The game begins with an initial configuration of live cells. This constitutes the starting state of the game. As the rules of the game are deterministic, the starting state completely determines all subsequent states of the game. Some initial configurations result in interesting behavior and have led to the identification of certain classes of patterns.

Still Life These patterns are not affected by further iterations of the game and stay unchanged as the game proceeds.

Oscillators These patterns recur after a fixed number of iterations which are called the *period* of the oscillator.

Spaceships These are oscillators that change position or *glide* across the grid.

Computation

> It's possible even, to create patterns which emulate logic gates (and, not, or, etc.) and counters. Building up from these, it was proved that the Game of Life is Turing Complete [4], which means that with a suitable initial pattern, one can do any computation that can be done on any computer. Later, Paul Rendell actually constructed a simple Turing Machine as a proof of concept, which can be found here [5].
>
> — Chaos and Fractals: Conway's Game of Life [2]

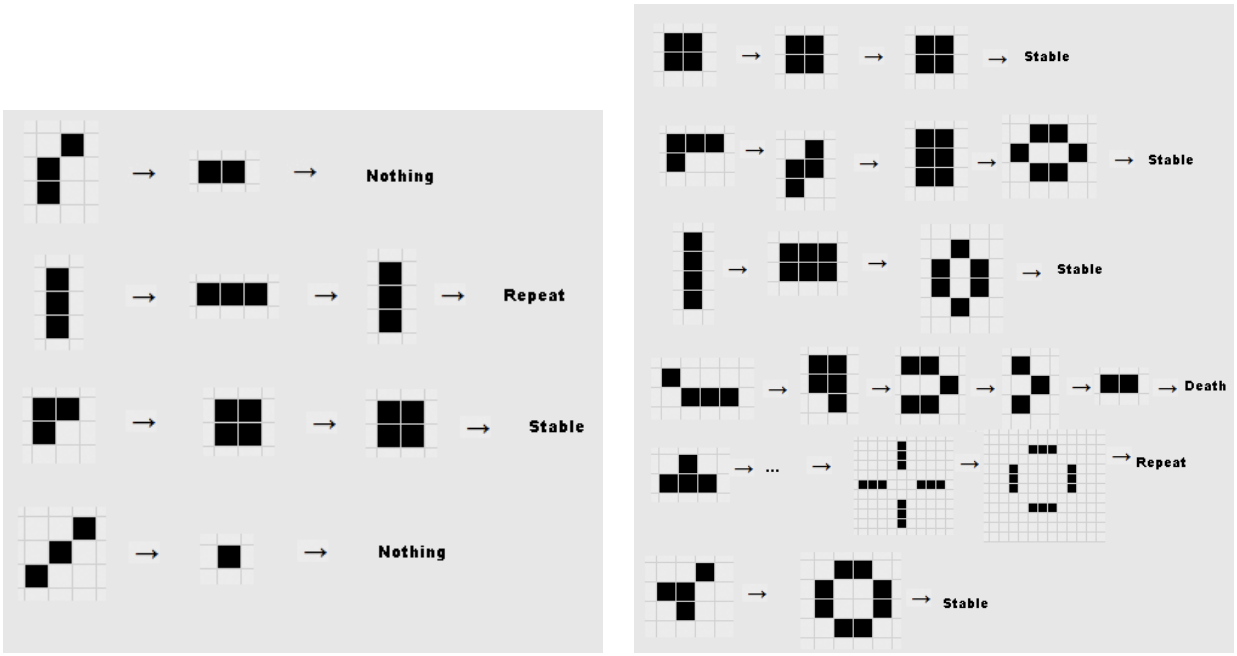

Figure 1: Evolution of the game from some initial states, taken from [2].

Visualization

As the Game of Life is a dynamic system, i.e. its state changes with each iteration, it is best viewed as an animation which this document is unable to present. See Wikipedia [1] or any of the other pages linked in this document for helpful animations. There are also many inspired and inspiring videos on YouTube including [6].

Implementation Details and Tasks

An implementation of Conway's Game of Life is provided in the accompanying `src/` folder. The required functionality and components are defined in several files as follows.

`game.py` provides the `Game` class and also a `main()` function to run the game.

`life.py` provides the `Life` and `Config` classes. `Life` encodes the state of the game and `Config` contains various configuration options. Both are used by the `Game` class.

`hashtables.py` provides the abstract classes `MySet` and `MyDict` and contains declarations of the classes `ChainedSet`, `ChainedDict`, `LinearSet`, and `LinearDict` which derive from the abstract classes. You have to implement the derived classes so as to override and implement the interface defined in the abstract classes. These classes are required by the `Life` class in order to store and update the state of the game.

Class details

`Life` Every instance is initialized with a starting configuration which is stored using two different hash tables.

`self._alive` stores the two-dimensional (x, y) coordinates of `live cells`.

`self._nbr_count` stores *neighbor information* as key-value pairs where each key is an (x, y) coordinate of a cell with *live neighbors* and its value is the count of its live neighbors.

The hash tables store not only the initial configuration but, once the game starts, they store the state of the game at the end of each iteration. During each iteration, the live cells have to be used to populate neighbor information which in turn must be used to update the live cells for the next iteration. This is already coded in the `step` method of `Life`. *Do not modify* the implementation of this class.

`Game` It runs a `Life` instance according to the provided configurations with an option for animation. In the animation system used, the origin, $(0, 0)$, is at the top left of the window. This class is provided for your testing. *You may modify* it as per your needs.

All files are fully documented. Please refer to in-file documentation for further details.

Task

Your task is to implement the `MySet` and `MyDict` subclass in `src/hashtables.py` using two different conflict resolution methods. Specifically, you have to implement the following classes which are utilized in the implementation of `Life`.

`ChainedSet` A hash table that implements the `MySet` interface using chaining for conflict resolution.

`ChainedDict` A hash table that implements the `MyDict` interface as an associative container and uses chaining for conflict resolution.

`LinearSet` A hash table that implements the `MySet` interface using linear probing for conflict resolution.

`LinearDict` A hash table that implements the `MyDict` interface as an associative container and uses linear probing for conflict resolution.

The conflict resolution strategy to use is indicated through a parameter passed to `Life` at the time of initialization.

Requirements

You will need to install the following modules.

`typing` to support certain type hints used in the code

`time` to support animation of the game

Tips

Below are some tips to gain a better understanding of the homework and facilitate your work for this assignment.

- Go over the provided implementation in order to understand the role and function of each class and its methods.
- Run the game using native python types as described below.
- Run the game using different initial configurations defined in the `Config` class.
- Feel free to add other initial configurations defined to the `Config` class.
- Feel free to modify the configuration options in `main()` in order to observe their effect.

Testing

You can test the framework by initializing `self._alive` and `self._nbr_count` with a native python `set` and `dict` instance respectively in `Life.__init__()`.

Once you have successfully implemented the hashtables, you can test your code by running the provided `pytest` files. These will also be used by GitHub to auto-grade your submission.

Credits

The Game of Life segment is adapted from [3].

Part II
Scientific Publications

You are working for a startup that provides a search engine for scientific publications. The search engine allows users to search for publications by entering keywords related to the topic they are interested in. Your job is to create an AVL Tree that will be used to index the publications and speed up the search process.

The startup has a large database of scientific publications, each containing multiple sentences. Your task is to create an AVL Tree where the key is a keyword and the value is a list of tuples where the first element is the index of the document in which the keyword is found and the second element is the index of the sentence in the document where the keyword is found.

The search engine will use the AVL Tree to quickly retrieve the list of publications that contain a given keyword. Your goal is to create an efficient implementation of the AVL Tree that can handle the large amount of data and provide fast search times. In this programming assignment, you are given a data file `data/publications.csv` that contains records of publications. Each records contains the following attributes:

- Title
- Abstract

Implementation Details and Tasks

- Implement the `AVLTree` class in the accompanying file, `avl_tree.py`, in the accompanying `src/` subfolder.
- In the `AVLTree` class, implement the following methods:
  - `insert (self,key:str,value: Tuple[int,int]) -> None`: Inserts a `(doc_id, sen_id)` tuple into the AVL Tree with the given keyword .
  - `search (self,key:str) -> List[Tuple[int,int]]`: Searches for a keyword in the AVL Tree and returns a list of `(doc_id, sen_id)` corresponding to the keyword. Returns an empty list if the keyword is not found.
  - `rotate_left(self,node:AVLTreeNode)-> None`: Performs a left rotation on the subtree rooted at the given node.
  - `rotate_left(self,node:AVLTreeNode)-> None`: Performs a right rotation on the subtree rooted at the given node.
  - `display(self)-> List[str]`: Performs an in-order traversal on the AVL Tree and returns a list of keys.
- The `Publication` class which loads the file, prepossesses the data and uses the `AVLTree` class has already been implemented in the accompanying file, `publication.py`, in the accompanying `src` subfolder.

Input and Output

In the following examples, the table 1 displays shows some sample documents based on which the AVL Tree is constructed. The program takes as input a keyword, searches the AVL Tree for the given keyword and then returns a list of tuples `(doc_id,sen_id)`

Title	Abstract
Doc1	This is the first document.
Doc2	This is the second document.
Doc3	This is the third document.

Table 1: Sample Documents

Input	Output
document	(0, 0), (1, 0), (2, 0)
third	(2,0)

Table 2: Sample Test Cases

Requirements

You will need to install the following modules.

`nltk` to remove stop words, punctuation, and stem the words.

Testing

Once you have successfully implemented the methods, you can test your code by reading from the accompanying data file, `data/publications.csv`, and performing queries on it. For grading purposes, your submission will be tested automatically by GitHub using the accompanying `pytest` file, `test_avl.py`.

References

[1] Conway's Game of Life, https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life, last accessed on 16 Feb 2022.

[2] Chaos and Fractals: Conway's Game of Life, http://pi.math.cornell.edu/~lipa/mec/lesson6.html, last accessed on 16 Feb 2022.

[3] A flexible implementation of Conway's Game of Life, https://www.refsmmat.com/posts/2016-01-25-conway-game-of-life.html, last accessed on 16 Feb 2022.

[4] Turing complete, https://simple.wikipedia.org/wiki/Turing_complete, last accessed on 16 Feb 2022.

[5] This is a Turing Machine implemented in Conway's Game of Life., http://www.rendell-attic.org/gol/tm.htm, last accessed on 16 Feb 2022.

[6] epic conway's game of life, https://www.youtube.com/watch?v=C2vgICfQawE, last accessed on 16 Feb 2022.