Data Structures and Algorithms

Lab 5: Hash and searching

Exercise 1

Refer back ***Lab 3&4 Sorting and searching*** for detail description.

In Lab 4, you do a simple spell checker with Linear Search and Binary search.

In this Lab 5, you continue that project by adding a new search – Search with Hash Table.

Requirement for hash table:

1. You will use djb2 hash function to convert a ***string*** to an ***Int*** in this exercise. Explain in report what you know about this hash function.
2. Choose your own hash table ***size***. Explain in report why you choose this size.
3. Your hash function to map result from ***djb2*** to Hash Table position is: ***hash % [table size]***
4. Experient with 4 ways to handle collision:
    a. Chaining approach
    b. Linear Probing
    c. Quadratic Probing
    d. Double Hashing (Choose your own 2nd Hash Function. Explain in report why you choose this hash function)

Compare the result from different method. Analyze and discussion.

Methods	**Run time (ms)**
Linear Search	
Binary search	
Chaining approach	
Linear Probing	
Quadratic Probing	
Double Hashing	

Exercise 2

Game of Life is a simulation game - a game that tries to resemble real life processes.

Life simulations take place on an infinite grid:

Each cell (square) in the grid can be in one of two possible states: Alive or Dead. A living state is usually indicated by a colored square; a dead state by a blank square.

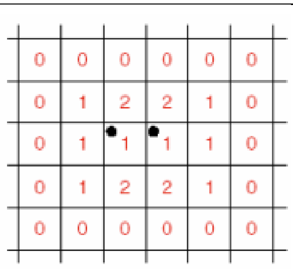
- Each square is adjacent to eight others. These eight squares surrounding the square in question are referred to as its neighbors.
- Every simulation begins with a set of live cells. This is referred to as the initial population.
- Then, the pattern evolves according to a certain set of rules. The set of rules determines what happens to each cell from one generation (configuration of live and dead cells) to the next.

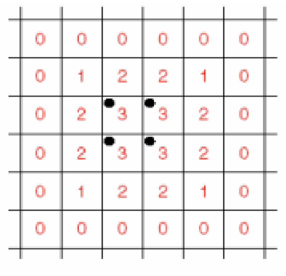
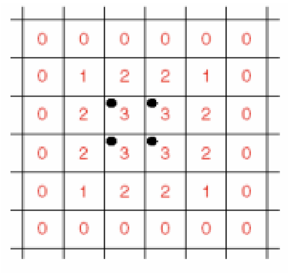
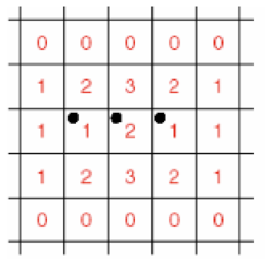
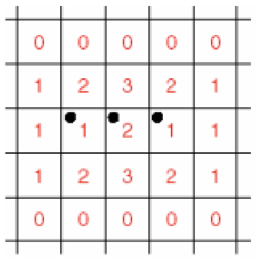
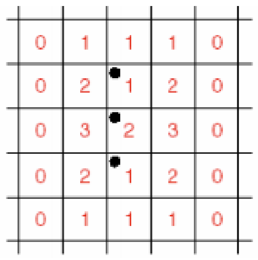
The Rules:

Number of live neighbors	State of cell in next generation
0 or 1	Dead (death by isolation)
2	Remains *stable* (if the cell is alive, it remains alive; if it is dead, it stays dead.)
3	Alive (referred to as a *birth*)
4 or more	Dead (death by overcrowding)

Students illustrate a problem on a 10x10 grid. Let the user enter the initial state (text file), and the program outputs the state afterwards.

Example:

Initial state	Output
 0 0 0 0 0 0 0 1 2 2 1 0 0 1 1 1 1 0 0 1 2 2 1 0 0 0 0 0 0 0	***All Dead***

Initial state	**Output**
	
	 and 

Exercise 3

The Monster family wants to keep a database of all Monsters and their handphone number. You are employed by the Monster Family to implement this phone book for them. You are to maintain a list of Monster names and phone numbers using a hash table, using Monster names as the keys.

For simplicity, you should resolve collision by linear probing.

To hash a Monster name, implement a hashCode() method to convert a String object to an int. Implement another function h() to hash the values returned by hashCode() to map it to one of the slots in our hash table.

In this problem, you're required to write 4 methods: add(), delete(), update() and find().

(1) add(Monster m, int p) -- adds a monster m with phone number p to the phone book.

(2) delete(Monster m) -- deletes monster m and its associated phone number from the phone book. Throws error if monster m is not in the phone book.

(3) updates(Monster m, int p) -- changes the phone number of monster m to p. Throws error if monster m is not in the phone book.

(4) find(Monster m) -- returns the phone number of monster m. Throws an error if monster m is not in the phone book.

For example, about phonebook.txt:

```
Edward Howard Dudley          1912352460
Mr. Gatema                    2888888888
Clyde Thornton                312345987
Yolanda "Yo-yo" Cribbins      9996669999
```

Terms of submission

- Student are required to submit both source code, document and some additional files for this Lab.

- Compress them with the name <StudentID>.zip or <StudentID>.rar. Then submit this compressed file.

Similar source code, plagiarism or spam submissions will score 0 in this SUBJECT