Hashpipe Report

Simon Høiberg

April 14, 2018

Abstract

HashPipe.java is a symbol table implemented with a Hashpipe data structure. HashPipe supports search and insertion in time proportional to $O(LogN)$ on average.

HashPipe contains the following API:

```
public HashPipe()
public int size()
public void put(String key, Integer val)
public Integer get(String key)
public String floor(String key)
```

Class design

HashPipe uses a pointer-based data structure where each key-value pair is represented by a pipe.

Every pipe has a given height thus contains a pointer to the neighbour pipe(s). On instantiation, a root-pipe is created which solely represents pointers and every key-value pair that is put will be represented as new pipes.

The height of the root-pipe will be fixed at 32, and every new pipe will have it's height calculated deterministically.

Every pipe is based in the following structure

```
class Pipe {
     String key;
     int value;
     int height;
     Pipe[] nextPipes;

     // Root pipe will be created with fixed height of 32
     public Pipe()

     // New pipe will be created given it's key-value pair
     public Pipe(String key, int value)

     public String getKey()

     public int getValue()

     public int getHeight()

     public void setNextPipe(Pipe pipe, int index)

     public Pipe getNextPipe(int index)
  }
```

The field **Pipe[] nextPipes** will represent the height of the pipe with *height* amount of pointers to the neighbour pipe(s).

The **put()** method works by following the root-pipe's pointers searching from top down and recursivly rearranging the pointers such that the key-value pipes will be ordered by key ascending fra left to right.

The **get()** method works by recursivly following the root-pipe's pointers searching from top down until a key-match is found or the bottom of the pipe is reached. The value of the pipe will be returned if key-match is found, null is returned otherwise.

Performence tests

Following performence tests is done using CodeJudge

TestsSmall

Test	*Status*	*CPU TIME*	*WALL TIME*
Test01	Succeeded	0.32 s	0.42 s
Test02	Succeeded	0.17 s	0.21 s
Test03	Succeeded	0.13 s	0.18 s

TestsBest

Test	*Status*	*CPU TIME*	*WALL TIME*
Test01	Succeeded	0.88 s	0.93 s
Test02	Runtime error	2.78s s	2.97 s

TestsBest

Test	*Status*	*CPU TIME*	*WALL TIME*
Test01	Succeeded	0.54 s	0.59 s
Test02	Succeeded	1.12s s	1.17 s