자료구조설계

제 출 일	2017.11.15.
과 제 번 호	09
분 반	01
학 과	컴퓨터공학과
학 번	201602038
이 름	이 미 진

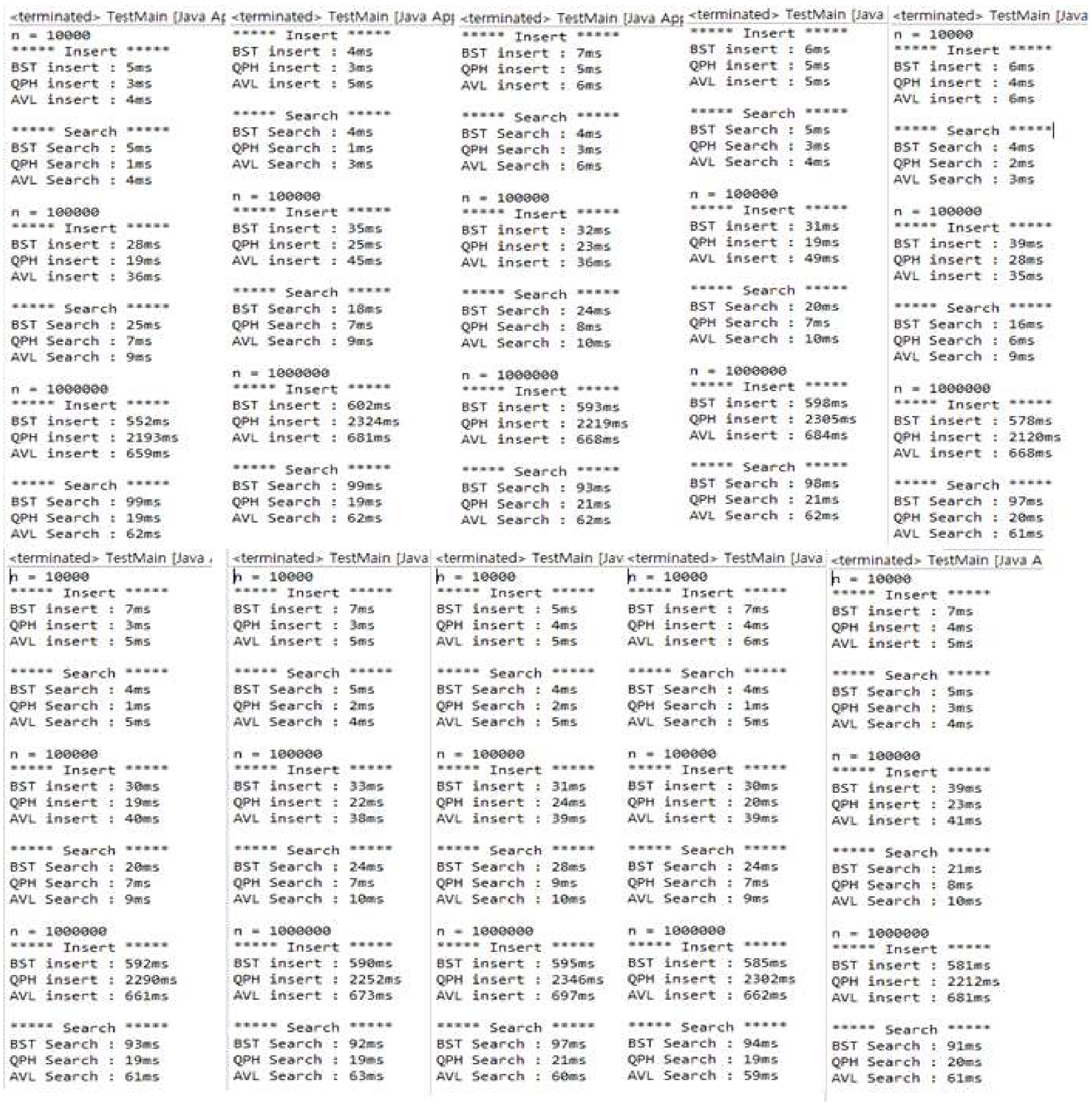
1. 구현 내용 설명

1-1. 코드 설명

```
package DS01_09_201602038;

import java.util.ArrayList;
import java.util.Collections;

public class TestMain {
	public static void main(String args[]) {
	long start,end;
	ArrayList<Object> arrayl = new ArrayList<>();
	BinarySearchTree bst;
	QuadraticProbingHashTable qph = new QuadraticProbingHashTable();
	AVLTree avl;

	int i;
	int n=10000;
	for( ;n<10000000; ){ // n이 10000, 100000, 1000000 일 동안 반복
		int[] arrayi = new int[n];

		System.out.println("n = "+n);
		System.out.println("***** Insert *****");

		for( i = 0; i < n; i++ ) {
			arrayl.add(i);
		}
		Collections.shuffle(arrayl);	// 무작위 추출
		for( i = 0; i < n; i++ ) {
			arrayi[i]= (int) arrayl.get(i);
		}

		/* bst Insert start */
		start = System.currentTimeMillis();
		bst = new BinarySearchTree((Comparable<?>) arrayl.get(0));

		for( i = 0; i < n; i++ ) {
			bst.insert(arrayi[i]);
		}
		end = System.currentTimeMillis();

		System.out.println("BST insert : "+(end-start)+"ms");
		/* bst Insert end */
		/* qph Insert start */
		start = System.currentTimeMillis();

		for(i = 0; i < n; i++) {
			qph.put(arrayi[i], arrayi[i]);
		}
		end = System.currentTimeMillis();

		System.out.println("QPH insert : "+(end-start)+"ms");
		/* qph Insert end */

		/* avl Insert start */
		start = System.currentTimeMillis();

		avl = new AVLTree((int) arrayl.get(0));

		for( i = 1; i < n; i++ ) {
			avl.grow(arrayi[i]);
		}
		end = System.currentTimeMillis();

		System.out.println("AVL insert : "+(end-start)+"ms\n");
		/* avl Insert end */
		System.out.println("***** Search *****");

		/* bst Search start */
		start = System.currentTimeMillis();

		for( i = 0; i < n; i++ ) {
			bst.contains(i);
		}
		end = System.currentTimeMillis();

		System.out.println("BST Search : "+(end-start)+"ms");
		/* bst Search end */
		/* qph Search start */
		start = System.currentTimeMillis();

		for( i = 0; i < n; i++ ) {
			qph.get(i);
		}
		end = System.currentTimeMillis();
		System.out.println("QPH Search : "+(end-start)+"ms");
		/* qph Search end */
		/* avl Search start */
		start = System.currentTimeMillis();

		for( i= 0; i < n; i++ ) {
			avl.search(i);
		}

		end = System.currentTimeMillis();

		System.out.println("AVL Search : "+(end-start)+"ms\n");
		/* avl Search end */
		n = n*10;
	}
}
}
```

각 알고리즘별 성능을 측정하기 위해 System.currentTimeMillis()로 BinarySearchTree, QuadraticProbingHashTable, AVLTree 의 삽입, 검색을 수행할 동안의 시작 시간과 종료 시간을 기록한다. 그 후에 end-start 로 삽입과 검색에 걸린 시간을 출력한다.
이번 과제에서는 Collections 클래스에 있는 Collections.shuffle 메소드를 사용하였는데, List나 배열의 내용을 랜덤으로 리턴해주는 기능을 한다.
n만큼 I를 증가시키면서 arrayI에 I를 add 해주고, arrayI에 있는 값들을 랜덤으로 리턴해주면서 arrayi에 arrayI에서 get한 값을 저장해준다. 알고리즘을 수행할 때에는 랜덤으로 리턴된 값이 들어가 있는 arrayi을 이용한다.
for문을 통해 n이 10000,100000,1000000 일 동안 반복하고, 모든 과정을 끝냈을 땐 n에 10을 곱해주어 수를 늘린다.

2. 실행 결과 화면



```
<terminated> TestMain [Java Ap
n = 10000
***** Insert *****
BST insert : 5ms
QPH insert : 3ms
AVL insert : 4ms

***** Search *****
BST Search : 5ms
QPH Search : 1ms
AVL Search : 4ms

n = 100000
***** Insert *****
BST insert : 28ms
QPH insert : 19ms
AVL insert : 36ms

***** Search *****
BST Search : 25ms
QPH Search : 7ms
AVL Search : 9ms

n = 1000000
***** Insert *****
BST insert : 552ms
QPH insert : 2193ms
AVL insert : 659ms

***** Search *****
BST Search : 99ms
QPH Search : 19ms
AVL Search : 62ms
```

```
<terminated> TestMain [Java Ap
***** Insert *****
BST insert : 4ms
QPH insert : 3ms
AVL insert : 5ms

***** Search *****
BST Search : 4ms
QPH Search : 1ms
AVL Search : 3ms

n = 100000
***** Insert *****
BST insert : 35ms
QPH insert : 25ms
AVL insert : 45ms

***** Search *****
BST Search : 18ms
QPH Search : 7ms
AVL Search : 9ms

n = 1000000
***** Insert *****
BST insert : 602ms
QPH insert : 2324ms
AVL insert : 681ms

***** Search *****
BST Search : 99ms
QPH Search : 19ms
AVL Search : 62ms
```

```
<terminated> TestMain [Java Ap
***** Insert *****
BST insert : 7ms
QPH insert : 5ms
AVL insert : 6ms

***** Search *****
BST Search : 4ms
QPH Search : 3ms
AVL Search : 6ms

n = 100000
***** Insert *****
BST insert : 32ms
QPH insert : 23ms
AVL insert : 36ms

***** Search *****
BST Search : 24ms
QPH Search : 8ms
AVL Search : 10ms

n = 1000000
***** Insert *****
BST insert : 593ms
QPH insert : 2219ms
AVL insert : 668ms

***** Search *****
BST Search : 93ms
QPH Search : 21ms
AVL Search : 62ms
```

```
<terminated> TestMain [Java
***** Insert *****
BST insert : 6ms
QPH insert : 5ms
AVL insert : 5ms

***** Search *****
BST Search : 5ms
QPH Search : 3ms
AVL Search : 4ms

n = 100000
***** Insert *****
BST insert : 31ms
QPH insert : 19ms
AVL insert : 49ms

***** Search *****
BST Search : 20ms
QPH Search : 7ms
AVL Search : 10ms

n = 1000000
***** Insert *****
BST insert : 598ms
QPH insert : 2305ms
AVL insert : 684ms

***** Search *****
BST Search : 98ms
QPH Search : 21ms
AVL Search : 62ms
```

```
<terminated> TestMain [Java
n = 10000
***** Insert *****
BST insert : 6ms
QPH insert : 4ms
AVL insert : 6ms

***** Search *****
BST Search : 4ms
QPH Search : 2ms
AVL Search : 3ms

n = 100000
***** Insert *****
BST insert : 39ms
QPH insert : 28ms
AVL insert : 35ms

***** Search *****
BST Search : 16ms
QPH Search : 6ms
AVL Search : 9ms

n = 1000000
***** Insert *****
BST insert : 578ms
QPH insert : 2120ms
AVL insert : 668ms

***** Search *****
BST Search : 97ms
QPH Search : 20ms
AVL Search : 61ms
```

```
<terminated> TestMain [Java
n = 10000
***** Insert *****
BST insert : 7ms
QPH insert : 3ms
AVL insert : 5ms

***** Search *****
BST Search : 4ms
QPH Search : 1ms
AVL Search : 5ms

n = 100000
***** Insert *****
BST insert : 30ms
QPH insert : 19ms
AVL insert : 40ms

***** Search *****
BST Search : 20ms
QPH Search : 7ms
AVL Search : 9ms

n = 1000000
***** Insert *****
BST insert : 592ms
QPH insert : 2290ms
AVL insert : 661ms

***** Search *****
BST Search : 93ms
QPH Search : 19ms
AVL Search : 61ms
```

```
<terminated> TestMain [Java
n = 10000
***** Insert *****
BST insert : 7ms
QPH insert : 3ms
AVL insert : 5ms

***** Search *****
BST Search : 5ms
QPH Search : 2ms
AVL Search : 4ms

n = 100000
***** Insert *****
BST insert : 33ms
QPH insert : 22ms
AVL insert : 38ms

***** Search *****
BST Search : 24ms
QPH Search : 7ms
AVL Search : 10ms

n = 1000000
***** Insert *****
BST insert : 590ms
QPH insert : 2252ms
AVL insert : 673ms

***** Search *****
BST Search : 92ms
QPH Search : 19ms
AVL Search : 63ms
```

```
<terminated> TestMain [Jav
n = 10000
***** Insert *****
BST insert : 5ms
QPH insert : 4ms
AVL insert : 5ms

***** Search *****
BST Search : 4ms
QPH Search : 2ms
AVL Search : 5ms

n = 100000
***** Insert *****
BST insert : 31ms
QPH insert : 24ms
AVL insert : 39ms

***** Search *****
BST Search : 28ms
QPH Search : 9ms
AVL Search : 10ms

n = 1000000
***** Insert *****
BST insert : 595ms
QPH insert : 2346ms
AVL insert : 697ms

***** Search *****
BST Search : 97ms
QPH Search : 21ms
AVL Search : 60ms
```

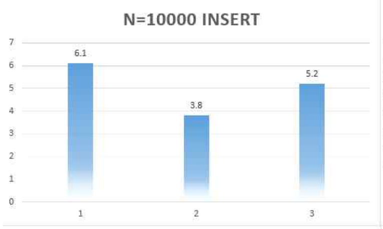
```
<terminated> TestMain [Java
n = 10000
***** Insert *****
BST insert : 7ms
QPH insert : 4ms
AVL insert : 6ms

***** Search *****
BST Search : 4ms
QPH Search : 1ms
AVL Search : 5ms

n = 100000
***** Insert *****
BST insert : 30ms
QPH insert : 20ms
AVL insert : 39ms

***** Search *****
BST Search : 24ms
QPH Search : 7ms
AVL Search : 9ms

n = 1000000
***** Insert *****
BST insert : 585ms
QPH insert : 2302ms
AVL insert : 662ms

***** Search *****
BST Search : 94ms
QPH Search : 19ms
AVL Search : 59ms
```

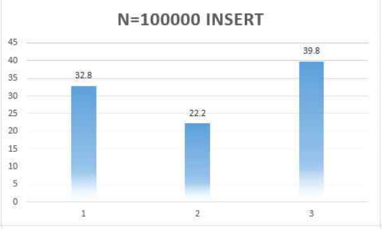
```
<terminated> TestMain [Java A
n = 10000
***** Insert *****
BST insert : 7ms
QPH insert : 4ms
AVL insert : 5ms

***** Search *****
BST Search : 5ms
QPH Search : 3ms
AVL Search : 4ms

n = 100000
***** Insert *****
BST insert : 39ms
QPH insert : 23ms
AVL insert : 41ms

***** Search *****
BST Search : 21ms
QPH Search : 8ms
AVL Search : 10ms

n = 1000000
***** Insert *****
BST insert : 581ms
QPH insert : 2212ms
AVL insert : 681ms

***** Search *****
BST Search : 91ms
QPH Search : 20ms
AVL Search : 61ms
```

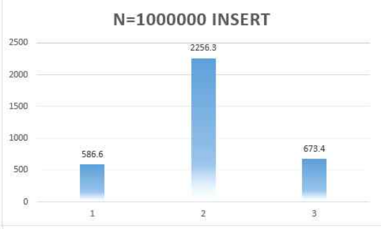
10번 실행하여 시간의 평균을 계산하였다.

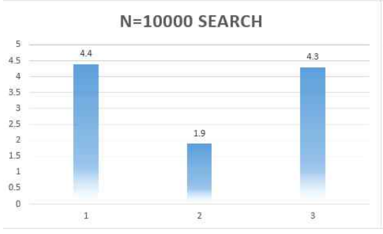
Insert	n=10000	n=100000	n=1000000
BST	6.1	32.8	586.6
QPH	3.8	22.2	2256.3
AVL	5.2	39.8	673.4

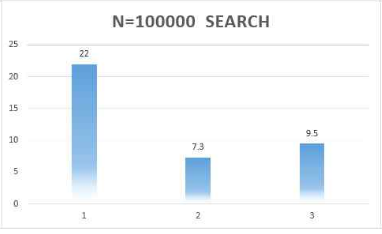
Search	n=10000	n=100000	n=1000000
BST	4.4	22	95.3
QPH	1.9	7.3	19.8
AVL	4.3	9.5	61.3

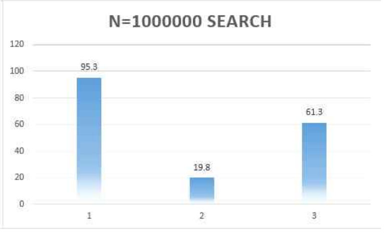












n=10000 Insert (빠른순) : qph > avl > bst
n=10000 Search (빠른순) : qph > avl > bst
n=100000 Insert (빠른순) : qph > bst > avl
n=100000 Search (빠른순) : qph > avl > bst
n=1000000 Insert (빠른순) : bst > avl > qph
n=1000000 Search (빠른순) : qph > avl > bst

삽입 시간의 경우 n=100000 까지는 QuadraticProbingHashTable이 다른 알고리즘보다 적은 시간이 걸리는걸 볼 수 있었지만, n이 1000000과 같이 큰 수가 되자 시간이 급증하였다. 탐색 시간의 경우 전체적으로 QuadraticProbingHashTable가 적은 시간에 수행되는 것을 볼 수 있다. n이 10000일 때는 BinarySearchTree와 AVLTree의 차이가 별로 없었지만 n이 커질수록 차이가 생겼다.
수가 커질 때, Insert에서는 QuadraticProbingHashTable이 성능이 안좋게 측정되었지만 Search를 할 때는 다른 알고리즘보다 월등히 빠르므로 탐색용도로 쓴다면 QuadraticProbingHashTable 알고리즘을 사용하는 것이 좋을 것 같다.