1. (해싱, 선형조사법) 다음은 선형조사법으로 해싱을 구현한 코드이다. 아래 코드를 입력하고 실행하면서 해싱 구현법을 학습하시오. (Reference: http://opendatastructures.org/versions/edition-0.1e/ods-java/5_1_ChainedHashTable_Hashin.html, CC-BY-2.5-CA, Reference: https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/SeparateChainingHashST.java.html, GPLv3)

```
public class Test {
	public static void main(String[] args) {
		SimpleLinearProbingHashTable  ht=new SimpleLinearProbingHashTable(1000);
		ht.put("Korea");
		ht.put("Japan");
		System.out.println(ht.get("Korea"));
		System.out.println(ht.get("Japan"));
		System.out.println(ht.get("China"));
	}
}
// Reference: http://opendatastructures.org/versions/edition-0.1e/ods-java/5_1_ChainedHashTable_Hashin.html, CC-BY-2.5-CA
// Reference: https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/SeparateChainingHashST.java.html, GPLv3
public class SimpleLinearProbingHashTable {
	private int HashTableSize;
	Object	hashTable[];
	public SimpleLinearProbingHashTable(int size) {
		HashTableSize=size;
		hashTable=new Object[HashTableSize];
	}
	public boolean put(Object key) {
		int		index=hash(key);
		while(hashTable[index]!=null){
			if(hashTable[index].equals(key)) return false;
			index=(index+1)%HashTableSize;
		}
		hashTable[index]=key;
		return true;
	}
	private int hash(Object key) {
		return (key.hashCode()&0x7FFFFFFF)%HashTableSize;
	}
	public Object get(Object key) {
		int		index=hash(key);
		while(hashTable[index]!=null){
			if(hashTable[index].equals(key)) return hashTable[index];
			index=(index+1)%HashTableSize;
		}
		return null;
	}
}
```

2. (해싱, 체인법) 다음은 체인법으로 해싱을 구현한 코드이다. 아래 코드를 입력하고 실행하면서 해싱 구현법을 학습하시오. (Reference: http://opendatastructures.org/versions/edition-0.1e/ods-java/5_1_ChainedHashTable_Hashin.html, CC-BY-2.5-CA, Reference: https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/SeparateChainingHashST.java.html, GPLv3)

```
public class Test {
	public static void main(String[] args) {
			SimpleChainHashTable		ht=new SimpleChainHashTable(1000);
			ht.put("Korea");
			ht.put("Japan");
			System.out.println(ht.get("Korea"));
			System.out.println(ht.get("Japan"));
			System.out.println(ht.get("China"));
	}
}
import java.util.LinkedList;

// Reference: http://opendatastructures.org/versions/edition-0.1e/ods-java/5_1_ChainedHashTable_Hashin.html, CC-BY-2.5-CA
// Reference: https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/SeparateChainingHashST.java.html, GPLv3
public class SimpleChainHashTable {
	private int HashTableSize;
	LinkedList<Object>	hashTable[];
	public SimpleChainHashTable(int size) {
			HashTableSize=size;
			hashTable=new LinkedList[HashTableSize];
			for (int i = 0; i < hashTable.length; i++) hashTable[i]=new LinkedList<>();
	}
	public boolean put(Object key) {
			if(get(key)!=null) return false;
			hashTable[hash(key)].add(key);
			return true;
	}
	private int hash(Object key) {
			return (key.hashCode()&0x7FFFFFFF)%HashTableSize;
	}
	public Object get(Object key) {
			for (Object v : hashTable[hash(key)]) if(v.equals(key)) return v;
			return null;
	}
}
```

3. (**실습**: 해시테이블 resize) 이전 문제의 SimpleChainHashTable 클래스의 경우 해시테이블 포화 시 더 이상 자료 삽입이 불가하다. 해시테이블 포화 시 HashTableSize를 2배로 늘리는 resize() 함수와 print() 함수를 추가하여 아래 코드가 정상 동작하도록 하시오.

- keys 배열이 {"a", "a", "a", "a"}인 경우 실행 결과

```
hashTable[0] => [a]
해시테이블에 저장된 자료 수 = 1
```

- keys 배열이 {"a", "b", "c", "d", "a", "a", "e", "c", "d", "a"}인 경우 실행 결과

```
hashTable[0] => []
hashTable[1] => [a]
hashTable[2] => [b]
hashTable[3] => [c]
hashTable[4] => [d]
hashTable[5] => [e]
hashTable[6] => []
hashTable[7] => []
해시테이블에 저장된 자료 수 = 5
```

```
public class Test {
	public static void main(String[] args) {
		String	keys[]={"a", "a", "a", "a"};
		//String	keys[]={"a", "b", "c", "d", "a", "a", "e", "c", "d", "a"};

		SimpleChainHashTable		ht=new SimpleChainHashTable(1);
		for (int i = 0; i < keys.length; i++) ht.put(keys[i]);
		ht.print();
	}
}
```

4. (자바클래스 HashMap, HashSet) 다음은 해싱을 구현한 자바클래스 HashSet와 HashMap의 사용 예시 코드이다. 아래 코드를 입력하고 실행하면서 HashSet와 HashMap의 사용법을 학습하시오.

```
public class Test {
	public static void main(String[] args) {
		HashSet<String>	hashSet=new HashSet<>();
		hashSet.add("Korea");
		hashSet.add("Japan");
		System.out.println(hashSet.contains("Korea"));
		System.out.println(hashSet.contains("Japan"));
		System.out.println(hashSet.contains("China"));
	}
}
public class Test {
	public static void main(String[] args) {
		HashMap<String, Integer>	hashMap=new HashMap<>();
		hashMap.put("Korea", 99);
		hashMap.put("Japan", 87);
		System.out.println(hashMap.get("Korea"));
		System.out.println(hashMap.get("Japan"));
		System.out.println(hashMap.get("China"));
	}
}
```

5. (**실습**: BST vs. Hashing) 다음은 균형이진트리와 해싱의 자료 삽입 시간을 비교한 코드이다.

- **실습** #1: 아래 코드의 주석 처리된 문장을 resize 기능이 구현된 SimpleChainHashTable 클래스를 통해 실행하여 TreeSet와의 자료 삽입 시간을 비교해 보시오.

```
public class Test {
	public static void main(String[] args) {
		int		n=1000000;
		String	keys[]=new String[n];
		for (int i = 0; i < n; i++) keys[i]=genKeys();
		long		start;

		start=System.currentTimeMillis();
		TreeSet<String>	treeSet=new TreeSet<>();
		for (int i = 0; i < n; i++) treeSet.add(keys[i]);
		System.out.println("BST "+(System.currentTimeMillis()-start)/1000.+" sec.");

		start=System.currentTimeMillis();
		//SimpleChainHashTable		st=new SimpleChainHashTable(1);
		SimpleChainHashTable		st=new SimpleChainHashTable(n);
		for (int i = 0; i < n; i++) st.put(keys[i]);
		System.out.println("Hash "+(System.currentTimeMillis()-start)/1000.+" sec.");

	}
	private static String genKeys() {
		Random	random=new Random();
		int		len=random.nextInt(10)+5;
		String	s="";
		for (int j = 0; j < len; j++) s+=(char)(random.nextInt('z'-'a')+'a');
		return s;
	}
}
```

6. (**실습**: 배열 내 최빈값 탐색) 다음은 배열 n에 저장된 정수 중 재출현 빈도수가 최대인 정수(최빈값,mode)를 출력하는 코드이다. 이 코드를 아래 각 실습 조건에 맞게 완성하시오. 예를 들어, {1,3,7,4,8,3,7,3,3,7}에서 최빈값은 3이다. (References: 프로그래밍대회에서 배우는 알고리즘 문제해결전략, 2012, 구종만 저, (p.93-94))

- **실습** #1: 배열 n에 저장된 각 정수 n[i]에 대해, 배열 n에서의 n[i]의 빈도수를 계산하면서 그러한 빈도수가 최대인 값을 갱신하는 방식으로 구현하시오.
- **실습** #2: 배열 n을 정렬한 후, 정렬된 배열을 순차방문하면서 연속된 동일 값의 빈도수를 계산하여 그러한 빈도수가 최대인 값을 갱신하는 방식으로 구현하시오.
- **실습** #3: HashMap을 이용하여 구현하시오. HashMap의 key는 n[i]이며, value는 n[i]의 빈도수가 되도록 한다. 배열 n의 각 정수 n[i]에 대해 key값 n[i]가 HashMap에 존재하지 않는 경우 빈도수 1을 value로 설정하여 HashMap에 저장하고, n[i]가 HashMap에 존재하는 경우 기존 빈도수의 1 증가된 값을 value로 설정하여 HashMap에 재저장한다.
- **실습** #4: TreeMap을 이용하여 구현하시오. TreeMap의 key는 n[i]이며, value는 n[i]의 빈도수가 되도록 한다. 구현 방법은 HashMap의 경우와 동일하다.
- **실습** #5: 배열 n에 저장된 정수는 그 범위가 0~100이라고 가정하고, 계수정렬 기법을 활용하여 구현하시오. 이 실습의 경우 아래 코드에서 n[i]=random.nextInt(N/2) 문장을 n[i]=random.nextInt(101)로 변경하여 구현하시오.
- **실습** #6: N=1000000으로 설정하여 실습 #1~#5까지 구현 방법들의 소요 시간을 비교해 보시오.

```
public class Test {
	public static void main(String[] args) {
		Random	random=new Random();
		int		N=10000;
		int		n[]=new int[N];
		for (int i = 0; i < n.length; i++) n[i]=random.nextInt(N/2);
	}
}
```

References

- C로 쓴 자료구조론 (Fundamentals of Data Structures in C, Horowitz et al.). 이석호 역. 사이텍 미디어. 1993.
- 쉽게 배우는 알고리즘: 관계 중심의 사고법. 문병로. 한빛아카데미. 2013.
- C언어로 쉽게 풀어 쓴 자료구조. 천인국 외 2인. 생능출판사. 2017.
- 김윤명. (2008). 뇌를 자극하는 Java 프로그래밍. 한빛미디어.
- 남궁성. 자바의 정석. 도우출판.
- 김윤명. (2010). 뇌를 자극하는 JSP & Servlet. 한빛미디어.