ADS 2023 spring Week 6 Exercises

Exercises for Algorithms and Data Structures at ITU. The exercises are from *Algorithms, 4th Edition* by Robert Sedgewick and Kevin Wayne unless otherwise specified. Color-coding of difficulty level and alterations to the exercises (if any) are made by the teachers of the ADS course at ITU.

3.1.10 - Green Give a trace of the process of inserting the keys E A S Y Q U E S T I O N into an initially empty table using SequentialSearchST. How many compares are involved?

3.1.11 - Green Give a trace of the process of inserting the keys E A S Y Q U E S T I O N into an initially empty table using BinarySearchST. How many compares are involved?

3.4.1 - Green Insert the keys E A S Y Q U T I O N in that order into an initially empty table of $M = 5$ lists, using separate chaining. Use the hash function 11k % M to transform the *kth* letter of the alphabet into a table index.

3.4.10 - Green Insert the keys E A S Y Q U T I O N in that order into an initially empty table of size $M = 16$ using linear probing. Use the hash function 11k % M to transform the *kth* letter of the alphabet into a table index. Then redo this exercise for $M = 10$

3.1.13 - Green Would you use a sequential search ST or a binary search ST for an application that does 10^3 put() operations and 10^6 get() operations, randomly intermixed? Justify your answer.

3.1.14 - Green Would you use a sequential search ST or a binary search ST for an application that does 10^6 put() operations and 10^3 get() operations, randomly intermixed? Justify your answer.

3.4.4 - Yellow Design an algorithm to find values of a and M, with M as small as possible, such that the hash function (a * k) % M for transforming the *kth* letter of the alphabet into a table index produces distinct values (no collisions) for the keys S E A R C H X M P L. The result is known as a perfect hash function.

3.4.5 - Yellow Is the following implementation of getting the hash code of an object legal? If so, describe the effect of using it. If not, explain why.

```python
# Python

def __hash__(self):
    return 17
```

```java
// Java

public int hashCode()
    { return 17; }
```

3.4.13 - Yellow Which of the following scenarios leads to expected linear running time for a random search hit in a linear-probing hash table?

a) All keys hash to the same index.

b) All keys hash to different indices.

c) All keys hash to an even-numbered index.

d) All keys hash to different even-numbered indices.

3.4.15 - Yellow How many compares could it take, in the worst case, to insert N keys into an initially empty table of size N, using linear probing with array resizing?

3.4.26 - Yellow Lazy delete for linear probing. Add to LinearProbingHashST a delete() method that deletes a key-value pair by setting the value to null (but not removing the key) and later removing the pair from the table in resize() . Your primary challenge is to decide when to call resize() . Note : You should overwrite the null value if a subsequent put() operation associates a new value with the key. Make sure that your program takes into account the number of such tombstone items, as well as the number of empty positions, in making the decision whether to expand or contract the table.

3.4.6 - Red Suppose that keys are binary integers. For a modular hash function with prime $m > 2$, prove that any two binary integers that differ in exactly one bit have different hash values.

3.4.32 - Red Hash attack. Find 2^N strings, each of length 2^N, that have the same hashCode() value, supposing that the hashCode() implementation for String is the following:

```python
# Python
def hash_code(str):
  hash = 0;
  for i in range(len(str)):
    hash = (hash * 37) + ord(str[i])
  return hash;
```

```java
// Java
public int hashCode()
{
  int hash = 0;
  for (int i = 0; i < length(); i++)
    hash = (hash * 37) + charAt(i);
  return hash;
}
```

Strong hint: ef and fA have the same value.

3.4.33 - Red Bad hash function. Consider the following hashCode() implementation for String, which was used in early versions of Java:

```python
# Python
def hash_code(str):
  hash = 0;
  skip = max(1, len(str) // 8)
  for i in range(0, len(str), skip):
    hash = (hash * 31) + ord(str[i])
  return hash;
```

```java
// Java
public int hashCode()
{
  int hash = 0;
  int skip = Math.max(1, length()/8);
  for (int i=0; i < length(); i+=skip)
    hash = (hash * 31) + charAt(i);
  return hash;
}
```

Explain why you think the designers chose this implementation and then why you think it was abandoned in favor of the one in the previous exercise.