COS126 Symbol Table Exercises 4.4

(answers on reverse side)

- For each of the applications of symbol tables listed in the top group, if we want to store it in a variable that is a parameterization of the generic `ST` (symbol table) type, find a parameterization in the bottom group that will do the job.

  1. For each calendar year since 1748, the number of students who graduated from Princeton in that year
  2. Dictionary of all synonyms of all English words
  3. Count frequency of words in a book
  4. Store a list of people living at every postal address and track the age of each person there
  5. Track which way each voting senator voted (yes or no) on a motion
  6. Look up the state containing any given zip code*
  7. Symbol table of all local variables names and values in a virtual machine

  *: pretend zip codes are integers, although this is not done in practice because of leading zeroes and ZIP+4.

  a. `ST<String, Integer>`
  b. `ST<String, Boolean>`
  c. `ST<Integer, String>`
  d. `ST<Integer, Integer>`
  e. `ST<String, String[]>`
  f. `ST<String, ST<String, Integer>>`
  g. `ST<String, Object>`

- Can a `ST<Integer, Double>` be used to do anything that a `double[]` can do? Is the reverse true? Are there tradeoffs?

Answers:

- 1d

  2e

  3a

  4f

  5b

  6c

  7g (`Object` is the general Java superclass, which can refer to any non-primitive. Java itself doesn't truly use symbol tables in this way, but other "interpreted" languages do. We won't use `Object` in this course.)

- Anything that a `Double[]` can do, can also be done by a `ST<Integer, Double>`. For example, we'd use `st.put(i, 867.5309)` to simulate `stringArray[i] = 867.5309`. There are things that the symbol table can do which the array cannot: skip indices, use negative indices, and change the number of elements after creation.

  While the symbol table is more expressive, it is somewhat slower: it takes more time to look at or change an element, and the memory usage is higher per element.