Sample Assignment 7
Discussed on 2020-11-05,

Not graded

Question 1 (Polynomial HashCodes for Strings).
You have 4 different ASCII strings: `ABCD`, `ABD`, `ACD`, `BCD`, and you want to compute Java/Scala-style hash function and insert them (in this order) into a hash table H with 5 slots: $H[0], \ldots, H[4]$.
(A) Compute the (uncompressed) hashcode values for all the 4 string values. Your hashcode function $h_1(s) = p(31)$ (value of a polinomial for argument $z = 31$), where $p(z)$ is defined as follows:

$$p(z) = \sum_{i=0}^{L-1} \mathrm{ord}(c[i]) z^{L-1-i}.$$

Here L is the length of the string s. By $c[i]$ we denote the ith character of the input string s ($i = 0, 1, \ldots, L-1$). Polynomial is computed in a 4-byte integer register; if the integer register overflows, we only use the last four bytes. In this exercise all the strings are sufficiently short and the overflow does not happen.
Note. By $\mathrm{ord}(c)$ in this polynomial we denote the ASCII code of some character `c`; these codes are integers in $[0; 255]$. (See the ASCII codes: `http://www.asciitable.com/`.)

(B) Write the arithmetic expression to evaluate h_1(`ABCD`) with the *Horner's method* using only three multiplications and three additions (and no intermediate variables can be stored).

(C) Compute the compressed hash values for the same 4 strings (`ABCD`, `ABD`, `ACD`, `BCD`) modulo 5. Namely, the compressed hash value is

$$h_2(h_1(s)) = h_1(s) \bmod 5.$$

(D) Draw the four string objects in a hashtable H with 5 cells ($H[0], \ldots, H[4]$).

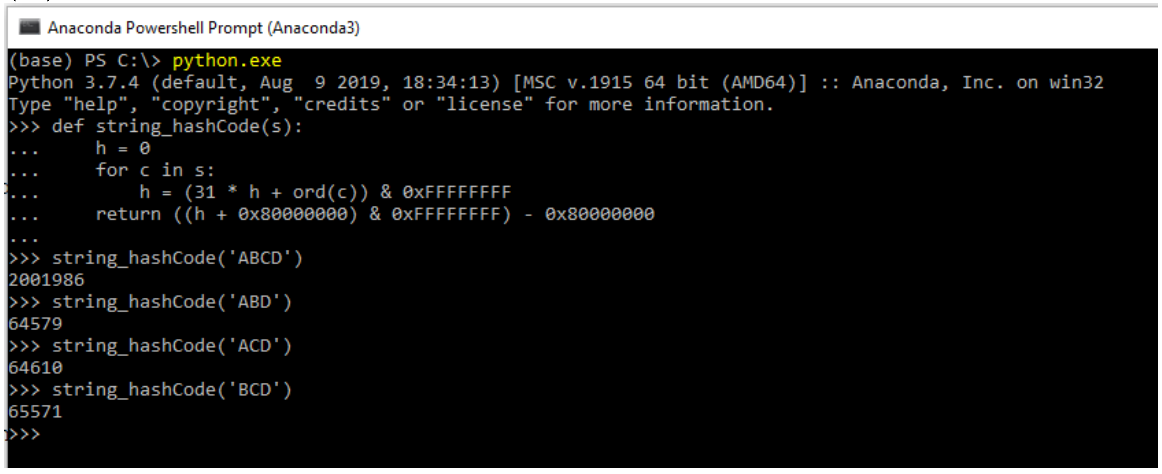
Note. Here is the pseudocode of the abovementioned string hashing function $h_1(s)$ (in Python):

```
def h1(s):
    h = 0
    for c in s:
        h = (31 * h + ord(c)) & 0xFFFFFFFF
    return ((h + 0x80000000) & 0xFFFFFFFF) - 0x80000000
```

All the manipulation with masks `0xFFFFFFFF` etc. is meant to guarantee that $h_1(s)$ is always 4 bytes long signed integer (in Python integers are generally longer). Your polynomial computations will not be affected by these masks (for short strings the value $h_1(s)$ is a positive number that does not cause 4-byte register overflow).

Question 1

(A) Compute the values by hand or use Python pseudocode:



```
Anaconda Powershell Prompt (Anaconda3)
(base) PS C:\> python.exe
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> def string_hashCode(s):
...     h = 0
...     for c in s:
...         h = (31 * h + ord(c)) & 0xFFFFFFFF
...     return ((h + 0x80000000) & 0xFFFFFFFF) - 0x80000000
...
>>> string_hashCode('ABCD')
2001986
>>> string_hashCode('ABD')
64579
>>> string_hashCode('ACD')
64610
>>> string_hashCode('BCD')
65571
>>>
```

(B) You can use Horner's method like this:

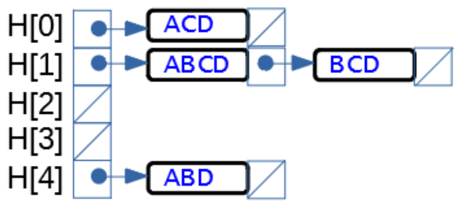
$$h_1(\texttt{ABCD}) = ((65 \cdot 31 + 66) \cdot 31 + 67) \cdot 31 + 68.$$

Notice that the characters `'A'`, `'B'`, `'C'`, `'D'` have ASCII codes $65, 66, 67, 68$ respectively. In hexadecimal notation, the byte `'A'` is `0x41`, which equals $4 \cdot 16 + 1 = 65$. See `http://www.asciitable.com/` for details.
Note. If Horner's method is not used, then a 3rd degree polynomial would take 3 additions and 6 multiplications. This is even more inefficient for longer strings, so we should always use the Horner's method (also used in the Python pseudocode on the previous page). Here is the same polynomial without Horner's method:

$$h_1(\texttt{ABCD}) = 65 \cdot 31 \cdot 31 \cdot 31 + 66 \cdot 31 \cdot 31 + 67 \cdot 31 + 68.$$

(C) The compressed hash values are the remainders of numbers 2001986, 64579, 64610 and 65571 when divided by 5 (i.e. they are numbers 1, 4, 0, 1 respectively).

(D) Here is the image with the hashtable (with all the 5 slots/buckets displayed). There is one collision: two different values map to the same (compressed) value 1. Hashtable is represented as an array of linked lists. The key `ABCD` is inserted first, only then the collision with another key `BCD` happens (so it becomes the next member in the linked list for the bucket $H[1]$).



Note. The hashfunction $h_1(s)$ is one of the simplest hash functions that is widely used.
See `https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#hashCode()`.