Programming Assignments: Hashing

Both assignments are related to fixed sized hash tables. The hash tables are used to store string (`str`) values. When inserting a new string data to the table, the hash value (slot) is calculated with the following hash function for strings:

```
procedure hash(data):
    sum = 0
    for i = 0 to N-1 do
        sum += ascii(data[i])
    return sum % X
```

where `N` is the length of the string (data), `%` is the symbol for the mod operation and `X` is a design specific constant of the hash table (see assignment instructions). The ascii value of a character can be calculated with the function `ord` in Python.

Assignment 4.1: Linear Probing (4 points)

Create a class `HashLinear` in Python which implements a fixed sized hash table that uses linear probing for collision resolution. The class must have following methods:

- `__init__(M: int)`: initializer, creates a hash table of size M
- `print()`: prints the content of the hash table
  - the data string in each slot separated with a space
  - display empty slots using character `'F'`
  - tombstone slots using character `'T'`
  - (see the example below)
- `insert(data: str)` (**2 pts**): inserts `data` into the hash table, ignores duplicates
- `delete(data: str)` (**2 pts**): removes `data` from the hash table

For hashing use $X = M$.

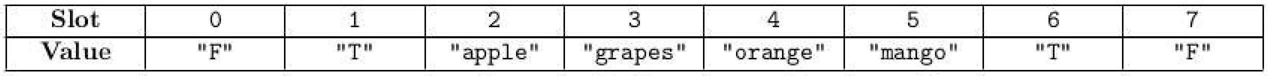
A code template with an example program (the hash table has the size of $M = 8$):

```
class HashLinear:
    # TODO



if __name__ == "__main__":
    table = HashLinear(8)
    table.print()

    table.insert("apple")
    table.insert("orange")
    table.insert("banana")
    table.insert("grapes")
    table.insert("mango")
    table.insert("peach")
    table.insert("apple")
    table.print()

    table.delete("banana")
    table.delete("kiwi")
    table.delete("peach")
    table.print()
```

The output:

```
$ python hashlinear.py
F F F F F F F F
F banana apple grapes orange mango peach F
F T apple grapes orange mango T F
```

Visualization of the final hash table from the example:



Slot	0	1	2	3	4	5	6	7
Value	"F"	"T"	"apple"	"grapes"	"orange"	"mango"	"T"	"F"

Submit your solution in CodeGrade as `hashlinear.py`.

Assignment 4.2: Bucket Hashing (4 points)

Create a class `HashBucket` in Python which implements a fixed sized hash table that uses bucket hashing with linear probing for collision resolution. The hash table also has an overflow array which stores the data which have not fit to the buckets during the insertion. The overflow array has the same capacity as the hash table. The class must have following methods:

- `__init__(M: int, B: int)`: initializer, creates a hash table of size M divided in B buckets
- `print()`: prints the content of the hash table and the overflow array
  - the data string in each slot followed by the data in the overflow bucket slots separated with a space
  - display empty slots using character `'F'`
  - tombstone slots using character `'T'`
  - (see the example below)
- `insert(data: str)` (**2 pts**): inserts `data` in the hash table, ignores duplicates
- `delete(data: str)` (**2 pts**): removes `data` from the hash table

For hashing use $X = B$. Finding the slot from the assigned bucket starts from the top/start. Overflow values are appended to the end of the overflow array (i.e. stack).

```
# hashbucket.py


class HashBucket:
    # TODO


if __name__ == "__main__":
    table = HashBucket(8, 4)
    table.print()

    table.insert("apple")
    table.insert("orange")
    table.insert("banana")
    table.insert("grapes")
    table.insert("mango")
    table.insert("peach")
    table.insert("apple")
    table.print()

    table.delete("banana")
    table.delete("kiwi")
    table.delete("peach")
    table.print()
```

Output:

```
$ python hashbucket.py
F F F F F F F F
orange F banana peach apple grapes F F mango
orange F T T apple grapes F F mango
```

Visualization of the final hash table from the example:

Bucket	0		1		2		3	
Value	"Orange"	"F"	"T"	"T"	"apple"	"grapes"	"F"	"F"

+

Overflow
"mango"

Submit your solution in CodeGrade as `hashbucket.py`.

Last modified: Tuesday, 25 June 2024, 1:48 PM