Data Structures and Algorithms

Assignment 04: Hash Table and Trie

Program templates for Programming Questions 1 to 3 are available on **Hackerearth**. You are required to use these templates to implement your functions.

You need to submit your code Hacerearth. Email invitation will be sent to your school account. Deadline for program submission: **April 27, 2025 (Sunday) 11.59 pm.**

Programming Question 1: Open Addressing of Double Hashing

Implement an open addressing hash table by double hashing to perform insertion and deletion. The structure of hash slots is given and a hash table with 37 hash slots is created in the main function. The hash functions are provided. The hash2() is the incremental hash function for double hashing, where H(key,i) = (hash1(key) + i*hash2(key)) mod TABLESIZE, i = 0, 1, 2,..., 36.

```
TABLESIZE = 37

PRIME = 13

EMPTY = 0

USED = 1

DELETED = 2

class HashSlot:

    def __init__(self):

        self.key = 0

        self.indicator = EMPTY

def hash1(key):

    return key % TABLESIZE

def hash2(key):

    return (key % PRIME) + 1
```

The insertion and deletion function prototypes are given as follows:

```
def hash_insert(key, hash_table):

def hash_delete(key, hash_table):
```

Both functions' return value is the number of key comparisons done during insertion and deletion respectively. Inserting a duplicate key will return -1. If the number of key comparisons is more than the table size, it implies that the table is full. For the deletion function, it will return -1 if the key to be deleted does not exist. The number of key comparisons in deletion cannot be more than table size.

Hint: When inserting a key, if a slot if found to be EMPTY or DELETED, the key cannot be inserted into the slot unless it is confirmed that the key is not in the table.

Programming Question 2: Coalesced Hashing

Coalesced hashing is a combination of closed addressing and linear probing, where H(key, i) = (H'(key)+i) % TABLESIZE, H'(key) = key % TABLESIZE, i = 0, 1, 2,..., TABLESIZE-1. Each slot not only stores the key, but also the link (index) to the next slot. The structure of hash slots is given below.

```
TABLESIZE = 37

PRIME = 13

EMPTY = 0

USED = 1

class HashSlot:

    def __init__(self):

        self.key = 0

        self.indicator = EMPTY

        self.next = -1
```

The insertion and searching function prototypes are given as follows:

```
def hash_insert(key, hash_table):

def hash_find(key, hash_table):
```

Both functions' return value is an index of the slot where the key is inserted and searched respectively. For insertion function, inserting a duplicate key will return -1. Return value larger than the table size implies that the table is full. For searching function, finding an non-existing key will return -1.

For example,

Hash(key) = key % 7
1) Insert 14
2) Insert 7
3) Insert 5
4) Insert 15
5) Insert 13
6) Insert 19

index	key	next
0	14	1
1	7	2
2	15	-1
3	19	-1
4		-1
5	5	3
6	13	-1

Programming Question 3: Lexicographical Range Query

Given a trie of lowercase English words. You will be asked several queries, each with two strings L and R. For each query, return the number of words in the trie that are lexicographically between L and R (inclusive). The function prototype is given as follows:

```
def count_in_range(trie, L, R):
```

For example, if the trie contains the words: cat, car, care, camera, dog, and L is car, R is cart. The number of words to be returned is 2 as car and care are lexicographically between "car" and "cart".

The TrieNode, Stack, and Query are given. You can write your own methods, but you may not use Python's built-in list.sort(), set, or dict in the implementation.