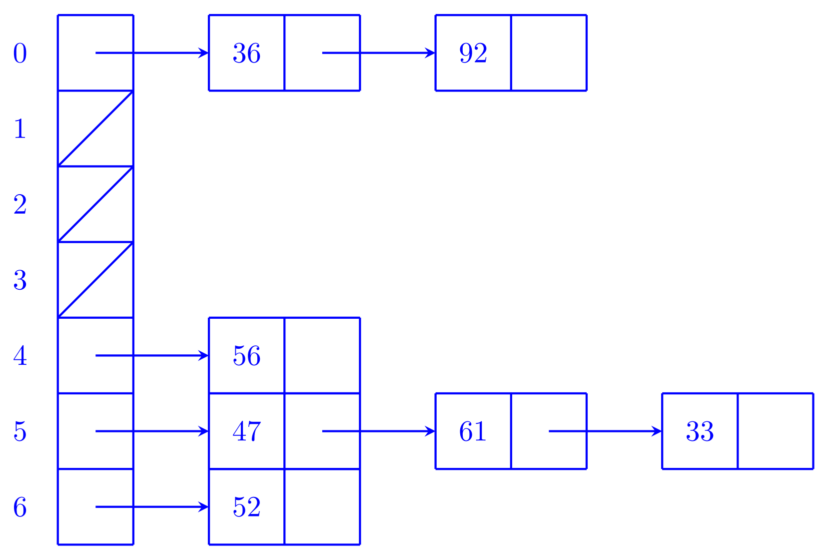
Problem Set 3

Name: Your Name

Collaborators: Name1, Name2

Problem 3-1.

(a) Known: $h(k) = (10k + 4) \bmod 7$
$h(47) = 5, h(61) = 5, h(36) = 0, h(52) = 6, h(56) = 4, h(33) = 5, h(92) = 0$



(b) Known: $h(k) = ((10k + 4)$ mod c) mod 7

```
def main():
    data = [47, 61, 36, 52, 56, 33, 92]
    'In python, set and dictionary are both based on hash table.'
    record = set()
    for c in range(1, 100):
        flag = True
        for a in data:
            x = ((10 * a + 4) % c) % 7
            if x in record:
                flag = False
                break
            record.add(x)
        if flag:
            print(c)
            break
        record.clear()

if __name__ == '__main__':
    main()
```

Answer is 13, then there is no collision.

Problem 3-2.

(a) Known: $H = \{h_{ab}(k) = (ak + b)\ mod\ n \mid a, b \in \{0, ..., n-1\}\ and\ a \neq 0\}$

$\exists k_1, k_2, h(k1) = h(k2) \Leftrightarrow ak_1+b \equiv ak_2+b(mod\ n) \Leftrightarrow a(k_1-k_2) \equiv 0(mod\ n), \forall a \in \mathbb{Z} \Leftrightarrow n|(k_1-k_2)$

(b) $n \mid (\lfloor \frac{k_1 n}{u})\rfloor - \lfloor \frac{k_2 n}{u} \rfloor$

But $k_1, k_2 < u \Rightarrow \lfloor \frac{k_{1(2)} n}{u} \rfloor < n$, we can deduce that the difference must be 0: $\lfloor \frac{k_1 n}{u} \rfloor = \lfloor \frac{k_2 n}{u} \rfloor$.

We can take $k_1 = 1, k_2 = 2$ because then $\frac{u}{k_{1(2)}} \gg n \Rightarrow \lfloor \frac{k_1 n}{u} \rfloor = \lfloor \frac{k_2 n}{u} \rfloor = 0$

(c) Seen the cour: the probability maximum is $\frac{1}{m}$

Problem 3-3.

(a) Use **radix sort**: a string is divided into multiple ASCII characters, which can be described by a number from 0 to 127. The length of string is fixed. So we can get the time: $\Theta(nlog_4 n)$. In a comparative model, each comparison of string will cost extra $\Theta(log_4 n)$ time, which leads to the time total: $O(nlog^2 n)$

(b) 1. $n \gg 800,000$, then $800,000$ is not a big number, so we can use **count sort**. The space cost will depend on n. The time cost $\Theta(n + 800,000) \simeq \Theta(n)$

2. Otherwise, we can use merge sort et etc.

(c) Multiply by n^3, then use radix sort: $\Theta(n)$

(d) This is a comparative model, where we can use merge sort: $\Theta(nlogn)$

Problem 3-4.

(a) Use a hash table to store $(i, r - b_i)$ pair.

(b)

Problem 3-5.

(a) Use hash table. For a string A, we extract a substring of which length is k. We sort it by using bucket sort: $\Theta(k + 26) \simeq \Theta(k)$. The result of sort is the key of hash table. The value is a index sort.

When we search the anagram substring count of B, we just ust bucket sort to sort B at first and then return the size of $h(key)$

(b) Use the method of problem above is fine.

```
def count_anagram_substrings(T, S):
    '''
    Input:  T | String
            S | Tuple of strings S_i of equal length k < |T|
    Output: A | Tuple of integers a_i:
              | the anagram substring count of S_i in T
    '''
    A = []
    ##################
    # YOUR CODE HERE #
    ##################
    k = len(S[0])
    record = {}
    for i in range(0, len(T) - k + 1):
        R = bucket_sort(T[i:i + k])
        if R not in record:
            record[R] = 1
        else:
```

```
                record[R] += 1
    for s in S:
        R = bucket_sort(s)
        if R in record:
            A.append(record[R])
        else:
            A.append(0)
    return tuple(A)

def bucket_sort(T):
    A = [0] * 26
    for t in T:
        A[ord(t) - 97] += 1
    S = ""
    for i in range(26):
        S += chr(i + 97) * A[i]
    return S
```

(c) Submit your implementation to `alg.mit.edu`.