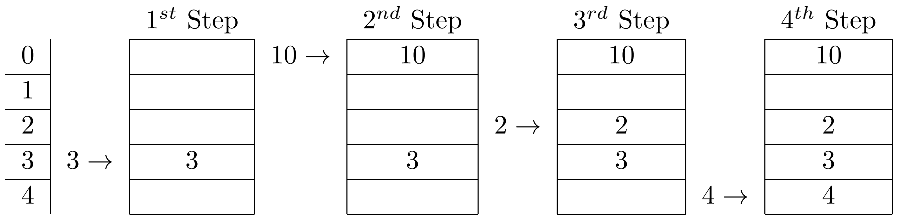
# Algorithms and Data Structures: Homework #10

Due on April 20, 2020 at 23:00

**Henri Sota**

## Problem 10.1

a) Given the sequence $< 3, 10, 2, 4 >$, apply the double-hashing strategy for open addressing to store the sequence in the given order in a hash table of size $m = 5$ with hash functions $h_1(k) = k \bmod 5$ and $h_2(k) = 7k \bmod 8$. Document all collisions and how they are resolved. Write down your computations.



**First Step**
First hash function is calculated with input 3: $h_1(3) = 3 \bmod 5 = 3$. Position 3 in the hash table is not occupied and therefore there is no need to use the second hash function to probe to another position. Place value 3 in position 3 of the hash table.

**Second Step**
First hash function is calculated with input 10: $h_1(10) = 10 \bmod 5 = 0$. Position 0 in the hash table is not occupied and therefore there is no need to use the second hash function to probe to another position. Place value 10 in position 0 of the hash table.

**Third Step**
First hash function is calculated with input 2: $h_1(2) = 2 \bmod 5 = 2$. Position 2 in the hash table is not occupied and therefore there is no need to use the second hash function to probe to another position. Place value 2 in position 2 of the hash table.

**Fourth Step**
First hash function is calculated with input 4: $h_1(4) = 4 \bmod 5 = 4$. Position 4 in the hash table is not occupied and therefore there is no need to use the second hash function to probe to another position. Place value 4 in position 4 of the hash table.

During insertion of these 4 values onto the hash table, there was no collision faced and therefore there was no need to use the second hash function to probe the hash table positions.

b) Implement a hash table that supports insertion and querying with open addressing using linear probing. Select an $h'$ function and explain why your selected $h'$ is well-suited for your test data. The implementation should be consistent with the following or equivalent class specifications:

```
class Node {
    public:
        int key;
        int value;
        Node(int key, int value);
}
class HashTable {
    private:
        Node **arr;
        int maxSize;
        int currentSize;
    public:
        HashTable();
        hashCode(int key);
        void insertNode(int key, int value);
        int get(int key);
        bool isEmpty();
}
```

Below you can find the definition of my HashTable class which implements the methods expressed in the homework sheet, a destructor and a helper method to print the Hash Table. The definition of this class is inside the C++ file "HashTable.cpp". Several tests against insert and get functionality are given in the file "testHashTable.cpp". My $h'$ function works using the division method: $key \bmod m$, where $m$ is the size of the Hash Table. The chosen $m$ is 25, because it is a number not too close to the powers of 2 and the type of keys we are storing are integers (in this case unique).

```cpp
class Node {
    public:
        int key;
        int value;
        Node(int key, int value);
};

class HashTable {
    private:
        Node **arr;
        int maxSize;
        int currentSize;
    public:
        // Constructor
        HashTable();

        // Destructor
        ~HashTable();

        // Hash Code method to calculate hash using division method
        int hashCode(int key);

        void insertNode(int key, int value);
        int get(int key);
        bool isEmpty();

        void printHashTable();
};
```

Listing 1: Hash Table Class Implementation Declaration in C++

## Problem 10.2

a) Show that a greedy algorithm for the activity-selection problem that makes the greedy choice of selecting the activity with shortest duration may fail at producing a globally optimal solution.

In order to show that the greedy choice of this algorithm may fail, an example of failure must be given. The goal of this algorithm is to choose intervals in such a way that it maximizes the number of intervals chosen, while they are non-conflicting or compatible with each other. In the following example, the greedy choice of selecting the activity with the shortest duration fails at producing a globally optimal solution.

The interval set $I$ is: $< (3, 8), (6, 10), (8, 15) >$ and $|I|$ is 3.

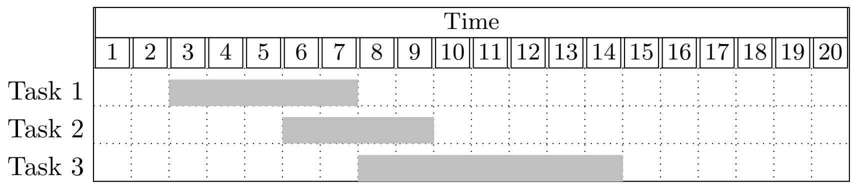


Figure 1: Activity Chart with respect to Time

The solution that the greedy choice of the shortest duration gives in this case is the set $S$: $< (6, 10) >$. The algorithm arrives at this solution as the interval with shortest duration is interval $< (6, 10) >$ and no other interval offers the possibility to be added to the solution set as no other interval is non-conflicting (start time is bigger or equal to finish time or finish time is smaller or equal to start time of interval $(6, 10)$). The optimal solution to the goal of the algorithm is to choose the set of intervals $S'$: $< (3, 8), (8, 15) >$ with optimal solution set cardinality 2.