Programming for Data Science (02/02/2024)

Upload the solutions to the programming exercices to following link:
https://evo.di.unipi.it/student/courses/16/exams/NVO5ZWv

Exercise 1. (Math, solve the exercise on paper) Consider the scenario of implementing a phone book. As an example, imagine you have to store:

"John Doe", 5551234
"Jane Smith", 5555678
"Alice Johnson", 5559876
…..

A. Present, with examples, what a hash table, a hash function, a collision are, in this scenario.
B. Describe a collision resolution technique.
C. What are the properties of the hash function “h(k)” ? Is it injective (aka ono-to-one)? Surjective (aka onto)? Invertible?
D. Use first order logic to formalize that a function f: N→ N is injective.

SOLUTIONS

A. A hash table is a data structure that stores key-value pairs. It uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found. In the context of a phone book, the keys would typically be names (e.g., "John Doe") and the values would be corresponding phone numbers (e.g., 5551234). A hash function takes an input (or 'key') and returns a fixed-size string of characters, typically a hash code. In the scenario of implementing a phone book, the hash function would take the name of a person and convert it into an index where the associated phone number can be stored/retrieved, For instance "John Doe" □ J. A collision occurs when two different keys hash to the same index in the hash table. For example, if both "John Doe" and "Jane Smith" hashed to the same index J, this would result in a collision.
B. One common collision resolution technique is chaining. In chaining, each bucket in the hash table contains a linked list of key-value pairs that hash to the same index. When a collision occurs, the new key-value pair is simply appended to the linked list at that index.
C. h(k) is not injective, for instance both "John Doe" and "Jane Smith" are mapped in J (hence the collisions); it is in general surjective, but this is not guaranteed, depends on the domain and on the function. It is not invertible (not being injective)
D. ∀x, y ∈ N, if f(x) = f(y), then x = y.

Exercise 2. (Python) A DNA string is a string built on an alphabet of 4 characters: A, T, G, and C. In bioinformatics a *k-mer* is a substring of *k* characters from a string that is longer than *k*. Write a function **KMer(s,k)** with two parameters: a DNA string and the value of k. Return a dictionary of *k-mer* counts. **BONUS**: store in the dictionary the list of positions from where each *k-mer* starts.

Example: **KMer(GTAGAGTAGT, 3)** → {“GTA”:2, “TAG”:2, “AGT”:2, “AGA”:1, “GAG”:1}
BONUS *Example*: **KMer(GTAGAGTAGT, 3)** → {“GTA”:[0,5], “TAG”:[1,6], “AGT”:[4,7], “AGA”:[2], “GAG”:[3]}

SOLUZIONE:

```
def KMer1(s, k):
   results = {}
   for i in range(len(s)-k+1):
       sub = s[i:i+k]
       results[sub] = results.get(sub, 0) + 1
   return results

def KMer2(s, k):
   results = {}
   for i in range(len(s)-k+1):
       sub = s[i:i+k]
       if sub not in results:
           results[sub] = [i]
       else:
           results[sub].append(i)
   return results

print(KMer1("GTAGAGTAGT", 3))
print(KMer2("GTAGAGTAGT", 3))
```

Exercise 3. (C) Write a C program that:

- Asks the user to input a number n greater than 50
- Allocate the memory space needed to store an array of n integers
- Fill the array with randomly generated values between -200 and 300
- Prints the content of the list
- Then, sorts the values in the list, in increasing order. No extra data structures can be used (i.e., do not store the values in an array to sort them) and do not use library functions. The sort should be implemented by your own.
- Prints the content of the sorted list

SOLUZIONE:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// function to swap the the position of two elements
void swap(int *a, int *b) {
 int temp = *a;
 *a = *b;
 *b = temp;
}

void printArray(int* array, int size) {
 for (int i=0; i<size; ++i) {
  printf("%d ", array[i]);
 }
 printf("\n");
}

void* selection_sort(int* array, int size) {
 for (int step=0; step<size-1; step++) {
  int min_idx = step;
  for (int i=step+1; i<size; i++) {
   // Select the minimum element in each loop.
   if (array[i] < array[min_idx])
    min_idx = i;
  }

  // put min at the correct position
  swap(&array[min_idx], &array[step]);
 }
}

int main() {
 int n;

 do {
  printf("Insert a number > 50: ");
  scanf("%d", &n);
 } while (n <= 50);

 int* array = (int*) malloc(n * sizeof(int));
 if (array == NULL) {
  printf("Memory allocation failed\n");
  exit(EXIT_FAILURE);
 }

 // Initialize random seed
 srand(time(NULL));
 for (int i=0; i<n; i++)
  array[i] = (rand() % 501) - 200;

 // Print the content of the array
 printf("Array Content: ");
 printArray(array, n);

 selection_sort(array, n);

 // Print the content of the sorted array
 printf("Sorted Content: ");
 printArray(array, n);

 // free the allocated memory
 free(array);
 return 0;
}
```