Compito di Programmazione I - Bioinformatica

26 settembre 2022 (tempo disponibile: 2 ore)

Esercizio 1 (14 punti)

(si consegni `bits.c`)

La *popolazione* di un numero naturale è la quantità di bit ad 1 nella sua rappresentazione binaria. Per esempio, il numero 18 ha popolazione due, poiché la sua rappresentazione binaria è 10010, che contiene due bit ad 1. Si completi il seguente file `bits.c`:

```
// AGGIUNGERE QUI GLI #include NECESSARI

// riempie arr, lungo length, con interi casuali tra 0 e 65535 inclusi
void init_random(int arr[], int length) { // COMPLETARE
}

// ritorna la popolazione di n, cioe' il numero di bit ad 1 nella sua
// rappresentazione binaria. Si assuma che n sia tra 0 e 65535 inclusi
int population(int n) {
  return 0; // MODIFICARE E COMPLETARE
}

// ordina arr in ordine non decrescente per popolazione
void sort_by_population(int arr[], int length) { // COMPLETARE
}
```

I file `bits.h` e `main_bits.c` sono già scritti e completi, non vanno modificati e non vanno consegnati. Se servisse, si possono aggiungere funzioni ausiliarie dentro `bits.c`.

Se tutto è corretto, un esempio di esecuzione di `main_bits.c` potrebbe essere:

```
Inserisci la lunghezza dell'array, non negativa: 11
40352 (popolazione 7)
26034 (popolazione 8)
11374 (popolazione 8)
51867 (popolazione 9)
36806 (popolazione 9)
52429 (popolazione 9)
34173 (popolazione 9)
47988 (popolazione 10)
32628 (popolazione 11)
44847 (popolazione 11)
63389 (popolazione 12)
```

Esercizio 2 (17 punti)

(si consegni `max_element.c`)

Si consideri una lista di elementi, ciascuno composto da una lettera (id) a da un valore (value). Si completino tutte le funzioni dichiarate ma non definite in `max_element.c`. Se tutto è corretto, la funzione `main()` (già scritta, da non modificare) produce il seguente output:

```
Max element A: 5
```

Nota: non è necessario liberare la memoria allocata.