# *Esercitazione di laboratorio n. 4*

**Esercizio n.1**: **Massimo Comun Divisore**
*Competenze: ricorsione matematica (Ricorsione e problem-solving: 2.2)*

Dati 2 interi positivi `a` e `b`, il loro massimo comun divisore (`gcd(a,b)`) si definisce ricorsivamente come:

$$gcd(a,b) = \begin{cases} 2 \cdot gcd(\frac{a}{2}, \frac{b}{2}) & \text{if a,b are even} \\ gcd(a, \frac{b}{2}) & \text{if a is odd, b is even} \\ gcd(\frac{a-b}{2}, b) & \text{if a, b are odd} \end{cases}$$

Si individui un'opportuna condizione di terminazione e si scriva una funzione ricorsiva `int gcd(int a, int b);` che realizzi la definizione di cui sopra. Si ricordi che nel calcolo del massimo comun divisore si assume che `a > b`. Per tener conto di tale condizione, se non soddisfatta, si scambiano `a` e `b`.

**Esercizio n.2**: **Elemento maggioritario**
*Competenze: ricorsione matematica (Ricorsione e problem-solving: 2.2)*

Sia dato un vettore `vet` di `N` naturali. Si definisce elemento maggioritario, se esiste, quel valore che ha numero di occorrenze > `N`/2.

Esempio: se `N`= 7 e `vet` contiene 3, 3, 9, 4, 3, 5, 3 l'elemento maggioritario è 3. Se `N`=8 e `vet` contiene 0, 1, 0, 2, 3, 4, 0, 5 non esiste elemento maggioritario.

Si scriva una funzione `maggioritario` che, dati `N` e `vet`, visualizzi l'elemento maggioritario se esiste, -1 se non esiste. Il prototipo sia:

```
int majority( int *a, int N);
```

Vincoli: complessità *O(nlogn)*, algoritmo in loco.

**Esercizio n.3**: **Valutazione di espressioni regolari**
I problemi di ricerca di stringe all'interno di testi e/o collezioni di stringhe (solitamente di dimensione maggiore rispetto alla stringa cercata) si basano raramente su un confronto esatto, ma molto spesso necessitano di rappresentare in modo compatto non una, ma un insieme di stringhe cercate, evitandone per quanto possibile l'enumerazione esplicita. Le ***espressioni regolari*** sono una notazione molto utilizzata per rappresentare (in modo compatto) insiemi di stringhe correlate tra loro (ad esempio aventi una parte comune).
Una ***espressione regolare*** (o regexp) è una sequenza di simboli (quindi una stringa) che identifica un insieme di stringhe. Si scriva una funzione in C in grado di individuare (cercare) eventuali occorrenze di una data regexp all'interno di una stringa di input.
La funzione sia caratterizzata dal seguente prototipo:

```
char *cercaRegexp(char *src, char *regexp);
```

dove :

- il parametro `src` rappresenta la stringa sorgente in cui cercare.
- il parametro `regexp` rappresenta l'espressione regolare da cercare.
- il valore di ritorno della funzione è un puntatore alla prima occorrenza di `regexp` in `src` (`NULL` se non trovata).

Ai fini dell'esercizio si consideri di valutare solamente stringhe composte da caratteri alfabetici. Si considerino inoltre solamente espressioni regolari composte da caratteri alfabetici e dai seguenti metacaratteri:

- . trova un singolo carattere (cioè qualunque carattere può corrispondere a un punto)
- [] trova un singolo carattere contenuto nelle parentesi (cioè uno qualsiasi dei caratteri tra parentesi va bene)
- [^ ] trova ogni singolo carattere non contenuto nelle parentesi (cioè tutti i caratteri tra parentesi non vanno bene)
- \a trova un carattere minuscolo
- \A trova un carattere maiuscolo

Esempi di espressioni regolari:

`.oto` corrisponde a ogni stringa di quattro caratteri terminante con "`oto`", es. "`voto`", "`noto`", "`foto`", ...
`[mn]oto` rappresenta solamente "`moto`" e "`noto`"
`[^f]oto` rappresenta tutte le stringhe terminanti in "`oto`" ad eccezione di "`foto`"
`\aoto` rappresenta ogni stringa di quattro caratteri (come "`voto`", "`noto`", "`foto`", ... ) iniziante per lettere minuscola e terminante in "`oto`"
`\Aoto` rappresenta ogni stringa di quattro caratteri (come "`Voto`", "`Noto`", "`Foto`", ...) iniziante per lettere maiuscola e terminante in "`oto`".

NOTA: i metacaratteri possono apparire in qualsiasi punto dell'espressione regolare. I casi qui sopra sono solo una minima parte a titolo di esempio. Sono quindi espressioni regolari valide: `A[^f]\anR.d`, `\A[aeiou]5t[123]`, ecc.

**Esercizio n.4**: **Azienda di trasporti - ordinamento**
Si consideri lo scenario introdotto nell'esercizio 2 del Laboratorio 2. Si realizzi un programma in C che, una volta acquisite le informazioni in una opportuna struttura dati, renda disponibili le seguenti operazioni:

- stampa, a scelta se a video o su file, dei contenuti del log
- ordinamento del vettore per data, e a parità di date per ora
- ordinamento del vettore per codice di tratta
- ordinamento del vettore per stazione di partenza
- ordinamento del vettore per stazione di arrivo
- ricerca di una tratta per stazione di partenza (anche parziale).

Per quanto riguarda le ricerche, si richiede che siano implementate sia una funzione di ricerca dicotomica sia una funzione di ricerca lineare. Per quanto riguarda l'ordinamento, si presti attenzione alla stabilità dell'algoritmo prescelto nel caso di ordinamento per più chiavi successive. Si selezioni l'algoritmo di ricerca più opportuno: se la base dei dati è ordinata secondo la chiave di ricerca corrente si usi la ricerca dicotomica, altrimenti quella lineare. Si suggerisce a tal proposito di

mantenere nel programma uno stato relativo all'ordinamento corrente della base dati (ossia, su quale chiave sia attualmente ordinato).

**Esercizio n.5**: **Azienda di trasporti - multiordinamento**

A partire dalle specifiche dell'esercizio precedente, estendere le funzionalità del programma per mantenere in contemporanea più ordinamenti della base dati

Suggerimento: si legga il vettore originale una sola volta, mantenendolo nell'esatto ordine di lettura per tutta la durata dell'esecuzione. Si affianchino al vettore originale tanti vettori di puntatori a struttura quanti sono gli ordinamenti richiesti, con i quali sono gestiti gli ordinamenti stessi.

**Valutazione: gli esercizi 2, 3 e 5 saranno oggetto di valutazione**
**Scadenza: caricamento di quanto valutato: entro le 23:59 del 26/11/2019.**