Esame di Laboratorio di Fisica Computazionale

3 maggio 2016, ore 9.30

shell scripting

Si scriva uno script che riceve due argomenti in input da linea di comando.
`./script.sh arg1 arg2`
Il primo argomento rappresenta una stringa che vogliamo cercare.
Il secondo argomento rappresenta un insieme di files all'interno dei quali cerchiamo la stringa rappresentata dal primo argomento.
Lo script stampa sullo schermo ciascuna riga in cui compare la stringa cercata, ma non il resto del file.

1. Soluzione banale: si scriva uno script che utilizza `grep` per risolvere il problema.
2. Si aggiunga un controllo che verifichi la presenza dei due argomenti.
3. Si scriva uno script che utilizza `sed` per risolvere il problema. *suggerimento: la fine di una riga può essere rappresentata da un $*

Mathematica

1. Si risolva il sistema di equazioni differenziali

$$\begin{cases} x'(t) &= 3\,x(t) - y(t) \\ y'(t) &= x(t) + y(t) \\ x(0) &= a, \quad y(0) = b \end{cases}$$

   e si definisca $f(a,b) = (x(t), y(t))$ con le funzioni soluzioni del problema.
   Si disegni $f(1,-1)$ facendo variare $t \in [0,1]$.

2. Sia $\gamma(t) = (x(t), y(t))$ la soluzione del sistema di equazioni differenziali

$$\begin{cases} \dot{x}(t) &= 2x(t) - 1 \\ \dot{y}(t) &= x(t)^2 - y(t) \end{cases} \quad , \quad t \in [0,1]$$

   e sia $\vec{F}(x,y) = (y - x^2, 2x - 1)$ un campo vettoriale.
   Si calcoli, lungo la curva $\gamma(t)$, l'integrale $\int_\gamma \vec{F} \cdot dP = \int_0^1 dt\, \vec{F}(x(t), y(t)) \cdot \dot{\gamma}(t) dt$

3. Ortonormalizzazione di Gram-Schmidt.
   Si definisca una funzione che, data una lista $\{\mathbf{v}_i\}$ di n vettori a n componenti, costruisce una base di vettori $\{\mathbf{u}_i\}$, ortogonali rispetto al prodotto scalare ordinario, secondo il metodo di Gram-Schmidt:

$$\begin{aligned} \mathbf{u}_1 &= \mathbf{v}_1 \\ \mathbf{u}_j &= \mathbf{v}_j - \sum_{k=2}^{j} \frac{\mathbf{v}_j \cdot \mathbf{u}_{k-1}}{\mathbf{u}_{k-1} \cdot \mathbf{u}_{k-1}} \mathbf{u}_{k-1} \qquad j = 2, ..., n \end{aligned}$$

   Infine, si normalizzi a 1 ciascun vettore $\mathbf{e}_i = \mathbf{u}_i / ||\mathbf{u}_i||$.

4. Scomposizione QR.
   Una matrice reale quadrata A può essere scomposta nel prodotto di una matrice ortogonale Q per una matrice triangolare superiore R, $A = QR$. Si scrivano delle funzioni di `Mathematica` che permettano di calcolare per una generica matrice $n \times n$ la sua scomposizione nel prodotto QR secondo il seguente procedimento.
   Si considerino i vettori colonna della matrice $A = [\mathbf{a}_1, ..., \mathbf{a}_n]$.
   Con la procedura di Gram-Schmidt si determini una base ortogonale $\mathbf{a}_i \to \mathbf{u}_i$ e quindi si ricavino i versori normalizzati a 1: $\mathbf{u}_i \to \mathbf{e}_i = \mathbf{u}_i / ||\mathbf{u}_i||$.
   Si costruiscano la matrice $Q = [\mathbf{e}_1, ..., \mathbf{e}_n]$ e la matrice R

$$R = \begin{pmatrix} \mathbf{e}_1 \cdot \mathbf{a}_1 & \mathbf{e}_1 \cdot \mathbf{a}_2 & \mathbf{e}_1 \cdot \mathbf{a}_3 & ... \\ 0 & \mathbf{e}_2 \cdot \mathbf{a}_2 & \mathbf{e}_2 \cdot \mathbf{a}_3 & ... \\ 0 & 0 & \mathbf{e}_3 \cdot \mathbf{a}_3 & ... \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Si consideri il caso

$$A = \begin{pmatrix} 24 & -102 & 8 \\ 12 & 334 & -136 \\ -8 & 48 & -82 \end{pmatrix},$$

si calcolino esplcitamente le matrici Q e R e si verifichi che effettivamente $A = QR$.

5. Congettura di Collatz. Viene generata una successione di numeri interi, secondo una regola indicata di seguito, e l'algoritmo si arresta quando il nuovo numero è uguale a 1. La congettura ipotizza che la successione si arresti, qualunque sia l'intero n utilizzato come punto di partenza.

$$\begin{cases} a_0 &= n \\ a_i &= f(a_{i-1}) \end{cases}$$

dove

$$f(n) = \begin{cases} n/2 & n \text{ pari} \\ 3n+1 & n \text{ dispari} \end{cases}$$

   (a) Si implementi una funzione che calcola la successione degli a_i, avendo la possibilità di scegliere il valore di a_0.

   (b) Si implementi una funzione `collatz(n)` che calcola quanti elementi ci sono nella successione nel momento in cui l'algoritmo si arresta (ovvero il valore di i del primo elemento della successione tale che $a_i = 1$).
   suggerimento: si utilizzino, nell'ordine, i comandi `Return[value]` e `Break[]` in un ciclo `Do`.

   (c) Si utilizzi la funzione del punto precedente per generare una lista il cui elemento generico è la coppia $\{n, \text{collatz}[n]\}$, con n compreso tra 1 e 10000. Si disegnino questi punti.

C++

Si risolva l'esercizio proposto. Per facilitare la correzione, se possibile includere tutto in un unico file sorgente. La sufficienza è raggiunta risolvendo correttamente i primi quattro punti.

Cammini su reticolo quadrato

1. Si scriva una classe `Pos2D` che rappresenterà un vettore (equivalentemente, un sito) in un reticolo quadrato. Tra i membri public si mettano le due coordinate intere. Si scriva un opportuno costruttore che le assegni (senza valori di default) e il costruttore di copie.

2. Si scrivano gli overloading degli operatori `+` e `==`, che dovranno realizzare rispettivamente la somma e il confronto tra due vettori.

3. Si scriva una classe `Walk` che rappresenterà un cammino. Tra i membri protected si metta la posizione corrente del cammino sul reticolo. Tra i membri public si scriva un costruttore che richieda come parametro la posizione iniziale e la assegni. Si specifichi come valore di default la posizione $(0, 0)$.

4. Si scriva una access function `position` (membro pubblico di `Walk`) che restituisca la posizione corrente.

5. Si implementi l'overloading dell'operatore `+=` (anch'esso membro di `Walk`) in modo che la sintassi `w +=` δ incrementi la posizione corrente del cammino `w` di un passo δ (vettore sul reticolo). Per risolvere i punti successivi, questo operatore dovrà avere comportamento polimorfico: lo si dichiari in modo tale che questo avvenga.

6. Nel `main` si istanzi un oggetto di tipo `Walk` con posizione iniziale $(0, 0)$. Gli si vuole far compiere il cammino identificato dai seguenti vettori spostamento

$$\delta_i = (\lfloor 2\cos(i/5) + 1/2 \rfloor, \lfloor 2\sin(i/5) + 1/2 \rfloor)$$

   per $i = 0, 1, 2, \ldots$ (il simbolo $\lfloor \cdot \rfloor$ indica la parte intera, realizzata in c++ dalla funzione `floor`). Si trovi dopo quanti passi (cioè a quale valore di i) il cammino si trova nella posizione $(-5, 17)$.

7. Si scriva una classe `AvoidingWalk`, che erediti pubblicamente da `Walk`. Questa implementerà il vincolo di non passare su siti già visitati. A questo scopo, si metta tra i membri private un `std::set` che conterrà i siti già visitati. (Attenzione: per poter usare questo container associativo, sarà necessario implementare un'ulteriore caratteristica di `Pos2D`.)

8. Commentare brevemente sul motivo per cui un container di questo tipo sia preferibile ad un array sequenziale (come `std::vector`) in questo contesto. Si scriva inoltre il costruttore di `AvoidingWalk`.

9. Si scriva l'operatore `+=` (override di quello ereditato da `Walk`) in modo che realizzi il passo solo se questo fa finire il cammino su un sito non ancora visitato (in caso contrario la posizione rimane invariata). Si usino i due seguenti membri di `std::set`:

   - la funzione `insert`, che inserisce un elemento;
   - la funzione `find`, che cerca un elemento e restituisce un iteratore ad esso, se lo trova, oppure un iteratore a `end()` se non lo trova.

   (Facoltativo: per migliorare la performance, si usi la versione di `insert` che vuole come primo argomento un iteratore alla posizione dove ci si aspetta che verrà inserito l'elemento.)

10. Si verifichi che lo stesso codice scritto nel main al punto 6 funziona per un `AvoidingWalk`, e si misuri, anche per questo tipo di cammino, il numero di passi per raggiungere il punto $(-5, 17)$.

11. Si scriva una classe `WeaklyAvoidingWalk`, che erediti in modo pubblico da `AvoidingWalk`, che implementerà un vincolo meno stringente. In particolare, ad ogni passo verrà scelta una mossa vincolata (quella di `AvoidingWalk`, che ricorda ed evita i siti visitati) o una mossa non vincolata (quella di `Walk`, che non ricorda i siti visitati) con probabilità rispettivamente p e $1 - p$. Si metta questa probabilità p tra i membri privati di `WeaklyAvoidingWalk` e si scriva il costruttore che la inizializza. Si scriva quindi l'override dell'operatore `+=` in modo che richiami il codice già scritto per le due classi base.

    [Può essere utile usare `drand48()` come generatore di numeri pseudo-casuali compresi tra 0 e 1.]

12. Scrivere una funzione globale `compare`, che prenda come argomenti due oggetti di tipo `Walk` (si valuti se passarli per copia o per referenza, considerando il punto successivo) e che faccia eseguire ai due cammini 100 passi il cui vettore δ abbia le due componenti scelte a caso (indipendentemente) tra il valore 1 e il valore -1. I passi fatti dai due dovranno essere esattamente gli stessi e nello stesso ordine. Il valore di ritorno della funzione indicherà se i due cammini hanno posizione finale identica oppure no.

13. Nel `main`, eseguire 10 000 confronti, usando la funzione `compare`, tra un `Walk` e un `WeaklyAvoidingWalk` con parametro $p = 0.087$, e verificare che arrivano nello stesso punto in circa metà dei casi. (Entrambi partono dall'origine ogni volta.)