Rossi Federico

Esercizio 1

Il primo esercizio ha creato problemi con il versionamento di python su docker, quindi ho provato con una versione apposta per i container ed ha funzionato, stesso codice, cambia solo versione di python e i comandi nel file docker.

Il grafico funziona, ci gli input file generano un errore non fatale se non hanno niente in input, non implica nessun blocco, ovviamente se non sono selezionate due categorie/colonne diverse il grafico non rappresenta dei dati conformi.

I dati dal sito indicato vengono scaricati e messi nella variabile “data”, e poi richiamati quando vengono selezionati gli input.

Il file Dokerfile indica la versione di python, recupera i nomi delle dipendenze per scaricare le librerie necessarie dal file requirements.txt, lancia i download, e imposta le variabili per l’esecuzione.

In commands.txt ci sono i comandi per creare il container su docker, avviarlo, creare l’immagine e caricarlo su dockerhub.

Reso pubblico il file su dockerhub, creato account azure, provato “web app” e “web app for container” per hostare l’app web, impostato il docker di riferimento, nessuna impostazione di sicurezza. Azure non sono riuscito a farlo funzionare, ritorna errore che manca le risorse.

Esercizio 2

Niente da segnalare, moltiplicazione di due funzioni periodiche oscillatorie che differiscono di 1:20 come velocità di oscillazione per fare l’effetto indicato, più una costante per inclinare (“crescente”) la funzione.

Esercizio 3

Dal momento che Mario toglie una caramella tutte le volte prima di darle agli altri, per il numero di caramelle in ogni scatola devo calcolare il minimo comune multiplo dei numeri degli amici e aggiungerci la caramella che toglie sempre
Per calcolare il minimo comune multiplo, scompongo in fattori primi.

4 = 2^2
5 = 5
6 = 3*2
7 = 7

Mcm = 2^2 * 5 * 3 * 7 = 420
Mario esce di casa con 420 + 1 caramelle * scatola = 1684

Matematicamente non c'è un massimo, ogni multiplo di 420 + 1 è un numero di caramelle possibile, ovvero 1680 * x +4 con x = 1, 2, 3, ... infinito.

Fisicamente ed eticamente mi chiedo:

- Quanto pesano le caramelle, e quanta forza ha Mario per trasportarle?
- Che utilizzo ne fa di tutte queste caramelle?

Esercizio 4

Considerando il problema come descritto, avrei 146 guasti registrati.

Considero sempre che ci saranno, un intervallo valido per ogni dato per decidere se simile/equivalente o no, e una certa tolleranza sul confronto di tutto il set riguardante quanti dati sono simili (e se ce ne sono di più rilevanti) rispetto ad un altro.

Soluzione consigliata al sottotitolo SOLUZIONE EFFICENTE, anche se non rispetta perfettamente le condizioni

SOLUZIONE SEMPLICE

Per ogni t registrato andrei ad aggiungere un valore che useremo come priorità => t * (n+p).

Impostandolo ad esempio su 100% per le circa 2 ore prima dei fallimenti, e scendere fino allo 0% delle seguenti due ore, in modo tale da verificare prima i valori critici e vicino ai fallimenti.

In base alla capacità di calcolo della macchina prendo un set di dati in tempo reale (ad esempio 5 minuti) e lo confronterei con i dati precedentemente registrati, se trovo delle serie di dati simili cerco di capire in che stato si trova la macchina.

```
oldData = new dataInterval(365*(n+p));

timeSegment = valore calibrato; // tempo preso come intervallo per testare i nuovi dati

while (!allarm) {

newData = new dataInterval(timeSegment);

timer = new timer();

oldDataT = oldData.tocken(timeSegment)
```

```
while (timer < timeSegment) {
if (similar(newData,oldDataT) { allarm.setAllarm; }
else { oldDataT.next; } // i pacchetti sono ordinati per priorità
}
}
 similar(newData, oldDataT) {
confronto di ogni newData[x] con oldDataT[y] con 0 <= x,y < timeSegment
        se sono simili e il dato è vicino al margine scelto ritorna per settare l'allarme,
        altrimenti continua a cercare e poi restituisce negativo
}
```

Ovviamente non è assolutamente efficente, e il problema principale è che non considera che ci possono essere delle porzioni di set simili, e analizza più volte delle ennuple inutilmente.

SOLUZIONE EFFICENTE

Questa soluzione non rispetta la condizione richiesta "with a predictive span of 1h", ma ci sono delle considerazioni da fare a riguardo.

Il processo produttivo che restituisce n parametri al secondo e non sono una scienza esatta, ovvero ci sono dei valori che, come detto prima, possono essere più o meno rilevanti, se dovessimo tracciare un grafico per ogni parametro saranno principalmente onde; quindi, non è possibile dare con esattezza il momento del fallimento, ma solo una stima che è costruita in base alle similitudini dei dati registrati, ovvero dal singolo parametro che oscilla più o meno veloce, fino alla serie di dati in un certo intervallo; quindi, anche la predicibilità del fallimento è composta dalla probabilità che ogni piccolo parametro segua una certa curva e un tempo probabilistico di evoluzione/modifica di questi dati, di conseguenza il tempo mancante all'arresto.

Il ragionamento consiste nel creare un grafo direzionato con tutti i dati registrati in precedenza.

In ogni nodo avrò gli n dati registrati in un singolo momento.
[scelta: per ogni n, posso unire alcuni valori? ad esempio range di temperature, counter troppo alti, oppure anche a coppie o più, o con delle medie; tutto in base ai tipi di dati]

Ad ogni nodo assegno anche due probabilità (di confrontarlo con i dati in tempo reale, che saranno unite per calcolare la probabilità di confronto del nodo):

- la prima probabilità è settata più alta per i nodi che hanno vicino un nodo terminatore (di fallimento) e più bassa per quelli lontani (è solo concettuale, si può utilizzare il numero di passi per arrivare al nodo terminatore).
  [scelta: quanto vogliamo confrontarci con i dati vicini ai fallimenti e non con quelli più distanti?]
- la seconda probabilità inizializzata a "neutro" verrà settata durante l'esecuzione (può contenere valori che alzano e abbassano la probabilità precedente).

Considerando il peso dell'operazione di confronto e alle risorse disponibili per effettuarli (computer potenti, con tanti thread), per ogni dato ricevuto al secondo del processo in esecuzione effettuo i confronti con dei nodi scelti sparsi su tutto il grafo in base alla probabilità che li distingue.

In base all'esito del confronto, imposto il secondo parametro probabilistico di conseguenza:

- abbassandolo se non è positivo
- aumentandolo se sono simili *[potrebbe essere controproducente aumentare molto la probabilità, rischio di andare in loop su una zona o di cercarci troppo e perdere tempo, quindi da valutare l'aumento di probabilità, magari con un limite o con un piccolissimo incremento, non posso toglierla perché mi serve cercare nelle zone più "calde"]*.

Considerando che i nodi del grafo nelle vicinanze sono simili, per ottimizzare il processo, modifico anche il secondo valore dei nodi vicini *[scelta in base alla dimensione del grafo e alla differenza dei due nodi]*, più mi allontano più che la modifica perde di valenza.

Più che il tempo avanza più che i valori del secondo parametro torneranno verso il neutro *[scelta: ogni quanto tempo?]*

Così facendo riesco a trovare in tempi brevi la zona del grafo che più assomiglia alla mia situazione reale e posso dare una stima del tempo che rimane e con quanta probabilità all'arresto.

Per rendere il grafo più efficiente posso continuare a costruirlo con i nuovi dati, ma solo dopo il fallimento, fino a quel momento sarebbero dati superflui.

Considerazione sulle dimensioni del grafo, avendo i fallimenti dopo circa 48/72 ore, posso evitare di mettere i dati troppo lontani dai fallimenti per ottimizzare tempo e spazio, però come scelta progettuale gli inserirei e metterei la probabilità di testarli estremamente bassa.

Pseudocodice

Considerando un computer mutithread posso dare ad ogni thread un segmento ti tempo con n valori, quando arriva un nuovo valore, il thread con il valore più vecchio lo prende in carico scartando quello vecchio.

```
oldNewNodeTime = 0;
Thread().run() {
	while () {
		if ( !pendingNewNode && newNode.time > oldNewNodeTime) {
			oldNode = getNode(); // graph node
			similarity = compare(newNode, oldNode);
			if (similarity > valoreSettato) { //caso di nodo somigliante
				allarm(newNode, similarity);
			}
			setNearbyNode(similarity, newNode);
		} else {
			oldNewNodeTime = newNode.time + 1;
			newNode = getNewNode(); // real time node;
		}
	}
}

Compare(nodeA,nodeB) {
	return %ofSimilarity; // if nodeA == nodeB return 1; if nodeA == -nodeB return -1
}

setNearbyNode(similarity, node) {
	for (node.parent + node.child ) {
		similarity = similarity * exFactor; // fattore di espanzione della probabilità2
		if ( |similarity| > minModifierSimilarity &&
			similarity < node.probability2 { // evita il nodo di  provenienza
			setProbability2(similarity);
			setNearbyNode(similarity);
		}
	}
}
```

```
allarm(node, similarity) {
        calcola x; // probabilità di finire in un fallimento
        calcola t; // tempo per finire ad un noto terminatore, si può contare i nodi come
                        //stima, considerare la possibile compressione dei dati
        setAllarm("possibile spengimento fra { t /60 } minuti, probabilità { x }%");
}
```