Compito di Programmazione 18/6/2021

La biblioteca di Fantaland vuole gestire i prestiti dei suo libri. Ogni libro ha un identificativo: `book_id`, un insieme di autori (vettore di stringhe): `authors`, il titolo: `title`, anno di pubblicazione: `year`, posizione nella libreria (stringa): `position`, valore del libro: `value`. I libri vengono memorizzati in un vettore in ordine di autori, a parità di autori si ordina in base all'anno. Un prestito contiene l'indicazione del libro prestato, la data del prestito e la data di restituzione (per definizione 30 giorni a partire dalla data del prestito). I prestiti vengono memorizzati in una lista(mantenuta in maniera ordinata per data di restituzione).

- Implementare le strutture dati per rappresentare le informazioni descritte e le relative funzioni membro
- Implementare la funzione `borrow` che prende in ingresso un libro e la data del prestito e aggiunge un nodo alla lista dei prestiti
- Implementare la funzione `valueExpired` della lista dei prestiti prendendo in ingresso la data odierna e la lista per calcolare il valore complessivo dei libri la cui data di restituzione e' scaduta
- Implementare un programma driver che crea il vettore di libri e utilizza le funzioni implementate.
- **Opzionale**: implementare la funzione `returnBook` che cerca il libro dalla lista dei prestiti, se presente lo rimuove e calcola l'eventuale multa come 1$ per ogni giorno che eccede la data di restituzione

Nota: consegnare uno zip file che contiene una cartella *CognomeNome* in cui si trovano il codice sorgente e il makefile per la compilazione. **Il codice deve essere opportunamente commentato**.

Compito di Programmazione 16/7/2021
Tempo a disposizione 3 ore

La città di BlueSky vuole monitorare i livelli di CO2 dell'aria della città. La città e' divisa in quartieri secondo una matrice 20x20, ogni elemento della matrice corrisponde ad un quartiere della città. I rilevatori di CO2, mantenuti in un vettore, sono distribuiti in diversi quartieri, le informazioni di ciascun rivelatore sono: id_station (intero), coord_x (intero), coord_y(intero) e una lista di rilevamenti mantenuta in ordine di data di rilevamento. Ogni rilevamento contiene le informazioni sulla data del rilevamento, e la quantità di CO2 in parti per milione - ppm (double).

Implementare le strutture dati per gestire le informazioni. Le classi dei rilevatori devono contenere il costruttore copia, l'operatore di assegnazione e il distruttore.
Implementare le seguenti funzioni:

- `addMeasurement` che prende in ingresso l'id del rilevatore, la data, e il livello di ppm e aggiunge questo rilevamento alla lista dei rilevamenti di quella stazione
- `meanCO2Zone` che prende in ingresso due date, d1 e d2, e le coordinate di una zona. Una zona e' un insieme rettangolare di quartieri definiti dalle coordinate del quartiere in alto a sinistra e quello in basso a destra della zona. Questa funzione calcola il valore medio di ppm per tutte le stazioni di quella zona fra le date d1 e d2.
- Un programma driver di esempio che istanzia gli oggetti e invoca le due funzioni.

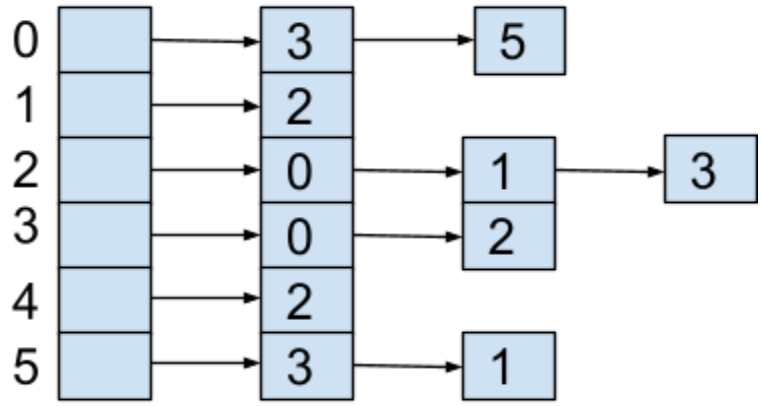
Consegnare una cartella in formato compresso del tipo `CognomeNome.zip` (prima il cognome e poi il nome!). Il file zip contiene una cartella `CognomeNome` con il codice sorgente ed il relativo Makefile. Tutto il codice deve essere opportunamente documentato. **Non consegnare programmi che non compilano correttamente**.

Compito di Programmazione 17/9/2021
Tempo a disposizione 3 ore

La start-up di nome *Elgoog* vuole lanciare sul mercato un nuovo motore di ricerca basato su un fattore di peso (*rank*) da dare a ciascuna pagina in base alla sua importanza nella rete. Per ogni pagina si memorizzano:

- Un ID della pagina (potrebbe essere un intero fra 0 e N-1, dove N e' il numero di pagine del web)
- Il numero di link contenuti in questa pagina (i link uscenti alle altre pagine del web). Per la pagina i indichiamo questo valore con $L(i)$
- Un valore di peso P che rappresenta l'importanza della pagina. Indichiamo questo valore per la pagina i con $P(i)$. All'inizio tutte le pagine hanno lo stesso rank pari a $1/N$.

L'insieme delle pagine sono mantenute in un vettore. Un *Web* è un vettore di N liste in cui nella lista *i*-ma ci sono gli ID delle pagine linkate da questa pagina. Ad esempio la pagina 0 contiene due link uno alla pagina 3 e uno alla pagina 5, la pagina 1 contiene un link alla pagina 2, etc.:



Implementare le strutture dati per rappresentare l'insieme delle pagine e il Web.

Implementare la funzione **`rankUpdate`** che aggiorna i pesi delle pagine in maniera tale che ad ogni iterazione ciascuna pagina trasferisce parte del suo rank in maniera uniforme a tutte le pagine citate nella pagina (i link uscenti da questa pagina). Quindi il rango di ogni pagina viene aggiornato nel seguente modo:

$$P(i) = \frac{(1-\delta)}{N} + \delta \sum_{j \in M(i)} \frac{P(j)}{L(j)}$$

Dove $M(i)$ sono tutte le pagine del Web che contengono un link alla pagina i, e δ è un valore costante tra 0 e 1. Nota i link da una pagina a sè stessa non vanno considerati. Per l'esempio in figura il rank della pagina 2 viene aggiornato come:

$$P(2) = \frac{1-\delta}{6} + \delta \cdot \left(\frac{P(1)}{1} + \frac{P(3)}{2} + \frac{P(4)}{1}\right)$$

Suggerimento importante: non e' necessario che per aggiornare una certa pagina i si debba calcolare esplicitamente l'insieme delle pagine che contengono il link alla pagina i.
Il vostro programma inizializza un Web e invoca iterativamente la funzione **`rankUpdate`** fino a convergenza. Visualizzare il rango di ogni pagina prima e dopo l'update.

Facoltativo 1. Invocate **`rankUpdate`** iterativamente fino alla convergenza. Per verificare la convergenza potere calcolare la media delle differenze in valore assoluto dei valori di P fra una iterazione e la successiva. Visualizzare la differenza sullo schermo e fermare le iterazioni quando la differenza e' minore di un certo valore (es. 0.01).

Facoltativo 2. Implementare una funzione **`initWeb`** che istanzia le strutture del Web in maniera random.

Consegnare una cartella in formato compresso del tipo `CognomeNome.zip` (prima il cognome e poi il nome senza spazi!). Il file zip contiene una cartella `CognomeNome` con il codice sorgente ed il relativo Makefile. Tutto il codice deve essere opportunamente documentato. **Non consegnare programmi che non compilano correttamente**.

Complimenti! Avete implementato una versione semplificata dell'algoritmo *PageRank* di Google.

Compito di Programmazione 15/10/2021
Tempo a disposizione 3 ore

Vogliamo rappresentare le informazioni sugli insegnamenti che ciascuno studente ha superato e calcolare quanti studenti hanno superato un certo numero di crediti.
Le informazioni minime per un insegnamento sono il nome_insegnamento (stringa), il numero di crediti n_crediti (intero) e il docente (stringa), possiamo rappresentare tutti gli insegnamenti in un vettore. Per ogni studente invece conserveremo il nome (stringa), cognome (stringa), matricola (stringa) e la lista degli insegnamenti (riferimenti) che lo studente ha superato.
Useremo una funzione membro nella classe studente che calcola il numero di crediti che lo studente ha superato. Possiamo rappresentare l'insieme degli studenti in un vettore.

- Implementare le strutture dati per rappresentare le informazioni
- Implementare la funzione **istogramma_crediti()** che riceve in ingresso l'insieme degli studenti e calcola l'istogramma del numero di crediti superati dall'insieme dei nostri studenti. Visto che ogni studente non puo' superare piu' di 180 crediti l'istogramma e' un vettore di 181 elementi in cui in ogni posizione *k* c'e' il numero di studenti che ha superato k crediti. Ad esempio istogramma[0] contiene il numero di studenti che hanno superato 0 crediti, istogramma[9] contiene il numero di studenti che hanno superato 9 crediti, istogramma[18] contiene il numero di studenti che hanno superato 18 crediti, etc..
  NOTA: basta scorrere l'insieme degli studenti UNA SOLA VOLTA per calcolare l'istogramma
- Implementare un programma driver che istanzia le strutture dati con dati di esempio e invoca la funzione **istogramma_crediti()**.
- **Facoltativo**: ordinare il vettore degli studenti in base al numero di crediti superato.

Consegnare una cartella in formato compresso del tipo `CognomeNome.zip` (prima il cognome e poi il nome senza spazi!). Il file zip contiene una cartella `CognomeNome` con il codice sorgente ed il relativo Makefile. Tutto il codice deve essere opportunamente documentato. **Non consegnare programmi che non compilano correttamente**.

Compito di Programmazione 24/11/2021

Tempo a disposizione 3 ore

Un'azienda di sviluppo software intende gestire i propri progetti. Ogni **progetto** ha un nome (stringa), un committente (stringa), un importo (float), una data di inizio e una data di consegna del progetto. Il progetto contiene inoltre una lista di **tasks**. Una task contiene il nome della task, la data di scadenza, il numero di ore uomo (intero), il riferimento al programmatore responsabile di quel task. La lista dei task e' **mantenuta** ordinata per data di scadenza. Per mantenuta intendiamo che ogni volta che si inserisce un task ad un progetto esso va inserito nell'ordine corretto (quindi non bisogna ordinare dopo aver inserito tutti i task). Per ogni programmatore manteniamo il nome, ed il costo orario. I programmatori sono contenuti in un vettore ordinato per costo.

- Implementare le classi per rappresentare le varie informazioni, con le opportune funzioni membro
- Implementare la funzione **workloadDeveloper()** che restituisce la somma delle ore uomo dei task di cui quel programmatore e' responsabile.

Consegnare una cartella in formato compresso del tipo `Matricola_CoognomeNome.zip` (es. `N46XXXXX_RossiMario.zip`). Il file zip contiene una cartella `Matricola_CoognomeNome` con il codice sorgente ed il relativo Makefile. Tutto il codice deve essere opportunamente documentato. **Non consegnare programmi che non compilano correttamente**.

Compito di Programmazione 13/1/2022
Tempo a disposizione 3 ore

Un'agenzia di spedizioni deve gestire le consegne settimanali dei pacchi. Ogni pacco ha un mittente, un destinatario, una data di consegna e un peso in kilogrammi. Per ogni pacco l'agenzia riceve un compenso base pari a 1 euro per ogni chilogrammo. Esistono poi dei pacchi relativi alle spedizioni extra-europee che oltre alle informazioni di base contengono la Nazione di spedizione e il continente. Per questi pacchi l'agenzia oltre al compenso base riceve €10 per le spese di dogana dei pacchi provenienti dagli Stati Uniti e €15 per tutti gli altri continenti.
Manteniamo i pacchi di questa settimana in una lista. La lista e' mantenuta in maniera ordinata in base alla data di consegna. Cioe' ogni volta che si aggiunge un pacco alla lista, esso viene inserito nella posizione corretta (non bisogna ri-ordinare la lista ad ogni aggiunta!).

Implementare:

- Le strutture dati per la rappresentazione delle informazioni relative alle consegne dell'agenzia
- La funzione `weekly_earnings()` che a partire dalla lista di pacchi calcola la sommatoria dei comprensi
- Un programma driver che istanzia un insieme di pacchi di varie tipologie, le inserisce nella lista e invoca la funzione `weekly_earnings()`

Consegnare una cartella in formato compresso del tipo `Matricola_CoognomeNome.zip` (es. `N46XXXXX_RossiMario.zip`). Il file zip contiene una cartella `Matricola_CoognomeNome` con il codice sorgente ed il relativo Makefile. Tutto il codice deve essere opportunamente documentato. **Non consegnare programmi che non compilano correttamente**.

Compito di Programmazione 16/2/2022
Tempo a disposizione 3 ore

Una fabbrica di mobili che produce tavoli deve mantere l'inventario dei piani di appoggio. Ogni piano di appoggio porta delle informazioni sul colore (codice intero), sul tipo di legno (stringa), sul forintore (stringa), il prezzo (float) per centimetro quadrato. I piani di appoggio possono essere di tre tipi rotondi, rettangolari o triangolari. Quelli rettangolari avranno una base e un'altezza, cosi' come quelli triangolari, mentre per quelli rotondi dovremo rappresentare il raggio.
L'elenco dei piani di appoggio e' mantenuto in una lista ordinata per area (funzione membro della classe) del piano di appoggio. Ogni volta che si produce un nuovo piano esso viene aggiunto alla lista nella posizione opportuna.

Implementare:

- Le strutture dati per la rappresentazione delle informazioni relative ai piani di appoggio
- La funzione `insertTable()` che inserisce un piano di appoggio nella lista
- La funzione `summarizeWharehouse()` che calcola la somma dei prezzi di tutti i piani di appoggio della lista

Consegnare una cartella in formato compresso del tipo `Matricola_CoognomeNome.zip` (es. `N46XXXXX_RossiMario.zip`). Il file zip contiene una cartella `Matricola_CoognomeNome` con il codice sorgente ed il relativo Makefile. Tutto il codice deve essere opportunamente documentato.

Domanda **1**

Risposta non data

Punteggio max.: 1,00

L'Agenzia Europea dei Biscotti (EBA) vuole catalogare in gruppi tutti i tipi di biscotti prodotti in Europa in base alle loro caratteristiche.

Un tipo di biscotto ha un identificativo (potrebbe essere un intero fra 0 e N-1, dove N e' il numero di tipi di biscotto), un nome (stringa), un produttore (stringa) ed un vettore di 10 caratteristiche numeriche (float) che ne descrivono le proprietà che sono colore, peso, %grassi, contenuto in proteine, e altro.

Le informazioni sono contenute in un file di testo (europeanBiscuits.txt) contenente sulla prima riga il numero totale di biscotti e sulle righe successive le informazioni su un tipo di biscotti, es:

15

0 Gocciole Barilla 0.1 0.35 ... 12.3

1 Krumiri Bistefani 0.05 0.34 ... 3.2

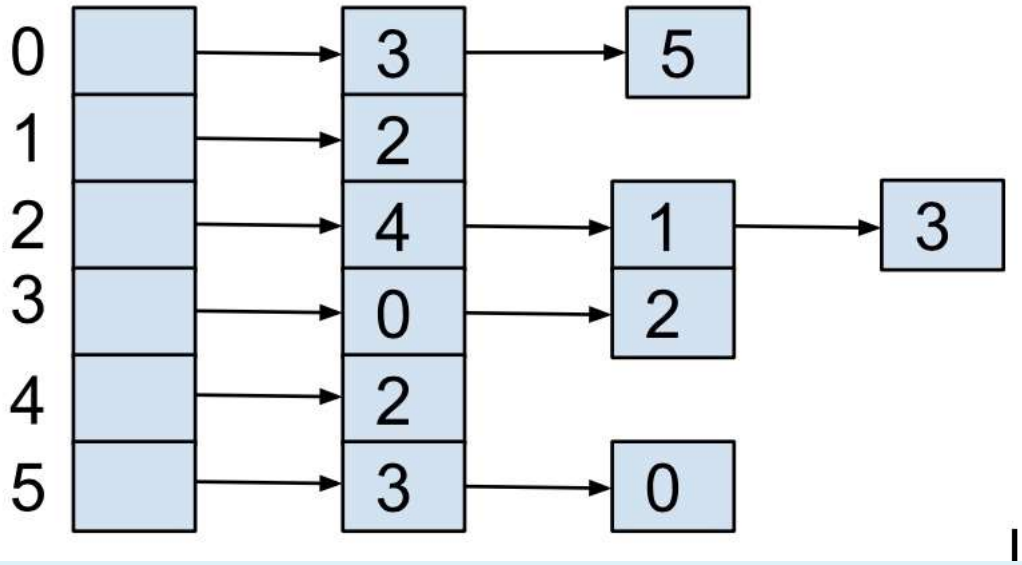
....

14 Macine Barilla 0.1 0,2 ... 1.3

I biscotti vengono mantenuti in un vettore.

Per la classificazione viene usato un parametro (float) Epsilon, due biscotti sono Epsilon-simili se la distanza euclidea fra i loro vettori delle caratteristiche e' minore di Epsilon (la distanza euclidea e' la radice quadrata della somma dei quadrati delle differenze).

Il **grafo della similarità** è un vettore di N liste (di interi) in cui nella lista i-ma ci sono gli ID dei biscotti Epsilon-simili al biscotto i-mo. Es:



Nota che la relazione e' simmetrica se A e' Epsilon-simile a B allora B è Epsilon-simile ad A.

Implementare le strutture dati per rappresentare l'insieme dei biscotti e il grafo della Epsilon-similarità.

Scrivere un programma che

- **Legge** i dati dal file
- **Costruisce** il grafo di similarità in base ai dati letti in ingresso
- **Stampa** per ogni tipo di biscotto il numero di Epsilon-simili.

Facoltativo: scrivere una funzione *cluster* che riceve in ingresso l'id di un biscotto e stampa tutti i biscotti raggiungibili da esso attraverso il grafo di similarità. Diciamo che un elemento B è raggiungibile da un elemento A se esiste una sequenza di biscotti $B_0=A, B_1, B_2 \ldots B_k=B$ tali che B_i è Epsilon-simile a B_{i+1}.

Suggerimento: è facile implementarla con una funzione ricorsiva che usa un array statico di N flag che marca gli elementi gia' visitati e scorre la lista dell'elemento iniziale ricevuto in ingresso appena trova un elemento non ancora visitato lo stampa e visita la lista del nuovo elemento.

Consegnare una cartella in formato compresso del tipo Matricola_CognomeNome.zip (es. N46XXXXX_RossiMario.zip). Il file zip contiene una cartella Matricola_CognomeNome con il codice sorgente ed il relativo Makefile. Tutto il codice deve essere opportunamente documentato.

Iniziato	Tuesday, 19 July 2022, 14:35
Stato	Completato
Terminato	Tuesday, 19 July 2022, 17:06
Tempo impiegato	2 ore 30 min.
Valutazione	Non ancora valutato

Domanda **1**

Completo

Punteggio max.: 1,00

Contrassegna domanda

Vogliamo rappresentare i rapporti sociali (amicizia) in una comunità di N persone. Ogni persona e' rappresentata con identificativo (un intero da 0 a N-1), un nome, un cognome, indirizzo (stringa), eta' (intero).

Manteniamo l'insieme delle persone in un vettore P in maniera tale che la persona con identificativo i è memorizzato nella posizione i-ma del vettore. Le informazioni sulle amicizie sono contenute in un file (friends.txt) organizzato con una coppia di interi su ogni riga (es.):

0 3
4 7
1 2
...

Ogni coppia (x,y) rappresenta l'amiciza fra x e y, e simmetricamente fra y e x, 0<= x, y <= N-1. Data una comunità di N persone, la *matrice di adiacenza* delle amicizie è una matrice A (simmetrica), di dimensioni NxN, di 0 e 1, un cui A[i][j] = 1 rappresenta che la persona con identificativo i e' amico della persona con identificativo j. A[i][j] = 0 invece rappresenta che le due persone i e j non hanno un rapporto di amicizia.

Implementare le strutture dati per rappresentare la comunità delle N persone e la matrice di adiacenza delle amicizie.

Scrivere un programma che:

- **Inizializza**, il vettore delle persone nella comunità
- **Legge** le amicizie dal file
- **Costruisce** la matrice di adiacenza delle amicizie
- **Stampa** per ogni persona l'età media dei suoi amici.