Corso di Algoritmi 2 - Prova scritta: quesito di programmazione

In una situazione in cui lo spazio per la memorizzazione è limitato, si vuole un software per determinare quali record da alcuni database memorizzare e quali invece ricostruire ogni volta che servono, in modo da minimizzare il tempo necessario per ricostruire i record non memorizzati (ovvero per massimizzare il tempo di ricostruzione dei record che vengono memorizzati) . Per ciascun database è noto lo spazio che occupa e il tempo necessario per ricostruire complessivamente il database. Si sa che la ricostruzione parziale di un database è possibile e che il tempo necessario è proporzionale al numero di record da ricostruire (cioè se il tempo per ricostruire l'intero database è T e i record in tutto sono R, il tempo per ricostruire ciascun record è T/R e per ricostruire K di questi record è K*T/R).

Scrivete una classe Java **DB** con il costruttore:
public DB(int[] *dim*, double[] *time*), dove *dim* è un array che contiene le dimensioni dei database in termini di record e *time* contiene invece il tempo necessario per ricostruire ciascun database. Il numero totale di database è dato da *dim.length.* Dovete supporre che sia *dim.length* = *time.length* e che tutti i valori dei due array siano positivi.
Nella classe, implementate il metodo:
public double *timeToRebuild*(int *memSpace)
dove *memSpace* è lo spazio di memoria complessivo a disposizione; se *memSpace* è maggiore o uguale a zero, il metodo restituisce il tempo necessario per ricostruire i record che non sono stati scelti; se *memSpace* è minore di zero, solleva una java.lang.IllegalArgumentException.

Esempi :
dim = {70, 12, 3}, *time* = {10, 2, 0.3}, ***timeToRebuild* (3)** =11.8*;*
dim = {3, 2, 4}, *time* = {2, 5, 1}, ***timeToRebuild* (10)** = 0*;*
dim = {4, 3, 4}, *time* = {5 , 3, 1}, ***timeToRebuild* (10)** =8.75*;*
dim = {5,8,3}, *time* = {3,11,9}, ***timeToRebuild* (-1)** lancia una **IllegalArgumentException.**

TEST: Lavorate implementando anche una classe test JUnit con almeno due test **significativi**, va bene anche se sono semplici.

Struttura del progetto Java e consegna: La classe richiesta e la classe test devono essere contenute in un package il cui nome è **il vostro nomecognome** (senza punti né spazi). Consegnate l'intero package.

Suggerimenti e commenti generali per tutti i progetti:
0) Dovete risolvere il problema rigorosamente applicando uno degli algoritmi visti a lezione.
1) Potete consultare la documentazione Java: http://master.edu-al.unipmn.it/docs/TIJ/TIJ3.htm per il linguaggio e http://master.edu-al.unipmn.it/docs/java2se/docs-1.8/api/index.html per le API.
2) Se serve, potete usare la libreria graphLib.jar (che sarà comunque sempre fornita). Alcune note:
- è allegato un file con le istruzioni per visualizzare i commenti Javadoc per i metodi;
- la sintassi per la descrizione di un grafo nella libreria graphLib è del tipo: "3; 0 1 4; 1 2 5" dove 3 è il numero dei nodi, (0,1) e (1,2) sono archi di peso 4 e 5 rispettivamente;
- osservate che il metodo **union** delle implementazioni di UnionFind che trovate nella libreria vuole come argomenti *dei rappresentanti* di insiemi, non degli elementi qualunque.

3) Potete aggiungere tutti i metodi private che volete.
4) Selezionate la cartella "Consegna" come vostro workspace; in questo modo, in caso di problemi tecnici, il vostro lavoro non andrà completamente perduto, e comunque non rischiate di dimenticarvi di consegnare (se invece doveste decidere di ritirarvi, basta dirlo).
5) Verranno valutatati: l'algoritmo utilizzato, la correttezza e l'aderenza alle specifiche (comprese la gestione degli input particolari e l'inizializzazione), la presenza di una classe test con test significativi.