CS 6301.502. Implementation of advanced data structures and algorithms

Fall 2017

Long Project 6: Multi-dimensional search

Thu, Nov 9, 2017

Version 1.0: Initial description (Thu, Nov 9).

Due: 1st deadline: 11:59 PM, Sun, Nov 26. Final deadline: 11:59 PM, Sun, Dec 3.

Max marks: 100. Maximum excellence credit: 10.

Criteria for allocation of EC: 1st deadline submission, good design,

good-quality code, correct outputs, fast RT.

First 15 projects can earn up to 10 EC each.

Other projects meeting above criteria can earn up to 5 EC.

Implement the following operations. Starter code is provided.

Do not change the name of the class or move it away from cs6301/gXX.

You can keep other source files in subfolders, if you wish to do so.

Do not change the signatures of public methods in the starter code.

Multi-dimensional search: Consider the web site of a seller like Amazon.

They carry tens of thousands of products, and each product has many

attributes (Name, Size, Description, Keywords, Manufacturer, Price, etc.).

The search engine allows users to specify attributes of products that

they are seeking, and shows products that have most of those

attributes. To make search efficient, the data is organized using

appropriate data structures, such as balanced trees. But, if products

are organized by Name, how can search by price implemented efficiently?

The solution, called indexing in databases, is to create a new set of

references to the objects for each search field, and organize them to

implement search operations on that field efficiently. As the objects

change, these access structures have to be kept consistent.

We have a set of items available for purchase. Each item is

identified by an id (Long), and has a description (one or more Longs).

There are a number of suppliers, and each supplier is identified by a

vendor id (Long). Each supplier charges a price for each item they

sell (int). The following operations are supported:

a. add(id, description): add a new item. If an entry with the same

id already exists, the new description is merged with the

existing description of the item.

Returns true if the item is new, and false otherwise.

b. add(supplier, reputation): add a new supplier (Long) and their

reputation (float in [0.0-5.0], single decimal place). If the

supplier exists, their reputation is replaced by the new value.

Return true if the supplier is new, and false otherwise.

c. add(supplier, Pairs(id, price)): add products and their prices

at which the supplier sells the product. If there is an entry

for the price of an id by the same supplier, then the price is

replaced by the new price. Returns the number of new entries

created.

d. description(id): return an array with the description of id.

Return null if there is no item with this id.

e. findItem(arr): given an array of Longs, return an array of items

whose description contains one or more elements of the array,

sorted by the number of elements of the array that are in the

item's description (non-increasing order).

f. findItem(n, minPrice, maxPrice, minReputation): given a Long n,

return an array of items whose description contains n, which

have one or more suppliers whose reputation meets or exceeds the

given minimum reputation, that sell that item at a price that

falls within the price range [minPrice, maxPrice] given. Items

should be sorted in order of their minimum price charged by a

supplier for that item (non-decreasing order).

g. findSupplier(id): given an id, return an array of suppliers who

sell that item, ordered by the price at which they sell the item

(non-decreasing order).

h. findSupplier(id, minReputation): given an id and a minimum

reputation, return an array of suppliers who sell that item,

whose reputation meets or exceeds the given reputation. The

array should be ordered by the price at which they sell the item

(non-decreasing order).

i. identical(): find suppliers selling 5 or more products, who have

the same identical profile as another supplier: same reputation,

and, sell the same set of products, at identical prices. This

is a rare operation, so do not do additional work in the other

operations so that this operation is fast. Creative solutions

that are elegant and efficient will be awarded excellence credit.

Return array of suppliers satisfying above condition. Make sure

that each supplier appears only once in the returned array.

j. invoice(arr, minReputation): given an array of ids, find the

total price of those items, if those items were purchased at the

lowest prices, but only from sellers meeting or exceeding the

given minimum reputation. Each item can be purchased from a

different seller.

k. purge(maxReputation): remove all items, all of whose suppliers

have a reputation that is equal or lower than the given maximum

reputation. Returns an array with the items removed.

l. remove(id): remove item from storage. Returns the sum of the

Longs that are in the description of the item deleted (or 0, if

such an id did not exist).

m. remove(id, arr): remove from the given id's description those

elements that are in the given array. It is possible that some

elements of the array are not part of the item's description.

Return the number of elements that were actually removed from

the description.

n. removeAll(arr): remove the elements of the array from the

description of all items. Return the number of items that lost

one or more terms from their descriptions.

Implement the operations using data structures that are best suited

for the problem. It is recommended that you use the data structures

from Java's library when possible.