Amazon | OA 2020 | Fetch Items To Display

LEETCODE : <https://leetcode.com/discuss/interview-question/823159/>

https://leetcode.com/discuss/interview-question/823159/

Amazon's website contains one to many items in each page. To mimic the logic of the website, an Amazon programmer has a list of items and each item has its name, relevance and price. After sorting the items by (name: 0, relevance: 1, price: 2), the programmer is trying to find out a list of items displayed in a chosen page.

Given a list of items, the sort column, the sort order (0: ascending, 1: descending), the number of items to be displayed in each page and a page number, write an algorithm to determine the list of item names in the specified page while respecting the item's order (Page number starts at 0).

Input

The input to the function/method consists of six arguments:

numOfltems, an integer representing the number of items;

items, a map of string as key representing the name and pair of integers as values representing the relevance, price.

sortParameter, an integer representing the value used for sorting (0 for name, 1 for relevance, 2 for price)

sortOrder, an integer representing the order of sorting (0 for ascending order and 1 descending order);

itemsPerPage, an integer representing the number of items per page;

pageNumber, an integer representing the page number.

Output

Return a list of strings representing the item names on the requested page in the order they are displayed.

Constraints

1 <= numOfitems < 10^5

0 <= relevance, price < 10^8

0 <= pageNumber < 10

Note

itemsPerPage is always greater than 0, and is always less than the minimum of numOfltems and 20.

Example

Input:

numOfltems = 3

items = [["item1", 10,15], ["item2",3,4]. ["item3", 17, 8]]

sortParameter=1

sortOrder = 0

itemsPerPage=2

pageNumber=1

Output:

["item3"]

Explanation:

There are 3 items.

Sort them by relevance(sortParameter = 1) in ascending order ( items = [["item2", 3, 4], ["item1", 10,15], ["item3", 17, 8]]).

Display up to 2 items on each page.

The page 0 contains 2 item names ["item2", "item1"] and page 1 contains only 1 item name ["item3"].

So, the output is "item3".

**Signature:**

List<String> fetchItemsToDisplay(int numOfItems, HashMap<String, PairInt> items, int sortParameter, int sortOrder, int itemsPerPage, int pageNumber);

Ans :

import java.util.\*;

public class FetchItemsToDisplay {

public static void main(String[] args) {

int numOfItems = 3;

HashMap<String, int[]> items = new HashMap<>();

items.put("item1", new int[]{10, 5});

items.put("item2", new int[]{3, 4});

items.put("item3", new int[]{17, 8});

items.put("item5", new int[]{1, 2});

items.put("item7", new int[]{12, 15});

int sortParam = 1;

int sortOrder = 1; //0 is Asc and 1 is Desc

int itemsPerPage = 2;

int pageNumber = 1;

FetchItemsToDisplay f = new FetchItemsToDisplay();

List<String> res = f.fetchItemsToDisplay(numOfItems, items, sortParam, sortOrder, itemsPerPage, pageNumber);

System.out.println("List of items is: " + res);

// Output: List of items is: [item1, item2]

// Since the list is expected to be in descending order on the relevance value and needs items on Page1

}

public List fetchItemsToDisplay(int numOfItems, HashMap<String, int[]> items, int sortParameter,

int sortOrder, int itemsPerPage, int pageNumber) {

PriorityQueue<DisplayItems> pq = new PriorityQueue<>();

if (sortOrder == 1)

pq = new PriorityQueue<>(Collections.reverseOrder());

//Since the int array has ["relevance\_value", "price\_value"]

//If the sort parameter is 1, I am picking 0th value as my key in the pq else the 1st value (sortParameter - 1)

for (Map.Entry<String, int[]> map : items.entrySet()) {

pq.add(new DisplayItems(map.getValue()[sortParameter - 1], map.getKey()));

}

List<String> result = new ArrayList<>();

while (!pq.isEmpty()) {

result.add(pq.peek().itemName);

pq.poll();

}

//Fetching the items on the given pageNumber

int startIndex = pageNumber \* itemsPerPage;

int endIndex = (startIndex + itemsPerPage) > result.size() ? result.size() : startIndex + itemsPerPage;

return result.subList(startIndex, endIndex);

}

public class DisplayItems implements Comparable<DisplayItems> {

private String itemName;

private Integer value;

public DisplayItems(Integer value, String itemName) {

this.itemName = itemName;

this.value = value;

}

public String getItemName() {

return itemName;

}

public Integer getValue() {

return value;

}

@Override

public int compareTo(DisplayItems o) {

return this.getValue().compareTo(o.value);

}

}

}

Another attempt using expression tree and lambda expression.

public class FetchItemsToDisplay1 {

public void RunExample () {

var numOfltems = 3;

var items = new Dictionary<string, Tuple<int, int>>{

{"item1", new Tuple<int,int>(10, 15)},

{"item2", new Tuple<int,int>(3, 4)},

{"item3", new Tuple<int,int>(17, 8)}

};

var sortParameter = 1;

var sortOrder = 0;

var itemsPerPage = 2;

var pageNumber = 1;

var products = GetItemsToDisplay(numOfltems, items, sortParameter, sortOrder, itemsPerPage, pageNumber);

foreach(var product in products)

{

Console.Write(" " + product);

}

}

public List<string> GetItemsToDisplay(

int numOfItems,

Dictionary<string, Tuple<int, int>> items,

int sortParameter,

int sortOrder,

int itemsPerPage,

int pageNumber)

{

var products = items.Select(x => new StoreProduct

{

Name = x.Key,

Relevence = x.Value.Item1,

Price = x.Value.Item2

}).AsQueryable();

var type = typeof(StoreProduct);

var OrderByParam = type.GetProperties()[sortParameter];

var property = type.GetProperty(OrderByParam.Name);

var orderByAddress = products.OrderBy(x => property.GetValue(x));

var parameter = Expression.Parameter(type, "x");

var propertyAccess = Expression.MakeMemberAccess(parameter, property);

var orderByExpression = Expression.Lambda(propertyAccess, parameter);

string command = sortOrder == 1 ? "OrderByDescending" : "OrderBy";

var resultExpression = Expression.Call(

typeof(Queryable),

command,

new Type[] { type, property.PropertyType },

products.Expression,

Expression.Quote(orderByExpression));

var OrderedProducts = products.Provider.CreateQuery<StoreProduct>(resultExpression);

return OrderedProducts.Select((y, index) => new Tuple<string, int>(y.Name, index))

.GroupBy(x => x.Item2 / itemsPerPage)

.Where(x => x.Key == pageNumber)

.SelectMany(x => x.Select(x => x.Item1))

.ToList();

}

public class StoreProduct

{

public string Name { get; set; }

public int Relevence { get; set; }

public int Price{ get; set; }

}

}