Algoritmusok/Adatszerkezetek
Mohó Beadandó

Fájl leírás

Mivel IntelliJ-t használtam ezért a kód kap, csomó plusz file-t köztük egy .idea-t verzió kezeléshez, egy out mappa, amiben a gépi kód található, valamint a Code.iml ami a maven tartozékokat tárolja, ami a unit tesztekhez tartalmaz library-ket. Az src mappán belül található maga a program a JUnit teszt kód, így próbáltam kicsit profibbá tenni a kódomat, sajnos nem tökéletes, de a célnak megfelel, emellett található a 19. Feladat (Konténer oszlopok) megoldása.

Megoldás menete

A megoldás nagyon egyértelműen a Verem adatszerkezettel volt a leg egyszerűbb és mivel több vermünk (oszlopunk) lehet ezért egy listába fűztem így könnyen lehet egy előre nem ismert mennyiségű veremet létre hozni, úgy, hogy azok rendezettek és könnyen megszámolhatók. A program ránéz a beérkező adatokra majd a kellő információt egy tömbben tárolja és átadja azt egy metódusnak (a hely mennyiségével együtt) majd a tömbön végig haladva 3 fő lehetőséget vizsgál, van-e már oszlop, amibe teheti a számot, ha nincs akkor kreál egyet, amennyiben van megnézi, hogy eltudja-e helyezni bármelyikbe a megadott feltétel alapján, ha a nem tudja akkor létrehozz egy új oszlopot és a tetejére beteszi.

```
MakingRows(N,C[])
        Ciklus E in C[]
                Bekerült:=false
        Ha Üres(L) akkor
                Listába(V)
                L(Verembe(0,E))
        különben
                Ciklus i:=0 amig i <hossz(L)
                        Ha L(tető(i)) >= E és Bekerült = false akkor
                                L(Verembe(i,E))
                                Bekerült := true
                Ciklus vége
                Ha  Bekerült = false akkor
                        Listába(V)
                        L(Verembe(hossz(L),E))
Ciklus vége
Eljárás vége
```

Kód magyarázat

Beolvasás

Mivel szerettem volna, ha a fő biznisz logikám külön metódusban van és mivel a Java Scanner-jével nagyon sok mindent lehet szűrni ezért a main első 15 sorra a konzolon biztosított inputot dolgozza fel és adja oda a megfelelő változóknak. A .skip részen egy Regex-el szűröm ki az összes fölösleges inputott, amit nem tudok felhasználni, majd egy for ciklussal a c nevű tömbbe helyezem az adatokat, ezután a makingRows metódusnak át adja rendelkezésre álló helyet és a beérkezett konténer számokat.

```
public static void main(String... args){
    Scanner scanner = new Scanner(System.in);

    int n = scanner.nextInt();
    scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

    int k = scanner.nextInt();
    scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

    int[] c = new int[n];

    String[] cItems = scanner.nextLine().split( regex: " ");
    scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

    for (int i = 0; i < n; i++) {
        int cItem = Integer.parseInt(cItems[i]);
        c[i] = cItem;
    }
```

Feldolgozás

Itt először is felvesszük az új ans (answer) nevű Integer listánkat, amibe majd a végeredmény második részét helyezzük, alatta Létre hozzuk a saját Veremlistánkat A beérkezett tömb feldolgozása egy foreach ciklussal történik így nem kell adódnunk a tömb teljes hossza miatt. Létre hozzuk a getsIn nevű boolean változót, ami azért felel, hogy az adott elem egy teljesen új verembe kerüljön, ezen felül biztosítja, hogy egy elem nem lesz több veremben, mint kéne. Megnézzük, hogy az első elemnél járunk, vagyis, hogy nem léteznek még vermek a listában, amennyiben ez nem a helyzet akkor végig iterálunk a listának a vermein és megnézzük a legfelső elemeket, hogy nagyobbak-e a jelenlegi értéknél esetleg, hogy az adott elem bekerült-e már valahova. Ha ez nem sikerül akkor a listához adunk egy új vermet és a tetejére tesszük az elemet. Minden esetben, ha az elem elhelyezésre kerül az ans nevű listába rögzítjük a helyét. A foreach ciklus után az amount változót felvesszük, aminek oda adjuk értékül a lista méretett, majd ellenőrizzük, hogy befér-e a helyre, amennyiben nem a metódus egy üres listával és egy hiba üzenettel térvisza, ellenkező esetben ki írja a mennyiséget és visszaküldi a mainbe az eredményt.

```java
static List<Integer> makingRows(int spaceLimit,int[] c) {
    List<Integer> ans = new ArrayList<>();
    Konteneroszlop list =  new Konteneroszlop();

    for(int container : c){
        boolean getsIn = false;
        if(list.cheack()){
            list.addStack();
            list.addElement( stackIndex: 0,container);
            ans.add(1);
        }else{
            for(int i = 0; i < list.numOfStacks(); i++){
                if(list.peekElement(i) >= container && !getsIn){
                    list.addElement(i,container);
                    ans.add(i+1);
                    getsIn = true;
                }
            }
            if(!getsIn){
                list.addStack();
                list.addElement( stackIndex: list.numOfStacks()-1, container);
                ans.add(list.numOfStacks());
            }
        }
    }
    int amount = list.numOfStacks();
    if(amount > spaceLimit){
        List<Integer> error = new ArrayList<>();
        System.out.println("Not Enough Space");
        return error;
    }
    System.out.println(amount);
    return ans;

}
```

Verem létezik-e

A cheack függvény megnézi, hogy van e bármi a verem listában, hogy később tudjuk, hogy az első elemnél tartunk

```java
public boolean cheack() { return this.columns.isEmpty(); }
```

Verem létre hozása

Ez az egész programnak a leg lényegesebb része mert itt hozzuk létre azt az Listát, amiben Vermek vannak, és az addStack függvénnyel tudunk ebbe a listába új vermeket tenni.

```
    List<Stack<Integer>> columns;

Kontineroszlop() { this.columns = new ArrayList<>(); }

void addStack() { this.columns.add(new Stack<>()); }
```

Új elem verembe

Az addElement két fő paramétere a verem indexe és az új elem, amit bele teszünk.

```
void addElement(int stackIndex, int element) { this.columns.get(stackIndex).add(element); }
```

Verem teteje

peekElement a megadott sorszámú verem tétjéével tér vissza, amivel ellenőrizzük, hogy az új elem bele tehető-e.

```
public int peekElement(int stackIndex) { return this.columns.get(stackIndex).peek(); }
```

Vermek száma

A feladat eredményének első feléhez szükséges nomOfStacks visszaadja az oszlopok számát, amit kreálni kell.

```
public  int numOfStacks() { return this.columns.size(); }
```

Kiírás

A makingRows metódusban ki írjuk a kellő minimum oszlopok menyiségét majd az ans listát visza adjuk a results nevű listába, amin foreach-el végig menve egyenkét space-el elválasztva kiírja az eredményt.

```
    List<Integer> results = makingRows(k,c);
    for(int result :results){
        System.out.print(result+" ");
    }
```

JUnit tesztek

A unit tesztek implementálása pusztán azért volt eleinte mert szerettem volna gyakorolni és mivel java környezetben még nem csináltam, de a mester sajnálatos software-es hiánya miatt tökéletesen kipótolja. Mivel maga az oszlopok száma írreleváns az ellenőrzésben, mivel, ha a minta stimmel akkor az eredménynek is muszáj ezért csak a makingRows metódusba viszem be a saját adatim és megnézem, hogy stimmelnek-e azzal, amit elvárok.

```
@Test
public void BasicInputFromPapper() {
    Konteneroszlop make = new Konteneroszlop();
    int testK = 9;
    int[] testC = {6, 3, 4, 3, 8, 1, 2, 7, 3, 5};
    List<Integer> output = Arrays.asList(1, 1, 2, 1, 3, 1, 2, 3, 3, 4);

    Assert.assertEquals(output,make.makingRows(testK,testC));
}

@Test
public void NotEnoughSpace() {
    Konteneroszlop make = new Konteneroszlop();
    int testK = 1;
    int[] testC = {6, 3, 4, 3, 8, 1, 2, 7, 3, 5};
    List<Integer> output = Arrays.asList();

    Assert.assertEquals(output,make.makingRows(testK,testC));
}

@Test
public void makingRowsTest1() {
    Konteneroszlop make = new Konteneroszlop();
    int testK = 3;
    int[] testC = {13,1,5};
    List<Integer> output = Arrays.asList(1, 1, 2);

    Assert.assertEquals(output,make.makingRows(testK,testC));
```