Algoritmi care lucreaza cu tipuri de date definite de utilizator

Problema 1: Definiti o structura de date “EnhancedArray” care sa permita urmatoarele operatii:

- add(int v) -> adaugarea unui nou element v in array
- update(int i, int v) -> actualizarea elementului de pe pozitia i cu noua valoare v
- get(int i) -> returnarea elementului de pe pozitia i
- getMaximum() -> returnarea maximului din array in complexitatea O(1)
- printAllValues() -> afisarea tuturor valorilor din array

Explicatie: Pentru a putea returna maximul din array in O(1) atunci putem sa tinem maximul intro variabila pe care o actualizam de fiecare data cand adaugam si actualizam un element din array, iar cand este nevoie de acel maxim, returnam doar valoarea acelei variabile.

Nume fisier: enhancedArray.h

```
#ifndef ENHANCEDARRAY_ENHANCEDARRAY_H
#define ENHANCEDARRAY_ENHANCEDARRAY_H
#include <climits>

const int MAX_ARR_SIZE = 1000;

class enhancedArray {

public:
   int arr[MAX_ARR_SIZE]{};
   int maxVal = INT_MIN;

   void add(int);
   void update(int, int);
   int get(int);
   int getMaximum();

   void printAllValues();

private:
   int index = 0;
};


#endif
```

Nume fisier enhancedArray.cpp

```cpp
#include "enhancedArray.h"
#include <stdexcept>
#include <iostream>

using namespace std;

void enhancedArray::add(int v) {
   arr[index++] = v;
   if (v > maxVal) {
       maxVal = v;
   }
}

void enhancedArray::update(int i, int v) {
   if (i >= index) {
       throw invalid_argument("Array index out of bounds");
   }
   arr[i] = v;
   if (v > maxVal) {
       maxVal = v;
   }
}

int enhancedArray::get(int i) {
   if (i >= index) {
       throw invalid_argument("Array index out of bounds");
   }

   return arr[i];
}

int enhancedArray::getMaximum() {
   return maxVal;
}

void enhancedArray::printAllValues() {
   cout << "Array = [ ";
   for (int i = 0; i < index; i++) {
       cout << arr[i];
       if (i != index -1) {
           cout << ", ";
       } else {
           cout << " ]\n";
       }
   }
}
```

Nume fisier main.cpp

```cpp
#include <iostream>
#include "enhancedArray.h"

using namespace std;
```

```
int main() {
   enhancedArray enhancedArr;
   enhancedArr.add(1);
   enhancedArr.add(2);
   enhancedArr.add(3);
   enhancedArr.add(2);
   enhancedArr.add(7);
   enhancedArr.add(10);
   enhancedArr.add(6);
   enhancedArr.add(1);
   enhancedArr.add(9);

   cout << "arr[7]=" << enhancedArr.get(7) << "\n";
   cout << "Maximum in O(1) is: " << enhancedArr.getMaximum() << "\n";
   cout << "Array is: ";
   enhancedArr.printAllValues();
   enhancedArr.update(7, 70);

   cout << "Maximum in O(1) is: " << enhancedArr.getMaximum() << "\n";

   cout << "Array is: ";
   enhancedArr.printAllValues();

   return 0;
}
```

Output:

arr[7]=1
Maximum in O(1) is: 10
Array is: Array = [1, 2, 3, 2, 7, 10, 6, 1, 9]
Maximum in O(1) is: 70
Array is: Array = [1, 2, 3, 2, 7, 10, 6, 70, 9]

Problema 2: Definiti o structura de date “SparseArray” care sa permita lucrul cu array-uri foarte mari, care contin foarte multe valori de 0 (zero) si foarte putine valori diferite de 0 (zero). Valorile sunt de tip intreg. Aceasta structura de date, “SparseArray”, ar trebui construita in asa fel incat sa fie cat mai eficienta din punct de vedere al consumului de memorie si ar trebui sa permita urmatoarele operatii:

- add(v) -> adaugarea unui nou element v in array
- update(i, v) -> actualizarea elementului de pe pozitia i cu noua valoare v
- get(int i) -> returnarea elementului de pe pozitia i
- remove(i) -> stergerea elementului de pe pozitia i
- size() -> returneaza lungime totala a array-ului
- printElements() -> afisarea tuturor valorilor din array care sunt tinute in memorie

Explicatie: Pentru ca amprenta asupra memoriei sa fie cat mai mica pentru un astfel de array, putem sa facem urmatoarele lucruri:

1. Memoram doar valorile diferite de 0 impreuna cu pozitiile pe care acestea apar.
2. In momentul in care dorim sa accesam o anumita valoare de pe o anumita pozitie, putem sa verificam daca avem salvat in memorie ceva pentru pozitia respectiva si sa returnam acea valoare, iar daca nu avem nimic in memorie pentru pozitia respectiva, atunci returnam 0.

Nume fisier sparseArray.h

```
#ifndef SPARSEARRAY_SPARSEARRAY_H
#define SPARSEARRAY_SPARSEARRAY_H

#include <unordered_map>
using namespace std;

class sparseArray {

public:
   // ne folosim de un "unordered_map" care ne permite sa accesam elementele de pe
anumite chei in O(1)
   unordered_map<int, int> arr;

   void add(int);
   int get(int);
   void update(int, int);
   void remove(int);
   int size();
   void printElements();

private:
   int arrSize = 0;

   void validate(int);
};


#endif
```

Nume fisier: sparseArray.cpp

```
#include <iostream>
#include "sparseArray.h"
#include <stdexcept>
```

```
void sparseArray::add(int v) {
   if (v != 0) {
       // tinem in memorie doar elementele care sunt diferite de 0
       arr[arrSize] = v;
   }
   arrSize++;
}

int sparseArray::get(int i) {
   validate(i);

   int v = arr[i];
   // in momentul in care trebuie sa returnam un numar de pe o anumita pozitie,
   // verificam daca exista o valoare diferita de zero care este tinuta in memorie pe
   // pozitia respectiva si o returnma, iar daca nu exista, atunci returnam 0.
   if (v != 0) {
       return v;
   } else {
       return 0;
   }
}

void sparseArray::update(int i, int v) {
   validate(i);

   // in momentul in care dorim sa actualizam o valoare de pe o anumita pozitie,
verificam
   //  1. daca noua valoare este diferita de 0, atunci o punem in memorie;
   //  2. daca noua valoare este 0, atunci verificam daca exista o valoare memorata
deja pe pozitia i,
   //        iar daca exista, atuni o stergem
   if (v != 0) {
       arr[i] = v;
   } else {
       if (arr[i] != 0) {
           arr.erase(i);
       }
   }
}

void sparseArray::remove(int i) {
   // verificam daca avem memorata o valoare diferita de 0 pe pozitia i, iar daca da,
atunci o stergem din memorie;
   // daca nu, atunci nu avem ce sa stergem pentru ca valorile de 0 nu le memoram.
Dupa care decrementam
   // arrSize in care tinem lungimea totala a intregului array.
   int v = arr[i];
   if (v != 0) {
       arr.erase(i);
   }
   arrSize--;
}

int sparseArray::size() {
   // variabila arrSize contine lungime totala a intregului array
```

```
    return arrSize;
}

void sparseArray::printElements() {
    cout << "Map size=" << arr.size() << "\n";
    cout << "Array: ";
    for (auto p: arr) {
        cout << "arr[" << p.first << "]=" << p.second << ", ";
    }
    cout << "\n";
}


void sparseArray::validate(int i) {
    if (i >= arrSize) {
        throw invalid_argument("Array index out of bounce.");
    }
}
```

Nume fisier: main.cpp

```
#include <iostream>
#include "sparseArray.h"

using namespace std;
int main() {

    sparseArray sparseArr;

    // generam un array cu mai multe elemente
    for (int i = 1; i <= 1000; i++) {
        int value = 0;
        if (i % 100 == 0) {
            value = i;
        }
        sparseArr.add(value);
    }

    cout << "Array size = " << sparseArr.size() << "\n";

    sparseArr.printElements();

    sparseArr.update(99, 0);
    cout << "After update 99 -> 0\n";
    sparseArr.printElements();

    cout << "After update 3 -> 0\n";
    sparseArr.update(3, 5);
    sparseArr.printElements();

    int shouldBeZero = sparseArr.get(7);
    cout << "shouldBeZero=" << shouldBeZero << "\n";
```

```
    int notZero = sparseArr.get(3);
    cout << "notZero=" << notZero << "\n";

    sparseArr.printElements();

//    Daca am comenta aceste 2 linii de cod am abtine exceptia "Array index out of
bounce."
//    sparseArr.get(1011);
//    sparseArr.update(1011, 223);

    return 0;
}
```

Output:

Array size = 1000
Map size=10
Array: arr[999]=1000, arr[899]=900, arr[799]=800, arr[699]=700, arr[599]=600, arr[499]=500, arr[399]=400, arr[299]=300, arr[199]=200, arr[99]=100,

After update 99 -> 0
Map size=9
Array: arr[999]=1000, arr[899]=900, arr[799]=800, arr[699]=700, arr[599]=600, arr[499]=500, arr[399]=400, arr[299]=300, arr[199]=200,

After update 3 -> 5
Map size=10
Array: arr[999]=1000, arr[899]=900, arr[799]=800, arr[699]=700, arr[599]=600, arr[499]=500, **arr[3]=5**, arr[399]=400, arr[299]=300, arr[199]=200,

shouldBeZero=0
notZero=5