L0 Aplicatii cu structuri de date pentru programatori Java (STL in C++ / Java)

Exemplu cu clasa vector C++

Exemplu: Citirea si afisarea unui vector cu n elemente de tip structura, cu componentele reale r si i.

```
#include <iostream>       // std::cout
#include <vector>
struct Complex {
   float r;
   float i;
}z;
int i, n;
std::vector<Complex> a;
int main ()
{ std::cout<<"\n n = "; std::cin>>n;
  std::cout<<"\n dati perechile de numere: ";
  for (i=0;i<n;i++)
  { std::cout<<"\n z.r = "; std::cin>>z.r;
    std::cout<<"\n z.i = "; std::cin>>z.i;
    a.push_back(z);
}
   std::cout<<"\n vectorul: ";
   for (i=0;i<a.size();i++)
            std::cout<<"("<<a[i].r<<" "<<a[i].i<<")   ";
  return 0;
}
```

Versiune Java (introducere element intr-un vector si afisare vector)

```
import java.util.Vector;
…….
Vector<Complex> a = new Vector<Complex>();
……
Complex element = new Complex();
……
a.add(element);
…..
for (Complex element : a)
            System.out.println(element);
```

Clasa Priority_queue C++

Exemplu: Cititi n numere complexe (partea reala si imaginara) si sortati-le descrescator folosind o structura de date priority_queue.

```
#include <iostream>
#include <queue>
struct Complex {
   float r;
   float i;
}z;

bool operator<(const Complex &a, const Complex &b)
{   //TODO Intoarceti true daca a < b, fals altfel
    if (a.r<b.r) return true;
    else  if(a.r == b.r && a.i<b.i) return true;
            else return false;
}
int i, x, y, n, re, im,j;
std::priority_queue<Complex> pq;
int main ()
{ std::cout<<"\n n = "; std::cin>>n;
  std::cout<<"\n dati n perechi de numere: ";
  for (i=0;i<n;i++)
```

```
  { std::cout<<"\n z.r = "; std::cin>>z.r; std::cout<<"\n z.i = "; std::cin>>z.i;
    pq.push(z);
}
std::cout<<"\n Priority Queue:  \n ";
while (!pq.empty())
  {  z = pq.top(); std::cout<<"("<<z.r<<" "<<z.i<<")  \n";
     pq.pop();
  }
  return 0;
}
```

Versiune Java – functie care supraincarca metoda de comparare a 2 variabile de tip class

```
private static class ComplexComparator implements Comparator<Complex> {
        /**
         * Functie ce compara doua numere complexe: intai compara partea reala
         * si daca ambele numere au aceeasi parte reala, atunci compara partea imaginara
         */
                @Override
                public int compare(Complex a, Complex b) {
                        //TODO Intoarceti un numar pozitiv daca a > b, negativ altfel
                if (a.r<b.r) return -1;
                  else
                        if(a.r == b.r && a.i<b.i) return -1;
                        else return 1;
                }
   }
```

Creare coada de prioritate cu functia de comparare, adaugare element in coada, afisare coada prioritate

```
PriorityQueue<Complex> pq = new PriorityQueue<Complex>(new ComplexComparator());
…..
pq.add(element);
….
while (!pq.isEmpty()) {
   Complex z = pq.poll();
  System.out.println(z);
        }
```

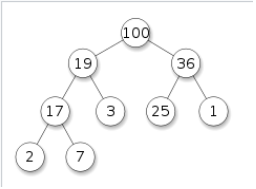
Clasa map C++ ordoneaza elementele dupa cheie; elementele sunt de forma (cheie, valoare)

Exemplu: Cititi n numere complexe (partea reala si imaginara) si sortati-le descrescator folosind o structura de date map si asociati un numar de ordine fiecarui numar complex, ordine care semnifica pozitia initiala la citirea datelor.

```
#include <iostream>       // std::cout
#include <map>
using namespace std;

struct Complex {
   // Partea reala
   float r;
   // Partea imaginara
   float i;
}z;

int i,x,y,n,j=0;

bool operator<(const Complex &a, const Complex &b)
{   //TODO Intoarceti true daca a < b, fals altfel
   if (a.r<b.r) return true;
   else
          if(a.r == b.r && a.i<b.i) return true;
        else return false;
}
```

```
int main ()
{
  std::map<Complex,int> mymap;
  std::map<Complex,int>::iterator it;
  std::cout<<"\n n = "; std::cin>>n;
  for(i=1;i<=n;i++)
  {  std::cout<<"\n x = "; std::cin>>x;
     std::cout<<"\n y = "; std::cin>>y;
     z.r=x; z.i=y;
     mymap[z] = ++j;
}
   // afisare map
  for (it=mymap.begin(); it!=mymap.end(); ++it)
    std::cout << it->first.r << "   " << it->first.i<< " => " << it->second << '\n';
  return 0;
}
```

Heap-uri – structura de date abstracta bazata pe arbori. Heap-ul este o implementare eficienta a unei cozi de prioritate.

Max-heap binar: 

Exemplu: Se dau n numere. Sa se sorteze crescator folosind un Heap.

```
#include <iostream>     // std::cout
#include <algorithm>    // std::make_heap, std::pop_heap, std::push_heap, std::sort_heap
#include <vector>       // std::vector
int main () {
  int a[] = {10,20,30,5,15};
  std::vector<int> v(a, a+5);
  std::cout << " Vectorul initial: ";
  for (unsigned i=0; i<v.size(); i++)    std::cout << ' ' << v[i];
  std::make_heap (v.begin(),v.end());
  std::cout << "\n Heap initial: ";
  for (unsigned i=0; i<v.size(); i++)    std::cout << ' ' << v[i];
  std::cout << "\n Max heap: " << v.front() << '\n';
  std::pop_heap (v.begin(),v.end()); v.pop_back();
  std::cout << " Max heap dupa stergerea varfului: " << v.front() << '\n';
  v.push_back(100); std::push_heap (v.begin(),v.end());
  std::cout << " Max heap dupa inserarea lui 100: " << v.front() << '\n';
  std::sort_heap (v.begin(),v.end());
  std::cout << " Ordine finala, dupa sortare heap: ";
  for (unsigned i=0; i<v.size(); i++)    std::cout << ' ' << v[i];
  std::cout << '\n';  return 0;
}
```

Output:

```
Vectorul initial:  10 20 30 5 15
 Heap initial:  30 20 10 5 15
 Max heap: 30
 Max heap dupa stergerea varfului: 20
 Max heap dupa inserarea lui 100: 100
 Ordine finala, dupa sortare heap:  5 10 15 20 100
```

Set-uri - elementele se memoreaza in ordine crescatoare si sunt unice

Iterators: **begin** Return iterator to beginning (public member function)
end Return iterator to end (public member function)

Capacity: **empty** Test whether container is empty (public member function)
size Return container size (public member function)
Modifiers: **insert** Insert element (public member function) **erase** Erase elements (public member function)
emplace Construct and insert element (public member function **Operations**: **find** Get iterator to element (public member function) **count** Count elements with a specific value (public member function)

Problema: Sortați descrescător elementele unui vector de numere complexe folosind un maxheap. La părți reale egale, criteriul este partea imaginară. Pentru maxheap puteți folosi clasa PriorityQueue în Java sau std::priority_queue în C++. Folosind std::map în C++, sau HashMap în Java, creați o mapare între elementele din vector și poziția lor în șirul sortat. (infoarena; vmchecker)

Fisierul main.cpp

```
#include <iostream>
#include <map>
#include <vector>
#include <queue>
#include "complex.h"
bool operator<(const Complex &a, const Complex &b)
{  //TODO Intoarceti true daca a < b, fals altfel
   if (a.r<b.r) return true;
   else
            if(a.r == b.r && a.i<b.i) return true;
          else return false;
}
std::vector<Complex> get_sorted(const std::vector<Complex> &v)
{   std::vector<Complex> res;
   //TODO Folosind priority_queue, adaugati elementele din v in ordine descrescatoare.
   //  creare priority_queue
   std::priority_queue<Complex> pq;
   for (int i=0;i<v.size();i++)
           pq.push(v[i]);
   //extragere vector sortat din priority_queue
   Complex z;
  while (!pq.empty())
  {     z = pq.top(); res.push_back(z);
           pq.pop();   }
   return res;
}
std::map<Complex, int> get_mapping(const std::vector<Complex> &v)
{    std::map<Complex, int> res;
   //TODO Adaugati in map, pentru fiecare element din v, pozitia sa din vectorul sortat.
   int j = 0;
   for (int i=0;i<v.size();i++)  res[v[i]] = ++j;
   return res;
}
int main(void)
{    std::vector<Complex> v;
   // Citeste date de test.
   v = read_data(".date.in");
   std::cout << "Vectorul initial" << std::endl;
   write_vector(v);
   // Verifica sortarea.
   std::vector<Complex> sorted = get_sorted(v);
   std::cout << "Vectorul sortat" << std::endl;
   write_vector(sorted);
   // Verifica maparea.
   std::cout << "Maparea" << std::endl;
   std::map<Complex, int> mapping = get_mapping(sorted);
   for (unsigned int i = 0; i < sorted.size(); i++)
   {      std::cout << sorted[i] << " e pe pozitia "
```

```
            << mapping[sorted[i]] << std::endl;
    }
    return 0;
}
```

Fisierul complex.h

```
#include <vector>
#include <fstream>
#include <iostream>
#include <fstream>
#include <map>
struct Complex {  float r; // Partea reala
                  float i; // Partea imaginara  };
std::ostream& operator<<(std::ostream &stream, Complex c)
{  stream << "(" << c.r << ", " << c.i << ")";
   return stream;
}
std::vector<Complex> get_sorted(const std::vector<Complex> &v);
std::map<Complex, int> get_mapping(const std::vector<Complex> &v);
bool operator<(const Complex &a, const Complex &b);
std::vector<Complex> read_data(const char* filename)
{ std::ifstream f(filename);
   int n;  std::vector<Complex> res;
   f >> n;      // Citeste numarul de elemente din vector
   // Citeste elementele
   while (n--) { Complex current;
      f >> current.r; f >> current.i; res.push_back(current);
   }
   return res;
}
void write_vector(const std::vector<Complex> &v)
{  for (unsigned int i = 0; i < v.size(); i++) {   std::cout << v[i] << " ";    }
   std::cout << "\n"; }
```

Fisierul .dat.in

```
5
1 3
4 5
4 3
2 1
3 3
```

Versiune Java (NetBeans IDE 8.2)

<u>Fisier MainJava.java</u>

```
package mainjava;
import java.util.Comparator;
import java.util.HashMap;
import java.util.PriorityQueue;
import java.util.Vector;

public class MainJava {

   public static void main(String[] args) {
      Vector<Complex> v;
          /* Citeste date de test. */
      v = Complex.readData(".date.in");
      System.out.println("Vectorul initial:");
      Complex.writeVector(v);
```

```
    /* Verifica sortarea. */
    Vector<Complex> sorted = getSorted(v);
    System.out.println("Vectorul sortat:");
    Complex.writeVector(sorted);

    /* Verifica maparea. */
    HashMap<Complex,Integer> mapping = getMapping(v);
    System.out.println("Maparea:");
    for (Complex element : sorted) {
        System.out.println(String.format("%s e pe pozitia %d", element, mapping.get(element)));
    }
}

private static class ComplexComparator implements Comparator<Complex> {
        /**
         * Functie ce compara doua numere complexe: intai compara partea reala
         * si daca ambele numere au aceeasi parte reala, atunci compara partea imaginara
         */
                @Override
                public int compare(Complex a, Complex b) {
                        //TODO Intoarceti un numar pozitiv daca a > b, negativ altfel
                if (a.r<b.r) return -1;
                  else
                        if(a.r == b.r && a.i<b.i) return -1;
                        else return 1;
                }
}

/**
 * Functie ce sorteaza crescator elementele unui vector de numere complexe
 * @param v             vectorul de numere complexe de intrare
 * @return              vectorul de numere complexe sortat descrescator
 */
private static Vector<Complex> getSorted( Vector<Complex> v ) {
        Vector<Complex> res = new Vector<Complex>();    // vectorul rezultat

        /*TODO Folosind PriorityQueue, adaugati elementele din v in ordine crescatoare.
         * Pentru PriorityQueue folositi comparatorul definit mai sus */
        PriorityQueue<Complex> pq = new PriorityQueue<Complex>(new ComplexComparator());
        for (Complex element : v)
                pq.add(element);
        while (!pq.isEmpty()) {
                 Complex z = pq.poll();
                res.add(z);
        }
    return res;
}

/**
 * Functie ce construieste un HashMap: pentru fiecare element complex atribuie pozitia sa
 * in vectorul sortat
 * @param v             vectorul de elemente complexe, nesortat
 * @return              HashMap care atribuie pentru fiecare element pozitia sa in vectorul sortat
 */
private static HashMap<Complex, Integer> getMapping ( Vector<Complex> v) {
        HashMap<Complex, Integer> res = new HashMap<>();
        /*TODO Adaugati in map, pentru fiecare element din v, pozitia sa in vectorul sortat. */
    int k = 0;
        for (Complex element : v)
              {k++;
```

```
                res.put(element, k);
               }
        return res;
    }
}
```

Fisier Complex.java

```
package mainjava;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Vector;

public class Complex {
        public float r;     // Partea reala
        public float i;     // Partea imaginara
        /**
         * Functie care citeste numere complexe dintr-un fisier
         * @param filename      numele fisierului de intrare
         * @return                      vectorul de numere complexe
         */
        public static Vector<Complex> readData ( String filename ) {
                BufferedReader br = null; // pentru citire din fisier
                int n = 0;                              // numar de elemente
                Vector<Complex> res = new Vector<Complex>();    // vectorul rezultat
                /* you should use try-catch for proper error handling! */
                try {   String sCurrentLine;
                        br = new BufferedReader(new FileReader(filename));
                        /* citim numarul de elemente */
                        sCurrentLine = br.readLine();
                        n = Integer.parseInt(sCurrentLine);
                        /* citim pe rand fiecare element */
                        for (int i=0; i<n; i++) {
                                sCurrentLine = br.readLine();
                                Complex current = new Complex();
                                current.r = Integer.parseInt(sCurrentLine.split(" ")[0]);
                                current.i = Integer.parseInt(sCurrentLine.split(" ")[1]);
                                res.add(current);
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                } finally {
                        /* trebuie sa inchidem buffered reader-ul */
                        try {
                                if (br != null) br.close();
                        } catch (IOException ex) {
                                ex.printStackTrace();
                        }
                }
                return res;
        }

        /** Functie care afiseaza pe ecran elementele unui vector de numere complexe
         * @param v             vectorul de numere complexe
         */
        public static void writeVector ( Vector<Complex> v ) {
                for (Complex element : v) {
                        System.out.println(element);
                }
```

```
        }
        /** Functie care afiseaza pe ecran numarul complex
         */
        @Override
        public String toString() {
                return String.format("(%.2f, %.2f)", r, i);
        }
}
```

……

```
run:
Vectorul initial:
(1.00, 3.00)
(4.00, 5.00)
(4.00, 3.00)
(2.00, 1.00)
(3.00, 3.00)
Vectorul sortat:
(1.00, 3.00)
(2.00, 1.00)
(3.00, 3.00)
(4.00, 3.00)
(4.00, 5.00)
Maparea:
(1.00, 3.00) e pe pozitia 1
(2.00, 1.00) e pe pozitia 4
(3.00, 3.00) e pe pozitia 5
(4.00, 3.00) e pe pozitia 3
(4.00, 5.00) e pe pozitia 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Problema: Sa se creeze un generator de n numere aleatoare (< 2*k). Afisati numerele care respecta conditia ca mediana numerelor sa fie mai mica decat k, unde n si k sunt date. Folositi STL C++.

```
#include <iostream>
#include <map>
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
using namespace std;

int nr, k, med, i, n, x, j,jj, R;
multimap<int,int> map1;
multimap<int,int>::iterator it;

bool median(int &med, int R)
{   //TODO Intoarce true daca mediana numerelor generate < k dat
        map1.insert (pair<int,int>(R, med));
        nr = map1.size();
   cout<<"\n nr "<<nr<<":  ";
   i = 0;
        for (it = map1.begin(); it != map1.end(); ++it)
   if (i < nr/2)
          { i++;
            med = (*it).first;
      }
   if (med < k) return true;
   else return false;
}

int main ()
{   cout<<"\n numarul maxim de elemente generate aleator = "; cin>>n;
  cout<<"\n limita k = "; cin>>k;
  /* initialize random seed: */
  srand (time(NULL));
  j = 0;
```

```
 do
 {
  R = rand() % (2*k) + 1;  cout<<" rand = "<<R;
  if (median(med, R))
   { for (it = map1.begin(); it != map1.end(); ++it)  cout << (*it).first << "  " ;
 cout<<"  med = "<<med<<"\n";
 }
  j++;
}
 while (j<n);
    cout<<"\n generator de numere cu mediana < k: \n";
   cout<<"\n mediana celor "<<n<<" numere = "<<med;
   return 0;
}
```

Probleme

1. Implementati o cautare a unui element intr-un vector care a fost sortat, dar si rotit la dreapta cu un numar oarecaare de pozitii.
2. Avand un vector cu numere pozitive, determinati cea mai mare suma ce poate fi obtinuta prin adunarea numerelor aflate in pozitii non-consecutive. Daca se alege numarul v[i], nu este permisa folosirea lui v[i-1] sau v[i+1]
3. Cum se salveaza si cauta (cu auto-complete, fara corectii automate) peste 1 milion de nume? Solutia cea mai buna este in $O(n^2)$.
4. Determinati 3 elemente dintr-un vector de numere intregi a caror suma este 0. Solutia cea mai buna este in $O(n^2)$.
5. Determinati maximul dintre doua numere intregi fara a folosi if-else si nici operatori de comparatie.

Solutie: Rescriem astfel: a – k*(a-b)

int getMax(int a, int b) { int c = a-b; int k = (c >>31) & 0x1; int max = a – k*c; return max; }

Exista si solutii care folosesc modulo si impartire.

6. Consideram ca avem urmatoarea codificare a caracterelor in numere a-> 1, b->2, …, z->26. Fiind dat un numar intreg, afisati numarul total de traduceri posibile in stringuri conform codificarii date. Ex: 11223
7. Avand 9 bile, dintre care una cantareste mai putin decat celelalte, si o balanta, cum puteti determina care bila este cea mai usoara?
8. Determinati numarul de cifre ‚2’ continute de toate numerele intre 0 si n.
9. Avem un generator de numere aleatoare care trimite fiecare valoare nou generata unei functii ce trebuie sa mentina mediana tuturor numerelor primite pana la orice pas.
10. Determinati cele mai mari 1 milion de numere dintr-un vector cu un miliard de numere.