Lucrarea nr. 8 - Structuri de date de tip liste sortate

O listă sortată reprezintă un aranjament al înregistrărilor din cadrul listei în funcţie de o anumită valoare a unei chei. Pentru operațiile asupra unei astfel de liste se folosesc două metode: **`find()`** şi **`delete()`.** Avantajul unei liste sortate, în cazul ştergerii sau inserării unui element, constă în faptul că nu mai este nevoie să se intervină asupra celorlalte elemente ale listei. Găsirea locației unde noul element trebuie introdus, sau extras, este suficientă, alături de refacerea câmpurilor next.
Spre exemplu metoda de inserție a unei înregistrări va avea următorul cod:

```
public void insert(double key)
{ Link newLink = new Link(key);
Link previous = null;
Link current = first;
while(current != null && key > current.dData)
{  previous = current;
   current = current.next;
}
if(previous==null)
      first = newLink;
else
      previous.next = newLink;
      newLink.next = current;
}
```

După cum se vede în analiza acestei metode sunt 3 zone destinate celor 3 situații particulare:

- Inserția elementului la începutul listei: caz în care **`previous`** ar fi fost **`null`**, **`First`** va fi inițializat cu înregistrarea inserată – **`newLink`**
- Inserția elementului în corpul listei: înregistrarea curentă (**`newLink`**) este legată de înregistrarea anterioară (**`previous.next`**), iar câmpul next al înregistrării inserate este conectat cu înregistrarea existentă (**`current`**)

Eficiența listelor înlănțuite sortate

Inserarea sau ştergerea unor itemi arbitrari în lista sortată necesită O(N) comparări (N/2 la majoritatea), deoarece locația potrivită trebuie găsită parcurgand lista pas cu pas. Totuşi, valoarea minimă poate fi găsită sau ştearsă, în O(1) timpi deoarece este la începutul listei. Dacă o aplicație accesează frecvent itemul minim şi inserarea rapidă nu este imperativă, atunci o listă sortată este o alegere eficientă.

Sortarea prin inserarea în listă

O listă sortată poate fi folosită ca un mecanism eficient de sortare. Să presupunem că aveți un vector de date nesortate. Dacă luați „itemii” din vector şi îi inserați unul câte unul în lista sortată, ei vor fi plasați automat în ordine. Dacă îi scoateți din listă şi îi puneți înapoi în vector, vectorul va fi sortat. Programul

urmator realizeaza acest lucru, începepand prin a insera dintr-un vector itemii nesortați (folosind un constructor) şi apoi îi aşează iar în vector.
Acest lucru se dovedeşte a fi mult mai eficient decât inserarea obişnuită într-un vector. Asta deoarece sunt necesare mai puține copii. Este tot un proces de tip $O(n^2)$ deoarece inserarea fiecarui item presupune compararea unui nou item cu jumătate din itemii care sunt deja în listă şi sunt *n* itemi de inserat, rezultând în $n^2/4$ comparări. Totuşi, fiecare item este copiat decât de doua ori: o dată din vector în listă şi încă o dată din listă în vector. n*2 copii sunt un avantaj față de inserarea sortată într-un vector, unde sunt necesare n^2 copii.

Aplicatie: Proiectul ***SortedListApp*** implementează o listă înlănțuită sortată:

```
import java.io.*;
class Link
{ public double dData;
  public Link next;

public Link(double dd)
{ dData = dd; }

public void displayLink()
{ System.out.print(dData + " "); }
}

class SortedList
{ private Link first;

public SortedList()
{ first = null; }

public boolean isEmpty()
{ return (first==null); }

public void insert(double key)
{ Link newLink = new Link(key);
  Link previous = null;
  Link current = first;
  while(current != null && key > current.dData)
  { previous = current;
    current = current.next;
  }
  if(previous==null)
     first = newLink;
  else
     previous.next = newLink;
     newLink.next = current;
}

public Link remove()
```

```
{ Link temp = first;
first = first.next;
return temp;
}

public void displayList()
{ System.out.print("List (first-->last): ");
Link current = first;
while(current != null)
  { current.displayLink();
    current = current.next;
  }
 System.out.println("");
 }
}

class SortedListApp
{ public static void main(String[] args)
 {SortedList theSortedList = new SortedList();
  theSortedList.insert(20);
  theSortedList.insert(40);
  theSortedList.displayList();
  theSortedList.insert(10);
  theSortedList.insert(30);
  theSortedList.insert(50);
  theSortedList.displayList();
  theSortedList.remove();
  theSortedList.displayList();
 }
}
```

Aplicatie: *Sortarea prin inserarea în listă*

```
import java.io.*;
class Link
{
public double dData; // elementul data
public Link next; // urmatorul element in lista

public Link(double dd) // constructor
{ dData = dd; }
} // sfârşit class Link

class SortedList
{
private Link first; // face referinţă la primul element din listă

public SortedList() // constructor (fara args)
{ first = null; }
```

```
public SortedList(Link[] linkArr) //constructor (array ca argument)
 first = null;; // inițializează lista
 for(int j=0; j<linkArr.length; j++)
  // copiază vectorul
  insert( linkArr[j] ); // în listă
}

public void insert(Link k)
// inserează, în ordine
{
Link previous = null; // începe cu primul item
Link current = first; // si parcurge toata lista

while(current != null && k.dData > current.dData)
 { // or key > current,
     previous = current;
     current = current.next;
   //  trece la următorul item
 }
if(previous==null) // începutul listei
    first = k; // first --> k
else // nu la început
    previous.next = k; // vechiul prev --> k
    k.next = current; // k --> vechiul current
} // sfârşit insert()


public Link remove()
// returnează şi şterge primul element
{
  Link temp = first; // salvează primul element
  first = first.next; // şterge primul element
  return temp; // returnează valoare
}
} // sfârşit SortedList

class ListInsertionSortApp
{
public static void main(String[] args)
{
int size = 10;
// crează vector de linkuri
Link[] linkArray = new Link[size];
for(int j=0; j<size; j++)
// ocupă vectorul cu elemente
{ // numar aleator
  int n = (int)(java.lang.Math.random()*99);
  Link newLink = new Link(n);
  linkArray[j] = newLink; // }
```

```
// afişează vectorul
   System.out.print("Unsorted array: ");
for(int j=0; j<size; j++)
  System.out.print( linkArray[j].dData + " " );
  System.out.println("");
// crează listă nouă,
// iniţializată cu vectorul
SortedList theSortedList=new SortedList(linkArray);
for(int j=0; j<size; j++)
  linkArray[j] = theSortedList.remove();
// afişează vectorul
  System.out.print("Sorted Array: ");
for(int j=0; j<size; j++)
  System.out.print(linkArray[j].dData + " ");
  System.out.println("");
 } // sfârşit main()
} // sfârşit ListInsertionSortApp
```

Acest program afişează valorile vectorului înainte de sortare şi apoi după sortare.

Vector nesortat: 59 69 41 56 84 15 86 81 37 35
Vector sortat: 15 35 37 41 56 59 69 81 84 86

Ieşirea va fi diferită de fiecare dată din cauza valorilor iniţiale care sunt generate la întamplare. Un nou constructor pentru **SortedList** ia un vector de obiecte tip **Link** ca argument şi inserează întregul conţinut al acestui vector într-o listă nou creată. Se face de asemenea o modificare în metoda **insert()**a acestui program. Acum acceptă obiecte `Link` ca argument, spre deosebire de **double**. Facem acest lucru pentru a stoca obiectele Link în vector şi a le insera direct în listă. În programul **sortedList.java,** este mai convenabil să avem metoda
insert()pentru a crea fiecare obiect **Link,** folosind valoarea double ca argument.
Dezavantajul este că ocupă de două ori mai multă memorie: vectorul şi lista trebuie să fie în memorie în acelaşi timp. Totuşi, această metodă este convenabilă atunci când vrem să sortăm vectori care nu sunt prea mari.

Probleme propuse

1. Modificați programul **ListInsertionSortApp** astfel încât să realizați o metodă sortare() care să aranjeze lista în ordine crescătoare (puteți folosi orice metodă de ordonare).
2. Modificați programul `SortedListApp` astfel încât să pentru lucrul cu caractere de tip litere .
3. Realizati un program in care itemii din lista sa fie studentii dintr-o grupa, iar poinetrii utilizati sa fie notele de la disciplina Java. Realizati ordonarea acestora de la nota cea mai mica la nota cea mai mare